Руководство по языку SQL СУБД Firebird 3.0

18 сентября 2015 — v.0001-1 для Firebird 3.0 Beta 2

Спонсоры документации:
Platinum Sponsor

Gold Sponsor

IBSurgeon

Руководство по языку SQL СУБД Firebird 3.0

Над документом работали:

Денис Симонов, Пол Винкенуг, Дмитрий Филиппов, Дмитрий Еманов, Томас Воинк, Александр Карпейкин, Дмитрий Кузьменко, Алексей Ковязин

Редактор:

Симонов Денис

Содержание

Введение

Это руководство описывает язык SQL, поддерживаемый СУБД Firebird 3.0.

В руководстве также приводятся практические примеры использования SQL, многие из которых взяты из реальной практики.

Что содержит данный документ

Данный документ содержит описание языка SQL Firebird. Он охватывает следующие основные области:

- Основные положения;
- Зарезервированные и ключевые слова;
- Типы и подтипы данных;
- Операторы DDL (Data Definition Language — язык создания данных);
- Операторы DML (Data Manipulation Language — язык обращения с данными);
- Операторы управления транзакциями;
- Обработка исключений;
- Операторы PSQL (Procedural SQL — процедурный SQL, используется в хранимых процедурах, триггерах и выполнимых блоках);
- Безопасность и операторы управления доступом;
- Операторы и предикаты (утверждения);
- Агрегатные функции;
- Встроенные функции;
- Коды ошибок и обработка исключительных ситуаций;
- Описание системных таблиц и таблиц мониторинга;
- Наборы символов и соответствующие им порядки сортировки.

Вопросы, не связанные с SQL в данном документе не рассматриваются.

Авторство

В работе над руководством принимали участие:

- Симонов Денис;
- Винкенуг Пол;
- Дмитрий Филиппов;
- Дмитрий Еманов;
- Томас Воинк;
- Александр Карпейкин;
- Алексей Ковязин;
- Дмитрий Кузьменко.

Редакторы – Александр Карпейкин, Дмитрий Кузьменко, Алексей Ковязин, Денис Симонов.

Спонсоры

Платиновым спонсором создания «Руководства по языку СУБД Firebird» является Московская Биржа www.moex.com.

Московская Биржа — крупнейший в России и Восточной Европе биржевой холдинг, образованный 19 декабря 2011 года в результате слияния биржевых групп ММВБ (основана в 1992) и РТС (основана в 1995). Московская Биржа входит в двадцатку ведущих мировых площадок по объёму торгов ценными бумагами, суммарной капитализации торгуемых акций и в десятку крупнейших бирж производных финансовых инструментов.

Золотым спонсором «Руководства по языку СУБД Firebird» является IBSurgeon (iBase.ru) (www.ib-aid.com, www.ibase.ru): техническая поддержка и инструменты разработчика и администратора для СУБД Firebird.

Благодарности

Благодарим Влада Хорсуна, Александра Пешкова, Павла Зотова за помощь в создании этого документа.

Лицензионные замечания

Обновления

Так как СУБД Firebird постоянно развивается, то изменяется и улучшается его документация. Вы можете получить обновлённые версии этого документа и других частей документации по Firebird на русском языке на сайте проекта документации: https://www.assembla.com/spaces/firebird-russian-documentation/wiki

Глава 1

Общие сведения

Подмножества SQL

SQL имеет четыре подмножества SQL, используемых в различных областях применения:

- Динамический SQL (DSQL, Dynamic SQL)
- Процедурный SQL (PSQL, Procedural SQL)
- Встроенный SQL (ESQL, Embedded SQL)
- Интерактивный SQL (ISQL, Interactive SQL)

Динамический SQL является основной частью языка, которая соответствует Части 2 (SQL/Foundation – SQL/Основы) спецификации SQL. DSQL представляет собой конструкции, которые передаются клиентскими приложениями с помощью Firebird API и обрабатываются сервером базы данных.

Процедурный SQL является расширением Динамического SQL, в котором дополнительно присутствуют составные операторы, содержащие локальные переменные, присваивание, циклы и другие процедурные конструкции. PSQL относится к Части 4 (SQL/PSM) спецификации SQL. Изначально расширение PSQL было доступно только лишь в постоянно хранимых в базе модулях (процедурах и триггерах), но сравнительно недавно они стали также доступны в Динамическом SQL (смотри EXECUTE BLOCK).

Встроенный SQL определяет подмножество DSQL, поддерживаемое средством Firebird GPRE. GPRE — приложение-препроцессор, которое позволяет вам внедрять SQL конструкции в ваш непосредственный язык программирования (C, C++, Pascal, Cobol и так далее) и производить обработку этих внедрённых конструкций в правильные вызовы Firebird API. Обратите внимание, что ESQL поддерживает только часть конструкций и выражений DSQL.

Интерактивный SQL подразумевает собой язык, который может быть использован для работы с приложением командной строки Firebird ISQL для интерактивного доступа к базам данных. isql является обычным клиентским приложением. Для него обычный язык — это язык DSQL. Однако приложение поддерживает несколько дополнительных команд.

Оба языковых подмножества, как DSQL, так и PSQL полностью представлены в данном руководстве. Из набора инструментария ни ESQL, ни ISQL не описаны здесь отдельно, за исключением тех мест, где это не указано явно.

Диалекты SQL

SQL диалект — это термин, определяющий специфические особенности языка SQL, которые доступны во время доступа с его помощью к базе данных. SQL диалект может быть определён как на уровне базы данных, так и на уровне соединения с базой данных. В настоящее время доступны три диалекта:

- В 1-м диалекте дата и время хранятся в типе данных DATE, и имеется тип данных TIMESTAMP, который идентичен DATE. Двойные кавычки используются для разграничения строковых данных. Точность типов данных NUMERIC и DECIMAL меньше, чем в 3-м диалекте и в случае, если значение точности более 9, Firebird хранит такие значения как длинные значения с плавающей точкой. BIGINT не является доступным типом данных. Идентификаторы являются регистро-независимыми. Значение генераторов хранится как 64 битное целое, а при выдаче значения усекается до 32 битного целого;

- Диалект 2 доступен только в клиентском соединении к Firebird и не может быть применён к базе данных. Он предназначен для того, чтобы помочь в отладке в случае возможных проблем с целостностью данных при проведении миграции с диалекта 1 на 3;

- Диалект 3 базы данных позволяет хранить числа (типы данных DECIMAL и NUMERIC) в базе данных как длинные значения с фиксированной точкой (масштабируемые целые числа) в случае если точность числа меньше чем 9. Тип данных TIME доступен и используется для хранения значения только времени. Тип данных DATE хранит информацию о дате. Тип данных BIGINT доступен в качестве целого 64-х битного типа данных. Двойные кавычки могут использоваться, но только для идентификаторов, которые являются зависимыми от регистра, а не для строковых данных, для которых используют одинарные кавычки. Значения генераторов хранятся как 64-ти битные целые значения. Новую базу данных Firebird создаёт в 3-м диалекте.

Целью 1-го диалекта является обеспечение поддержки для унаследованных (пре-версия IB6) Interbase приложений для работы с Firebird. Диалект 2 используется как промежуточный и предназначен для разрешения проблем при миграции с 1-го в 3-й диалект. Для вновь разрабатываемых баз данных и приложений настоятельно рекомендуется использовать 3-й диалект. Диалект при соединении с базой данных должен быть таким же, как и базы данных. Исключением является случай миграции с 1-го в 3-й диалект, когда в строке соединения с базой данных используется 2-й диалект.

> **Замечание:**
>
> По умолчанию это руководство описывает семантику SQL третьего диалекта, если только в тексте явно не указывается диалект.

Действия при ошибках

Обработка любого оператора либо успешно завершается, либо прерывается из-за вызванной определёнными условиями ошибки. Обработку ошибок можно производить, как в клиентском приложении, так и на стороне сервера средствами SQL.

Подробнее смотрите в приложении Обработка ошибок, коды и сообщения.

Глава 2

Структура языка

Основные сведения: операторы, предложения, ключевые слова

Основная конструкция SQL — оператор (statement). Оператор описывает, что должна выполнить система управления базами данных с конкретным объектом данных или метаданных, обычно не указывая, как именно это должно быть выполнено. Достаточно сложные операторы содержат более простые конструкции — предложения (clause) и варианты, альтернативы (options). Предложение описывает некую законченную конструкцию в операторе. Например, предложение WHERE в операторе SELECT и в ряде других операторов (UPDATE, DELETE) задаёт условия поиска данных в таблице (таблицах), подлежащих выборке, изменению, удалению. Предложение ORDER BY задаёт характеристики упорядочения выходного, результирующего, набора данных. Альтернативы, будучи наиболее простыми конструкциями, задаются при помощи конкретных ключевых слов и определяют некоторые дополнительные характеристики элементов предложения (допустимость дублирования данных, варианты использования и др.).

В SQL существуют ключевые слова и зарезервированные слова. Ключевые слова — это все слова, входящие в лексику (словарь) языка SQL. Ключевые слова можно (но не рекомендуется) использовать в качестве имён, идентификаторов объектов базы данных, внутренних переменных и параметров. Зарезервированные слова — это те ключевые слова, которые нельзя использовать в качестве имён объектов базы данных, переменных или параметров.

Например, следующий оператор будет выполнен без ошибок потому, что ABS является ключевым, но не зарезервированным словом.

```
CREATE TABLE T (ABS INT NOT NULL);
```

При выполнении такого оператора будет выдана ошибка потому, что ADD является ключевым и зарезервированным словом.

```
CREATE TABLE T (ADD INT NOT NULL);
```

Список зарезервированных и ключевых слов представлен в приложении Зарезервированные и ключевые слова.

Идентификаторы

Все объекты базы данных имеют имена, которые иногда называют идентификаторами. Максимальная длина идентификатора составляет 31 байт. Существует два типа имён — имена, похожие по форме на имена переменных в обычных языках программирования, и имена с разделителями (delimited name), которые являются отличительной особенностью языка SQL.

Обычное имя должно начинаться с буквы латинского алфавита, за которой могут следовать буквы (латинского алфавита), цифры, символ подчёркивания и знак доллара. Такое имя нечувствительно к регистру, его можно записывать как строчными, так и прописными буквами. В имени нельзя использовать буквы кириллицы, пробелы, другие специальные символы.

Следующие имена с точки зрения системы являются одинаковыми:

```
fullname
FULLNAME
FuLlNaMe
FullName
```

Синтаксис:

```
<name> ::=
  <letter> |
  <name><letter> |
  <name><digit> |
  <name>_ |
  <name>$

<letter> ::= <upper letter> | <lower letter>

<upper letter> ::= A | B | C | D | E | F | G | H | I | J | K | L | M
    | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

<lower letter> ::= a | b | c | d | e | f | g | h | i | j | k | l | m
    | n | o | p | q | r | s | t | u | v | w | x | y | z

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Имя с разделителями заключается в двойные кавычки. Оно может содержать любые символы, включая буквы кириллицы, пробелы, специальные символы. В нем также могут присутствовать зарезервированные слова. Такое имя является чувствительным к регистру. Имена с разделителем доступны только в диалекте 3. Подробнее о диалектах см. Диалекты SQL

```
<delimited name> ::= "<ASCII char>[<ASCII char> …]"
```

Следует иметь в виду, что конечные пробелы в именах с разделителями, как и в любых строковых константах, отбрасываются.

Существует определённая похожесть и отличие обычных имён и имён с разделителями. Такие имена с разделителями и обычные, как "FULLNAME" и FULLNAME являются одинаковыми, а "FullName" и FULLNAME (так же как, например, и FullName) отличаются.

Литералы

Литералы служат для непосредственного представления данных, ниже приведён список стандартных литералов:

- целочисленные — 0, -34, 45;
- вещественные — 0.0, -3.14, 3.23e-23;
- строковые — 'текст', 'don''t!';
- дата — DATE '10.01.2014';
- время — TIME '15:12:56';
- временная отметка — TIMESTAMP '10.01.2014 13:32:02';
- неопределённое состояние — null.

Подробней о литералах для каждого из типов данных см. Типы и подтипы данных.

Операторы и специальные символы

Существует набор специальных символов, используемых в качестве разделителей.

```
<special char> ::= <space> | " | % | & | ' | ( | )
    | * | + | , | - | . | / | : | ; | < | = | > | ? | [ | ]
    | ^ | { | } | |
```

Часть этих символов, а так же их комбинации могут быть использованы как операторы (арифметические, строковые, логические), как разделители команд SQL, для квотирования идентификаторов, и для обозначения границ строковых литералов или комментариев.

Синтаксис:

```
<operator> ::=
    <string operator> |
    <arithmetic operator> |
    <comparison operator> |
    <logical operator>

<string operator> ::= {||}

<arithmetic operator> ::= * | / | + | - |

<comparison operator> ::=
    {=} | {<>} | {!=} | {~=} | {^=} |
    {>} | {<} | {>=} | {<=} | {!>} | {~>} | {^>} |
    {!<} | {~<} | {^<}
```

```
<logical operator> ::= NOT | AND | OR
```

Подробнее об операторах см. Выражения.

Комментарии

В SQL скриптах, операторах SQL и PSQL модулях могут встречаться комментарии. Комментарий — это произвольный текст заданный пользователем, предназначенный для пояснения работы отдельных частей программы. Синтаксический анализатор игнорирует текст комментариев. В Firebird поддерживается два типа комментариев: блочные и однострочные.

Синтаксис:

```
<comment> ::= <block comment> | <single-line comment>

<block comment> ::=
    /*<ASCII char>[<ASCII char> …]*/

<single-line comment> ::=
    --<ASCII char>[<ASCII char> …]<end line>
```

Блочные комментарии начинается с символов /* и заканчивается символами */. Блочные комментарии могут содержать текст произвольной длины и занимать несколько строк.

> **Примечание**
>
> Однако, если в блоке комментария присутствует последовательность символов '*/', то блочный комментарий будет немедленно завершён при обнаружении символов */.

Однострочные комментарии начинаются с символов -- и действуют до конца текущей строки.

Пример 2.1. Комментарии

```
CREATE PROCEDURE P(APARAM INT)
RETURNS (B INT)
AS
BEGIN
  /* Данный текст не будет учитываться
     при работе процедуры, т.к. является комментарием
   */
  B = A + 1; -- Однострочный комментарий
  SUSPEND;
END
```

Глава 3

Типы и подтипы данных

Типы данных используются в случае:

- определения столбца в таблице базы данных в операторе CREATE TABLE или для его изменения с использованием ALTER TABLE;
- при объявлении и редактировании домена оператором CREATE DOMAIN/ALTER DOMAIN;
- при объявлении локальных переменных в хранимых процедурах, функциях, PSQL-блоках и триггерах, при указании аргументов хранимых процедур и функций;
- при описании внешних функций (UDF – функций, определённых пользователем) для указания аргументов и возвращаемых значений;
- при явном преобразовании типов данных в качестве аргумента для функции CAST.

Таблица 3.1. Типы данных Firebird

Название	Размер	Диапазон и точность	Описание
BIGINT	64 бита	-2^{63} .. $2^{63} - 1$	Тип данных доступен только в 3 диалекте.
BOOLEAN	8 бит	false, true, unknown	Логический тип данных.
BLOB	Переменный	Нет. Размер сегмента BLOB ограничивается 64К. Максимальный размер поля BLOB 4 Гб. Для размера страницы 4096 максимальный размер BLOB поля несколько ниже 2 Гб.	Тип данных с динамически изменяемым размером для хранения больших данных, таких как графика, тексты, оцифрованные звуки. Базовая структурная единица — сегмент. Подтип Blob описывает содержимое.
CHAR(n) CHARACTER(n)	n символов (размер в байтах зависит от кодировки, кол-во байт на символ)	от 1 до 32 767 байт	Символьный тип данных фиксированной длины. При отображении данных, строка дополняется пробелами справа до указанной длины. Если количество символов n не указано, то по умолчанию принимается 1.

Название	Размер	Диапазон и точность	Описание
DATE	32 бита	От 01.01.0001 н.э. до 31.12.9999 н.э.	ISC_DATE
DECIMAL (*precision*, *scale*)	Переменный (16, 32 или 64 бита)	*precision* = от 1 до 18, указывает, по меньшей мере, количество цифр для хранения; *scale* = от 0 до 18. Задаёт количество знаков после разделителя	*scale* должно быть меньше или равно *precision*. Число с десятичной точкой, имеющей после точки *scale* разрядов. Пример: DECIMAL(10,3) содержит число точно в следующем формате: ppppppp.sss.
DOUBLE PRECISION	64 бита	$2{,}225 \times 10^{-308}$ .. $1{,}797 \times 10^{308}$	IEEE двойной точности, 15 цифр, размер зависит от платформы
FLOAT	32 бита	$1{,}175 \times 10^{-38}$ .. $3{,}402 \times 10^{38}$	IEEE одинарной точности, 7 цифр
INTEGER INT	32 бита	–2 147 483 648 .. 2 147 483 647	signed long
NUMERIC (*precision*, *scale*)	Переменный (16, 32 или 64 бита)	*precision* = от 1 до 18, указывает, по меньшей мере, количество цифр для хранения; *scale* = от 0 до 18. Задаёт количество знаков после разделителя.	*scale* должно быть меньше или равно precision. Число с десятичной точкой, имеющей после точки scale разрядов. Пример: NUMERIC(10,3) содержит число точно в следующем формате: ppppppp.sss.
SMALLINT	16 бит	–32 768 .. 32 767	signed short (word)
TIME	32 бита	От 0:00 до 23:59:59.9999	ISC_TIME
TIMESTAMP	64 бита	От 01.01.0001 н.э. до 31.12.9999 н.э.	Включает информацию и о времени
VARCHAR(*n*) CHAR VARYING	*n* символов (размер в байтах зависит от кодировки,	От 1 до 32 765 байтов	Размер символов в байтах с учётом их кодировки не может быть больше 32765. Для этого типа данных, в отличие

Название	Размер	Диапазон и точность	Описание
CHARACTER VARYING	кол-ва байт на символ)		от CHAR (где по умолчанию предполагается количество символов 1), количество символов n обязательно должно быть указано.

Примечание

Следует иметь в виду, что временной ряд из дат прошлых веков рассматривается без учёта реальных исторических фактов и так, как будто бы во всем этом диапазоне ВСЕГДА действовал только Григорианский календарь.

Целочисленные типы данных

Для целых чисел используют целочисленные типы данных SMALLINT, INTEGER и BIGINT (в 3 диалекте). Firebird не поддерживает беззнаковый целочисленный тип данных.

SMALLINT

Тип данных SMALLINT представляет собой целочисленное компактное хранилище данных и применяется в случае, когда не требуется широкий диапазон возможных значений для хранения данных.

Числа типа SMALLINT находятся в диапазоне 2^{-16} .. 2^{16} - 1, или -32 768 .. 32 767.

Примеры:

```
CREATE DOMAIN DFLAG AS SMALLINT DEFAULT 0 NOT NULL
  CHECK (VALUE=-1 OR VALUE=0 OR VALUE=1);

CREATE DOMAIN RGB_VALUE AS SMALLINT;
```

INTEGER

Тип данных INTEGER представляет собой 4-байтовое целое. Сокращённый вариант записи типа данных INT.

Числа типа INTEGER находятся в диапазоне 2^{-32} .. 2^{32} - 1, или -2 147 483 648 .. 2 147 483 647.

Примеры:

```
CREATE TABLE CUSTOMER (
  CUST_NO INTEGER NOT NULL,
```

```
  CUSTOMER VARCHAR(25) NOT NULL,
  CONTACT_FIRST VARCHAR(15),
  CONTACT_LAST VARCHAR(20),
  CONSTRAINT INTEG_60,
  ...
  PRIMARY KEY (CUST_NO)
);
```

BIGINT

BIGINT - это SQL-99-совместимый 64 битный целочисленный тип данных. Он доступен только в 3-м диалекте. При использовании клиентом диалекта 1, передаваемое сервером значение генератора усекается до 32-х битного целого (INTEGER). При подключении в 3-м диалекте значение генератора имеет тип BIGINT.

Числа типа BIGINT находятся в диапазоне 2^{-64} .. 2^{64} - 1, или -9 223 372 036 854 775 808 .. 9 223 372 036 854 775 807.

Числа типа BIGINT могут быть заданы в шестнадцатеричном виде с 9 — 16 шестнадцатеричными цифрами. Более короткие шестнадцатеричные числа интерпретируются как тип данных INTEGER.

Примеры:

Пример 3.1. Использование типа BIGINT

```
CREATE TABLE WHOLELOTTARECORDS (
  ID BIGINT NOT NULL PRIMARY KEY,
  DESCRIPTION VARCHAR(32)
);
```

Пример 3.2. Использование чисел типа BIGINT заданных шестнадцатеричном виде

```
INSERT INTO MYBIGINTS VALUES (
  -236453287458723,
  328832607832,
  22,
  -56786237632476,
  0X6F55A09D42, -- 478177959234
  0X7FFFFFFFFFFFFFFF, -- 9223372036854775807
  0XFFFFFFFFFFFFFFFF, -- -1
  0X80000000, -- -2147483648, т.е. INTEGER
  0X080000000, -- 2147483648, т.е. BIGINT
  0XFFFFFFFF, -- -1, т.е. INTEGER
  0X0FFFFFFFF -- 4294967295, т.е. BIGINT
);
```

Шестнадцатеричный INTEGER автоматически приводится к типу BIGINT перед вставкой в таблицу. Однако это происходит после установки численного значения, так 0x80000000 (8 цифр) и 0x080000000 (9 цифр) будут сохранены в разных форматах. Значение 0x80000000 (8 цифр) будет сохранено в формате INTEGER, а 0x080000000 (9 цифр) как BIGINT.

Типы данных с плавающей точкой

Типы данных с плавающей точкой являются примерами данных, которые хранятся в СУБД с точностью, подходящей масштабу числа. Примерами таких данных являются FLOAT и DOUBLE PRECISION (DOUBLE).

Учитывая особенности хранения чисел с плавающей точкой, этот тип данных не рекомендуется использовать для хранения денежных данных. По тем же причинам не рекомендуется использовать столбцы с данными такого типа в качестве ключей и применять к ним ограничения уникальности.

При проверке данных столбцов с типами данных с плавающей точкой рекомендуется вместо точного равенства использовать выражения проверки вхождения в диапазон, например BETWEEN.

При использовании таких типов данных в выражениях рекомендуется крайне внимательно и серьёзно подойти к вопросу округления результатов расчётов.

FLOAT

Данный тип данных обладает приблизительной точностью 7 цифр после запятой. Для обеспечения надёжности хранения полагайтесь на 6 цифр.

DOUBLE PRECISION

При хранении данных, предполагается приблизительная точность 15 цифр.

Типы данных с фиксированной точкой

Данные типы данных позволяют применять их для хранения денежных значений и обеспечивают предсказуемость операций умножения и деления.

Firebird предлагает два типа данных с фиксированной точкой: NUMERIC и DECIMAL. В соответствии со стандартом оба типа ограничивают хранимое число объявленным масштабом (количеством чисел после запятой). При этом подход к тому, как ограничивается точность для типов разный: для столбцов NUMERIC точность является такой, «как объявлено», в то время как DECIMAL столбцы могут получать числа, чья точность, по меньшей мере, равна тому, что было объявлено.

Например, NUMERIC(4, 2) описывает число, состоящее в общей сложности из четырёх цифр, включая 2 цифры после запятой; итого 2 цифры до запятой, 2 после. При записи в столбец с этим типом данных значений 3,1415 в столбце NUMERIC(4, 2) будет сохранено значение 3,14.

Для данных с фиксированной точкой общим является форма декларации, например NUMERIC(p, s). Здесь важно понять, что в этой записи s — это масштаб, а не интуитивно предсказываемое «количество знаков после запятой». Для «визуализации» механизма хранения данных запомните для себя процедуру:

- При сохранении в базу данных число умножается на 10^s (10 в степени s), превращаясь в целое;

- При чтении данных происходит обратное преобразование числа.

Способ физического хранения данных в СУБД зависит от нескольких факторов: декларируемой точности, диалекта базы данных, типа объявления.

Таблица 3.2. Способ физического хранения вещественных чисел

Точность	Тип данных	Диалект 1	Диалект 3
1 - 4	NUMERIC	SMALLINT	SMALLINT
1 - 4	DECIMAL	INTEGER	INTEGER
5 - 9	NUMERIC и DECIMAL	INTEGER	INTEGER
10 - 18	NUMERIC и DECIMAL	DOUBLE PRECISION	BIGINT

NUMERIC

Формат объявления данных:

```
NUMERIC(p, s)
```

В зависимости от точности p и масштаба s СУБД хранит данные по-разному.

Приведём примеры того, как СУБД хранит данные в зависимости от формы их объявления:

```
NUMERIC(4)      SMALLINT
NUMERIC(4,2)    SMALLINT (data * 10^2)
NUMERIC(10,4)   DOUBLE PRECISION в 1-ом диалекте
                BIGINT в 3-ем диалекте (data * 10^4)
```

Внимание

Всегда надо помнить, что формат хранения данных зависит от точности. Например, вы задали тип столбца NUMERIC(2, 2), предполагая, что диапазон значений в нем будет — 0.99...0.99. Однако в действительности диапазон значений в столбце будет -327.68..327.68, что объясняется хранением типа данных NUMERIC(2, 2) в формате SMALLINT. Фактически типы данных NUMERIC(4, 2), NUMERIC(3, 2) и NUMERIC(2, 2) являются одинаковыми. Т.е. Для реального хранения данных в столбце с типом данных NUMERIC(2, 2) в диапазоне -0.99...0.99 для него надо создавать ограничение.

DECIMAL

Формат объявления данных:

```
DECIMAL(p, s)
```

Формат хранения данных в базе во многом аналогичен NUMERIC, хотя существуют некоторые особенности, которые проще всего пояснить на примере.

Приведём примеры того, как СУБД хранит данные в зависимости от формы их объявления:

```
DECIMAL(4)       INTEGER
DECIMAL(4,2)     INTEGER (data * 10^2)
DECIMAL(10,4)    DOUBLE PRECISION в 1-ом диалекте
                 BIGINT в 3-ем диалекте (data * 10^4)
```

Типы данных для работы с датой и временем

В СУБД Firebird для работы с данными, содержащими дату и время, используются типы данных DATE, TIME, TIMESTAMP. В 3-м диалекте присутствуют все три вышеназванных типа данных, а в 1-м для операций с датой и временем доступен только тип данных DATE, который не тождественен типу данных DATE 3-го диалекта, а напоминает тип данных TIMESTAMP из 3-го диалекта.

В диалекте 1 тип DATE может быть объявлен как TIMESTAMP. Такое объявление является рекомендуемым для новых баз данных в 1-м диалекте.

Доли секунды, если хранятся в типах данных даты и времени, являются десятитысячными долями секунды.

DATE

В 3-м диалекте тип данных DATE, как это и следует предположить из названия, хранит только одну дату без времени. В 1-м диалекте нет типа данных "только дата".

Допустимый диапазон хранения от 1 января 1 г. н.э. до 31 декабря 9999 года.

> **Подсказка**
>
> В случае необходимости сохранять в 1 диалекте только значения даты, без времени, при записи в таблицу добавляйте время к значению даты в виде литерала '00:00:00.0000'.

TIME

Этот тип данных доступен только в 3-м диалекте. Позволяет хранить время дня в диапазоне от 00:00:00.0000 до 23:59:59.9999.

При необходимости получения времени из DATE. в 1-м диалекте можно использовать функцию EXTRACT.

Примеры:

```
EXTRACT (HOUR FROM DATE_FIELD)
EXTRACT (MINUTE FROM DATE_FIELD)
EXTRACT (SECOND FROM DATE_FIELD)
```

TIMESTAMP

Этот тип данных доступен только в 3-м диалекте, состоит из двух 32-битных слов и хранит дату со временем. Такое хранение эквивалентно типу DATE 1-го диалекта.

Операции, использующие значения даты и времени

Благодаря способу хранения даты и времени с этими типами возможны арифметические операции вычитания из более поздней даты (времени) более раннюю. Дата представлена количеством дней с "нулевой даты" – 17 ноября 1858 г. Время представлено количеством секунд (с учётом десятитысячных долей), прошедших с полуночи.

Таблица 3.3. Арифметические операции для типов данных даты и времени

Операнд 1	Оператор	Операнд 2	Результат
DATE	+	TIME	TIMESTAMP
DATE	+	*n*	DATE, увеличенная на *n* целых дней (дробная часть игнорируется).
TIME	+	DATE	TIMESTAMP
TIME	+	*n*	TIME, увеличенное на *n* секунд (дробная часть учитывается)
TIMESTAMP	+	*n*	TIMESTAMP, где дни увеличены на целую часть числа *n*, плюс дробная часть числа *n* (если указана) как количество секунд в дне (с точностью до десятитысячных долей секунды).
DATE	-	DATE	Количество дней в интервале DECIMAL (9, 0)
DATE	-	*n*	DATE, уменьшенная на *n* целых дней (дробная часть игнорируется)
TIME	-	TIME	Количество секунд в интервале DECIMAL (9, 4)
TIME	-	*n*	TIME, уменьшенное на *n* секунд (дробная часть учитывается)
TIMESTAMP	-	TIMESTAMP	Количество дней и части дня в интервале DECIMAL (18, 9)
TIMESTAMP	-	*n*	TIMESTAMP, где дни уменьшены на целую часть числа *n*, плюс

Операнд 1	Оператор	Операнд 2	Результат
			дробная часть числа *n* (если указана) как количество секунд в дне (с точностью до десятитысячных долей секунды).

Одно значение даты/времени может быть вычтено из другого если:

- Оба значения имеют один и тот же тип даты/времени;
- Первый операнд является более поздним, чем второй.

> **Примечание**
>
> В диалекте 1 тип DATE рассматривается как TIMESTAMP.

См. также: DATEADD, DATEDIFF.

Символьные типы данных

В СУБД Firebird для работы с символьными данными есть тип данных фиксированной длины CHAR и строковый тип данных VARCHAR переменной длины. Максимальный размер текстовых данных, хранящийся в этих типах данных, составляет 32767 байт для типа CHAR и 32765 байт для типа VARCHAR. Максимальное количество символов, которое поместится в этот объём, зависит от используемого набора символов CHARACTER SET и/или заданного порядка сортировки, который для символьных данных задаётся предложением COLLATE.

В случае отсутствия явного указания набора символов при описании текстового объекта базы данных будет использоваться набор символов по умолчанию, заданный при создании базы данных. При отсутствии явного указания набора символов, а также отсутствия набора символов по умолчанию в базе данных, поле получает набор символов CHARACTER SET NONE.

Если база данных будет содержать строки только с русским алфавитом, то для неё рекомендуется к использованию кодировка WIN1251. При её использовании на один символ расходуется 1 байт, соответственно максимальный размер текстовых полей для данной кодировки будет 32767 символов. Для стандартных операций сортировки при работе с WIN1251 не требуется задавать порядок сортировки (COLLATE).

UNICODE

В настоящее время все современные средства разработки поддерживают Unicode. При возникновении необходимости использования восточноевропейских текстов в строковых полях базы данных или для более экзотических алфавитов, рекомендуется работать с набором символов UTF8. При этом следует иметь в виду, что на один символ в данном наборе приходится до 4 байт. Следовательно, максимальный размер символов в символьных полях составит 32676/4 (8192) байта на символ. При этом следует обратить внимание, что фактически значение параметра «байт на символ» зависит от диапазона, к которому принадлежит символ: английские буквы занимают 1 байт, русские буквы кодировки WIN1251 — 2 байта, остальные символы — могут занимать до 4-х байт.

Набор символов UTF8 поддерживает последнюю версию стандарта Unicode, до 4 байт на символ, поэтому для интернациональных баз рекомендуется использовать именно эту реализацию поддержки Unicode в Firebird.

Набор символов клиента

При работе со строками необходимо помнить и о параметре соединения клиентской программы к базе данных. В нём также задаётся набор символов. В случае различия набора символов, при выдаче результата для строковых столбцов происходит автоматическая перекодировка как при передаче данных с клиента на сервер, так и в обратном направлении с сервера на клиента. То есть, совершенно нормальной является ситуация, когда база создана в кодировке WIN1251, а в настройках клиента в параметрах соединения стоит KOI8R или UTF8.

Специальные наборы символов

Набор символов NONE относится к специальным наборам символов. Его можно охарактеризовать тем, что каждый байт является частью строки, но в системе хранится без указаний, к какому фактическому набору символов они относятся. Разбираться с такими данными должно клиентское приложение, на него возлагается ответственность в правильной трактовке символов из таких полей.

Также к специальным наборам символов относится OCTETS. В этом случае данные рассматриваются как байты, которые могут в принципе не интерпретироваться как символы. OCTETS позволяет хранить бинарные данные и/или результаты работы некоторых функций Firebird. Правильное отображение данных пользователю, хранящихся в полях с CHARACTER SET OCTETS, также становится заботой клиентской стороны. При работе с подобными данными следует также помнить, что СУБД не контролирует их содержимое и возможно возникновение исключения при работе кода, когда идёт попытка отображения бинарных данных в желаемой кодировке.

Последовательность сортировки

Каждый текстовый набор символов (CHARACTER SET) имеет последовательность сортировки (COLLATE) по умолчанию, задающий порядок сортировки и способы сравнения. Если необходимо нестандартное поведение строк при указанных выше действиях, то в описании строкового столбца может быть указан параметр COLLATE, который его опишет. Помимо описания объявления столбца, выражение COLLATE может быть добавлено в предложениях SELECT в секции WHERE, когда происходят операции сравнения больше — меньше, в секции ORDER BY при сортировке по символьному полю, а также при операциях группировки для указания специальной последовательности сортировки при выводе в предложении GROUP BY.

Регистронезависимый поиск

Для регистронезависимого поиска можно воспользоваться функцией UPPER:

```
WHERE UPPER(name) = UPPER(:flt_name)
```

Для строк с набором символов WIN1251 можно для этих же целей воспользоваться предложением COLLATE PXW_CYRL.

Пример:

```
WHERE FIRST_NAME COLLATE PXW_CYRL >= :FLT_NAME
```

Пример сортировки независимой от регистра символов:

```
ORDER BY NAME COLLATE PXW_CYRL
```

См. также: CONTAINING.

Последовательности сортировки для UTF-8

Ниже приведена таблица возможных последовательностей сортировки для набора символов UTF8.

Таблица 3.4. Последовательности сортировки для UTF8

COLLATION	Комментарии
UCS_BASIC	Сортировка работает в соответствии с положением символа в таблице (бинарная): Пример: A, B, a, b, a...
UNICODE	Сортировка работает в соответствии с алгоритмом UCA (Unicode Collation Algorithm) (алфавитная). Пример: a, A, a, b, B...
UTF-8	Сортировка происходит без учёта регистра символа.
UNICODE_CI_AI	Сортировка происходит без учёта регистра символа, в алфавитном порядке.

Пример сортировки строк для набора символов UTF8 без учёта регистра символов (эквивалент COLLATE PXW_CYRL)

```
ORDER BY NAME COLLATE UNICODE_CI_AI
```

Индексирование символьных типов

При построении индекса по строковым полям необходимо учитывать ограничение на длину ключа индекса. Максимальная используемая длина ключа индекса равна 1/4 размера страницы, т.е. от 1024 до 4096 байтов. Максимальная длина индексируемой строки на 9 байтов меньше, чем максимальная длина ключа. В таблице приведены данные для максимальной длины индексируемой строки (в символах) в зависимости от размера страницы и набора символов, её можно вычислить по следующей формуле:

```
max_char_length = FLOOR((page_size / 4 - 9) / N),
```

где *N* — число байтов на представление символа.

Таблица 3.5. Длина индексируемой строки и набор символов

Размер страницы	Максимальная длина индексируемой строки для набора символов, байт/символ				
	1	2	3	4	6
4096	1015	507	338	253	169
8192	2039	1019	679	509	339
16384	4087	2043	1362	1021	682

> **Примечание**
>
> В кодировках, нечувствительных к регистру ("_CI"), один символ в *индексе* будет занимать не 4, а 6 (шесть) байт, поэтому максимальная длина ключа для страницы, скажем, 4096 байт, составит 169 символов.

Последовательность сортировки (COLLATE) тоже может повлиять на максимальную длину индексируемой строки. Полный список доступных наборов символов и нестандартных порядков сортировки доступен в приложении Наборы символов и порядки сортировки.

CHAR

CHAR является типом данных фиксированной длины. Если введённое количество символом меньше объявленной длины, то поле дополнится концевыми пробелами. В общем случае символ заполнитель может и не являться пробелом, он зависит от набора символов, так например, для набора символов OCTETS — это ноль.

Полное название типа данных CHARACTER, но при работе нет необходимости использовать полные наименования; инструменты по работе с базой прекрасно понимают и короткие имена символьных типов данных.

Синтаксис:

```
CHAR [(length)] [CHARACTER SET <charset>] [COLLATE <collate>]
```

В случае если не указана длина, то считается, что она равна единице.

Данный тип символьных данных можно использовать для хранения в справочниках кодов, длина которых стандартна и определённой «ширины». Примером такого может служить почтовый индекс в России – 6 символов.

VARCHAR

Является базовым строковым типом для хранения текстов переменной длины, поэтому реальный размер хранимой структуры равен фактическому размеру данных плюс 2 байта, в которых задана длина поля.

Все символы, которые передаются с клиентского приложения в базу данных, считаются как значимые, включая начальные и конечные пробельные символы.

Полное название CHARACTER VARYING. Имеется и сокращённый вариант записи CHAR VARYING.

Синтаксис:

```
VARCHAR (length) [CHARACTER SET <charset>] [COLLATE <collate>]
```

NCHAR

Представляет собой символьный тип данных фиксированной длины с предопределённым набором символов ISO8859_1.

Синтаксис:

```
NCHAR [(length)]
```

Синонимом является написание NATIONAL CHAR.

Аналогичный тип данных доступен для строкового типа переменной длины: NATIONAL CHARACTER VARYING.

Строковые литералы

Строковые литералы могут содержать произвольные символы, допустимые для применяемого набора символов. Весь литерал заключается в апострофы. Апостроф внутри символьного литерала должен повторяться два раза, чтобы отличить его от признака завершения литерала. Максимальная длина строкового литерала составляет 65535 Байт.

Примеры строковых констант:

```
'Mrs. Hunt''s husband'
```

Альтернативы для апострофов в строковых литералах

Вместо двойного (экранированного) апострофа вы можете использовать другой символ или пару символов.

Ключевое слово q или Q предшествующее строке в кавычках сообщает парсеру, что некоторые левые и правые пары одинаковых символов являются разделителями для встроенного строкового литерала.

Синтаксис:

```
<alternate string literal> ::=
```

```
  { q | Q } <quote> <alternate start char>
  [ { <char> }... ]
  <alternate end char> <quote>
```

Описание:

Когда *<alternate start char>* является одним из символов '(', '{', '[' или '<', то *<alternate end char>* должен быть использован в паре с соответствующим "партнёром", а именно ')', '}', ']' или '>'. В других случаях *<alternate end char>* совпадает с *<alternate start char>*.

Внутри строки, т.е. *<char>* элементах, одиночные (не экранированные) кавычки могут быть использованы. Каждая кавычка будет частью результирующей строки.

Примеры:

Пример 3.3. Использование альтернативных апострофов в строковых литералах

```
-- result: abc{def}ghi
SELECT Q'{abc{def}ghi}' FROM rdb$database;

-- result: That's a string
SELECT Q'!That's a string!' FROM rdb$database;
```

Динамическая сборка запроса использующего строковые литералы.

```
EXECUTE BLOCK
RETURNS (
  RDB$TRIGGER_NAME CHAR(31)
)
AS
  DECLARE VARIABLE S VARCHAR(8191);
BEGIN
  S = 'SELECT RDB$TRIGGER_NAME FROM RDB$TRIGGERS WHERE RDB$RELATION_NAME IN ';
  S = S || Q'! ('SALES_ORDER', 'SALES_ORDER_LINE')!';
  FOR
    EXECUTE STATEMENT :S
    INTO :RDB$TRIGGER_NAME
  DO
    SUSPEND;
END
```

BOOLEAN

SQL-2008 совместимый тип данных BOOLEAN (8 бит) включает различные значения истинности TRUE и FALSE. Если не установлено ограничение NOT NULL, то тип данных BOOLEAN поддерживает также значение истинности UNKNOWN как NULL значение. Спецификация не делает различия между значением NULL этого типа и значением истинности UNKNOWN, которое является результатом SQL предиката, поискового условия или выражения логического типа. Эти значения взаимозаменяемы и обозначают одно и то же.

Как и в других языках программирования, значения типа BOOLEAN могут быть проверены в неявных значениях истинности. Например, **field1 OR field2** или **NOT field1** являются допустимыми выражениями.

Предикаты могут использовать оператор IS [NOT] для проверки соответствия. Например, **field1 IS FALSE** или **field1 IS NOT TRUE**.

> **Примечание**
>
> - Представлен в API типом FB_BOOLEAN и константами FB_TRUE и FB_FALSE;
> - Операторы эквивалентности («=», «!=», «<>» и др.) допустимы во всех сравнениях;
> - Значение TRUE больше чем FALSE;
> - Несмотря на то, что тип данных BOOLEAN не преобразуется неявно ни к одному типу, возможно явное преобразование к строке с помощью функции CAST.

Примеры:

Пример 3.4. Использование типа BOOLEAN

```
CREATE TABLE TBOOL (ID INT, BVAL BOOLEAN);
COMMIT;

INSERT INTO TBOOL VALUES (1, TRUE);
INSERT INTO TBOOL VALUES (2, 2 = 4);
INSERT INTO TBOOL VALUES (3, NULL = 1);
COMMIT;

SELECT * FROM TBOOL
```

```
ID           BVAL
============ =======
1            <true>
2            <false>
3            <null>
```

```
-- Проверка TRUE значения
SELECT * FROM TBOOL WHERE BVAL
```

```
ID           BVAL
============ =======
1            <true>
```

```
-- Проверка FALSE значения
SELECT * FROM TBOOL WHERE BVAL IS FALSE
```

```
ID           BVAL
============ =======
2            <false>
```

```
-- Проверка UNKNOWN значения
SELECT * FROM TBOOL WHERE BVAL IS UNKNOWN
```

```
ID           BVAL
============ =======
3            <null>
```

```
-- Логические значения в SELECT списке
SELECT ID, BVAL, BVAL AND ID < 2
FROM TBOOL
```

```
ID           BVAL
============ ======= =======
1            <true> <true>
2            <false> <false>
3            <null> <false>
```

```
-- PSQL объявления с начальным значением
DECLARE VARIABLE VAR1 BOOLEAN = TRUE;

-- Допустимый синтаксис, но как и сравнение
-- с NULL, никогда не вернёт ни одной записи
SELECT * FROM TBOOL WHERE BVAL = UNKNOWN
SELECT * FROM TBOOL WHERE BVAL <> UNKNOWN
```

Бинарные типы данных

BLOB

BLOB (Binary Large Objects, большие двоичные объекты) представляют собой сложные структуры, предназначенные для хранения текстовых и двоичных данных неопределённой длины, зачастую очень большого объёма.

Синтаксис:

```
BLOB [SUB_TYPE <subtype>]
[SEGMENT SIZE <seg_length>]
[CHARACTER SET <charset>]
```

Сокращённый синтаксис:

```
BLOB (<seg_length>)
BLOB (<seg_length>, <subtype>)
BLOB (, <subtype>)
```

Размер сегмента: Указание размера сегмента BLOB является некоторым атавизмом, оно идёт с тех времён, когда приложения для работы с данными BLOB писались на C (Embedded SQL) при помощи GPRE. В настоящий момент размер сегмента при работе с данными BLOB определяется клиентской частью, причём размер сегмента может превышать размер страницы данных.

Подтипы BLOB

Подтип BLOB отражает природу данных, записанную в столбце. Firebird предоставляет два предопределённых подтипа для сохранения пользовательских данных:

- *Подтип 0 (BINARY):* Если подтип не указан, то данные считаются нетипизированными и значение подтипа принимается равным 0. Псевдоним подтипа 0 — BINARY. Этот подтип указывает, что данные имеют форму бинарного файла или потока (изображение, звук, видео, файлы текстового процессора, PDF и т.д.).
- *Подтип 1 (TEXT):* Подтип 1 имеет псевдоним TEXT, который может быть использован вместо указания номера подтипа. Например, BLOB SUBTYPE TEXT. Это специализированный подтип, который используется для хранения текстовых данных большого объёма. Для текстового подтипа BLOB может быть указан набор символов и порядок сортировки COLLATE, аналогично символьному полю.

Кроме того, существует возможность добавления пользовательских подтипы данных, для них зарезервирован интервал от -1 до -32768. Пользовательские подтипы с положительными числами не поддерживаются, поскольку Firebird использует числа больше 2 для внутренних подтипов метаданных.

Особенности BLOB

Максимальный размер поля BLOB ограничен 4Гб и не зависит от варианта сервера, 32 битный или 64 битный (во внутренних структурах, связанных с BLOB присутствуют 4-х байтные счётчики). Для размера страницы 4096 максимальный размер BLOB поля несколько ниже 2 Гб.

Текстовые BLOB любой длины и с любым набором символов (включая multi-byte) могут быть использованы практически с любыми встроенными функциями и операторами:

- Полная поддержка для операторов:

  = (присвоение);
  =, <>, <, <=, >, >= (сравнение);
  || (конкатенация);
  BETWEEN, IS [NOT] DISTINCT FROM, IN, ANY|SOME и ALL;
- Частичная поддержка для STARTING [WITH], LIKE и CONTAINING. (возникает ошибка, в случае если второй аргумент больше или равен 32 Кб);
- SELECT DISTINCT, ORDER BY и GROUP BY в своей работе используют BLOB ID, а не содержимое самого поля. Это одновременно и хорошо и плохо, кроме того, SELECT DISTINCT ошибочно выдаёт несколько значений NULL, если они присутствуют. GROUP BY ведёт себя странно в том, что он объединяет одинаковые строки, если они находятся рядом, но не делает этого, если они располагаются вдали друг от друга.

Хранение BLOB:

- По умолчанию, для каждого BLOB создаётся обычная запись, хранящаяся на какой-то выделенной для этого странице данных (data page). Если весь BLOB на эту страницу

поместится, его называют BLOB уровня 0. Номер этой специальной записи хранится в записи таблицы и занимает 8 байт.

- Если BLOB не помещается на одну страницу данных (data page), то его содержимое размещается на отдельных страницах, целиком выделенных для него (blob page), а в записи о BLOB помещают номера этих страниц. Это BLOB уровня 1.

- Если массив номеров страниц с данными BLOB не помещается на страницу данных (data page), то его (массив) размещают на отдельных страницах (blob page), а в запись о BLOB помещают уже номера этих страниц. Это BLOB уровня 2.

- Уровни выше 2 не поддерживаются.

См. также: FILTER, DECLARE FILTER.

Массивы

Поддержка массивов в СУБД Firebird является расширением традиционной реляционной модели. Поддержка в СУБД такого инструмента позволяет проще решать некоторые задачи по обработке однотипных данных. Массивы в Firebird реализованы на базе полей типа BLOB. Массивы могут быть одномерными и многомерными.

```
CREATE TABLE SAMPLE_ARR (
  ID INTEGER NOT NULL PRIMARY KEY,
  ARR_INT INTEGER [4]);
```

Так будет создана таблица с полем типа массива из четырёх целых. Индексы данного массива от 1 до 4. Для определения верхней и нижней границы значений индекса следует воспользоваться следующим синтаксисом:

```
[<нижняя>:<верхняя>]
```

Добавление новой размерности в синтаксисе идёт через запятую. Пример создания таблицы с массивом размерности два, в котором нижняя граница значений начинается с нуля:

```
CREATE TABLE SAMPLE_ARR2 (
  ID INTEGER NOT NULL PRIMARY KEY,
  ARR_INT INTEGER [0:3, 0:3]);
```

СУБД не предоставляет большого набора инструментов для работы с содержимым массивов. База данных `employee.fdb`, которая находится в дистрибутиве Firebird, содержит пример хранимой процедуры, показывающей возможности работы с массивами. Ниже приведён её текст:

```
CREATE OR ALTER PROCEDURE SHOW_LANGS (
  CODE VARCHAR(5),
```

```
  GRADE SMALLINT,
  CTY VARCHAR(15))
RETURNS (
  LANGUAGES VARCHAR(15))
AS
  DECLARE VARIABLE I INTEGER;
BEGIN
  I = 1;
  WHILE (I <= 5) DO
  BEGIN
    SELECT LANGUAGE_REQ[:I]
    FROM JOB
    WHERE (JOB_CODE = :CODE)
      AND (JOB_GRADE = :GRADE)
      AND (JOB_COUNTRY = :CTY)
      AND (LANGUAGE_REQ IS NOT NULL))
    INTO :LANGUAGES;

    IF (:LANGUAGES = '') THEN
      /* PRINTS 'NULL' INSTEAD OF BLANKS */
      LANGUAGES = 'NULL';
    I = I +1;
    SUSPEND;
  END
END
```

Если приведённых выше возможностей достаточно для ваших задач, то вы вполне можете применять массивы для своих проектов. В настоящее время совершенствования механизмов обработки массивов средствами СУБД не производится.

Специальные типы данных

Тип данных SQL_NULL

Данный тип данных содержит не данные, а только состояние: NULL или NOT NULL. Также, этот тип данных не может быть использован при объявлении полей таблицы, переменных PSQL, использован в описании параметров. Этот тип данных добавлен для улучшения поддержки нетипизированных параметров в предикате IS NULL. Такая проблема возникает при использовании «отключаемых фильтров» при написании запросов следующего типа:

```
WHERE col1 = :param1 OR :param1 IS NULL
```

после обработки, на уровне API запрос будет выглядеть как

```
WHERE col1 = ? OR ? IS NULL
```

В данном случае получается ситуация, когда разработчик при написании SQL запрос рассматривает *:param1* как одну переменную, которую использует два раза, а на уровне API

запрос содержит два отдельных и независимых параметра. Вдобавок к этому, сервер не может определить тип второго параметра, поскольку он идёт в паре с IS NULL.

Именно для решения проблемы «? IS NULL» и был добавлен этот специальный тип данных SQL_NULL.

После введения данного специального типа данных при передаче запроса и его параметров на сервер будет работать такая схема: приложение передаёт параметризованные запросы на сервер в виде «?». Это делает невозможным слияние пары «одинаковых» параметров в один. Так, например, для двух фильтров (двух именованных параметров) необходимо передать четыре позиционных параметра (далее предполагается, что читатель имеет некоторое знакомство с Firebird API):

```
SELECT
  SH.SIZE, SH.COLOUR, SH.PRICE
FROM SHIRTS SH
WHERE (SH.SIZE = ? OR ? IS NULL)
  AND (SH.COLOUR = ? OR ? IS NULL)
```

После выполнения isc_dsql_describe_bind() sqltype 2-го и 4-го параметров устанавливается в SQL_NULL. Как уже говорилось выше, сервер Firebird не имеет никакой информации об их связи с 1-м и 3-м параметрами — это полностью прерогатива программиста. Как только значения для 1-го и 3-го параметров были установлены (или заданы как NULL) и запрос подготовлен, каждая пара XSQLVARs должна быть заполнена следующим образом:

Пользователь задал параметры:

- Первый параметр (сравнение значений): set *sqldata в переданное значение и *sqlind в 0 (для NOT NULL);
- Второй параметр (проверка на NULL): set *sqldata в NULL (не SQL_NULL) и *sqlind в 0 (для NOT NULL).

Пользователь задал параметры:

- Оба параметра (проверка на NULL): set *sqldata в NULL (не SQL_NULL) и *sqlind в -1 (индикация NULL).

Другими словами: значение параметра сравнения всегда устанавливается как обычно. SQL_NULL параметр устанавливается также, за исключением случая, когда sqldata передаётся как NULL.

Преобразование типов данных

При написании выражения или при задании, например, условий сравнения, нужно стараться использовать совместимые типы данных. В случае необходимости использования смешанных данных различных типов, желательно первоначально выполнить преобразования типов, а уже потом выполнять операции.

При рассмотрении вопроса преобразования типов в Firebird большое внимание стоит уделить тому, в каком диалекте база данных.

Явное преобразование типов данных

В тех случаях, когда требуется выполнить явное преобразование одного типа в другой, используют функцию CAST.

Синтаксис:

```
CAST (<value> | NULL AS <data_type>)

<data_type> ::=
      sql_datatype
  | [TYPE OF] domain
  | TYPE OF COLUMN relname.colname
```

Преобразование к домену

При преобразовании к домену учитываются объявленные для него ограничения, например, NOT NULL или описанные в CHECK и если *<value>* не пройдёт проверку, то преобразование не удастся. В случае если дополнительно указывается TYPE OF (преобразование к базовому типу), при преобразовании игнорируются любые ограничения домена. При использовании TYPE OF с типом (VAR)CHAR набор символов и сортировка сохраняются.

Преобразование к типу столбца

При преобразовании к типу столбца допускается использовать указание столбца таблицы или представления. Используется только сам тип столбца; в случае строковых типов это также включает набор символов, но не сортировку. Ограничения и значения по умолчанию исходного столбца не применяются.

Примеры:

```
CREATE TABLE TTT (
  S VARCHAR (40)
  CHARACTER SET UTF8 COLLATE UNICODE_CI_AI);
COMMIT;

/* У меня много друзей (шведский)*/
SELECT
  CAST ('Jag har manga vanner' AS TYPE OF COLUMN TTT.S)
FROM RDB$DATABASE;
```

Допустимые преобразования для функции CAST

Таблица 3.6. Допустимые преобразования для функции CAST

Из типа	В тип
Числовые типы	Числовые типы [VAR]CHAR BLOB

Из типа	В тип
[VAR]CHAR BLOB	[VAR]CHAR BLOB BOOLEAN Числовые типы DATE TIME TIMESTAMP
DATE TIME	[VAR]CHAR BLOB TIMESTAMP
TIMESTAMP	[VAR]CHAR BLOB TIME DATE
BOOLEAN	[VAR]CHAR BLOB

Для преобразования строковых типов данных в тип BOOLEAN необходимо чтобы строковый аргумент был одним из предопределённых литералов логического типа (`'true'` или `'false'`).

> **Важно**
>
> При проведении следует помнить о возможности частичной потери данных, например, при преобразовании типа данных TIMESTAMP в DATE.

Преобразование литералов дат и времени

Для преобразования строковых типов данных в типы DATE, TIME или TIMESTAMP необходимо чтобы строковый аргумент был либо одним из предопределённых литералов даты и времени, либо строковое представление даты в одном из разрешённых форматов.

```
<datetime_literal> ::= {
  [YYYY<p>]MM<p>DD[<p>HH[<p>mm[<p>SS[<p>NNNN]]]] |
  MM<p>DD[<p>YYYY[<p>HH[<p>mm[<p>SS[<p>NNNN]]]]] |
  DD<p>MM[<p>YYYY[<p>HH[<p>mm[<p>SS[<p>NNNN]]]]] |
  MM<p>DD[<p>YY[<p>HH[<p>mm[<p>SS[<p>NNNN]]]]] |
  DD<p>MM[<p>YY[<p>HH[<p>mm[<p>SS[<p>NNNN]]]]] |
  NOW |
  TODAY |
  TOMORROW |
  YERSTERDAY
}

<date_literal> ::= {
  [YYYY<p>]MM<p>DD |
  MM<p>DD[<p>YYYY] |
  DD<p>MM[<p>YYYY] |
  MM<p>DD[<p>YY] |
  DD<p>MM[<p>YY] |
```

```
  TODAY |
  TOMORROW |
  YERSTERDAY
}

<time_literal> := HH[<p>mm[<p>SS[<p>NNNN]]]

<p> ::= whitespace | . | : | , | - | /
```

Таблица 3.7. Описание формата даты и времени

Аргумент	Описание
datetime_literal	Литерал даты-времени.
date_literal	Литерал даты.
time_literal	Литерал времени.
YYYY	Год из четырёх цифр.
YY	Последние две цифры года (00-99).
MM	Месяц. Может содержать 1 или 2 цифры (1-12 или 01-12). В качестве месяца допустимо также указывать трёхбуквенное сокращение или полное наименование месяца на английском языке, регистр не имеет значение.
DD	День. Может содержать 1 или 2 цифры (1-31 или 01-31).
HH	Час. Может содержать 1 или 2 цифры (0-23 или 00-23).
mm	Минуты. Может содержать 1 или 2 цифры (0-59 или 00-59).
SS	Секунды. Может содержать 1 или 2 цифры (0-59 или 00-59).
NNNN	Десятитысячные доли секунды. Может содержать от 1 до 4 цифр (0-9999).
p	Разделитель, любой из разрешённых символов, лидирующие и завершающие пробелы игнорируются

Таблица 3.8. Литералы с предопределёнными значениями даты и времени

Литерал	Значение	Тип данных для диалекта 1	Тип данных для диалекта 1
'NOW'	Текущая дата и время	DATE	TIMESTAMP
'TODAY'	Текущая дата	DATE (с нулевым временем)	DATE (только дата)
'TOMORROW'	Завтрашняя дата	DATE (с нулевым временем)	DATE (только дата)

Литерал	Значение	Тип данных для диалекта 1	Тип данных для диалекта 1
'YESTERDAY'	Вчерашняя дата	DATE (с нулевым временем)	DATE (только дата)

Правила:

- В формате Год-Месяц-День, год обязательно должен содержать 4 цифры;
- Для дат в формате с завершающим годом, если в качестве разделителя дат используется точка «.», то дата интерпретируется в форме День-Месяц-Год, для остальных разделителей она интерпретируется в форме Месяц-День-Год;
- Если год не указан, то в качестве года берётся текущий год;
- Если указаны только две цифры года, то для получения столетия Firebird использует алгоритм скользящего окна. Задача заключается в интерпретации двухсимвольного значения года как ближайшего к текущему году в интервале предшествующих и последующих 50 лет;
- Если не указан один из элементов времени, то оно принимается равным 0.

> **Подсказка**
>
> Настоятельно рекомендуем в литералах дат использовать только формы с полным указанием года в виде 4 цифр во избежание путаницы.

Примеры интерпретации литералов дат и времени:

```
SELECT
  CAST('04.12.2014' AS DATE) AS d1, -- DD.MM.YYYY
  CAST('04 12 2014' AS DATE) AS d2, -- MM DD YYYY
  CAST('4-12-2014' AS DATE) AS d3,  -- MM-DD-YYYY
  CAST('04/12/2014' AS DATE) AS d4, -- MM/DD/YYYY
  CAST('04,12,2014' AS DATE) AS d5, -- MM,DD,YYYY
  CAST('04.12.14' AS DATE) AS d6,   -- DD.MM.YY
  -- DD.MM в качестве года берётся текущий
  CAST('04.12' AS DATE) AS d7,
  -- MM/DD в качестве года берётся текущий
  CAST('04/12' AS DATE) AS d8,
  CAST('2014/12/04' AS DATE) AS d9, -- YYYY/MM/DD
  CAST('2014 12 04' AS DATE) AS d10, -- YYYY MM DD
  CAST('2014.12.04' AS DATE) AS d11, -- YYYY.MM.DD
  CAST('2014-12-04' AS DATE) AS d12, -- YYYY-MM-DD
  CAST('4 Jan 2014' AS DATE) AS d13, -- DD MM YYYY
  CAST('2014 Jan 4' AS DATE) AS dt14, -- YYYY MM DD
  CAST('Jan 4, 2014' AS DATE) AS dt15, -- MM DD, YYYY
  CAST('11:37' AS TIME) AS t1, -- HH:mm
  CAST('11:37:12' AS TIME) AS t2, -- HH:mm:ss
  CAST('11:31:12.1234' AS TIME) AS t3, -- HH:mm:ss.nnnn
  CAST('11.37.12' AS TIME) AS t4, -- HH.mm.ss
  -- DD.MM.YYYY HH:mm
  CAST('04.12.2014 11:37' AS TIMESTAMP) AS dt1,
  -- MM/DD/YYYY HH:mm:ss
  CAST('04/12/2014 11:37:12' AS TIMESTAMP) AS dt2,
  -- DD.MM.YYYY HH:mm:ss.nnnn
  CAST('04.12.2014 11:31:12.1234' AS TIMESTAMP) AS dt3,
  -- MM/DD/YYYY HH.mm.ss
```

```
  CAST('04/12/2014 11.37.12' AS TIMESTAMP) AS dt4
FROM rdb$database
```

См. также: CAST.

Сокращённое приведение типов даты и времени (datetime)

При преобразовании строки в тип DATE, TIME или TIMESTAMP, Firebird позволяет использовать сокращённое "C-style" приведение типов.

Синтаксис:

```
datatype 'date/time string'
```

Примеры:

```
UPDATE PEOPLE
SET AGECAT = 'Old'
WHERE BIRTHDATE < DATE '1-Jan-1943';

INSERT INTO APPOINTMENTS
(EMPLOYEE_ID, CLIENT_ID, APP_DATE, APP_TIME)
VALUES (973, 8804, DATE 'today' + 2, TIME '16:00');

NEW.LASTMOD = TIMESTAMP 'now';
```

Обратите внимание, что эти сокращённые выражения вычисляются сразу же во время синтаксического анализа, то есть, как будто оператор уже подготовлен к выполнению. Таким образом, даже если запрос выполняется несколько раз, значение, например, для TIMESTAMP 'NOW' не изменится, независимо от того, сколько времени проходит. Если вам нужно получать нарастающее значение времени (т.е. оно должно быть оценено при каждом вызове), используйте полный синтаксис CAST. Пример использования в триггере такого выражения:

```
NEW.CHANGE_DATE = CAST('now' AS TIMESTANP);
```

См. также: Явное преобразование типов данных.

Неявное преобразование типов данных

В 3-м диалекте невозможно неявное преобразование данных, здесь требуется указывать функцию CAST для явной трансляции одного типа в другой. Однако это не относится к операции конкатенации, при которой все другие типы данных будут неявно преобразованы к символьному.

При использовании 1-го диалекта во многих выражениях выполняется неявное преобразование одних типов в другой без применение функции CAST. Например, в выражении отбора в диалекте 1 можно записать:

```
WHERE DOC_DATE < '31.08.2014'
```

и преобразование строки в дату произойдёт неявно.

В 1-м диалекте можно смешивать целые данные и числовые строки, строки неявно преобразуются в целое, если это будет возможно, например:

```
2 + '1'
```

корректно выполнится. В 3-м диалекте подобное выражение вызовет ошибку, в нем потребуется запись следующего вида:

```
2 + CAST('1' AS SMALLINT)
```

Неявное преобразование типов при конкатенации

При конкатенации множества элементов разных типов, все не строковые данные будут неявно преобразованы к строке, если это возможно.

Примеры:

```
SELECT 30||' days hath September, April, June and November' CONCAT$
FROM RDB$DATABASE
```

```
CONCAT$
================================================
30 days hath September, April, June and November
```

Пользовательские типы данных — домены

Домены в СУБД Firebird реализуют широко известный по многим языкам программирования инструмент «типы данных, определённые пользователем». Когда несколько таблиц в базе данных содержат поля с характеристиками одинаковыми или практически одинаковыми, то есть целесообразность сделать домен, в котором описать набор свойств поля и использовать такой набор свойств, описанный один раз, в нескольких объектах базы данных. Домены могут использоваться помимо описания полей таблиц и представлений (VIEW) и при объявлении входных и выходных параметров, а также при объявлении переменных в коде PSQL.

Атрибуты домена

Определение домена содержит обязательные и необязательные атрибуты. К обязательному атрибуту относится тип данных. К необязательным относятся:

- значение по умолчанию;

- возможности использования NULL для домена;
- ограничения CHECK для данных домена;
- набор символов (для символьных типов данных и BLOB полей);
- порядок сортировки (для символьных типов данных).

Пример создания домена:

```
CREATE DOMAIN BOOL3 AS SMALLINT
   CHECK (VALUE IS NULL OR VALUE IN (0, 1));
```

См. также: Явное преобразование типов данных, где описаны отличия работы механизма преобразования данных при указании доменов для опций TYPE OF и TYPE OF COLUMN.

Переопределение свойств доменов

При описании таблиц базы данных некоторые свойства столбцов, базирующихся на доменах, могут быть переопределены. Возможности переопределения атрибутов столбцов на базе доменов приведены в таблице.

Таблица 3.9. Возможности переопределения атрибутов столбцов на базе доменов

Атрибут	Переопределяется?	Примечания
тип данных	нет	
значение по умолчанию	да	
текстовый набор символов	да	также может использоваться, чтобы восстановить для столбца значения по умолчанию для базы данных
текстовый порядок сортировки	да	
условия проверки CHECK	нет	для добавления в проверку новых условий, можно использовать в операторах CREATE и ALTER на уровне таблицы соответствующие предложения CHECK.
NOT NULL	нет	во многих случаях лучше оставить при описании домена возможность значения NULL, а контроль его допустимости осуществлять в описании полей на уровне таблицы.

Создание доменов

Создание домена производится оператором CREATE DOMAIN.

Краткий синтаксис:

```
CREATE DOMAIN <name> [AS] <type>
[DEFAULT {<const> | <literal> | NULL | <context_var>}]
[NOT NULL] [CHECK (<condition>)]
[COLLATE collation];
```

См. также: CREATE DOMAIN.

Изменение доменов

Для редактирования свойств домена используют оператор ALTER DOMAIN языка определения данных (DDL).

При редактировании домена можно:

- переименовать домен;
- изменить тип данных;
- удалить текущее значение по умолчанию;
- установить новое значение по умолчанию;
- установить ограничение NOT NULL;
- удалить ограничение NOT NULL;
- удалить текущее ограничение CHECK;
- добавить новое ограничение CHECK.

Краткий синтаксис:

```
ALTER DOMAIN name
  [{TO new_name}]
  [{SET DEFAULT {literal | NULL | <context_var>} |
    DROP DEFAULT}]
  [{SET | DROP} NOT NULL]
  [{ADD [CONSTRAINT] CHECK (<dom_condition>) |
    DROP CONSTRAINT}]
  [{TYPE <datatype>}];
```

Пример:

```
ALTER DOMAIN STORE_GRP SET DEFAULT -1;
```

При изменении доменов следует учитывать и его зависимости: имеются ли столбцы таблиц; находятся ли в коде PSQL объявления переменных, входных и/или выходных параметров с типом этого домена. При поспешном редактировании без внимательной проверки можно сделать данный код неработоспособным!

Важно

При смене в домене типа данных не допустимы преобразования, которые могут привести к потере данных. Также, например, при преобразовании VARCHAR в INTEGER проверьте, все ли данные, что используют данных домен, смогут пройти преобразование.

См. также: ALTER DOMAIN.

Удаление доменов

Оператор DROP DOMAIN удаляет из базы данных домена при условии, что он не используется в каком либо из объектов базы данных.

Синтаксис:

```
DROP DOMAIN name;
```

Пример:

```
DROP DOMAIN Test_Domain;
```

См. также: DROP DOMAIN.

Глава 4

Общие элементы языка

В этой главе рассматриваются элементы, которые являются общими для всех реализаций языка SQL — выражения, которые используются для извлечения и обработки фактов из данных и предикатов, которые проверяют истинность этих фактов.

Выражения

Выражения SQL представляют формальные методы для вычисления, преобразования и сравнения значений. Выражения SQL могут включать в себя столбцы таблиц, переменные, константы, литералы, различные операторы и предикаты, а так же другие выражения. Полный список допустимых символов (tokens) в выражениях описан ниже.

Таблица 4.1. Описание элементов языка

Элемент	Описание
Имя столбца	Идентификаторы столбцов из указанных таблиц, используемые в вычислениях, или сравнениях, или в качестве условия поиска. Столбец типа массив не может быть элементом выражения, если только он не проверяется на IS [NOT] NULL.
Элементы массива	В выражении может содержаться ссылка на элемент массива.
Арифметические операторы	Символы +, -, *, / используемые для вычисления значений.
Оператор конкатенации	Оператор || используется для соединения символьных строк.
Логические операторы	Зарезервированные слова NOT, AND и OR используются при комбинировании простых условий поиска для создания сложных утверждений.
Операторы сравнения	Символы =, <>, !=, ~=, ^=, <, <=, >, >=, !<, ~<, ^<, !>, ~> и ^>.
Предикаты сравнения	LIKE, STARTING WITH, CONTAINING, SIMILAR TO, BETWEEN, IS [NOT] NULL и IS [NOT] DISTINCT FROM
Предикаты существования	Предикаты, используемые для проверки существования значений в наборе. Предикат IN может быть использован как с наборами констант, так и со скалярными подзапросами. Предикаты EXISTS, SINGULAR, ALL ANY, SOME могут быть использованы только с подзапросами.
Константы	Числа, заключённые в апострофы строковые литералы, логические значения TRUE, FALSE и UNKNOWN, псевдозначение NULL.
Литералы дат	Выражения, подобные строковым литералам, заключённые в апострофах, которые могут быть интерпретированы как значения

Элемент	Описание
	даты, времени или даты-времени. Литералами дат могут быть предварительно объявленные литералы ('TODAY', 'NOW' и т.д.) или строки из символов и чисел, такие как '25.12.2016 15:30:35', которые могут быть преобразованы в дату, время или дату с временем.
Контекстные переменные	Встроенные контекстные переменные.
Локальные переменные	Локальные переменные, входные или выходные параметры PSQL модулей (хранимых процедур, триггеров, анонимных блоков PSQL).
Позиционные параметры	В DSQL в качестве параметров запроса могут быть использованы только позиционные параметры. Позиционные параметры представляют собой знаки вопроса (?) внутри DSQL оператора. Доступ к таким параметрам осуществляется по его номеру (позиции в запросе относительно предыдущего позиционного параметра) поэтому они называются позиционными. Обычно компоненты доступа позволяют работать с именованными параметрами, которые они сами преобразовывают в позиционные.
Подзапросы	Оператор SELECT заключённый в круглые скобки, который возвращает одно единственное (скалярное) значение или множество значений (при использовании в предикатах существования).
Идентификаторы функций	Идентификаторы встроенных или внешних функций в функциональных выражениях.

Приведения типа	Выражение явного преобразования одного типа данных в другой
	`CAST(<value> AS <datatype>)`
	или сокращённое (для даты/времени) преобразование типа
	`<datatype> <value>`
	например `DATE '25.12.2016'`

Условные выражения	Выражение CASE и встроенные функции COALESCE, NULLIF.
Круглые скобки	Используются для группировки выражений. Операции внутри скобок выполняются перед операциями вне скобок. При использовании вложенных скобок, сначала вычисляются значения самых внутренних выражений, а затем вычисления перемещаются наверх по уровням вложенности.
Предложение COLLATE	Может быть применено к значениям типа CHAR и VARCHAR, чтобы использовать строковые сравнения в указанной последовательности сортировки.

Элемент	Описание
NEXT VALUE FOR *sequence*	Конструкция NEXT VALUE FOR позволяет получить следующее значение последовательности, то же самое делает встроенная функция GEN_ID().

Операторы SQL

Выражения SQL включают операторы для сравнения и вычисления значений.

Приоритет операторов

Приоритет определяет порядок, в котором операторы и получаемые с помощью них значения вычисляются в выражении.

Все операторы разбиты на 4 типа. Каждый тип оператора имеет свой приоритет. Чем выше приоритет типа оператора, тем раньше он будет вычислен. Внутри одного типа операторы имеют собственный приоритет, который также определяет порядок их вычисления в выражении. Операторы с одинаковым приоритетом вычисляются слева направо. Для изменения порядка вычислений операции могут быть сгруппированы с помощью круглых скобок.

Таблица 4.2. Приоритеты типов операторов

Тип оператора	Приоритет	Пояснение
Конкатенация	1	Строки объединяются до выполнения любых других операций.
Арифметический	2	Арифметические операции выполняются после конкатенации строк, но перед выполнением операторов сравнения и логических операций.
Сравнение	3	Операции сравнения вычисляются после конкатенации строк и выполнения арифметических операций, но до логических операций.
Логический	4	Логические операторы выполняются после всех других типов операторов.

Оператор конкатенации

Оператор конкатенации (||) соединяет две символьные строки и создаёт одну строку. Символьные стоки могут быть константами или значениями, полученными из столбцов или других выражений.

Пример:

```
SELECT LAST_NAME || ', ' || FIRST_NAME AS FULL_NAME
FROM EMPLOYEE
```

Арифметические операторы

Таблица 4.3. Приоритет арифметических операторов

Оператор	Назначение	Приоритет
*	Умножение	1
/	Деление	2
+	Сложение	3
-	Вычитание	4

Пример:

```
UPDATE T
SET A = 4 + 1/(B-C)*D
```

Операторы сравнения

Таблица 4.4. Приоритет операторов сравнения

Оператор	Назначение	Приоритет
=	Равно, идентично	1
<>, !=, ~=, ^=	Не равно	2
>	Больше	3
<	Меньше	4
>=	Больше или равно	5
<=	Меньше или равно	6
!>, ~>, ^>	Не больше	7
!<, ~<, ^<	Не меньше	8

В эту же группу входят предикаты сравнения BETWEEN, LIKE, CONTAINING, SIMILAR TO и другие.

Пример:

```
IF (SALARY > 1400) THEN
...
```

См. также: Другие предикаты сравнения.

Логические операторы

Таблица 4.5. Приоритет логических операторов

Оператор	Назначение	Приоритет
NOT	Отрицание условия поиска.	1
AND	Объединяет два предиката и более, каждый из которых должен быть истинным, чтобы истинным был и весь предикат.	2
OR	Объединяет два предиката и более, из которых должен быть истинным хотя бы один предикат, чтобы истинным был и весь предикат.	3

Пример:

```
IF (A > B OR (A > C AND A > D) AND NOT (C = D)) THEN
...
```

NEXT VALUE FOR

Доступно в: DSQL, PSQL.

Синтаксис:

```
NEXT VALUE FOR sequence-name
```

Возвращает следующее значение в последовательности (SEQUENCE). SEQUENCE является SQL совместимым термином генератора в InterBase и Firebird. Оператор NEXT VALUE FOR полностью эквивалентен функции GEN_ID (seq, n) и является рекомендуемым синтаксисом.

Пример:

```
NEW.CUST_ID = NEXT VALUE FOR CUSTSEQ;
```

> **Примечание**
>
> NEXT VALUE FOR не поддерживает значение приращения, отличное от того, что было указано при создании последовательности в предложении INCREMENT [BY]. Если требуется другое значение шага, то используйте старую функцию GEN_ID.

См. также: SEQUENCE (GENERATOR), GEN_ID.

Условные выражения

Условное выражение — это выражение, которое возвращает различные значения в зависимости от истинности некоторого условия или условий. В данном разделе описано лишь

одно условное выражение CASE. Остальные условные выражения являются производными встроенными функциями и описаны в разделе Скалярные функции.

CASE

Доступно в: DSQL, ESQL.

Оператор CASE возвращает только одно значение из нескольких возможных. Есть два синтаксических варианта:

- Простой CASE, сравнимый с Pascal case или C switch;
- Поисковый CASE, который работает как серия операторов "if ... else if ... else if".

Простой CASE

Синтаксис:

```
CASE <test-expr>
  WHEN <expr> THEN <result>
  [WHEN <expr> THEN <result> ...]
  [ELSE <defaultresult>]
END
```

При использовании этого варианта *<test-expr>* сравнивается с *<expr>* 1, *<expr>* 2 и т.д. до тех пор, пока не будет найдено совпадение, и тогда возвращается соответствующий результат. Если совпадений не найдено, то возвращается *defaultresult* из ветви ELSE. Если нет совпадений, и ветвь ELSE отсутствует, возвращается значение NULL.

Совпадение эквивалентно оператору «=», т.е. если *<test-expr>* имеет значение NULL, то он не соответствует ни одному из *<expr>*, даже тем, которые имеют значение NULL.

Результаты не обязательно должны быть литеральными значением, они также могут быть именами полей, переменными, сложными выражениями или NULL литералами.

Сокращённый вид простого оператора CASE используется в функции DECODE.

Пример:

```
SELECT
  NAME,
  AGE,
  CASE UPPER(SEX)
    WHEN 'M' THEN 'Male'
    WHEN 'F' THEN 'Female'
    ELSE 'Unknown'
  END AS SEXNAME,
  RELIGION
FROM PEOPLE
```

Поисковый CASE

Синтаксис:

```
CASE
  WHEN <bool_expr> THEN <result>
  [WHEN <bool_expr> THEN <result> …]
  [ELSE <defaultresult>]
END
```

Здесь *<bool_expr>* выражение, которое даёт тройной логический результат: TRUE, FALSE или NULL. Первое выражение, возвращающее TRUE, определяет результат. Если нет выражений, возвращающих TRUE, то в качестве результата берётся *defaultresult* из ветви ELSE. Если нет выражений, возвращающих TRUE, и ветвь ELSE отсутствует, результатом будет NULL.

Как и в простом операторе CASE, результаты не должны быть буквальным значением: они могут быть полями или именами переменных, сложными выражениями, или иметь значение NULL.

Пример:

```
CANVOTE = CASE
  WHEN AGE >= 18 THEN 'Yes'
  WHEN AGE < 18 THEN 'No'
  ELSE 'Unsure'
END;
```

NULL в выражениях

В SQL NULL не является значением — это состояние, указывающее, что значение элемента неизвестно или не существует. Это не ноль, не пустота, не «пустая строка», и оно не ведёт себя как какое-то из этих значений.

При использовании NULL в числовых, строковых выражениях или в выражениях, содержащих дату/время, в результате вы всегда получите NULL. При использовании NULL в логических (булевых) выражениях результат будет зависеть от типа операции и других вовлечённых значений. При сравнении значения с NULL результат будет неопределённым (UNKNOWN).

> **Важно**
>
> Неопределённый логический результат (UNKNOWN) тоже представлен псевдо-значением NULL.

Выражения возвращающие NULL

Выражения в этом списке всегда возвратят NULL:

```
1 + 2 + 3 + NULL
'Home ' || 'sweet ' || NULL
MyField = NULL
MyField <> NULL
NULL = NULL
```

```
not (NULL)
```

Если вам трудно понять, почему, вспомните, что NULL – значит «неизвестно».

NULL в логических выражениях

Мы уже рассмотрели, что not (NULL) даёт в результате NULL. Для операторов and (логическое И) и or (логическое ИЛИ) взаимодействие несколько сложнее:

```
NULL or false = NULL
NULL or true = true
NULL or NULL = NULL
NULL and false = false
NULL and true = NULL
NULL and NULL = NULL
```

Примеры:

```
(1 = NULL) OR (1 <> 1) -- возвратит NULL
(1 = NULL) OR (1 = 1) -- возвратит TRUE
(1 = NULL) OR (1 = NULL) -- возвратит NULL
(1 = NULL) AND (1 <> 1) -- возвратит FALSE
(1 = NULL) AND (1 = 1) -- возвратит NULL
(1 = NULL) AND (1 = NULL) -- возвратит NULL
```

Подзапросы

Подзапрос — это специальный вид выражения, которое фактически является запросом SELECT к другой таблице, включённый в спецификацию основного запроса. Подзапросы пишутся как обычные SELECT запросы, но должны быть заключены в круглые скобки. Выражения подзапроса используется следующими способами:

- Для задания выходного столбца выходного столбца в списке выбора SELECT;
- Для получения значений или условий для предикатов поиска (предложения WHERE, HAVING).

Коррелированные подзапросы

Подзапрос может быть коррелированным (соотнесённым). Запрос называется соотнесённым, когда оба, и внутренний, и внешний, запросы взаимозависимы. Это означает, что для обработки каждой записи внутреннего запроса, должна быть получена также запись внешнего запроса, т.е. внутренний запрос всецело зависит от внешнего.

Пример коррелированного подзапроса:

```
SELECT *
FROM Customers C
WHERE EXISTS
      (SELECT *
       FROM Orders O
       WHERE C.cnum = O.cnum
         AND O.adate = DATE '10.03.1990');
```

При использовании подзапросов для получения значений выходного столбца в списке выбора SELECT, подзапрос должен возвращать скалярный результат.

Подзапросы возвращающие скалярный результат (Singletons)

Подзапросы, используемые в предикатах поиска, кроме предикатов существования и количественных предикатов, должны возвращать скалярное результат, то есть не более чем один столбец из одной отобранной строки или одно агрегированное значение, в противном случае, произойдёт ошибка времени выполнения ("Multiple rows in a singleton select...").

Примеры:

Пример 4.1. Подзапрос в качестве выходного столбца в списке выбора

```
SELECT
    e.first_name,
    e.last_name,
    (SELECT
        sh.new_salary
     FROM
        salary_history sh
     WHERE
        sh.emp_no = e.emp_no
     ORDER BY sh.change_date DESC ROWS 1) AS last_salary
FROM
    employee e
```

Пример 4.2. Подзапрос в предложении WHERE для получения значения максимальной зарплаты сотрудника и фильтрации по нему

```
SELECT
    e.first_name,
    e.last_name,
    e.salary
FROM
    employee e
WHERE
    e.salary = (SELECT
                    MAX(ie.salary)
                FROM
                    employee ie)
```

Предикаты

Предикат — это простое выражение, утверждающее некоторый факт. Предикат может быть истинным (TRUE), ложным (FALSE) и неопределённым (UNKNOWN). В SQL ложный и неопределённый результаты трактуются как ложь.

В SQL предикаты проверяют в ограничении CHECK, предложении WHERE, выражении CASE, условии соединения во фразе ON для предложений JOIN, а также в предложении HAVING. В PSQL операторы управления потоком выполнения проверяют предикаты в предложениях IF, WHILE и WHEN.

Утверждения

Проверяемые условия не всегда являются простыми предикатами. Они могут быть группой предикатов, каждый из которых при вычислении делает вклад в вычислении общей истинности. Такие сложные условия называются утверждениями. Утверждения могут состоять из одного или нескольких предикатов, связанных логическими операторами AND, OR и NOT.

Каждый из предикатов может содержать вложенные предикаты. Результат вычисления истинности утверждения получается в результате вычисления всех предикатов по направлению от внутренних к внешним. Каждый «уровень» вычисляется в порядке приоритета, до тех пор, пока не возможно будет получить окончательное утверждение.

Предикаты сравнения

Предикат сравнения представляет собой два выражения, соединяемых оператором сравнения. Имеется шесть традиционных операторов сравнения:

```
=, >, <, >=, <=, <>
```

(Полный список операторов сравнения см. Операторы сравнения).

Если в одной из частей (левой или правой) предиката сравнения встречается NULL, то значение предиката будет неопределённым (UNKNOWN).

Примеры:

Получить информацию о компьютерах, имеющих частоту процессора не менее 500 МГц и цену ниже $800

```
SELECT *
FROM Pc
WHERE speed >= 500 AND price < 800;
```

Получить информацию обо всех принтерах, которые являются матричными и стоят меньше $300

```
SELECT *
```

```
FROM Printer
WHERE type = 'matrix' AND price < 300;
```

Следующий запрос не вернёт ни одной записи, поскольку сравнение происходит с псевдо-значением NULL, даже если существуют принтеры с неуказанным типом.

```
SELECT *
FROM Printer
WHERE type = NULL AND price < 300;
```

Другие предикаты сравнения

Другие предикаты сравнения состоят из ключевых слов.

BETWEEN

Доступно в: DSQL, PSQL, ESQL.

Синтаксис:

```
<value> [NOT] BETWEEN <value_1> AND <value_2>
```

Оператор BETWEEN проверяет, попадает (или не попадает при использовании NOT) ли значение во включающий диапазон значений.

Оператор BETWEEN использует два аргумента совместимых типов. В отличие от некоторых других СУБД в Firebird оператор BETWEEN не является симметричным. Меньшее значение должно быть первым аргументом, иначе предикат BETWEEN всегда будет ложным. Поиск является включающим. Таким образом, предикат BETWEEN можно переписать следующим образом:

```
<value> >= <value_1> AND <value> <= <value_2>
```

При использовании предиката BETWEEN в поисковых условиях DML запросов, оптимизатор Firebird может использовать индекс по искомому столбцу, если таковой доступен.

```
SELECT *
FROM EMPLOYEE
WHERE HIRE_DATE BETWEEN date '01.01.1992' AND CURRENT_DATE
```

LIKE

Доступно в: DSQL, PSQL, ESQL.

Синтаксис:

```
```

```
<match value> [NOT] LIKE <pattern>
  [ESCAPE <escape character>]
```

Параметры предиката LIKE

match value
: Выражение символьного типа.

pattern
: Шаблон поиска.

escape character
: Символ экранирования.

Предикат LIKE сравнивает выражение символьного типа с шаблоном, определённым во втором выражении. Сравнение с шаблоном является чувствительным к регистру (за исключением случаев, когда само поле определено с сортировкой (COLLATION) нечувствительной к регистру).

Трафаретные символы

В шаблоне, разрешается использование двух трафаретных символов:

- символ процента (%) заменяет последовательность любых символов (число символов в последовательности может быть от 0 и более) в проверяемом значении;
- символ подчёркивания (_), который можно применять вместо любого единичного символа в проверяемом значении.

Если проверяемое значение соответствует образцу с учётом трафаретных символов, то предикат истинен.

Использование управляющего символа в предложении ESCAPE

Если искомая строка содержит трафаретный символ, то следует задать управляющий символ в предложении ESCAPE. Этот управляющий символ должен использоваться в образце перед трафаретным символом, сообщая о том, что последний следует трактовать как обычный символ.

Найти номера отделов, названия которых начинаются со слова «Software»:

```
SELECT DEPT_NO
FROM DEPT
WHERE DEPT_NAME LIKE 'Software%';
```

В данном запросе может быть использован индекс, если он построен на поле DEPT_NAME.

> **Оптимизация LIKE**
>
> В общем случае предикат LIKE не использует индекс. Однако если предикат принимает вид LIKE 'строка%', то он будет преобразован в предикат STARTING WITH, который будет использовать индекс. Если вам необходимо выполнить поиск с начала строки, то вместо предиката LIKE рекомендуется использовать предикат STARTING WITH.

Поиск сотрудников, имена которых состоят из 5 букв, начинающихся с букв «Sm» и заканчивающихся на «th». В данном случае предикат будет истинен для имен «Smith» и «Smyth».

```
SELECT
    first_name
FROM
    employee
WHERE first_name LIKE 'Sm_th'
```

Поиск всех заказчиков, в адресе которых содержится строка «Ростов».

```
SELECT *
FROM CUSTOMER
WHERE ADDRESS LIKE '%Ростов%'
```

> **Подсказка**
>
> Если вам необходимо выполнить поиск внутри строки, то вместо предиката LIKE рекомендуется использовать предикат CONTAINING.

Поиск таблиц, содержащих в имени знак подчёркивания. В данном случае в качестве управляющего символа задан символ «#».

```
SELECT
  RDB$RELATION_NAME
FROM RDB$RELATIONS
WHERE RDB$RELATION_NAME LIKE '%#_%' ESCAPE '#'
```

См. также: STARTING WITH, CONTAINING, SIMILAR TO.

STARTING WITH

Доступно в: DSQL, PSQL, ESQL.

Синтаксис:

```
<value> [NOT] STARTING WITH <value>
```

Оператор STARTING WITH ищет строку или тип, подобный строке, которая начинается с символов в его аргументе. Поиск STARTING WITH чувствителен к регистру.

При использовании предиката STARTING WITH в поисковых условиях DML запросов, оптимизатор Firebird может использовать индекс по искомому столбцу, если он определён.

Поиск сотрудников, фамилия которых начинается с «Jo».

```
SELECT LAST_NAME, FIRST_NAME
FROM EMPLOYEE
```

```
WHERE LAST_NAME STARTING WITH 'Jo'
```

См. также: LIKE.

CONTAINING

Доступно в: DSQL, PSQL, ESQL.

Синтаксис:

```
<value> [NOT] CONTAINING <value>
```

Оператор CONTAINING ищет строку или тип, подобный строке, отыскивая последовательность символов, которая соответствует его аргументу. Он может быть использован для алфавитно-цифрового (подобного строковому) поиска в числах и датах. Поиск CONTAINING не чувствителен к регистру.

При использовании предиката CONTAINING в поисковых условиях DML запросов, оптимизатор Firebird не может использовать индекс по искомому столбцу.

Поиск проектов в именах, которых присутствует подстрока «Map»:

```
SELECT *
FROM PROJECT
WHERE PROJ_NAME CONTAINING 'map';
```

В данном случае будет возвращены две строки с именами «AutoMap» и «MapBrowser port».

Поиск записей об изменении зарплат с датой содержащей число 84 (в данном случае изменения, которые произошли в 1984 году):

```
SELECT *
FROM SALARY_HISTORY
WHERE CHANGE_DATE CONTAINING 84;
```

См. также: LIKE.

SIMILAR TO

Доступно в: DSQL, PSQL.

Синтаксис:

```
<match value> [NOT] SIMILAR TO <pattern> [ESCAPE <escape character>]
```

Параметры предиката SIMILAR TO

match value
Выражение символьного типа.

pattern
Регулярное выражение SQL.

escape character
Символ экранирования.

Оператор SIMILAR TO проверяет соответствие строки с шаблоном регулярного выражения SQL. В отличие от некоторых других языков для успешного выполнения шаблон должен соответствовать всей строке — соответствие подстроки не достаточно. Если один из операндов имеет значение NULL, то и результат будет NULL. В противном случае результат является TRUE или FALSE.

Синтаксис регулярных выражений SQL

Следующий синтаксис определяет формат регулярного выражения SQL. Это полное и корректное его определение. Кроме того, он является весьма формальным и довольно длинным и, вероятно, озадачивает тех, кто не имеет некоторого опыта работы с регулярными выражениями (или с очень формальными и довольно длинными нисходящими определениями). Не стесняйтесь пропустить его и начать читать следующий раздел, Создание регулярных выражений, использующий подход от простого к сложному.

```
<regular expression> ::= <regular term> ['|' <regular term> ...]

<regular term> ::= <regular factor> ...

<regular factor> ::= <regular primary> [<quantifier>]

<quantifier> ::= ?
                 | *
                 | +
                 | '{' <m> [,[<n>]] '}'

<m>, <n> ::= целое положительное число с <m> <= <n> если оба присутствуют

<regular primary> ::= <character>
                      | <character class>
                      | %
                      | (<regular expression>)

<character> ::= <escaped character>
                | <non-escaped character>

<escaped character> ::= <escape-char> <special character>
                        | <escape-char> <escape-char>

<special character> ::= любой из символов []()|^-+*%_?{

<non-escaped character> ::= любой символ за исключением <special character>
                            и не эквивалентный <escape-char> (если задан)

<character class> ::= '_'
                      | '[' <member> ... ']'
                      | '[^' <non-member> ... ']'
                      | '[' <member> ... '^' <non-member> ... ']'

<member>, <non-member> ::= <character>
```

```
                          | <range>
                          | <predefined class>

<range> ::= <character>-<character>

<predefined class> ::= '[:' <predefined class name> ':]'

<predefined class name> ::= ALPHA | UPPER | LOWER | DIGIT
                          |  ALNUM  |  SPACE  |  WHITESPACE
```

Создание регулярных выражений

Символы

В регулярных выражениях большинство символов представляет сами себя. Единственное исключение — специальные символы (special character):

```
[ ] ( ) | ^ - + * % _ ? {
```

и управляющие символы, если они заданы.

Регулярному выражению, не содержащему специальных или управляющих символов, соответствует только полностью идентичные строки (в зависимости от используемой сортировки). То есть это функционирует точно так же, как оператор "=":

```
'Apple' SIMILAR TO 'Apple' -- TRUE
'Apples' SIMILAR TO 'Apple' -- FALSE
'Apple' SIMILAR TO 'Apples' -- FALSE
'APPLE' SIMILAR TO 'Apple' -- в зависимости от сортировки
```

Шаблоны

Известным SQL шаблонам _ и % соответствует любой единственный символ и строка любой длины, соответственно:

```
'Birne' SIMILAR TO 'B_rne' -- TRUE
'Birne' SIMILAR TO 'B_ne' -- FALSE
'Birne' SIMILAR TO 'B%ne' -- TRUE
'Birne' SIMILAR TO 'Bir%ne%' -- TRUE
'Birne' SIMILAR TO 'Birr%ne' -- FALSE
```

Обратите внимание, что шаблон % также соответствует пустой строке.

Классы символов

Набор символов, заключённый в квадратные скобки определяют класс символов. Символ в строке соответствует классу в шаблоне, если символ является элементом класса:

```
'Citroen' SIMILAR TO 'Cit[arju]oen' -- TRUE
'Citroen' SIMILAR TO 'Ci[tr]oen' -- FALSE
'Citroen' SIMILAR TO 'Ci[tr][tr]oen' -- TRUE
```

Как видно из второй строки классу только соответствует единственный символ, а не их последовательность.

Два символа, соединённые дефисом, в определении класса определяют диапазон. Диапазон для активного сопоставления включает в себя эти два конечных символа и все символы, находящиеся между ними. Диапазоны могут быть помещены в любом месте в определении класса без специальных разделителей, чтобы сохранить в классе и другие символы.

```
'Datte' SIMILAR TO 'Dat[q-u]e' -- TRUE
'Datte' SIMILAR TO 'Dat[abq-uy]e' -- TRUE
'Datte' SIMILAR TO 'Dat[bcg-km-pwz]e' -- FALSE
```

`[:ALPHA:]`
Латинские буквы a...z и A...Z. С регистро-нечувствительными сортировками этот класс также включает подчёркнутые символы.

`[:DIGIT:]`
Десятичные цифры 0...9.

`[:ALNUM:]`
Объединение [:ALPHA:] и [:DIGIT:].

`[:UPPER:]`
Прописные (в верхнем регистре) латинские буквы A...Z. Также включает в себя нижний регистр при регистро-нечувствительных сортировках и подчёркнутые символы при регистро-нечувствительных сортировках.

`[:LOWER:]`
Строчные (в нижнем регистре) латинские буквы a...z. Также включает в себя нижний регистр при регистро-нечувствительных сортировках и подчёркнутые символы при регистро-нечувствительных сортировках.

`[:SPACE:]`
Символ пробела (ASCII 32).

`[:WHITESPACE:]`
Вертикальная табуляции (ASCII 9), перевод строки (ASCII 10), горизонтальная табуляция (ASCII 11), разрыв страницы (ASCII 12), возврат каретки (ASCII 13) и пробел (ASCII 32).

Включение в оператор SIMILAR TO предопределённого класса имеет тот же эффект, как и включение всех его элементов. Использование предопределённых классов допускается только в пределах определения класса. Если Вам нужно сопоставление только с предопределённым классом и ничего больше, то поместите дополнительную пару скобок вокруг него.

В эту группу предикатов включены предикаты, которые используют подзапросы для передачи значений для различного вида утверждений в условиях поиска. Предикаты существования называются так потому, что они различными способами проверяют существование значения в левой части предиката в выходных результатах подзапросов.

```
'Erdbeere' SIMILAR TO 'Erd[[:ALNUM:]]eere' -- TRUE
'Erdbeere' SIMILAR TO 'Erd[[:DIGIT:]]eere' -- FALSE
'Erdbeere' SIMILAR TO 'Erd[a[:SPACE:]b]eere' -- TRUE
'Erdbeere' SIMILAR TO '[[:ALPHA:]]' -- FALSE
'E' SIMILAR TO '[[:ALPHA:]]' -- TRUE
```

Если определение класса запускается со знаком вставки (^), то все, что следует за ним, исключается из класса. Все остальные символы проверяются.

```
'Framboise' SIMILAR TO 'Fra[^ck-p]boise' -- FALSE
'Framboise' SIMILAR TO 'Fr[^a][^a]boise' -- FALSE
'Framboise' SIMILAR TO 'Fra[^[:DIGIT:]]boise' -- TRUE
```

Если знак вставки (^) находится не в начале последовательности, то класс включает в себя все символы до него и исключает символы после него.

```
'Grapefruit' SIMILAR TO 'Grap[a-m^f-i]fruit' -- TRUE
'Grapefruit' SIMILAR TO 'Grap[abc^xyz]fruit' -- FALSE
'Grapefruit' SIMILAR TO 'Grap[abc^de]fruit' -- FALSE
'Grapefruit' SIMILAR TO 'Grap[abe^de]fruit' -- FALSE
'3' SIMILAR TO '[[:DIGIT:]^4-8]' -- TRUE
'6' SIMILAR TO '[[:DIGIT:]^4-8]' -- FALSE
```

Наконец, уже упомянутый подстановочный знак "_" является собственным классом символов, соответствуя любому единственному символу.

Кванторы

Вопросительный знак сразу после символа или класса указывает на то, что для соответствия предыдущий элемент может произойти 0 или 1 раз:

```
'Hallon' SIMILAR TO 'Hal?on' -- FALSE
'Hallon' SIMILAR TO 'Hal?lon' -- TRUE
'Hallon' SIMILAR TO 'Halll?on' -- TRUE
'Hallon' SIMILAR TO 'Hallll?on' -- FALSE
'Hallon' SIMILAR TO 'Halx?lon' -- TRUE
'Hallon' SIMILAR TO 'H[a-c]?llon[x-z]?' -- TRUE
```

Звёздочка (*) сразу после символа или класса указывает на то, что для соответствия предыдущий элемент может произойти 0 или более раз:

```
'Icaque' SIMILAR TO 'Ica*que' -- TRUE
```

```
'Icaque' SIMILAR TO 'Icar*que' -- TRUE
'Icaque' SIMILAR TO 'I[a-c]*que' -- TRUE
'Icaque' SIMILAR TO '_*' -- TRUE
'Icaque' SIMILAR TO '[[:ALPHA:]]*' -- TRUE
'Icaque' SIMILAR TO 'Ica[xyz]*e' -- FALSE
```

Знак плюс (+) сразу после символа или класса указывает на то, что для соответствия предыдущий элемент может произойти 1 или более раз:

```
'Jujube' SIMILAR TO 'Ju_+' -- TRUE
'Jujube' SIMILAR TO 'Ju+jube' -- TRUE
'Jujube' SIMILAR TO 'Jujuber+' -- FALSE
'Jujube' SIMILAR TO 'J[jux]+be' -- TRUE
'Jujube' SIMILAR TO 'J[[:DIGIT:]]+ujube' -- FALSE
```

Если символ или класс сопровождаются числом, заключённым в фигурные скобки, то для соответствия нужно повторение элемента точно в это число раз:

```
'Kiwi' SIMILAR TO 'Ki{2}wi' -- FALSE
'Kiwi' SIMILAR TO 'K[ipw]{2}i' -- TRUE
'Kiwi' SIMILAR TO 'K[ipw]{2}' -- FALSE
'Kiwi' SIMILAR TO 'K[ipw]{3}' -- TRUE
```

Если число сопровождается запятой, то для соответствия нужно повторение элемента как минимум в это число раз:

```
'Limone' SIMILAR TO 'Li{2,}mone' -- FALSE
'Limone' SIMILAR TO 'Li{1,}mone' -- TRUE
'Limone' SIMILAR TOto 'Li[nezom]{2,}' -- TRUE
```

Если фигурные скобки содержат два числа (m и n), разделённые запятой, и второе число больше первого, то для соответствия элемент должен быть повторен, как минимум, m раз и не больше m+1 раз:

```
'Mandarijn' SIMILAR TO 'M[a-p]{2,5}rijn' -- TRUE
'Mandarijn' SIMILAR TO 'M[a-p]{2,3}rijn' -- FALSE
'Mandarijn' SIMILAR TO 'M[a-p]{2,3}arijn' -- TRUE
```

Кванторы ?, * и + сокращение для {0,1}, {0,} и {1.}, соответственно.

Термин ИЛИ

В условиях регулярных выражений можно использовать оператор ИЛИ (|). Соответствие произошло, если строка параметра соответствует, по крайней мере, одному из условий:

```
'Nektarin' SIMILAR TO 'Nek|tarin' -- FALSE
```

```
'Nektarin' SIMILAR TO 'Nektarin|Persika' -- TRUE
'Nektarin' SIMILAR TO 'M_+|N_+|P_+' - TRUE
```

Подвыражения

Одна или более частей регулярного выражения могут быть сгруппированы в подвыражения (также называемые подмасками). Для этого их нужно заключить в круглые скобки:

```
'Orange' SIMILAR TO 'O(ra|ri|ro)nge' -- TRUE
'Orange' SIMILAR TO 'O(r[a-e])+nge' -- TRUE
'Orange' SIMILAR TO 'O(ra){2,4}nge' -- FALSE
'Orange' SIMILAR TO 'O(r(an|in)g|rong)?e' - TRUE
```

Экранирование специальных символов

Для исключения из процесса сопоставления специальных символов (которые часто встречаются в регулярных выражениях) их надо экранировать. Специальных символов экранирования по умолчанию нет — их при необходимости определяет пользователь:

```
'Peer (Poire)' SIMILAR TO 'P[^ ]+ \(P[^ ]+\)' ESCAPE '\' -- TRUE
'Pera [Pear]' SIMILAR TO 'P[^ ]+ #[P[^ ]+#]' ESCAPE '#' -- TRUE
'Paron-Appledryck' SIMILAR TO 'P%$-A%' ESCAPE '$' -- TRUE
'Parondryck' SIMILAR TO 'P%--A%' ESCAPE '-' -- FALSE
```

IS DISTINCT FROM

Доступно в: DSQL, PSQL.

Синтаксис:

```
op1 IS [NOT] DISTINCT FROM op2
```

Два операнда считают различными (DISTINCT), если они имеют различные значения, или если одно из них — NULL, и другое нет. Они считаются NOT DISTINCT (равными), если имеют одинаковые значения или оба имеют значение NULL.

IS [NOT] DISTINCT FROM всегда возвращает TRUE или FALSE и никогда UNKNOWN (NULL) (неизвестное значение). Операторы «=» и «<>», наоборот, вернут UNKNOWN (NULL), если один или оба операнда имеют значение NULL.

Таблица 4.6. Результаты выполнения различных операторов сравнения

Характеристики операнда	Результаты различных операторов			
	=	IS NOT DISTINCT FROM	<>	IS DISTINCT FROM
Одинаковые значения	TRUE	TRUE	FALSE	FALSE

Характеристики операнда	Результаты различных операторов			
	=	IS NOT DISTINCT FROM	<>	IS DISTINCT FROM
Различные значения	FALSE	FALSE	TRUE	TRUE
Оба NULL	UNKNOWN	TRUE	UNKNOWN	FALSE
Одно NULL	UNKNOWN	FALSE	UNKNOWN	TRUE

Примеры:

```
SELECT ID, NAME, TEACHER
FROM COURSES
WHERE START_DAY IS NOT DISTINCT FROM END_DAY

IF (NEW.JOB IS DISTINCT FROM OLD.JOB)
THEN POST_EVENT 'JOB_CHANGED';
```

См. также: IS NULL.

IS

Доступно в: DSQL, PSQL.

Синтаксис:

```
<value> IS [NOT] {TRUE | FALSE | UNKNOWN}
```

Данный оператор применим только к переменным, полям или выражениям логического типа. Оператор IS проверяет, что выражение в левой части соответствует логическому значению в правой части.

> **Замечание:**
>
> Значение истинности UNKNOWN представлено как псевдо-значение NULL.

Пример 4.3. Использование оператора IS с логическим типом данных

```
-- Проверка FALSE значения
SELECT * FROM TBOOL WHERE BVAL IS FALSE
```

```
ID           BVAL
============ =======
2            <false>
```

```
-- Проверка UNKNOWN значения
```

```
SELECT * FROM TBOOL WHERE BVAL IS UNKNOWN
```

```
ID           BVAL
============ =======
3            <null>
```

IS NULL

Доступно в: DSQL, PSQL, ESQL.

Синтаксис:

```
<value> IS [NOT] NULL
```

Поскольку NULL не является значением, эти операторы не являются операторами сравнения. Предикат IS [NOT] NULL проверяет утверждение, что выражение в левой части имеет значение (IS NOT NULL) или не имеет значения (IS NULL).

Пример 4.4. Предикат IS NULL

Поиск записей о продажах, для которых не установлена дата отгрузки:

```
SELECT *
FROM SALES
WHERE SHIP_DATE IS NULL;
```

Предикаты существования

В эту группу предикатов включены предикаты, которые используют подзапросы для передачи значений для различного вида утверждений в условиях поиска. Предикаты существования называются так потому, что они различными способами проверяют существование значения в левой части предиката в выходных результатах подзапросов.

EXISTS

Доступно в: DSQL, PSQL, ESQL.

Синтаксис:

```
[NOT] EXISTS (<select_stmt>)
```

Предикат EXISTS использует подзапрос в качестве аргумента и оценивает его как истинный, если в подзапросе есть выходные данные, в противном случае предикат оценивается как ложный. Результат подзапроса может содержать несколько столбцов, поскольку значения не проверяются, а просто фиксируется факт наличия строк результата. Данный предикат может принимать только два значения: истина (TRUE) и ложь (FALSE).

Примеры:

Пример 4.5. Предикат EXISTS

Найти тех сотрудников, у которых есть проекты.

```
SELECT *
FROM employee
WHERE EXISTS (SELECT *
              FROM
                employee_project ep
              WHERE
                ep.emp_no = employee.emp_no)
```

Пример 4.6. Предикат NOT EXISTS

Найти тех сотрудников, у которых нет проектов.

```
SELECT *
FROM employee
WHERE NOT EXISTS (SELECT *
                  FROM
                    employee_project ep
                  WHERE
                    ep.emp_no = employee.emp_no)
```

IN

Доступно в: DSQL, PSQL, ESQL.

Синтаксис:

```
<value> [NOT] IN (<select_stmt> | <value_list>)

<value_list> ::= <value_1> [, <value_2> ...]
```

Предикат IN проверяет, присутствует ли значение выражения слева в указанном справа наборе значений. Набор значений не может превышать 1500 элементов. Предикат IN может быть переписан в следующей эквивалентной форме:

```
(<value> = <value_1> [OR <value> = <value_2> ...])
```

При использовании предиката IN в поисковых условиях DML запросов, оптимизатор Firebird может использовать индекс по искомому столбцу, если он определён.

Во второй форме предикат IN проверяет, присутствует (или отсутствует, при использовании NOT IN) ли значение выражения слева в результате выполнения подзапроса справа. Результат подзапроса может содержать только один столбец, иначе будет выдана ошибка "count of column list and variable list do not match".

Запросы с использованием предиката IN с подзапросом, можно переписать на аналогичный запрос с использованием предиката EXISTS. Например, следующий запрос:

```
SELECT
  model, speed, hd
FROM PC
WHERE
  model IN (SELECT model
            FROM product
            WHERE maker = 'A');
```

Можно переписать на аналогичный запрос с использованием предиката EXISTS:

```
SELECT
  model, speed, hd
FROM PC
WHERE
  EXISTS (SELECT *
          FROM product
          WHERE maker = 'A'
            AND product.model = PC.model);
```

Однако, запрос с использованием NOT IN не всегда даст тот же результат, что запрос NOT EXISTS. Причина заключается в том, что предикат EXISTS всегда возвращает TRUE или FALSE, тогда как предикат IN может вернуть NULL в следующих случаях:

1. Когда проверяемое значений равно NULL и список в IN не пуст.
2. Когда проверяемое значение не имеет совпадений в списке IN и одно из значений является NULL.

В этих двух случаях предикат IN вернёт NULL, в то время как соответствующий предикат EXISTS вернёт FALSE. В поисковых условиях или операторе IF оба результата обозначают "провал" и обрабатываются одинаково.

Однако на тех же данных NOT IN вернёт NULL, в то время как EXISTS вернёт TRUE, что приведёт к противоположному результату.

Это можно продемонстрировать следующим примером.

Предположим у вас есть такой запрос:

```
-- Ищем людей, которые не родились в тот же день, что
-- известные жители Нью-Йорка
SELECT P1.name AS NAME
FROM Personnel P1
WHERE P1.birthday NOT IN (SELECT C1.birthday
                          FROM Celebrities C1
                          WHERE C1.birthcity = 'New York');
```

Можно предположить, что аналогичный результат даст запрос с использованием предиката NOT EXISTS:

```
-- Ищем людей, которые не родились в тот же день, что
-- известные жители Нью-Йорка
SELECT P1.name AS NAME
FROM Personnel P1
WHERE NOT EXISTS (SELECT *
                  FROM Celebrities C1
                  WHERE C1.birthcity = 'New York'
                    AND C1.birthday = P1.birthday);
```

Допустим, что в Нью-Йорке всего один известный житель, и его дата рождения неизвестна. При использовании предиката EXISTS подзапрос внутри него не выдаст результатов, так как при сравнении дат рождения с NULL результатом будет UNKNOWN. Это приведёт к тому, что результат предиката NOT EXISTS будет истинен для каждой строки основного запроса. В то время как результатом предиката NOT IN будет UNKNOWN и ни одна строка не будет выведена.

Примеры:

Пример 4.7. Предикат IN

Найти сотрудников с именами «Pete», «Ann» и «Roger»:

```
SELECT *
FROM EMPLOYEE
WHERE FIRST_NAME IN ('Pete', 'Ann', 'Roger');
```

Пример 4.8. Поисковый предикат IN

Найти все компьютеры, для которых существуют модели, производитель которых начинается на букву «A»:

```
SELECT
  model, speed, hd
FROM PC
WHERE
  model IN (SELECT model
            FROM product
            WHERE maker STARTING WITH 'A');
```

См. также: EXISTS.

SINGULAR

Доступно в: DSQL, PSQL, ESQL.

Синтаксис:

```
[NOT] SINGULAR (<select_stmt>)
```

Предикат SINGULAR использует подзапрос в качестве аргумента и оценивает его как истинный, если подзапрос возвращает одну и только одну строку результата, в противном случае предикат оценивается как ложный. Результат подзапроса может содержать несколько

столбцов, поскольку значения не проверяются. Данный предикат может принимать только два значения: истина (TRUE) и ложь (FALSE).

Пример 4.9. Предикат SINGULAR

Найти тех сотрудников, у которых есть только один проект.

```
SELECT *
FROM employee
WHERE SINGULAR (SELECT *
                FROM
                  employee_project ep
                WHERE
                  ep.emp_no = employee.emp_no)
```

Количественные предикаты подзапросов

Квантором называется логический оператор, задающий количество объектов, для которых данное утверждение истинно. Это логическое количество, а не числовое; оно связывает утверждение с полным множеством возможных объектов. Такие предикаты основаны на формальных логических квантификаторах общности и существования, которые в формальной логике записываются как ∀ и ∃.

В выражениях подзапросов количественные предикаты позволяют сравнивать отдельные значения с результатами подзапросов; их общая форма:

```
<value expression> <comp op> <quantifier> <subquery>
```

ALL

Доступно в: DSQL, PSQL.

Синтаксис:

```
<value> <op> ALL (<select_stmt>)
```

При использовании квантора ALL, предикат является истинным, если каждое значение выбранное подзапросом удовлетворяет условию в предикате внешнего запроса. Если подзапрос не возвращает ни одной строки, то предикат автоматически считается верным.

Пример 4.10. Квантор ALL

Вывести только тех заказчиков, чьи оценки выше, чем у каждого заказчика в Париже.

```
SELECT *
FROM Customers
WHERE rating > ALL
      (SELECT rating
       FROM Customers
```

```
        WHERE city = 'Paris')
```

> **Важно**
>
> Если подзапрос возвращает пустое множество, то предикат будет истинен для каждого левостороннего значения, независимо от оператора. Это может показаться странным и противоречивым, потому что в этом случае каждое левостороннее значение рассматривается как одновременно больше, меньше, равное и неравное любому значению из правого потока.
>
> Тем не менее, это нормально согласуется с формальной логикой: если множество пусто, то предикат верен 0 раз, т.е. для каждой строки в множестве.

ANY и SOME

Доступно в: DSQL, PSQL.

Синтаксис:

```
<value> <op> {ANY | SOME} (<select_stmt>)
```

Эти два квантора идентичны по поведению. Очевидно, оба представлены в стандарте SQL для взаимозаменяемого использования с целью улучшения читаемости операторов. При использовании квантора ANY или SOME, предикат является истинным, если любое из значений выбранное подзапросом удовлетворяет условию в предикате внешнего запроса. Если подзапрос не возвращает ни одной строки, то предикат автоматически считается ложным.

Пример 4.11. Квантор ANY

Вывести только тех заказчиков, чьи оценки выше, чем у любого заказчика в Риме.

```
SELECT *
FROM Customers
WHERE rating > ANY
      (SELECT rating
       FROM Customers
       WHERE city = 'Rome')
```

Глава 5

Операторы DDL

Data Definition Language (DDL) — язык описания данных. С помощью этого подмножества языка создаются, модифицируются и удаляются объекты базы данных (т.е. Метаданные).

DATABASE

В данном разделе описываются вопросы создания базы данных, подключения к существующей базе данных, изменения структуры файлов, перевод базы данных в состояние, необходимое для безопасного резервного копирования, и обратно и удаления базы данных.

CREATE DATABASE

Назначение: Создание новой базы данных.

Доступно в: DSQL, ESQL.

Синтаксис:

```
CREATE {DATABASE | SCHEMA} '<filespec>'
  [USER username [PASSWORD 'password'] [ROLE rolename]]
  [PAGE_SIZE [=] size]
  [LENGTH [=] num [PAGE[S]]
  [SET NAMES charset]
  [DEFAULT CHARACTER SET default_charset
  [COLLATION collation]]
  [<sec_file> [<sec_file> ...]]
  [DIFFERENCE FILE 'diff_file'];

<filespec> ::= [<server_spec>]{filepath | db_alias}

<server_spec> ::=
    host[\port]:
  | \\host\
  | <protocol>://[host[:port]/]

<protocol> = inet | wnet | xnet

<sec_file> ::= FILE 'filepath'
  [LENGTH [=] num [PAGE[S]] [STARTING [AT [PAGE]] pagenum]
```

Параметры оператора CREATE DATABASE

filespec
Спецификация первичного файла базы данных.

server_spec
Спецификация удалённого сервера. Включает в себя имя сервера и протокол. Необходима, если база данных создаётся на удалённом сервере.

filepath
Полный путь и имя файла, включая расширение. Имя файла должно быть задано в соответствии со спецификой используемой платформы.

db_alias
Псевдоним (alias) базы данных, присутствующий в файле `databases.conf`

host
Имя сервера или IP адрес, на котором создаётся база данных.

port
Номер порта, который слушает удалённый сервер.

protocol
Наименование протокола.

username
Имя пользователя-владельца базы данных. Может включать в себя до 31 символа. Может быть заключено в одинарные или двойные кавычки. Если имя пользователя заключено в двойные кавычки, то оно чувствительно к регистру.

password
Пароль пользователя-владельца базы данных. Может включать в себя до 32 символов. Чувствительно к регистру.

role
Имя роли, права которой могут учитываться при создании базы данных. Может быть заключено в одинарные или двойные кавычки. Если имя роли заключено в двойные кавычки, то оно чувствительно к регистру.

size
Размер страницы для базы данных. Допустимые значения 4096, 8192, 16384. Размер страницы по умолчанию 8192.

num
Максимальный размер первичного или вторичного файла в страницах.

charset
Задаёт набор символов подключения, доступного после успешного создания базы данных.

default_charset
Задаёт набор символов по умолчанию для строковых типов данных.

collation
Сортировка для набора символов по умолчанию.

sec_file
Спецификация вторичного файла.

pagenum
Номер страницы, с которой начинается вторичный файл базы данных.

diff_file
Путь и имя дельта файла.

Оператор CREATE DATABASE создаёт новую базу данных. Вы можете использовать CREATE DATABASE или CREATE SCHEMA. Это синонимы.

База данных может состоять из одного или нескольких файлов. Первый, основной, файл называется первичным, остальные файлы — вторичными.

> **Примечание**
>
> В настоящее время многофайловые базы данных являются атавизмом. Многофайловые базы данных имеет смысл использовать на старых файловых системах, в которых существует ограничение на размер любого файла. Например, в FAT32 нельзя создать файл больше 4х гигабайт.

Спецификация первичного файла — имя файла базы данных и его расширение с указанием к нему полного пути в соответствии с правилами используемой операционной системы. При создании базы данных файл базы данных должен отсутствовать. В противном случае будет выдано сообщение об ошибке и база данных не будет создана. Если полный путь к базе данных не указан, то база данных будет создана в одном из системных каталогов. В каком именно зависит от операционной системы.

Использование псевдонимов БД

Вместо полного пути к первичному файлу базы можно использовать псевдонимы (aliases). Псевдонимы описываются в файле `databases.conf` в формате:

```
alias = filepath
```

> **Примечание**
>
> Помимо указания псевдонимов для базы данных в этом файле можно задать параметры уровня базы данных для каждой описываемой базы данных. Эти параметры задаются в фигурных скобках сразу после объявления псевдонима.

Создание удалённых БД

При создании базы данных на удалённом сервере необходимо указать спецификацию удалённого сервера. Спецификация удалённого сервера зависит от используемого протокола. Если вы при создании базы данных используете протокол TCP/IP, то спецификация первичного файла должна выглядеть следующим образом:

```
host[/port]:{filepath | db_alias}
```

Если вы при создании базы данных используете протокол под названием именованные каналы (Name Pipes), то спецификация первичного файла должна выглядеть следующим образом.

```
```

```
\\host\{filepath | db_alias}
```

Существует также унифицированный синтаксис спецификации удалённого сервера. В этом синтаксисе первым параметром указывается наименование протокола, далее указывается имя сервера или IP адрес, номер порта и путь к первичному файлу базы данных или псевдоним. В качестве протокола можно указать следующие значения:

- INET — TCP/IP;
- WNET — NetBEUI или протокол именованных каналов;
- XNET — локальный протокол.

```
protocol://[host[:port]/]{filepath | db_alias}
```

Необязательные параметры CREATE DATABASE

Необязательные предложения USER и PASSWORD задают, соответственно, имя и пароль пользователя присутствующего в базе данных безопасности (`security3.fdb` или той, что указана в параметре *SecurityDatabase*). Пользователя и пароль можно не указывать, если установлены переменные окружения `ISC_USER` и `ISC_PASSWORD`. Необязательное предложение ROLE задаёт имя роли (обычно это RDB$ADMIN), права которой будут учитываться при создании базы данных. Роль должна быть назначена пользователю в соответствующей базе данных безопасности.

Пользователь, указанный при создании базы данных, будет её владельцем.

Необязательное предложение PAGE_SIZE задаёт размер страницы базы данных. Этот размер будет установлен для первичного файла и всех вторичных файлов базы данных. При вводе размера страницы БД меньшего, чем 4096, он будет автоматически изменён на 4096. Другие числа (не равные 4096, 8192 или 16384) будут изменены на ближайшее меньшее из поддерживаемых значений. Если размер страницы базы данных не указан, то по умолчанию принимается значение 8192.

Необязательное предложение LENGTH задаёт максимальный размер первичного или вторичного файла базы данных в страницах. При создании базы данных её первичный или вторичный файл будут занимать минимально необходимое количество страниц для хранения системных данных, не зависимо от величины, установленной в предложении LENGHT. Для единственного или последнего (в многофайловой базе данных) файла значение LENGTH никак не влияет на его размер. Файл будет автоматически увеличивать свой размер по мере необходимости.

Необязательное предложение SET NAMES задаёт набор символов подключения, доступного после успешного создания базы данных. По умолчанию используется набор символов NONE.

Необязательное предложение DEFAULT CHARACTER SET задаёт набор символов по умолчанию для строковых типов данных. Наборы символов применяются для типов CHAR, VARCHAR и BLOB. По умолчанию используется набор символов NONE. Для набора символов по умолчанию можно также указать сортировку по умолчанию (COLLATION). В этом случае сортировка станет умалчиваемой для набора символов по умолчанию (т.е. для всей БД за исключением случаев использования других наборов символов).

Предложение STARTING AT задаёт номер страницы базы данных, с которой должен начинаться следующий файл базы данных. Когда предыдущий файл будет полностью заполнен данными в соответствии с заданным номером страницы, система начнёт помещать вновь добавляемые данные в следующий файл базы данных.

Необязательное предложение DIFFERENCE FILE задаёт путь и имя дельта файла, в который будут записываться изменения, внесённые в БД после перевода её в режим "безопасного копирования" ("copy-safe") путём выполнения команды ALTER DATABASE BEGIN BACKUP. Полное описание данного параметра см. в ALTER DATABASE.

Для того чтобы база данных была создана в нужном вам диалекте SQL, следует перед выполнением оператора создания базы данных задать нужный диалект, выполнив оператор SET SQL DIALECT. По умолчанию база данных создаётся в 3 диалекте.

Кто может создать базу данных?

Создать новую базу данных могут:

- SYSDBA;
- Любой пользователь с привилегией на создание базы данных (GRANT CREATE DATABASE);
- Любой пользователь, указавший роль RDB$ADMIN, роль должна быть выдана в соответствующей базе данных безопасности (у пользователя должна присутствовать опция GRANT ADMIN ROLE);
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.1. Создание базы данных в операционной системе Windows

Создание базы данных в операционной системе Windows расположенной на диске D с размером страницы 8192. Владельцем базы данных будет пользователь wizard. База данных будет в 1 диалекте, и использовать набор символов по умолчанию WIN1251.

```
SET SQL DIALECT 1;
CREATE DATABASE 'D:\test.fdb'
USER wizard PASSWORD 'player' ROLE RDB$ADMIN
DEFAULT CHARACTER SET WIN1251;
```

Пример 5.2. Создание базы данных в операционной системе Linux

Создание базы данных в операционной системе Linux с размером страницы 4096. Владельцем базы данных будет пользователь wizard. База данных будет в 3 диалекте, и использовать набор символов по умолчанию UTF8 с умалчиваемой сортировкой UNICODE_CI_AI.

```
CREATE DATABASE '/home/firebird/test.fdb'
USER "wizard" PASSWORD 'player' ROLE 'RDB$ADMIN'
PAGE_SIZE = 4096
DEFAULT CHARACTER SET UTF8 COLLATION UNICODE_CI_AI;
```

Важно

В данном случае при создании базы данных будет учитываться регистр символов для имени пользователя, потому что оно указано в двойных кавычках.

Пример 5.3. Создание базы данных на удалённом сервере

Создание базы данных на удалённом сервере baseserver расположенном по пути, на который ссылается псевдоним test, описанный в файле `databases.conf`. Используется протокол TCP. Владельцем базы данных будет пользователь wizard.

```
CREATE DATABASE 'baseserver:test'
USER wizard PASSWORD 'player' ROLE RDB$ADMIN
DEFAULT CHARACTER SET UTF8;
```

То же самое с использованием унифицированного синтаксиса задания спецификации удалённого сервера.

```
CREATE DATABASE 'inet://baseserver:3050/test'
USER wizard PASSWORD 'player' ROLE RDB$ADMIN
DEFAULT CHARACTER SET UTF8;
```

Создание базы данных с указанием IP адреса (IPv4) вместо указания имени сервера.

```
CREATE DATABASE '127:0:0:1:test'
USER wizard PASSWORD 'player' ROLE RDB$ADMIN
DEFAULT CHARACTER SET UTF8;
```

Создание базы данных с указанием IP адреса (IPv6) вместо указания имени сервера.

```
CREATE DATABASE '[::1]:test'
USER wizard PASSWORD 'player' ROLE RDB$ADMIN
DEFAULT CHARACTER SET UTF8;
```

Пример 5.4. Создание многофайловой базы данных

Создание базы данных в 3 диалекте с набором символов по умолчанию UTF8. Первичный файл будет содержать 10000 страниц с размером страницы 8192. Как только в процессе работы с базой данных первичный файл будет заполнен, СУБД будет помещать новые данные во вторичный файл `test.fdb2`. Аналогичные действия будут происходить и со вторым вторичным файлом. Размер последнего файла будет увеличиваться до тех пор, пока это позволяет используемая операционная система или пока не будет исчерпана память на внешнем носителе.

```
SET SQL DIALECT 3;
CREATE DATABASE 'baseserver:D:\test.fdb'
USER wizard PASSWORD 'player' ROLE 'RDB$ADMIN'
PAGE_SIZE = 8192
DEFAULT CHARACTER SET UTF8
FILE 'D:\test.fdb2'
STARTING AT PAGE 10001
FILE 'D:\test.fdb3'
STARTING AT PAGE 20001;
```

Пример 5.5. Создание многофайловой базы данных 2

Создание базы данных в 3 диалекте с набором символов по умолчанию UTF8. Первичный файл будет содержать 10000 страниц с размером страницы 8192. Как только в процессе работы с базой данных первичный файл будет заполнен, СУБД будет помещать новые данные во вторичный файл `test.fdb2`. Аналогичные действия будут происходить и со вторым вторичным файлом.

```
SET SQL DIALECT 3;
CREATE DATABASE 'baseserver:D:\test.fdb'
USER wizard PASSWORD 'player' ROLE 'RDB$ADMIN'
PAGE_SIZE = 8192
LENGTH 10000 PAGES
DEFAULT CHARACTER SET UTF8
FILE 'D:\test.fdb2'
FILE 'D:\test.fdb3'
STARTING AT PAGE 20001;
```

См. также: ALTER DATABASE, DROP DATABASE.

ALTER DATABASE

Назначение: Изменение структуры файлов базы данных, переключение её в состояние "безопасное для копирования" или изменение некоторых свойств базы данных.

Доступно в: DSQL, ESQL.

Синтаксис:

```
ALTER {DATABASE | SCHEMA}
    {<add_sec_clause> [<add_sec_clausee> ...]}
  | {ADD DIFFERENCE FILE 'diff_file' | DROP DIFFERENCE FILE}
  | {{BEGIN | END} BACKUP}
  | {SET DEFAULT CHARACTER SET charset}
  | {SET LINGER TO seconds | DROP LINGER}
  | {ENCRYPT WITH plugin_name | DECRYPT};

<add_sec_clause> ::= ADD FILE <sec_file>

<sec_file> ::= 'filepath'
  [STARTING [AT [PAGE]] pagenum]
  [LENGTH [=] num [PAGE[S]]
```

Параметры оператора ALTER DATABASE

add_sec_clause
Инструкция для добавления вторичного файла базы данных.

sec_file
Спецификация вторичного файла.

filepath
Полный путь и имя дельта файла или вторичного файла базы данных.

pagenum
Номер страницы, с которой начинается вторичный файл базы данных.

num
Максимальный размер вторичного файла в страницах.

diff_file
Путь и имя дельта файла.

charset
Новый набор символов по умолчанию для базы данных.

seconds
Задержка в секундах.

plugin_name
Имя плагина шифрования.

Оператор ALTER DATABASE изменяет структуру файлов базы данных или переключает её в состояние "безопасное для копирования".

Добавление вторичного файла

Предложение ADD FILE добавляет к базе данных вторичный файл. Для вторичного файла необходимо указывать полный путь к файлу и имя вторичного файла. Описание вторичного файла аналогично тому, что описано в операторе CREATE DATABASE.

Изменение пути и имени дельта файла

Предложение ADD DIFFERENCE FILE задаёт путь и имя дельта файла, в который будут записываться изменения, внесённые в базу данных после перевода её в режим "безопасного копирования" ("copy-safe"). Этот оператор в действительности не добавляет файла. Он просто переопределяет умалчиваемые имя и путь файла дельты. Для изменения существующих установок необходимо сначала удалить ранее указанное описание файла дельты с помощью оператора DROP DIFFERENCE FILE, а затем задать новое описание файла дельты. Если не переопределять путь и имя файла дельты, то он будет иметь тот же путь и имя, что и БД, но с расширением `.delta`.

> **Примечание**
>
> При задании относительного пути или только имени файла дельты он будет создаваться в текущем каталоге сервера. Для операционных систем Windows это системный каталог.

Предложение DROP DIFFERENCE FILE удаляет описание (путь и имя) файла дельты, заданное ранее командой ADD DIFFERENCE FILE. На самом деле при выполнении этого оператора файл не удаляется. Он удаляет путь и имя файла дельты и при последующем переводе БД в режим "безопасного копирования" будут использованы значения по умолчанию (т.е. тот же путь и имя что и у файла БД, но с расширением `.delta`).

Перевод базы данных в режим "безопасного копирования"

Предложение BEGIN BACKUP предназначено для перевода базы данных в режим "безопасного копирования" ("copy-safe"). Этот оператор "замораживает" основной файл базы данных, что позволяет безопасно делать резервную копию средствами файловой системы, даже если пользователи подключены и выполняют операции с данными. При этом все изменения,

вносимые пользователями в базу данных, будут записаны в отдельный файл, так называемый дельта файл (*delta file*).

> **Примечание**
>
> Оператор BEGIN BACKUP, не смотря на синтаксис, не начинает резервное копирование базы данных, а лишь создаёт условия для его осуществления.

Предложение END BACKUP предназначено для перевода базы данных из режима "безопасного копирования "copy-safe" в режим нормального функционирования. Этот оператор объединяет файл дельты с основным файлом базы данных и восстанавливает нормальное состояние работы, таким образом, закрывая возможность создания безопасных резервных копий средствами файловой системы. (При этом безопасное резервное копирование с помощью утилиты GBAK остаётся доступным).

Изменение набора символов по умолчанию

Предложение SET DEFAULT CHARACTER SET изменяет набор символов по умолчанию для базы данных. Это изменение не затрагивает существующие данные. Новый набор символов по умолчанию будет использоваться только в последующих DDL командах, кроме того для них будет использоваться сортировка по умолчанию для нового набора символов.

LINGER

Предложение SET LINGER позволяет установить задержку закрытия базы данных. Этот механизм позволяет ядру SuperServer, сохранять базу данных в открытом состоянии в течение некоторого времени после того как последние соединение закрыто, т.е. иметь механизм задержки закрытия базы данных. Это может помочь улучшить производительность и уменьшить издержки в случаях, когда база данных часто открывается и закрывается, сохраняя при этом ресурсы «разогретыми» до следующего открытия.

> **Подсказка**
>
> Такой режим может быть полезен для Web приложений, в которых коннект к базе обычно "живёт" очень короткое время.

Предложение DROP LINGER удаляет задержку и возвращает базу данных к нормальному состоянию (без задержки). Эта команда эквивалентна установки задержки в 0.

> **Подсказка**
>
> Удаление LINGER не самое лучшее решение для временной необходимости его отключения для некоторых разовых действий, требующих принудительного завершения работы сервера. Утилита gfix теперь имеет переключатель `-NoLinger`, который сразу закроет указанную базу данных, после того как последнего соединения не стало, независимо от установок LINGER в базе данных. Установка LINGER будет сохранена и нормально отработает в следующий раз.
>
> Кроме того, одноразовое переопределение доступно также через сервисы API, с использованием тега isc_spb_prp_nolinger, например (в такой строке):
>
> ```
> fbsvcmgr host:service_mgr user sysdba password xxx
> action_properties dbname employee prp_nolinger
> ```

Шифрование базы данных

Предложение ENCRYPT WITH шифрует базу данных с помощью указанного плагина шифрования. Шифрование начинается сразу после этого оператора и будет выполняться в фоновом режиме. Нормальная работа с базами данных не нарушается во время шифрования.

> **Примечание**
>
> Процесс шифрования может быть проконтролирован с помощью поля MON$CRYPT_PAGE в псевдо-таблице MON$DATABASE или просмотрен на странице заголовка базы данных с помощью **gstat -e**.
>
> **gstat –h** также будет предоставлять ограниченную информацию о состоянии шифрования.

Предложение DECRYPT дешифрует базу данных.

Кто может выполнить ALTER DATABASE?

Выполнить оператор ALTER DATABASE могут:

- SYSDBA;
- Владелец базы данных;
- Любой пользователь с привилегией на изменение базы данных (GRANT ALTER DATABASE);
- Любой пользователь, подключенный с ролью RDB$ADMIN (роль должна быть назначена пользователю);
- Пользователь операционной системы root (Linux);
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.6. Добавление вторичного файла в базу данных

Как только в предыдущем первичном или вторичных файлах будет заполнено 30000 страниц, СУБД будет помещать данные во вторичный файл `test4.fdb`.

```
ALTER DATABASE
ADD FILE 'D:\test.fdb4'
STARTING PAGE 30001;
```

Пример 5.7. Установка пути и имени файла дельты

```
ALTER DATABASE
ADD DIFFERENCE FILE 'D:\test.diff';
```

Пример 5.8. Удаление описание файла дельты

```
ALTER DATABASE
DROP DIFFERENCE FILE;
```

Пример 5.9. Перевод базы данных в режим «безопасного копирования»

```
ALTER DATABASE
BEGIN BACKUP;
```

Пример 5.10. Возвращение базы данных в режим нормального функционирования из режима "безопасного копирования"

```
ALTER DATABASE
END BACKUP;
```

Пример 5.11. Изменение набора символов по умолчанию для базы данных

```
ALTER DATABASE SET DEFAULT CHARACTER SET WIN1251;
```

Пример 5.12. Установка задержки в 30 секунд

```
ALTER DATABASE SET LINGER 30;
```

Пример 5.13. Удаление задержки

```
ALTER DATABASE DROP LINGER;
```

или

```
ALTER DATABASE SET LINGER 0;
```

Пример 5.14. Шифрование базы данных

```
ALTER DATABASE ENCRYPT WITH DbCrypt;
```

Пример 5.15. Дешифрование базы данных

```
ALTER DATABASE DECRYPT;
```

См. также: CREATE DATABASE, DROP DATABASE.

DROP DATABASE

Назначение: Оператор DROP DATABASE удаляет текущую базу данных.

Доступно в: DSQL, ESQL.

Синтаксис:

```
```

```
DROP DATABASE;
```

Оператор DROP DATABASE удаляет текущую базу данных. Перед удалением базы данных, к ней необходимо присоединиться. Оператор удаляет первичный, все вторичные файлы и все файлы теневых копий.

Кто может удалить базу данных?

Базу данных могут удалить:

- SYSDBA;
- Владелец базы данных;
- Любой пользователь, подключенный с ролью RDB$ADMIN (роль должна быть назначена пользователю);
- Любой пользователь с привилегией на удаление базы данных (GRANT DROP DATABASE);
- Пользователь операционной системы root (Linux);
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.16. Удаление базы данных

Удаление базы данных, к которой подключен клиент.

```
DROP DATABASE;
```

См. также: CREATE DATABASE, ALTER DATABASE.

SHADOW

Теневая копия (shadow — дословно тень) является точной страничной копией базы данных. После создания теневой копии все изменения, сделанные в базе данных, сразу же отражаются и в теневой копии. Если по каким либо причинам первичный файл базы данных станет недоступным, то СУБД переключится на теневую копию.

В данном разделе рассматриваются вопросы создания и удаления теневых копий.

> **Примечание**
>
> Это относится только к текущим операциям с базой данных, но не к новым подключениям. В случае поломки исходной базы данных администратор БД должен восстановить изначальные файлы базы данных, в том числе и с помощью файлов теневых копий. Только после этого будет возможно подключение новых клиентов.

CREATE SHADOW

Назначение: Создание теневой копии.

Синтаксис:

```
CREATE SHADOW sh_num [AUTO | MANUAL] [CONDITIONAL]
  'filepath' [LENGTH [=] num [PAGE[S]]]
  [<secondary_file>];

<secondary_file> ::=
  FILE 'filepath'
  LENGTH [=] num [PAGE[S]] | STARTING [AT [PAGE]] pagenum
```

Параметры оператора CREATE SHADOW

sh_num
: Номер теневой копии – положительное число, идентифицирующее набор файлов теневой копии.

filepath
: Имя файла и путь к нему в соответствии с требованиями ОС.

num
: Максимальный размер теневой копии в страницах.

secondary_file
: Спецификация вторичного файла.

page_num
: Номер страницы, с которой должен начинаться вторичный файл копии.

Оператор CREATE SHADOW создаёт новую теневую копию. Теневая копия начинает дублировать базу данных сразу в момент создания этой копии.

Теневые копии, как и база данных, могут состоять из нескольких файлов. Количество и размер файлов теневых копий не связано с количеством и размером файлов базы данных.

Для файлов теневой копии размер страницы устанавливается равным размеру страницы базы данных и не может быть изменён.

Если по каким либо причинам файл базы данных становится недоступным, то система преобразует тень в копию базы данных и переключается на неё. Теневая копия становится недоступной. Что будет дальше зависит от выбранного режима.

Режимы AUTO и MANUAL

Когда теневая копия преобразуется в базу данных она становится недоступной. Теневая копия может также стать недоступной если будет удалён её файл, или закончится место на диске, где она расположена, или если этот диск повреждён.

- Если выбран режим AUTO (значение по умолчанию), то в случае, когда теневая копия становится недоступной, автоматически прекращается использование этой копии и из базы данных удаляются все ссылки на нее. Работа с базой данных продолжается обычным образом без осуществлёния копирования в данную теневую копию.

Если указано ключевое слово CONDITIONAL, то система будет пытаться создать новую теневую копию. чтобы заменить потерянную. Это не всегда возможно, тогда вам потребуется создать новую тень вручную.

- Если выбран режим MANUAL, то в случае, когда теневая копия становится недоступной, все попытки соединения с базой данных и обращения к ней будут вызывать сообщение об ошибке до тех пор, пока теневая копия не станет доступной или пока не будет удалена администратором БД с помощью оператора DROP SHADOW.

Необязательные параметры CREATE SHADOW

Необязательное предложение LENGTH задаёт максимальный размер первичного или вторичного файла теневой копии в страницах. Для единственного или последнего файла теневой копии значение LENGTH никак не влияет на его размер. Файл будет автоматически увеличивать свой размер по мере необходимости.

Предложение STARTING AT задаёт номер страницы теневой копии, с которой должен начинаться следующий файл теневой копии. Когда предыдущий файл будет полностью заполнен данными в соответствии с заданным номером страницы, система начнёт помещать вновь добавляемые данные в следующий файл теневой копии.

Кто может создать теневую копию?

Создать теневую копию могут:

- SYSDBA;
- Владелец базы данных;
- Любой пользователь с привилегией на изменение базы данных (GRANT ALTER DATABASE);
- Любой пользователь, подключенный с ролью RDB$ADMIN (роль должна быть назначена пользователю);
- Пользователь операционной системы root (Linux);
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.17. Создание теневую копию базы данных с номером 1

```
CREATE SHADOW 1 'g:\data\test.shd';
```

Пример 5.18. Создание многофайловой теневой копии

```
CREATE SHADOW 2 'g:\data\test.sh1'
LENGTH 8000 PAGES
FILE 'g:\data\test.sh2';
```

См. также: CREATE DATABASE, ALTER DATABASE, DROP SHADOW.

DROP SHADOW

Назначение: Удаление теневой копии.

Доступно в: DSQL, ESQL.

Синтаксис:

```
DROP SHADOW sh_num;
```

Параметры оператора DROP SHADOW

sh_num
: Номер теневой копии – положительное число, идентифицирующее набор файлов теневой копии.

Оператор DROP SHADOW удаляет указанную теневую копию из базы данных, с которой установлено текущее соединение. При удалении теневой копии удаляются все связанные с ней файлы, и прекращается процесс дублирования данных в эту теневой копии.

Кто может удалить теневую копию?

Удалить теневую копию могут:

- SYSDBA;
- Владелец базы данных;
- Любой пользователь с привилегией на изменение базы данных (GRANT ALTER DATABASE);
- Любой пользователь, подключенный с ролью RDB$ADMIN (роль должна быть назначена пользователю);
- Пользователь операционной системы root (Linux);
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.19. Удаление теневой копии с номером 1

```
DROP SHADOW 1;
```

См. также: CREATE SHADOW.

DOMAIN

Домен (Domain) — один из объектов реляционной базы данных, при создании которого можно задать некоторые характеристики, а затем использовать ссылку на домен при определении столбцов таблиц, объявлении локальных переменных, входных и выходных аргументов в модулях PSQL.

В данном разделе рассматриваются синтаксис операторов создания, модификации и удаления доменов. Подробное описание доменов и их использования можно прочесть в главе Пользовательские типы данных — домены.

CREATE DOMAIN

Назначение: Создание нового домена.

Доступно в: DSQL, ESQL.

Синтаксис:

```
CREATE DOMAIN name [AS] <datatype>
  [DEFAULT {literal | NULL | <context_var>}]
  [NOT NULL] [CHECK (<dom_condition>)]
  [COLLATE collation];

<datatype> ::=
    {SMALLINT | INTEGER | BIGINT} [<array_dim>]
  | BOOLEAN [<array_dim>]
  | {FLOAT | DOUBLE PRECISION} [<array_dim>]
  | {DATE | TIME | TIMESTAMP} [<array_dim>]
  | {DECIMAL | NUMERIC} [(precision [, scale])] [<array_dim>]
  | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(size)]
    [<array_dim>] [CHARACTER SET charset]
  | {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR} [VARYING]
    [(size)] [<array_dim>]
  | BLOB [SUB_TYPE {subtype_num | subtype_name}]
    [SEGMENT SIZE seglen] [CHARACTER SET charset]
  | BLOB [(seglen [, subtype_num])]

<array_dim> ::= [x:y [,x:y ...]]

<dom_condition> ::= {
    <val> <operator> <val>
  | <val> [NOT] BETWEEN <val> AND <val>
  | <val> [NOT] IN (<val> [, <val> ...] | <select_list>)
  | <val> IS [NOT] NULL
  | <val> IS [NOT] DISTINCT <val>
  | <val> [NOT] CONTAINING <val>
  | <val> [NOT] STARTING [WITH] <val>
  | <val> [NOT] LIKE <val> [ESCAPE <val>]
  | <val> [NOT] SIMILAR TO <val> [ESCAPE <val>]
  | <val> <operator> {ALL | SOME | ANY} (<select_list>)
  | [NOT] EXISTS (<select_expr>)
  | [NOT] SINGULAR (<select_expr>)
  | (<dom_condition>)
  | NOT <dom_condition>
  | <dom_condition> OR <dom_condition>
  | <dom_condition> AND <dom_condition>
}

<operator> ::= {= | < | > | <= | >= | !< | !> | <> | !=}

<val> ::= {
    VALUE
  | literal
  | <context_var>
  | <expression>
  | NULL
  | NEXT VALUE FOR genname
```

```
  | GEN_ID(genname, <val>)
  | CAST(<val> AS <datatype>)
  | (<select_one>)
  | func(<val> [, <val> ...])
}
```

Параметры оператора CREATE DOMAIN

name
Имя домена. Может содержать до 31 байта.

datatype
Тип данных SQL.

literal
Литерал.

context_var
Любая контекстная переменная, тип которой совместим с типом данных домена.

dom_condition
Условие домена.

collation
Порядок сортировки.

array_dim
Размерность массива.

x
Начальный номер элемента в массиве, положительное целое число.

y
Последний номер элемента в массиве, положительное целое число.

precision
Точность. От 1 до 18.

scale
Масштаб. От 0 до 18, должно быть меньше или равно *precision*.

size
Максимальный размер строки в символах.

charset
Набор символов.

subtype_num
Номер подтипа BLOB.

subtype_name
Мнемоника подтипа BLOB.

seglen
Размер сегмента, не может превышать 65535.

val
Значение.

select_one
Оператор SELECT выбирающий один столбец и возвращающий только одну строку.

select_list
Оператор SELECT выбирающий один столбец и возвращающий ноль и более строк.

select_expr
Оператор SELECT выбирающий несколько столбцов и возвращающий ноль и более строк.

experssion
Выражение.

genname
Имя последовательности (генератора).

func
Скалярная функция.

Оператор CREATE DOMAIN создаёт новый домен.

В качестве типа домена можно использовать любой тип данных SQL. Для любого типа данных кроме BLOB можно указать размерность массива, если домен должен быть массивом. Размерность массива указывается в квадратных скобках. Чтобы не перепутать их с символами, означающими необязательные элементы, они выделены жирным шрифтом. При указании размерности массива указываются два числа через двоеточие. Первое число означает начальный номер элемента массива, второе – конечный. Если указано только одно число, то оно означает последний номер в элементе массива, а первым номером считается 1. Если у массива два и более измерения, то они перечисляются через запятую.

Для типов CHAR, VARCHAR и BLOB с подтипом text можно указать набор символов в предложении CHARACTER SET. Если набор символов не указан, то по умолчанию принимается тот набор символов, который был указан при создании базы данных.

> **Предупреждение**
>
> Если же при создании базы данных не был указан набор символов, то при создании домена по умолчанию принимается набор символов NONE. В этом случае данные хранятся и извлекаются, так как они были поданы. В столбец, основанный на таком домене, можно загружать данные в любой кодировке, но невозможно загрузить эти данные в столбец с другой кодировкой. Транслитерация не выполняется между исходными и конечными кодировками, что может приводить к ошибкам.

Необязательное предложение DEFAULT позволяет указать значение по умолчанию для домена. Это значение будет помещено в столбец таблицы, который ссылает на данный домен, при выполнении оператора INSERT, если значение не будет указано для этого столбца. Локальные переменные и аргументы PSQL модулей, которые ссылаются на этот домен, будут инициализированы значением по умолчанию. В качестве значения по умолчанию может быть литерал совместимый по типу, неизвестное значение NULL и контекстная переменная, тип которой совместим с типом домена.

Предложение NOT NULL запрещает столбцам и переменным, основанным на домене, присваивать значение NULL.

Необязательное предложение CHECK задаёт ограничение домена. Ограничение домена задаёт условия, которому должны удовлетворять значения столбцов таблицы или переменных, которые ссылаются на данный домен. Условие должно быть помещено в круглые скобки.

Условие — это логическое выражение, называемое также предикат, которое может возвращать значения TRUE (истина), FALSE (ложь) и UNKNOWN (неизвестно). Условие считается выполненным, если предикат возвращает значение TRUE или UNKNOWN (эквивалент NULL). Если предикат возвращает FALSE, то значение не будет принято.

Ключевое слово VALUE в ограничении домена является заменителем столбца таблицы, который основан на данном домене, или переменной PSQL модуля. Оно содержит значение, присваиваемое переменной или столбцу таблицы. Ключевое слово VALUE может быть использовано в любом месте ограничения CHECK, но обычно его используют в левой части условия.

Необязательное предложение COLLATE позволяет задать порядок сортировки, если домен основан на одном из строковых типов данных (за исключением BLOB). Если порядок сортировки не указан, то по умолчанию принимается порядок сортировки умалчиваемый для указанного набора сортировки при создании домена.

Кто может создать домен?

Создать новый домен могут:

- SYSDBA;
- Владелец базы данных;
- Любой пользователь с привилегией на создание доменов (GRANT CREATE DOMAIN);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Пользователь, создавший домен, становится его владельцем.

Примеры

Пример 5.20. Создание домена, который может принимать значения больше 1000.

```
CREATE DOMAIN CUSTNO AS
INTEGER DEFAULT 10000
CHECK (VALUE > 1000);
```

Пример 5.21. Создание домена, который может принимать значения 'Да' и 'Нет'.

```
CREATE DOMAIN D_BOOLEAN AS
CHAR(3) CHECK (VALUE IN ('Да', 'Нет'));
```

Пример 5.22. Создание домена с набором символов UTF8 и порядком сортировки UNICODE_CI_AI.

```
CREATE DOMAIN FIRSTNAME AS
VARCHAR(30) CHARACTER SET UTF8
```

```
COLLATE UNICODE_CI_AI;
```

Пример 5.23. Создание домена со значением по умолчанию.

```
CREATE DOMAIN D_DATE AS
DATE DEFAULT CURRENT_DATE
NOT NULL;
```

Пример 5.24. Создание домена, определённого как массив из 2 элементов.

Создание домена, определённого как массив из 2 элементов типа NUMERIC(18, 3), нумерация элементов начинается с 1.

```
CREATE DOMAIN D_POINT AS
NUMERIC(18, 3) [2];
```

> **Примечание**
>
> Вы можете использовать домены определённые как массив только для определения столбцов таблиц. Вы не можете использовать такие домены для определения локальных переменных и аргументов PSQL модулей.

См. также: ALTER DOMAIN, DROP DOMAIN.

ALTER DOMAIN

Назначение: Изменение текущих характеристик домена или его переименование.

Доступно в: DSQL, ESQL.

Синтаксис:

```
ALTER DOMAIN name
  [{TO new_name}]
  [{SET DEFAULT {literal | NULL | <context_var>} |
    DROP DEFAULT}]
  [{SET | DROP} NOT NULL]
  [{ADD [CONSTRAINT] CHECK (<dom_condition>) |
    DROP CONSTRAINT}]
  [{TYPE <datatype>}];


<datatype> ::=
    {SMALLINT | INTEGER | BIGINT} [<array_dim>]
  | BOOLEAN [<array_dim>]
  | {FLOAT | DOUBLE PRECISION} [<array_dim>]
  | {DATE | TIME | TIMESTAMP} [<array_dim>]
  | {DECIMAL | NUMERIC} [(precision [, scale])] [<array_dim>]
  | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(size)]
    [<array_dim>] [CHARACTER SET charset]
  | {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR} [VARYING]
```

```
    [(size)] [<array_dim>]
  | BLOB [SUB_TYPE {subtype_num | subtype_name}]
    [SEGMENT SIZE seglen] [CHARACTER SET charset]
  | BLOB [(seglen [, subtype_num])]

<array_dim> ::= [x:y [,x:y ...]]

<dom_condition> ::= {
    <val> <operator> <val>
  | <val> [NOT] BETWEEN <val> AND <val>
  | <val> [NOT] IN (<val> [, <val> ...] | <select_list>)
  | <val> IS [NOT] NULL
  | <val> IS [NOT] DISTINCT <val>
  | <val> [NOT] CONTAINING <val>
  | <val> [NOT] STARTING [WITH] <val>
  | <val> [NOT] LIKE <val> [ESCAPE <val>]
  | <val> [NOT] SIMILAR TO <val> [ESCAPE <val>]
  | <val> <operator> {ALL | SOME | ANY} (<select_list>)
  | [NOT] EXISTS (<select_expr>)
  | [NOT] SINGULAR (<select_expr>)
  | (<dom_condition>)
  | NOT <dom_condition>
  | <dom_condition> OR <dom_condition>
  | <dom_condition> AND <dom_condition>
}

<operator> ::= {= | < | > | <= | >= | !< | !> | <> | !=}

<val> ::= {
    VALUE
  | literal
  | <context_var>
  | <expression>
  | NULL
  | NEXT VALUE FOR genname
  | GEN_ID(genname, <val>)
  | CAST(<val> AS <datatype>)
  | (<select_one>)
  | func(<val> [, <val> ...])
}
```

Параметры оператора ALTER DOMAIN

name
Имя домена.

new_name
Новое имя домена. Может содержать до 31 байта.

datatype
Тип данных SQL.

literal
Литерал.

context_var
Любая контекстная переменная, тип которой совместим с типом данных домена.

dom_condition
Условие домена.

collation
Порядок сортировки.

array_dim
Размерность массива.

x
Начальный номер элемента в массиве, положительное целое число.

y
Последний номер элемента в массиве, положительное целое число.

precision
Точность. От 1 до 18.

scale
Масштаб. От 0 до 18, должно быть меньше или равно *precision*.

size
Максимальный размер строки в символах.

charset
Набор символов.

subtype_num
Номер подтипа BLOB.

subtype_name
Мнемоника подтипа BLOB.

seglen
Размер сегмента, не может превышать 65535.

val
Значение.

select_one
Оператор SELECT выбирающий один столбец и возвращающий только одну строку.

select_list
Оператор SELECT выбирающий один столбец и возвращающий ноль и более строк.

select_expr
Оператор SELECT выбирающий несколько столбцов и возвращающий ноль и более строк.

experssion
Выражение.

genname
Имя последовательности (генератора).

func
Скалярная функция.

Оператор ALTER DOMAIN изменяет текущие характеристики домена, в том числе и его имя. В одном операторе ALTER DOMAIN можно выполнить любое количество изменений домена.

Предложение TO позволяет переименовать домен. Имя домена можно изменить, если не существует зависимостей от этого домена, т.е. столбцов таблиц, локальных переменных и аргументов процедур, ссылающихся на данный домен.

Предложение SET DEFAULT позволяет установить новое значение по умолчанию. Если домен уже содержал значение по умолчанию, то установка нового значения по умолчанию не требует предварительного удаления старого.

Предложение DROP DEFAULT удаляет ранее установленное для домена значение по умолчанию. В этом случае значением по умолчанию становится значение NULL.

Предложение ADD [CONSTRAINT] CHECK добавляет условие ограничения домена. Если домен уже содержал ограничение CHECK, то его предварительно необходимо удалить с помощью предложения DROP [CONSTRAINT] CHECK.

Предложение TYPE позволяет изменить тип домена на другой допустимый тип. Не допустимы любые изменения типа, которые могут привести к потере данных. Например, количество символов в новом типе для домена не может быть меньше, чем было установлено ранее.

Предложение SET NOT NULL устанавливает ограничение NOT NULL для домена. В этом случае для переменных и столбцах базирующихся на домене значение NULL не допускается. Предложение DROP NOT NULL удаляет ограничение NOT NULL для домена.

> **Примечание**
>
> Успешная установка ограничения NOT NULL для домена происходит только после полной проверки данных таблиц, столбцы которых базируются на домене. Это может занять довольно длительное время.

> **Предупреждение**
>
> При изменении описания домена, существующий PSQL код, может стать некорректным. Информация о том, как это обнаружить, находится в приложении Поле RDB$VALID_BLR.

Что не может изменить ALTER DOMAIN

- Если домен был объявлен как массив, то изменить ни его тип, ни размерность нельзя. Также нет возможности изменить любой другой тип на тип массив.
- Не существует способа изменить сортировку по умолчанию. В этом случае необходимо удалить домен и пересоздать его с новыми атрибутами.

Кто может изменить домен?

Изменить домен могут:

- SYSDBA;
- Владелец базы данных;
- Владелец домена;
- Любой пользователь с привилегией на изменение любого домена (GRANT ALTER ANY DOMAIN);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;

- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.25. Изменение значения по умолчанию для домена.

```
ALTER DOMAIN CUSTNO
INTEGER DEFAULT 2000;
```

Пример 5.26. Переименование домена.

```
ALTER DOMAIN D_BOOLEAN TO D_BOOL;
```

Пример 5.27. Удаление значения по умолчанию и добавления ограничения для домена.

```
ALTER DOMAIN D_DATE
DROP DEFAULT
ADD CONSTRAINT CHECK (VALUE >= date '01.01.2000');
```

Пример 5.28. Изменение ограничения домена.

```
ALTER DOMAIN D_DATE
DROP CONSTRAINT;

ALTER DOMAIN D_DATE
ADD CONSTRAINT CHECK
(VALUE BETWEEN date '01.01.1900' AND date '31.12.2100');
```

Пример 5.29. Изменение типа домена.

```
ALTER DOMAIN FIRSTNAME
TYPE VARCHAR(50) CHARACTER SET UTF8;
```

Пример 5.30. Добавление ограничения NOT NULL для домена.

```
ALTER DOMAIN FIRSTNAME SET NOT NULL;
```

См. также: CREATE DOMAIN, DROP DOMAIN.

DROP DOMAIN

Назначение: Удаление существующего домена.

Доступно в: DSQL, ESQL.

Синтаксис:

```
DROP DOMAIN domain_name;
```

Параметры оператора DROP DOMAIN

domain_name
Имя домена.

Оператор DROP DOMAIN удаляет домен, существующий в базе данных. Невозможно удалить домен, на который ссылаются столбцы таблиц базы данных или если он был задействован в одном из PSQL модулей. Чтобы удалить такой домен необходимо удалить из таблиц все столбцы, ссылающиеся на домен и удалить все ссылки на домен из PSQL модулей.

Кто может удалить домен?

Удалить домен могут:

- SYSDBA;
- Владелец базы данных;
- Владелец домена;
- Любой пользователь с привилегией на удаление любого домена (GRANT DROP ANY DOMAIN);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.31. Удаление домена

```
DROP DOMAIN COUNTRYNAME;
```

См. также: CREATE DOMAIN, ALTER DOMAIN.

TABLE

Firebird — это реляционная СУБД. Данные в таких базах хранятся в таблицах. Таблица – это плоская двухмерная структура, содержащая произвольное количество строк (row). Строки таблицы часто называют записями (record). Все строки таблицы имеют одинаковую структуру и состоят из столбцов (column). Столбцы таблицы часто называют полями (fields). Таблица должна иметь хотя бы один столбец. С каждым столбцом связан определённый тип данных SQL.

В данном разделе рассматриваются вопросы создания, модификации и удаления таблиц базы данных.

CREATE TABLE

Назначение: Создание новой таблицы.

Доступно в: DSQL, ESQL.

Синтаксис:

```
CREATE [GLOBAL TEMPORARY] TABLE tablename
  [EXTERNAL [FILE] '<filespec>']
  (<col_def> [, <col_def> | <tconstraint> ...])
  [ON COMMIT {DELETE | PRESERVE} ROWS];

<col_def> ::= colname
  {
    { <datatype> | domainname }
    [DEFAULT {literal | NULL | <context_var>}]
    [NOT NULL]
    [<col_constraint>]
    [COLLATE collation]
  } | <col_exp_def> | <col_indentity_def>

<col_exp_def> ::=
  [<datatype>] {COMPUTED [BY] | GENERATED ALWAYS AS} (<col_expr>)

<col_indentity_def> ::= [<datatype>] GENERATED BY DEFAULT AS IDENTITY
  [(START WITH startvalue)] [<col_constraint>]

<datatype> ::= {
    {SMALLINT | INTEGER | BIGINT} [<array_dim>]
  | BOOLEAN [<array_dim>]
  | {FLOAT | DOUBLE PRECISION} [<array_dim>]
  | {DATE | TIME | TIMESTAMP} [<array_dim>]
  | {DECIMAL | NUMERIC} [(precision [, scale])] [<array_dim>]
  | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(size)]
      [<array_dim>] [CHARACTER SET charset]
  | {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR} [VARYING] [(size)]
      [<array_dim>]
  | BLOB [SUB_TYPE {subtype_num | subtype_name}] [SEGMENT SIZE seglen]
      [CHARACTER SET charset]
  | BLOB [(seglen [, subtype_num])]
}

<array_dim> ::= [x:y [,x:y ...]]

<col_constraint> ::= [CONSTRAINT constr_name]
{   UNIQUE [<using_index>]
  | PRIMARY KEY [<using_index>]
  | REFERENCES other_table [(other_col)] [<using_index>]
      [ON DELETE { NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
      [ON UPDATE { NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
  | CHECK (<search_condition>)
}
```

```
<tconstraint> ::= [CONSTRAINT constr_name]
{   UNIQUE (colname [, colname ...]) [<using_index>]
  | PRIMARY KEY (colname [, colname ...]) [<using_index>]
  | FOREIGN KEY (colname [, colname ...])
    REFERENCES other_table [(other_col [, other_col ...])] [<using_index>]
      [ON DELETE { NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
      [ON UPDATE { NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
  | CHECK (<search_condition>)
}

<using_index> ::= USING [ASC[ENDING] | DESC[ENDING]] INDEX indexname

<search_condition> ::= {
    <val> <operator> <val>
  | <val> [NOT] BETWEEN <val> AND <val>
  | <val> [NOT] IN (<val> [, <val> ...] | <select_list>)
  | <val> IS [NOT] NULL
  | <val> IS [NOT] DISTINCT <val>
  | <val> [NOT] CONTAINING <val>
  | <val> [NOT] STARTING [WITH] <val>
  | <val> [NOT] LIKE <val> [ESCAPE <val>]
  | <val> [NOT] SIMILAR TO <val> [ESCAPE <val>]
  | <val> <operator> {ALL | SOME | ANY} (<select_list>)
  | [NOT] EXISTS (<select_expr>)
  | [NOT] SINGULAR (<select_expr>)
  | (<search_condition>)
  | NOT <search_condition>
  | <search_condition> OR <search_condition>
  | <search_condition> AND <search_condition>
}

<operator> ::= {= | < | > | <= | >= | !< | !> | <> | !=}

<val> ::= {
    colname [[<ind> [, <ind> ...]]]
  | literal
  | <context_var>
  | <expression>
  | NULL
  | NEXT VALUE FOR genname
  | GEN_ID(genname, <val>)
  | CAST(<val> AS <datatype>)
  | (<select_one>)
  | func(<val> [, <val> ...])
}
```

Параметры оператора CREATE TABLE

tablename
Имя таблицы, может содержать до 31 байта.

filespec
Спецификация файла (только для внешних таблиц).

col_def
Определение столбца.

colname
Имя столбца таблицы, может содержать до 31 байта.

datatype
Тип данных SQL.

col_identity_def
Определение столбца идентификации.

startvalue
Начальное значение столбца идентификации.

col_exp_def
Определение вычисляемого столбца.

col_expr
Выражение для вычисляемого столбца.

domain_name
Имя домена.

col_constraint
Ограничение столбца.

tconstraint
Ограничение таблицы.

constr_name
Имя ограничения, может содержать до 31 байта.

other_table
Имя таблицы, на которую ссылается внешний ключ.

other_col
Столбец таблицы, на которую ссылается внешний ключ.

literal
Литерал.

context_var
Любая контекстная переменная, тип которой совместим с типом данных столбца.

search_condition
Условие проверки ограничения.

collation
Порядок сортировки.

array_dim
Размерность массива.

x
Начальный номер элемента в массиве, положительное целое число.

y
Последний номер элемента в массиве, положительное целое число.

precision
Точность. От 1 до 18.

scale
Масштаб. От 0 до 18, должно быть меньше или равно *precision*.

size
Максимальный размер строки в символах.

charset
Набор символов.

subtype_num
Номер подтипа BLOB.

subtype_name
Мнемоника подтипа BLOB.

seglen
Размер сегмента, не может превышать 65535.

val
Значение.

select_one
Оператор SELECT выбирающий один столбец и возвращающий только одну строку.

select_list
Оператор SELECT выбирающий один столбец и возвращающий ноль и более строк.

select_expr
Оператор SELECT выбирающий несколько столбцов и возвращающий ноль и более строк.

experssion
Выражение.

genname
Имя последовательности (генератора).

func
Скалярная функция.

Оператор CREATE TABLE создаёт новую таблицу. Имя таблицы должно быть уникальным среди имён всех таблиц, представлений (VIEWs) и хранимых процедур базы данных.

Таблица может содержать, по меньшей мере, один столбец и произвольное количество ограничений таблицы.

Имя столбца должно быть уникальным для создаваемой таблицы. Для столбца обязательно должен быть указан либо тип данных, либо имя домена, характеристики которого будут скопированы для столбца, либо должно быть указано, что столбец является вычисляемым.

В качестве типа столбца можно использовать любой тип данных SQL.

Символьные столбцы

Для типов CHAR, VARCHAR и BLOB с подтипом TEXT можно указать набор символов в предложении CHARACTER SET. Если набор символов не указан, то по умолчанию принимается

тот набор символов, что был указан при создании базы данных. Если же при создании базы данных не был указан набор символов, то по умолчанию принимается набор символов NONE. В этом случае данные хранятся и извлекаются, так как они были поданы. В столбец можно загружать данные в любой кодировке, но невозможно загрузить эти данные в столбец с другой кодировкой. Транслитерация между исходными и конечными кодировками не выполняется, что может приводить к ошибкам.

Необязательное предложение COLLATE позволяет задать порядок сортировки для строковых типов данных (за исключением BLOB). Если порядок сортировки не указан, то по умолчанию принимается порядок сортировки по умолчанию для указанного набора сортировки.

Ограничение NOT NULL

По умолчанию столбец может принимать значение NULL.

Необязательное предложение NOT NULL указывает, что столбцу не может быть присвоено значение NULL.

Значение по умолчанию

Необязательное предложение DEFAULT позволяет указать значение по умолчанию для столбца таблицы. Это значение будет помещено в столбец таблицы при выполнении оператора INSERT, если значение не будет указано для этого столбца. В качестве значения по умолчанию может быть литерал совместимый по типу, неизвестное значение NULL или контекстная переменная, тип которой совместим с типом столбца. Если значение по умолчанию явно не устанавливается, то подразумевается пустое значение, NULL. Использование выражений в значении по умолчанию недопустимо.

Столбцы основанные на домене

Для определения столбца, можно воспользоваться ранее описанным доменом. Если определение столбца основано на домене, оно может включать новое значение по умолчанию, дополнительные ограничения CHECK, предложение COLLATE, которые перекрывают значения указанные при определении домена. Определение такого столбца может включать дополнительные ограничения столбца, например NOT NULL, если домен его ещё не содержит.

> **Важно**
>
> Следует обратить внимание на то, что если в определении домена было указано NOT NULL, на уровне столбца невозможно определить допустимость использования в нем значения NULL. Если вы хотите чтобы на основе домена можно было определять столбцы допускающие псевдозначение NULL и не допускающее его, то хорошей практикой является создание домена допускающего NULL и указание ограничения NOT NULL у столбцов таблицы там где это необходимо.

Столбцы идентификации (автоинкремент)

Столбцы идентификации могут быть определены с помощью предложения GENERATED BY DEFAULT AS IDENTITY. Столбец идентификации представляет собой столбец, связанный с внутренним генератором последовательностей. Его значение устанавливается автоматически каждый раз, когда оно не указано в операторе INSERT. Необязательное предложение START WITH позволяет указать начальное значение отличное от нуля.

Правила:

- Тип данных столбца идентификации должен быть целым числом с нулевым масштабом. Допустимыми типами являются SMALLINT, INTEGER, BIGINT, NUMERIC(x,0) и DECIMAL(x,0);
- Идентификационный столбец не может иметь DEFAULT и COMPUTED значений.

> **Примечание**
>
> - Идентификационный столбец не может быть изменён, чтобы стать обычным столбцом. Обратное тоже верно.
> - Идентификационные столбцы неявно являются NOT NULL столбцами.
> - Уникальность не обеспечивается автоматически. Ограничения UNIQUE или PRIMARY KEY требуются для гарантии уникальности.

Вычисляемые поля

Вычисляемые поля могут быть определены с помощью предложения COMPUTED [BY] или GENERATED ALWAYS AS (согласно стандарту SQL-2003). Они эквивалентны по смыслу. Для вычисляемых полей не требуется описывать тип данных (но допустимо), СУБД вычисляет подходящий тип в результате анализа выражения. В выражении требуется указать корректную операцию для типов данных столбцов, входящих в его состав. При явном указании типа столбца для вычисляемого поля результат вычисления приводится к указанному типу, то есть, например, результат числового выражения можно вывести как строку. Вычисление выражения происходит для каждой строки выбранных данных, если в операторе выборки данных SELECT, присутствует такой столбец.

> **Подсказка**
>
> Вместо использования вычисляемого столбца в ряде случаев имеет смысл использовать обычный столбец, значение которого рассчитывается в триггерах на добавление и обновление данных. Это может снизить производительность вставки/модификации записей, но повысит производительность выборки данных.

Столбцы типа массив

Для любого типа данных кроме BLOB можно указать размерность массива, если столбец должен быть массивом. Размерность массива указывается в квадратных скобках. Чтобы не перепутать их с символами, означающими необязательные элементы, они выделены жирным шрифтом. При указании размерности массива указываются два числа через двоеточие. Первое число означает начальный номер элемента массива, второе — конечный. Если указано только одно число, то оно означает последний номер в элементе массива, а первым номером считается 1. Для многомерного массива размерности массива перечисляются через запятую.

Ограничения

Существуют четыре вида ограничений:

- первичный ключ (PRIMARY KEY);
- уникальный ключ (UNIQUE);
- внешний ключ (REFERENCES или FOREIGN KEY);
- проверочное ограничение (CHECK).

Ограничения могут быть указаны на уровне столбца (ограничения столбцов) или на уровне таблицы (табличные ограничения). Ограничения уровня таблицы необходимы, когда ключи (ограничение уникальности, первичный ключ или внешний ключ) должны быть сформированы

по нескольким столбцам, или, когда ограничение CHECK включает несколько столбцов, т.е. действует на уровне записи. Синтаксис для некоторых типов ограничений может незначительно отличаться в зависимости от того определяется ограничение на уровне столбца или на уровне таблицы.

- Ограничение на уровне столбца указывается после определения других характеристик столбца. Оно может включать только столбец указанный в этом определении.
- Ограничения на уровне таблицы указываются после определений всех столбцов. Ограничения таблицы являются более универсальным способом записи ограничений, поскольку позволяют ограничение более чем для одного столбца таблицы.
- Вы можете смешивать ограничения столбцов и ограничения таблиц в одном операторе CREATE TABLE.

Если имя ограничения не задано, то ограничению будет присвоено имя, автоматически генерированное СУБД. Для любого ограничения вы можете указать имя.

Именованные ограничения

Необязательное предложение CONSTRAINT задаёт имя ограничения.

Автоматически создаваемые индексы

Для первичного ключа (PRIMARY KEY), уникального ключа (UNIQUE) и внешнего ключа (REFERENCES) система автоматически создаёт соответствующий индекс. Если задано имя ограничения, то автоматически создаваемому индексу будет присвоено это имя (при отсутствии предложения USING).

Предложение USING

Предложение USING позволяет задать определённое пользователем имя автоматически создаваемого индекса, и опционально определить, какой это будет индекс — по возрастанию (по умолчанию) или по убыванию.

Первичный ключ (PRIMARY KEY)

Ограничение первичного ключа PRIMARY KEY строится на поле с заданным ограничением NOT NULL и требует уникальности значений столбца. Таблица может иметь только один первичный ключ.

- Первичный ключ по единственному столбцу может быть определён как на уровне столбца, так и на уровне таблицы.
- Первичный ключ по нескольким столбцам может быть определён только на уровне таблицы.

Ограничение уникальности (UNIQUE)

Ограничение уникального ключа UNIQUE задаёт для значений столбца требование уникальности содержимого. Таблица может содержать любое количество уникальных ключей.

Как и первичный ключ, ограничение уникальности может быть определено на нескольких столбцах. В этом случае вы должны определять его как ограничение уровня таблицы.

В отличие от первичного ключа, в таких полях допускаются значения NULL для любого количества строк. Таким образом, можно определить ограничение уникальности для столбцов не имеющих ограничения NOT NULL.

NULL в уникальных ключах

Для уникальных ключей, содержащих несколько столбцов, логика немного сложнее:

- Во всех столбцах, входящих в уникальный ключ, нет ограничения NOT NULL;
- Разрешено хранение значения NULL в столбцах, входящих в уникальный ключ, в нескольких строках;
- Разрешены строки, имеющие в одном из столбцов уникального ключа значение NULL, а остальные столбцы заполнены значениями и эти значения различны хотя бы в одном из них;
- Запрещены строки, имеющие в одном из столбцов уникального ключа значение NULL, а остальные столбцы заполнены значениями, и эти значения имеют совпадения хотя бы в одном из них.

> **Примечание**
>
> Для уникальных ключей, содержащих несколько столбцов, допускаются дубликаты значений NULL только тогда, когда они помещаются во все столбцы ограничения. В остальных случаях значения NULL будут рассматриваться как одно из обычных значений (не UNKNOWN).

```
RECREATE TABLE t( x int, y int, z int, unique(x,y,z));
INSERT INTO t values( NULL, 1, 1 );
INSERT INTO t values( NULL, NULL, 1 );
INSERT INTO t values( NULL, NULL, NULL );
INSERT INTO t values( NULL, NULL, NULL ); -- Разрешено
INSERT INTO t values( NULL, NULL, 1 ); -- Запрещено
```

Внешний ключ (FOREIGN KEY)

На уровне столбца ограничение внешнего ключа определяется предложением REFERENCES. После ключевого слова REFERENCES указывается имя родительской таблицы, на первичный или уникальный ключ которой ссылается описываемый внешний ключ. Имя столбца родительской таблицы, являющегося первичным (уникальным) ключом, помещается сразу после имени таблицы и заключается в круглые скобки, например

```
... ,
ARTIFACT_ID INTEGER REFERENCES COLLECTION (ARTIFACT_ID),
```

Синтаксис определения внешнего ключа на уровне таблицы несколько отличается. После определения всех столбцов, с их ограничения уровня столбца, вы можете определить именованное ограничение внешнего ключа уровня таблицы, используя ключевые слова FOREIGN KEY и имён столбцов для которых оно применяется:

```
... ,
CONSTRAINT FK_ARTSOURCE FOREIGN KEY(DEALER_ID, COUNTRY)
  REFERENCES DEALER (DEALER_ID, COUNTRY),
```

Внешний ключ не требует ограничения NOT NULL для столбца или столбцов на которые ссылается внешний ключ, хотя оно будет, если внешний ключ ссылается на столбец или столбцы входящие в первичный ключ родительской таблицы. Тем не менее, внешний ключ

может ссылаться на столбец или столбцы входящие в уникальный ключ. NULL допускается в уникальных ключах, и, с некоторыми ограничениями, в уникальных ключах построенных на нескольких столбцах (см. обсуждение выше).

Для обеспечения дополнительной целостности данных можно указать необязательные опции ON DELETE и ON UPDATE, которые обеспечат согласованность данных между родительскими и дочерними таблицами по заданным правилам:

- Предложение ON UPDATE определяет, что произойдёт с записями подчинённой таблицы при изменении значения первичного/уникального ключа в строке главной таблицы.

- Предложение ON DELETE определяет, что произойдёт с записями подчинённой таблицы при удалении соответствующей строки главной таблицы.

Для обеспечения ссылочной целостности внешнего ключа, когда изменяется или удаляется значение связанного первичного или уникального ключа, могут быть выполнены следующие действия:

- NO ACTION — не будет выполнено никаких действий; в клиентской программе должны быть предприняты специальные меры по поддержанию ссылочной целостности данных, иначе обновление/удаление не будет произведено и будет выдано соответствующее сообщение об ошибке;
- CASCADE — при изменении или удалении значения первичного ключа над значением внешнего ключа будут произведены те же действия. При выполнении удаления строки в главной таблице в подчинённой таблице должны быть удалены все записи, имеющие те же значения внешнего ключа, что и значение первичного (уникального) ключа удалённой строки главной таблицы. При выполнении обновления записи главной таблицы в подчинённой таблице должны быть изменены все значения внешнего ключа, имеющие те же значения, что и значение первичного (уникального) ключа изменяемой строки главной таблицы;
- SET DEFAULT — значения внешнего ключа всех соответствующих строк в подчинённой таблице устанавливаются в значение по умолчанию, заданное в предложении DEFAULT для этого столбца;
- SET NULL — значения внешнего ключа всех соответствующих строк в подчинённой таблице устанавливаются в пустое значение NULL.

Ограничение CHECK

Ограничение CHECK задаёт условие, которому должны удовлетворять значения, помещаемые в данный столбец. Условие — это логическое выражение, называемое также предикат, которое может возвращать значения TRUE (истина), FALSE (ложь) и UNKNOWN (неизвестно). Условие считается выполненным, если предикат возвращает значение TRUE или UNKNOWN (эквивалент NULL). Если предикат возвращает FALSE, то значение не будет принято. Это условие используется при добавлении в таблицу новой строки (оператор INSERT) и при изменении существующего значения столбца таблицы (оператор UPDATE), а также операторов, в которых может произойти одно из этих действий (UPDATE OR INSERT, MERGE).

> **Важно**
>
> При использовании предложения CHECK для столбца, базирующегося на домене, следует помнить, что выражение в CHECK лишь дополняет условие проверки, которое может уже быть определено в домене.

На уровне столбца выражение в предложении CHECK ссылается на входящее значение с помощью ключевого слова VALUE, так же как предложение CHECK в определении домена:

```
...
CURRENCY CHAR(3) NOT NULL
CONSTRAINT CHECK_CURRENCY
CHECK (VALUE IN ('AUD','EUR','GBP','RUR','USD','YEN')),
...
```

На уровне таблицы выражение в предложении CHECK ссылается на входящее значение по идентификатору столбца:

```
...
CONSTRAINT CHECK_EXCHANGE
CHECK (CURRENCY IN ('AUD','EUR','GBP','RUR','USD','YEN')
AND CURRENCY <> EXCHANGE_CURRENCY),
...
```

Кто может создать таблицу?

Создать новую таблицу могут:

- SYSDBA;
- Владелец базы данных;
- Любой пользователь с привилегией на создание таблицы (GRANT CREATE TABLE);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Пользователь, создавший таблицу, становится её владельцем.

Примеры

Пример 5.32. Создание таблицы

```
CREATE TABLE COUNTRY (
  COUNTRY COUNTRYNAME NOT NULL PRIMARY KEY,
  CURRENCY VARCHAR(10) NOT NULL);
```

Пример 5.33. Создание таблицы с заданием именованного первичного и уникального ключей

```
CREATE TABLE STOCK (
  MODEL SMALLINT NOT NULL CONSTRAINT PK_STOCK PRIMARY KEY,
```

```
  MODELNAME CHAR(10) NOT NULL,
  ITEMID INTEGER NOT NULL,
  CONSTRAINT MOD_UNIQUE UNIQUE (MODELNAME, ITEMID));
```

Пример 5.34. Таблица с полем массивом

```
CREATE TABLE JOB (
    JOB_CODE          JOBCODE NOT NULL,
    JOB_GRADE         JOBGRADE NOT NULL,
    JOB_COUNTRY       COUNTRYNAME,
    JOB_TITLE         VARCHAR(25) NOT NULL,
    MIN_SALARY        NUMERIC(18, 2) DEFAULT 0 NOT NULL,
    MAX_SALARY        NUMERIC(18, 2) NOT NULL,
    JOB_REQUIREMENT   BLOB SUB_TYPE 1,
    LANGUAGE_REQ      VARCHAR(15) [1:5],
    PRIMARY KEY (JOB_CODE, JOB_GRADE, JOB_COUNTRY),
    FOREIGN KEY (JOB_COUNTRY) REFERENCES COUNTRY (COUNTRY)
      ON UPDATE CASCADE
      ON DELETE SET NULL,
    CONSTRAINT CHK_SALARY CHECK (MIN_SALARY < MAX_SALARY)
);
```

Пример 5.35. Создание таблицы с ограничением первичного, внешнего и уникального ключа для которых заданы пользовательские имена индексов

```
CREATE TABLE PROJECT (
    PROJ_ID    PROJNO NOT NULL,
    PROJ_NAME  VARCHAR(20) NOT NULL UNIQUE
      USING DESC INDEX IDX_PROJNAME,
    PROJ_DESC    BLOB SUB_TYPE 1,
    TEAM_LEADER  EMPNO,
    PRODUCT      PRODTYPE,
    CONSTRAINT PK_PROJECT PRIMARY KEY (PROJ_ID)
      USING INDEX IDX_PROJ_ID,
    FOREIGN KEY (TEAM_LEADER) REFERENCES EMPLOYEE (EMP_NO)
      USING INDEX IDX_LEADER
);
```

Пример 5.36. Создание таблицы со столбцом идентификации

```
CREATE TABLE objects (
  id INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
  name VARCHAR(15)
);

INSERT INTO objects (name) VALUES ('Table');
INSERT INTO objects (name) VALUES ('Book');
INSERT INTO objects (id, name) VALUES (10, 'Computer');
```

```
SELECT * FROM objects;
```

```
ID           NAME
============ ===============
           1 Table
           2 Book
          10 Computer
```

Пример 5.37. Создание таблицы с вычисляемыми полями

```
CREATE TABLE SALARY_HISTORY (
    EMP_NO          EMPNO NOT NULL,
    CHANGE_DATE     TIMESTAMP DEFAULT 'NOW' NOT NULL,
    UPDATER_ID      VARCHAR(20) NOT NULL,
    OLD_SALARY      SALARY NOT NULL,
    PERCENT_CHANGE  DOUBLE PRECISION DEFAULT 0 NOT NULL,
    SALARY_CHANGE   GENERATED ALWAYS AS
      (OLD_SALARY * PERCENT_CHANGE / 100),
    NEW_SALARY      COMPUTED BY
      (OLD_SALARY + OLD_SALARY * PERCENT_CHANGE / 100)
);
```

Поле SALARY_CHANGE объявлено согласно стандарту SQL::2003, поле NEW_SALARY в классическом стиле объявления вычисляемых полей в Firebird.

Глобальные временные таблицы (GTT)

Если в операторе создания таблицы указано необязательное предложение GLOBAL TEMPORARY, то вместо обычной таблицы будет создана глобальная временная таблица. Глобальные временные таблицы (в дальнейшем сокращённо GTT) так же, как и обычные таблицы, являются постоянными метаданными, но данные в них ограничены по времени существования транзакцией (значение по умолчанию) или соединением с БД. Каждая транзакция или соединение имеет свой собственный экземпляр GTT с данными, изолированный от всех остальных. Экземпляры создаются только при условии обращения к GTT, и данные в ней удаляются при подтверждении транзакции или отключении от БД.

Если в операторе создания глобальной временной таблицы указано необязательное предложение ON COMMIT DELETE ROWS, то будет создана GTT транзакционного уровня (по умолчанию). При указании предложения ON COMMIT PRESERVE ROWS – будет создана GTT уровня соединения с базой данных.

Предложение EXTERNAL [FILE] нельзя использовать для глобальной временной таблицы.

Глобальные временные таблицы имеют ряд ограничений:

- GTT и обычные таблицы не могут ссылаться друг на друга;
- GTT уровня соединения ("PRESERVE ROWS") GTT не могут ссылаться на GTT транзакционного уровня ("DELETE ROWS");
- Уничтожения экземпляра GTT в конце своего жизненного цикла не вызывает срабатывания триггеров до/после удаления.

Примеры

Пример 5.38. Создание глобальной временной таблицы уровня соединения

```
CREATE GLOBAL TEMPORARY TABLE MYCONNGTT (
    ID INTEGER NOT NULL PRIMARY KEY,
    TXT VARCHAR(32),
    TS TIMESTAMP DEFAULT CURRENT_TIMESTAMP);
ON COMMIT PRESERVE ROWS;
```

Пример 5.39. Создание глобальной временной таблицы уровня транзакции ссылающейся внешним ключом на глобальную временную таблицу уровня соединения.

```
CREATE GLOBAL TEMPORARY TABLE MYTXGTT (
    ID INTEGER NOT NULL PRIMARY KEY,
    PARENT_ID INTEGER NOT NULL REFERENCES MYCONNGTT(ID),
    TXT VARCHAR(32),
    TS TIMESTAMP DEFAULT CURRENT_TIMESTAMP);
```

Внешние таблицы

Необязательное предложение EXTERNAL [FILE] указывает, что таблица хранится вне базы данных во внешнем текстовом файле. Столбцы таблицы, хранящейся во внешнем файле, могут быть любого типа за исключением BLOB. Недопустимо также использование массивов с любым типом данных. Над таблицей, хранящейся во внешнем файле, допустимы только операции добавления новых строк (INSERT) и выборки (SELECT) данных. Операции же изменения существующих данных (UPDATE) или удаления строк такой таблицы (DELETE) не могут быть выполнены. Внешняя таблица не может содержать ограничений первичного, внешнего и уникального ключа. Для полей такой таблицы невозможно создать индексы. Файл с внешней таблицей должен располагаться на устройстве хранения, физически расположенном на сервере, на котором расположена СУБД. Если при обращении к таблице СУБД не находит файла, она его создаёт. Возможность использования для таблиц внешних файлов зависит от установки значения параметра *`ExternalFileAccess`* в файле конфигурации `firebird.conf`.

Примеры

Пример 5.40. Создание внешней таблицы

```
CREATE TABLE EXT_LOG
EXTERNAL FILE 'log.txt' (
   BYTIME TIMESTAMP,
   AMESSAGE VARCHAR(100)
);
```

См. также: ALTER TABLE, DROP TABLE, CREATE DOMAIN.

ALTER TABLE

Назначение: Изменение структуры таблицы.

Доступно в: DSQL, ESQL.

Синтаксис:

```
ALTER TABLE tablename
<operation> [, <operation>];

<operation> ::= {
    ADD <col_def>
  | ADD <tconstraint>
  | DROP colname
  | DROP CONSTRAINT constr_name
  | ALTER [COLUMN] colname {
        TO new_colname
      | TYPE <datatype>
      | POSITION new_col_position
      | SET DEFAULT {literal | NULL | <context_var>}
      | DROP DEFAULT
      | SET NOT NULL
      | DROP NOT NULL
    }
  | ALTER [COLUMN] gencolname [TYPE <datatype>]
      {GENERATED ALWAYS AS | COMPUTED [BY]} (<col_expr>)
  | ALTER [COLUMN] idencolname RESTART [ WITH startvalue ]
}

<col_def> ::= colname {
  { <datatype> | domainname }
  [DEFAULT {literal | NULL | <context_var>}]
  [NOT NULL]
  [<col_constraint>]
  [COLLATE collation]
} | <col_exp_def> | <col_indentity_def>

<col_exp_def> ::=
  [<datatype>] {COMPUTED [BY] | GENERATED ALWAYS AS} (<col_expr>)

<col_indentity_def> ::= [<datatype>] GENERATED BY DEFAULT AS IDENTITY
  [(START WITH startvalue)] [<col_constraint>]


<datatype> ::= {
    {SMALLINT | INTEGER | BIGINT} [<array_dim>]
  | BOOLEAN [<array_dim>]
  | {FLOAT | DOUBLE PRECISION} [<array_dim>]
  | {DATE | TIME | TIMESTAMP} [<array_dim>]
  | {DECIMAL | NUMERIC} [(precision [, scale])] [<array_dim>]
  | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(size)]
      [<array_dim>] [CHARACTER SET charset]
  | {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR} [VARYING] [(size)]
      [<array_dim>]
  | BLOB [SUB_TYPE {subtype_num | subtype_name}] [SEGMENT SIZE seglen]
      [CHARACTER SET charset]
  | BLOB [(seglen [, subtype_num])]
}

<array_dim> ::= [x:y [,x:y ...]]
```

```
<col_constraint> ::= [CONSTRAINT constr_name]
{   UNIQUE [<using_index>]
  | PRIMARY KEY [<using_index>]
  | REFERENCES other_table [(other_col)] [<using_index>]
      [ON DELETE { NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
      [ON UPDATE { NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
  | CHECK (<search_condition>)
}

<tconstraint> ::= [CONSTRAINT constr_name]
{   UNIQUE (colname [, colname ...]) [<using_index>]
  | PRIMARY KEY (colname [, colname ...]) [<using_index>]
  | FOREIGN KEY (colname [, colname ...])
    REFERENCES other_table [(other_col [, other_col ...])] [<using_index>]
      [ON DELETE { NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
      [ON UPDATE { NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
  | CHECK (<search_condition>)
}

<using_index> ::= USING [ASC[ENDING] | DESC[ENDING]] INDEX indexname

<search_condition> ::= {
    <val> <operator> <val>
  | <val> [NOT] BETWEEN <val> AND <val>
  | <val> [NOT] IN (<val> [, <val> ...] | <select_list>)
  | <val> IS [NOT] NULL
  | <val> IS [NOT] DISTINCT <val>
  | <val> [NOT] CONTAINING <val>
  | <val> [NOT] STARTING [WITH] <val>
  | <val> [NOT] LIKE <val> [ESCAPE <val>]
  | <val> [NOT] SIMILAR TO <val> [ESCAPE <val>]
  | <val> <operator> {ALL | SOME | ANY} (<select_list>)
  | [NOT] EXISTS (<select_expr>)
  | [NOT] SINGULAR (<select_expr>)
  | (<search_condition>)
  | NOT <search_condition>
  | <search_condition> OR <search_condition>
  | <search_condition> AND <search_condition>
}

<operator> ::= {= | < | > | <= | >= | !< | !> | <> | !=}

<val> ::= {
    colname [[<ind> [, <ind> ...]]]
  | literal
  | <context_var>
  | <expression>
  | NULL
  | NEXT VALUE FOR genname
  | GEN_ID(genname, <val>)
  | CAST(<val> AS <datatype>)
  | (<select_one>)
  | func(<val> [, <val> ...])
}
```

Параметры оператора ALTER TABLE

tablename
Имя таблицы.

operation
Одна из допустимых операций по изменению структуры таблицы.

col_def
Определение столбца.

colname
Имя столбца таблицы.

new_colname
Новое имя столбца таблицы, может содержать до 31 байта.

gencolname
Имя вычисляемого столбца таблицы.

idencolname
Имя столбца идентификации.

new_col_position
Новая позиция столбца в таблице. Целое число в диапазоне от 1 до количества столбцов таблицы.

datatype
Тип данных SQL.

col_identity_def
Определение столбца идентификации.

startvalue
Начальное значение столбца идентификации.

col_exp_def
Определение вычисляемого столбца.

col_expr
Выражение для вычисляемого столбца.

domain_name
Имя домена.

col_constraint
Ограничение столбца.

tconstraint
Ограничение таблицы.

constr_name
Имя ограничения, может содержать до 31 байта.

other_table
Имя таблицы, на которую ссылается внешний ключ.

other_col
Столбец таблицы, на которую ссылается внешний ключ.

literal
Литерал.

context_var
Любая контекстная переменная, тип которой совместим с типом данных столбца.

search_condition
Условие проверки ограничения.

collation
Порядок сортировки.

array_dim
Размерность массива.

x
Начальный номер элемента в массиве, положительное целое число.

y
Последний номер элемента в массиве, положительное целое число.

precision
Точность. От 1 до 18.

scale
Масштаб. От 0 до 18, должно быть меньше или равно *precision*.

size
Максимальный размер строки в символах.

charset
Набор символов.

subtype_num
Номер подтипа BLOB.

subtype_name
Мнемоника подтипа BLOB.

seglen
Размер сегмента, не может превышать 65535.

val
Значение.

select_one
Оператор SELECT выбирающий один столбец и возвращающий только одну строку.

select_list
Оператор SELECT выбирающий один столбец и возвращающий ноль и более строк.

select_expr
Оператор SELECT выбирающий несколько столбцов и возвращающий ноль и более строк.

experssion
Выражение.

genname
Имя последовательности (генератора).

func
Скалярная функция.

Оператор ALTER TABLE изменяет структуру существующей таблицы. Одиночный оператор ALTER TABLE позволяет производить множество операций добавления/удаления столбцов и ограничений, а также модификаций столбцов. Список операций выполняемых при модификации таблицы разделяется запятой.

ALTER TABLE

Некоторые изменения структуры таблицы увеличивают счётчик форматов, закреплённый за каждой таблицей. Количество форматов для каждой таблицы ограничено значением 255. После того, как счётчик форматов достигнет этого значения, вы не сможете больше менять структуру таблицы.

Для сброса счётчика форматов необходимо сделать резервное копирование и восстановление базы данных (утилитой gbak).

Предложение ADD

Предложение ADD позволяет добавить новый столбец или новое ограничение таблицы. Синтаксис определения столбца и синтаксис описания ограничения таблицы полностью совпадают с синтаксисом, описанным в операторе CREATE TABLE.

Воздействие на счётчик ссылок:

- При каждом добавлении нового столбца номер формата увеличивается на единицу.
- Добавление нового ограничения таблицы не влечёт за собой увеличение номера формата.

> **Примечание**
>
> При добавлении нового столбца, не допускающего значения NULL, в таблицу с данными необходимо также установить значение по умолчанию с помощью предложения DEFAULT. Дело в том, что в этом случае также происходит проверка данных на допустимость. А поскольку при добавлении нового столбца, он для всех строк таблицы содержит значение NULL, будет сгенерировано исключение.

> **Предупреждение**
>
> Будьте осторожны, при добавлении нового ограничения CHECK не осуществляется проверка соответствия ему ранее внесённых данных. Поэтому перед добавлением такого ограничения рекомендуем производить предварительную проверку данных в таблице.

Предложение DROP

Предложение DROP удаляет указанный столбец таблицы. Столбец таблицы не может быть удалён, если от него существуют зависимости. Другими словами для успешного удаления столбца на него должны отсутствовать ссылки. Ссылки на столбец могут содержаться:

- в ограничениях столбцов или таблицы;
- в индексах;
- в хранимых процедурах и триггерах;
- в представлениях.

При каждом удалении столбца номер формата увеличивается на единицу.

Предложение DROP CONSTRAINT

Предложение DROP CONSTRAINT удаляет указанное ограничение столбца или таблицы. Ограничение первичного ключа или уникального ключа не могут быть удалены, если они используются в ограничении внешнего ключа другой таблицы. В этом случае, необходимо удалить ограничение FOREIGN KEY до удаления PRIMARY KEY или UNIQUE ключа, на которые оно ссылается.

Удаление ограничения столбца или ограничения таблицы не влечёт за собой увеличение номера формата.

Предложение ALTER [COLUMN]

Предложение ALTER [COLUMN] позволяет изменить следующие характеристики существующих столбцов:

- изменение имени (не изменяет номер формата);
- изменение типа данных (увеличивает номер формата на единицу);
- изменение позиции столбца в списке столбцов таблицы (не изменяет номер формата);
- удаление значения по умолчанию столбца (не изменяет номер формата);
- добавление значения по умолчанию столбца (не изменяет номер формата);
- изменение типа и выражения для вычисляемого столбца (не изменяет номер формата);
- добавление ограничения NOT NULL (не изменяет номера формата);
- удаление ограничения NOT NULL (не изменяет номера формата).

Переименование столбца

Ключевое слово TO переименовывает существующий столбец. Новое имя столбца не должно присутствовать в таблице.

Невозможно изменение имени столбца, если этот столбец включён в какое-либо ограничение — первичный или уникальный ключ, внешний ключ, ограничение столбца или проверочное ограничение таблицы CHECK. Имя столбца также нельзя изменить, если этот столбец таблицы используется в каком-либо триггере, в хранимой процедуре или представлении.

Изменение типа столбца

Ключевое слово TYPE изменяет тип существующего столбца на другой допустимый тип. Не допустимы любые изменения типа, которые могут привести к потере данных. Например, количество символов в новом типе для столбца не может быть меньше, чем было установлено ранее.

Если столбец был объявлен как массив, то изменить ни его тип, ни размерность нельзя.

Нельзя изменить тип данных у столбца, который принимает участие в связке внешний ключ / первичный (уникальный) ключ.

Изменение позиции столбца

Ключевое слово POSITION изменяет позицию существующего столбца. Позиции столбцов нумеруются с единицы.

- Если будет задан номер позиции меньше 1, то будет выдано соответствующее сообщение об ошибке.
- Если будет задан номер позиции, превышающий количество столбцов в таблице, то изменения не будут выполнены, но ни ошибки, ни предупреждения не последуют.

Установка и удаление значения по умолчанию

Предложение DROP DEFAULT удаляет значение по умолчанию для столбца таблицы.

- Если столбец основан на домене со значением по умолчанию — доменное значение перекроет это удаление.

- Если удаление значения по умолчанию производится над столбцом, у которого нет значения по умолчанию, или чьё значение по умолчанию основано на домене, то это приведёт к ошибке выполнения данного оператора.

Предложение SET DEFAULT устанавливает значение по умолчанию для столбца таблицы. Если столбец уже имел значение по умолчанию, то оно будет заменено новым. Значение по умолчанию для столбца всегда перекрывает доменное значение по умолчанию.

Установка и удаление ограничения NOT NULL

Предложение SET NOT NULL добавляет ограничение NOT NULL для столбца таблицы.

> **Примечание**
>
> Успешное добавление ограничения NOT NULL происходит, только после полной проверки данных таблицы, для того чтобы убедится что столбец не содержит значений NULL.
>
> Явное ограничение NOT NULL на столбце, базирующегося на домене, преобладает над установками домена. В этом случае изменение домена для допустимости значения NULL, не распространяется на столбец таблицы.

Предложение DROP NOT NULL удаляет ограничение NOT NULL для столбца таблицы. Если столбец основан на домене с ограничением NOT NULL, то ограничение домена перекроет это удаление.

Изменение столбцов идентификации

Для столбцов идентификации (GENERATED BY DEFAULT AS IDENTITY) позволено изменять начальное значение. Если указано только предложение RESTART, то происходит сброс значения генератора в ноль. Необязательное предложение WITH позволяет указать для нового значения внутреннего генератора отличное от нуля значение. Невозможно изменить обычный столбец на столбец идентификации и наоборот.

Изменение вычисляемых столбцов

Для вычисляемых столбцов (GENERATED ALWAYS AS или COMPUTED BY) позволяется изменить тип и выражение вычисляемого столбца. Невозможно изменить обычный столбец на вычисляемый и наоборот.

Не изменяемые атрибуты

На данный момент не существует возможности изменить сортировку по умолчанию.

Кто может изменить таблицу?

Изменить таблицу могут:

- SYSDBA;
- Владелец базы данных;
- Владелец таблицы;
- Любой пользователь с привилегией на изменение любой таблицы (GRANT ALTER ANY TABLE);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.41. Добавление столбца в таблицу

```
ALTER TABLE COUNTRY
ADD CAPITAL VARCHAR(25);
```

Пример 5.42. Добавление столбца с ограничением уникальности и удаление другого столбца

```
ALTER TABLE COUNTRY
  ADD CAPITAL VARCHAR(25) UNIQUE,
  DROP CURRENCY;
```

Пример 5.43. Добавление столбца с ограничением NOT NULL

```
ALTER TABLE OBJECTS
ADD QUANTITY INT DEFAULT 1 NOT NULL;
```

> **Примечание**
>
> Обратите внимание на предложение DEFAULT, которое обязательно при добавлении ограничения NOT NULL, если в таблице есть данные.

Пример 5.44. Добавление проверочного ограничения и внешнего ключа

```
ALTER TABLE JOB
```

```
ADD CONSTRAINT CHK_SALARY CHECK (MIN_SALARY < MAX_SALARY),
ADD FOREIGN KEY (JOB_COUNTRY)
REFERENCES COUNTRY (COUNTRY);
```

Пример 5.45. Модификация сразу нескольких столбцов таблицы

```
ALTER TABLE STOCK
ALTER COLUMN MODEL SET DEFAULT 1,
ALTER COLUMN ITEMID TYPE BIGINT,
ALTER COLUMN ITEMID NULL,
ALTER COLUMN MODELNAME TO NAME;
```

Пример 5.46. Изменение столбца идентификации

```
ALTER TABLE objects
ALTER ID RESTART WITH 100;
```

Пример 5.47. Изменение вычисляемых столбцов

```
ALTER TABLE SALARY_HISTORY
ALTER NEW_SALARY GENERATED ALWAYS
AS (OLD_SALARY + OLD_SALARY * PERCENT_CHANGE / 100),
ALTER SALARY_CHANGE COMPUTED
BY (OLD_SALARY * PERCENT_CHANGE / 100);
```

См. также: CREATE TABLE, RECREATE TABLE.

DROP TABLE

Назначение: Удаление существующей таблицы.

Доступно в: DSQL, ESQL.

Синтаксис:

```
DROP TABLE tablename;
```

Параметры оператора DROP TABLE

tablename
Имя таблицы.

Оператор DROP TABLE удаляет существующую таблицу. Если таблица имеет зависимости, то удаление не будет произведено. При удалении таблицы будут также удалены все триггеры на её события и индексы, построенные для её полей.

Кто может удалить таблицу?

Удалить таблицу могут:

- SYSDBA;
- Владелец базы данных;
- Владелец таблицы;
- Любой пользователь с привилегией на удаление любой таблицы (GRANT DROP ANY TABLE);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.48. Удаление таблицы

```
DROP TABLE COUNTRY;
```

См. также: CREATE TABLE, RECREATE TABLE.

RECREATE TABLE

Назначение: Создание новой таблицы или пересоздание существующей.

Доступно в: DSQL.

Синтаксис:

```
RECREATE [GLOBAL TEMPORARY] TABLE tablename
  [EXTERNAL [FILE] '<filespec>']
  (<col_def> [, <col_def> | <tconstraint> ...])
  [ON COMMIT {DELETE | PRESERVE} ROWS];
```

Полное описание определений столбцов и ограничений таблицы смотрите в разделе CREATE TABLE.

Создаёт или пересоздаёт таблицу. Если таблица с таким именем уже существует, то оператор RECREATE TABLE попытается удалить её и создать новую. Оператор RECREATE TABLE не выполнится, если существующая таблица имеет зависимости.

Примеры

Пример 5.49. Создание или пересоздание таблицы

```
RECREATE TABLE COUNTRY (
  COUNTRY COUNTRYNAME NOT NULL PRIMARY KEY,
  CURRENCY VARCHAR(10) NOT NULL);
```

См. также: CREATE TABLE, DROP TABLE.

INDEX

Индекс (index) — объект базы данных, предназначенный для ускорения выборки данных из таблицы и/или для ускорения упорядочения результатов выборки данных из таблицы. Кроме того, индексы используются для обеспечения ограничений целостности — PRIMARY KEY, FOREIGN KEY, UNIQUE.

В данном разделе описываются вопросы создания индексов, перевода их в активное/неактивное состояние, удаление индексов и сбор статистики (пересчёт селективности) для индексов.

CREATE INDEX

Назначение: Создание индекса для таблицы.

Доступно в: DSQL, ESQL.

Синтаксис:

```
CREATE [UNIQUE] [ASC[ENDING] | DESC[ENDING]]
INDEX indexname ON tablename
{(col [, col …]) | COMPUTED BY (<expression>)};
```

Параметры оператора CREATE INDEX

indexname
Имя индекса. Может содержать до 31 байта.

tablename
Имя таблицы, для которой строится индекс.

col
Столбец таблицы. В качестве столбцов не могут быть использованы поля типа BLOB, ARRAY и вычисляемые поля.

expression
Выражение, содержащее столбцы таблицы.

Оператор CREATE INDEX создаёт индекс для таблицы, который может быть использован для ускорения поиска, сортировки и/или группирования. Кроме того, индекс может быть использован при определении ограничений, таких как первичный ключ, внешний ключ или

ограничениях уникальности. Индекс может быть построен на столбцах любого типа кроме BLOB и массивов. Имя индекса должно быть уникальным среди всех имён индексов.

Индексы в ключах

При добавлении ограничений первичного ключа, внешнего ключа или ограничения уникальности будет неявно создан одноименный индекс. Так, например, при выполнении следующего оператора будет неявно создан индекс PK_COUNTRY.

```
ALTER TABLE COUNTRY
ADD CONSTRAINT PK_COUNTRY PRIMARY KEY (ID);
```

Уникальные индексы

Если при создании индекса указано ключевое слово UNIQUE, то индекс гарантирует уникальность значений ключей. Такой индекс называется уникальным. Уникальный индекс не является ограничением уникальности.

Уникальные индексы не могут содержать дубликаты значений ключей, но могут содержать дубликаты значения NULL в соответствии со стандартом SQL-99 (в том числе и в многосегментном индексе).

Направление индекса

Индекс может быть построен в восходящем (ASCENDING) и нисходящем (DESCENDING) порядке.

Ключевое слово ASCENDING (сокращённо ASC), обозначает, что ключи индекса расположены по возрастанию значений. Такое расположение ключей используется по умолчанию.

Ключевое слово DESCENDING (сокращённо DESC), обозначает, что ключи индекса расположены по убыванию значений.

Вычисляемые индексы или индексы по выражению

При создании индекса вместо одного или нескольких столбцов вы также можете указать одно выражение, используя предложение COMPUTED BY. Такой индекс называется вычисляемым или индексом по выражению. Вычисляемые индексы используются в запросах, в которых условие в предложениях WHERE, ORDER BY или GROUP BY в точности совпадает с выражением в определении индекса. Выражение в вычисляемом индексе может использовать несколько столбцов таблицы.

Примечание

Не смотря на то, что можно создать вычисляемый индекс по вычисляемому полю, использоваться такой индекс не будет.

Ограничения на индексы

Максимальная длина ключа индекса ограничена 1/4 размера страницы.

Ограничения на длину индексируемой строки

Максимальная длина индексируемой строки на 9 байтов меньше, чем максимальная длина ключа. Максимальная длина индексируемой строки зависит от размера страницы и набора символов.

Таблица 5.1. Длина индексируемой строки и набор символов

Размер страницы	Максимальная длина индексируемой строки для набора символов, байт/символ				
	1	2	3	4	6
4096	1015	507	338	253	169
8192	2039	1019	679	509	339
16384	4087	2043	1362	1021	682

Максимальное количество индексов на таблицу

Для каждой таблицы максимально возможное количество индексов ограничено и зависит от размера страницы и количества столбцов в индексе.

Таблица 5.2. Число индексов и количество столбцов

Размер страницы	Число индексов в зависимости от количества столбцов в индексе		
	1	2	3
4096	203	145	113
8192	408	291	227
16384	818	584	454

Кто может создать индекс?

Создать индекс могут:

- SYSDBA;
- Владелец базы данных;
- Владелец таблицы, для которой создаётся индекс;
- Любой пользователь с привилегией на изменение любой таблицы (GRANT ALTER ANY TABLE);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.50. Создание индекса

```
CREATE INDEX IDX_UPDATER ON SALARY_HISTORY (UPDATER_ID);
```

Пример 5.51. Создание индекса с сортировкой ключей по убыванию

```
CREATE DESCENDING INDEX IDX_CHANGE
ON SALARY_HISTORY (CHANGE_DATE);
```

Пример 5.52. Создание многосегментного индекса

```
CREATE INDEX IDX_SALESTAT ON SALES (ORDER_STATUS, PAID);
```

Пример 5.53. Создание индекса, не допускающего дубликаты значений

```
CREATE UNIQUE INDEX UNQ_COUNTRY_NAME ON COUNTRY (NAME);
```

Пример 5.54. Создание вычисляемого индекса

```
CREATE INDEX IDX_NAME_UPPER ON PERSONS
COMPUTED BY (UPPER (NAME));
```

Такой индекс может быть использован для регистронезависимого поиска.

```
SELECT *
FROM PERSONS
WHERE UPPER(NAME) STARTING WITH UPPER('Iv');
```

См. также: ALTER INDEX, DROP INDEX.

ALTER INDEX

Назначение: Перевод индекса в активное/неактивное состояние, перестройка индекса.

Доступно в: DSQL, ESQL.

Синтаксис:

```
ALTER INDEX indexname {ACTIVE | INACTIVE};
```

Параметры оператора ALTER INDEX

indexname
Имя индекса.

Оператор ALTER INDEX переводит индекс в активное/неактивное состояние. Возможность изменения структуры и порядка сортировки ключей этот оператор не предусматривает.

- При выборе опции INACTIVE, индекс переводится из активного в неактивное состояние. Перевод индекса в неактивное состояние по своему действию похоже на команду DROP INDEX за исключением того, что определение индекса сохраняется в базе данных. Невозможно перевести в неактивное состояние индекс участвующий в ограничении.

  Подсказка

  Перевод индекса в неактивное состояние может быть полезен при массовой вставке, модификации или удалении записей из таблицы, для которой этот индекс построен.

- При выборе альтернативы ACTIVE индекс переводится из неактивного состояния в активное. При переводе индекса из неактивного состояния в активное индекс перестраивается.

  Подсказка

  Даже если индекс находится в активном состоянии оператор ALTER INDEX ACTIVE всё равно перестраивает индекс. Таким образом, эту команду можно использовать для перестройки индексов, автоматически созданных для ограничений PRIMARY KEY, FOREIGN KEY, UNIQUE, для которых выполнение оператора ALTER INDEX INACTIVE невозможно.

Кто может выполнить ALTER INDEX?

Перевести индекс в активное/неактивное состояние могут:

- SYSDBA;
- Владелец базы данных;
- Владелец таблицы, для которой построен индекс;
- Любой пользователь с привилегией на изменение любой таблицы (GRANT ALTER ANY TABLE);
- Любой пользователь, подключенный с ролью RDB$ADMIN (роль должна быть назначена пользователю);
- Пользователь операционной системы root (Linux);
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.55. Перевод индекса в неактивное состояние

```
ALTER INDEX IDX_UPDATER INACTIVE;
```

Пример 5.56. Возврат индекса в активное состояние

```

```

```
ALTER INDEX IDX_UPDATER ACTIVE;
```

См. также: CREATE INDEX, DROP INDEX.

DROP INDEX

Назначение: Удаление индекса из базы данных.

Доступно в: DSQL, ESQL.

Синтаксис:

```
DROP INDEX indexname;
```

Параметры оператора DROP INDEX

indexname
Имя индекса.

Оператор DROP INDEX удаляет существующий индекс из базы данных. При наличии зависимостей для существующего индекса (если он используется в ограничении) удаление не будет выполнено.

Кто может удалить индекс?

Индекс могут удалить:

- SYSDBA;
- Владелец базы данных;
- Владелец таблицы, для которой построен индекс;
- Любой пользователь с привилегией на изменение любой таблицы (GRANT ALTER ANY TABLE);
- Любой пользователь, подключенный с ролью RDB$ADMIN (роль должна быть назначена пользователю);
- Пользователь операционной системы root (Linux);
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.57. Удаление индекса

```
DROP INDEX IDX_UPDATER;
```

См. также: CREATE INDEX, ALTER INDEX.

SET STATISTICS

Назначение: Пересчёт селективности индекса.

Доступно в: DSQL, ESQL.

Синтаксис:

```
SET STATISTICS INDEX indexname;
```

Параметры оператора SET STATISTICS

indexname
Имя индекса.

Оператор SET STATISTICS пересчитывает значение селективности для указанного индекса.

Селективность индекса

Селективность (избирательность) индекса — это оценочное количество строк, которые могут быть выбраны при поиске по каждому значению индекса. Уникальный индекс имеет максимальную селективность, поскольку при его использовании невозможно выбрать более одной строки для каждого значения ключа индекса. Актуальность селективности индекса важна для выбора наиболее оптимального плана выполнения запросов оптимизатором.

Пересчёт селективности индекса может потребоваться после массовой вставки, модификации или удалении большого количества записей из таблицы, поскольку она становится неактуальной.

> **Примечание**
>
> Отметим, что в Firebird статистика индексов автоматически не пересчитывается ни после массовых изменений данных, ни при каких либо других условиях. При создании (CREATE) или его активации (ALTER INDEX ACTIVE) статистика индекса полностью соответствует его содержимому.

Кто может обновить статистику?

Пересчитать селективность индекса могут:

- SYSDBA;
- Владелец базы данных;
- Владелец таблицы, для которой построен индекс;
- Любой пользователь с привилегией на изменение любой таблицы (GRANT ALTER ANY TABLE);
- Любой пользователь, подключенный с ролью RDB$ADMIN (роль должна быть назначена пользователю);
- Пользователь операционной системы root (Linux);
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.58. пересчёт селективности индекса

```
SET STATISTICS INDEX IDX_UPDATER;
```

См. также: CREATE INDEX, ALTER INDEX.

VIEW

Представление (view) — виртуальная таблица, которая по своей сути является именованным запросом SELECT выборки данных произвольной сложности. Выборка данных может осуществляться из одной и более таблиц, других представлений, а также селективных хранимых процедур.

В отличие от обычных таблиц реляционных баз данных, представление не является самостоятельным набором данных, хранящимся в базе данных. Результат в виде набора данных динамически создаётся при обращении к представлению.

CREATE VIEW

Назначение: Создание нового представления.

Доступно в: DSQL.

Синтаксис:

```
CREATE VIEW viewname [<full_column_list>]
AS <select_statement>
[WITH CHECK OPTION];

<full_column_list> ::= (colname [, colname ...])
```

Параметры оператора CREATE VIEW

viewname
Имя представления. Может содержать до 31 байта.

select_statement
Оператор SELECT.

full_column_list
Список столбцов представления.

colname
Имя столбца представления. Дубликаты имён столбцов не позволяются.

Оператор CREATE VIEW создаёт новое представление. Имя представления должно быть уникальным среди имён всех представлений, таблиц и хранимых процедур базы данных.

После имени создаваемого представления может идти список имён столбцов, получаемых в результате обращения к представлению. Имена в списке могут быть никак не связаны с именами столбцов базовых таблиц. При этом их количество должно точно соответствовать количеству столбцов в списке выбора главного оператора SELECT представления.

Если список столбцов представления отсутствует, то будут использоваться имена столбцов базовых таблиц или псевдонимов (алиасов) полей оператора SELECT.

Примечание

- Если существует полный список столбцов, то задание псевдонимов не имеет смысла, поскольку они будут переопределены именами из списка столбцов;
- Полный список столбцов, используемых в представлениях, обязателен для оператора SELECT, содержащего выражения на основе столбцов или идентичные имена столбцов;
- Список столбцов необязателен при условии, что вы укажете их в операторе SELECT и их имена будут уникальными.

Обновляемые представления

Представление может быть изменяемым и неизменяемым. Если представление изменяемое, то данные, полученные при обращении к такому представлению, можно изменить при помощи DML операторов INSERT, UPDATE, DELETE, UPDATE OR INSERT, MERGE. Изменения, выполненные над представлением, отражаются в таблице, из которой происходит выборка данных.

Неизменяемое представление можно сделать изменяемым при помощи вспомогательных триггеров. Если на обновляемом представлении будет построен один или несколько триггеров, то изменения не будут автоматически попадать в таблицу — либо это делает триггер, либо запись не происходит.

Для того чтобы представление было изменяемым, необходимо выполнение следующих условий:

- оператор выборки SELECT обращается только к одной таблице или одному изменяемому представлению;
- оператор выборки SELECT не должен обращаться к хранимым процедурам;
- все столбцы базовой таблицы или изменяемого представления, которые не присутствуют в данном представлении, допускают значение NULL;
- оператор выборки SELECT не содержит полей определённых через подзапросы или другие выражения;
- оператор выборки SELECT не содержит полей определённых через агрегатные функции (MIN, MAX, AVG и др.);
- оператор выборки SELECT не содержит предложений ORDER BY, GROUP BY, HAVING;
- оператор выборки SELECT не содержит ключевого слова DISTINCT и ограничений количества строк ROWS, FIRST, SKIP.

WITH CHECK OPTIONS

Необязательное предложение WITH CHECK OPTIONS задаёт для изменяемого представления требования проверки вновь введённых или модифицируемых данных условию, указанному в предложении WHERE оператора выборки SELECT. При попытке вставки новой записи или модификации записи проверяется, выполняется ли для этой записи условие в предложении

WHERE, если условие не выполняется, то вставка/модификация не выполняется и будет выдано соответствующее диагностическое сообщение.

Предложение WITH CHECK OPTION может задаваться в операторе создания представления только в том случае, если в главном операторе SELECT представления указано предложение WHERE. Иначе будет выдано сообщение об ошибке.

Кто может создать представление?

Создать представление могут:

- SYSDBA;
- Владелец базы данных;
- Любой пользователь с привилегией на создание представления (GRANT CREATE VIEW);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Для создания представления необходимы также привилегии на чтение (SELECT) данных из базовых таблиц и представлений, и привилегии на выполнение (EXECUTE) используемых селективных хранимых процедур. Если же представление изменяемое, то пользователю необходимы привилегии ALL к базовым таблицам. Пользователь, создавший представление, становится его владельцем.

Примеры

Пример 5.59. Создание представления

```
CREATE VIEW ENTRY_LEVEL_JOBS AS
SELECT JOB_CODE, JOB_TITLE
FROM JOB
WHERE MAX_SALARY < 15000;
```

Пример 5.60. Создание представления с проверкой условия фильтрации

Создание представления возвращающего столбцы JOB_CODE и JOB_TITLE только для тех работ, где MAX_SALARY меньше $15000. При вставке новой записи или изменении существующей будет осуществляться проверка условия MAX_SALARY < 15000, если условие не выполняется, то вставка/изменение будет отвергнуто.

```
CREATE VIEW ENTRY_LEVEL_JOBS AS
SELECT JOB_CODE, JOB_TITLE
FROM JOB
WHERE MAX_SALARY < 15000
WITH CHECK OPTIONS;
```

Пример 5.61. Создание представления с использованием списка столбцов

```
CREATE VIEW PRICE_WITH_MARKUP (
  CODE_PRICE,
  COST,
  COST_WITH_MARKUP
) AS
SELECT
  CODE_PRICE,
  COST,
  COST * 1.1
FROM PRICE;
```

Пример 5.62. Создание представления с использованием псевдонимов полей

```
CREATE VIEW PRICE_WITH_MARKUP AS
SELECT
  CODE_PRICE,
  COST,
  COST * 1.1 AS COST_WITH_MARKUP
FROM PRICE;
```

Пример 5.63. Создание необновляемого представления с использованием хранимой процедуры

```
CREATE VIEW GOODS_PRICE
SELECT
    goods.name AS goodsname,
    price.cost AS cost,
    b.quantity AS quantity
FROM
    goods
    JOIN price ON goods.code_goods = price.code_goods
    LEFT JOIN sp_get_balance(goods.code_goods) b ON 1 = 1;
```

Пример 5.64. Создание обновляемого представления с использованием триггеров

```
-- базовые таблицы
RECREATE TABLE t_films(id INT PRIMARY KEY, title VARCHAR(100));
RECREATE TABLE t_sound(id INT PRIMARY KEY, audio BLOB);
RECREATE TABLE t_video(id INT PRIMARY KEY, video BLOB);
COMMIT;

-- создание необновляемого представления
RECREATE VIEW v_films AS
  SELECT f.id, f.title, s.audio, v.video
  FROM t_films f
```

```
  LEFT JOIN t_sound s ON f.id = s.id
  LEFT JOIN t_video v ON f.id = v.id;

/* Для того чтобы сделать представление обновляемым создадим
   триггер, который будет производить манипуляции над базовыми
   таблицами.
*/
SET TERM ^;
CREATE OR ALTER TRIGGER v_films_biud FOR v_films
ACTIVE BEFORE INSERT OR UPDATE OR DELETE POSITION 0 AS
BEGIN
  IF (INSERTING) THEN
    new.id = COALESCE(new.id, GEN_ID(g_films, 1));
  IF (NOT DELETING) THEN
  BEGIN
    UPDATE OR INSERT INTO t_films(id, title)
    VALUES(new.id, new.title)
    MATCHING(id);

    UPDATE OR INSERT INTO t_sound(id, audio)
    VALUES(new.id, new.audio)
    MATCHING(id);

    UPDATE OR INSERT INTO t_video(id, video)
    VALUES(new.id, new.video)
    MATCHING(id);
 END
 ELSE
 BEGIN
   DELETE FROM t_films WHERE id = old.id;
   DELETE FROM t_sound WHERE id = old.id;
   DELETE FROM t_video WHERE id = old.id;
 END
END^
SET TERM ;^

/*
 * Теперь мы можем производить манипуляции над
 * этим представлением как будто мы работаем с таблицей
 */
INSERT INTO v_films(title, audio, video)
VALUES('007 coordinates skyfall', 'pif-paf!', 'oh! waw!');
```

См. также: ALTER VIEW, CREATE OR ALTER VIEW, RECREATE VIEW, DROP VIEW.

ALTER VIEW

Назначение: Изменение существующего представления.

Доступно в: DSQL.

Синтаксис:

```
ALTER VIEW viewname [<full_column_list>]
```

```
AS <select_statement>
[WITH CHECK OPTION];

<full_column_list> ::= (colname [, colname ...])
```

Параметры оператора ALTER VIEW

viewname
Имя представления.

select_statement
Оператор SELECT.

full_column_list
Список столбцов представления.

colname
Имя столбца представления. Дубликаты имён столбцов не позволяются.

Оператор ALTER VIEW изменяет определение существующего представления, существующие права на представления и зависимости при этом сохраняются. Синтаксис оператора ALTER VIEW полностью аналогичен синтаксису оператора CREATE VIEW.

> **Предупреждение**
>
> Будьте осторожны при изменении количества столбцов представления. Существующий код приложения может стать неработоспособным. Кроме того, PSQL модули, использующие изменённое представление, могут стать некорректными. Информация о том, как это обнаружить, находится в приложении Поле RDB$VALID_BLR.

Кто может изменить представление?

Изменить представление могут:

- SYSDBA;
- Владелец базы данных;
- Владелец представления;
- Любой пользователь с привилегией на изменение любого представления (GRANT ALTER ANY VIEW);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.65. Изменение представления

```
ALTER VIEW PRICE_WITH_MARKUP (
  CODE_PRICE,
```

```
  COST,
  COST_WITH_MARKUP
) AS
SELECT
  CODE_PRICE,
  COST,
  COST * 1.15
FROM PRICE;
```

См. также: CREATE VIEW, CREATE OR ALTER VIEW, RECREATE VIEW.

CREATE OR ALTER VIEW

Назначение: Создание нового или изменение существующего представления.

Доступно в: DSQL.

Синтаксис:

```
CREATE OR ALTER VIEW viewname [<full_column_list>]
AS <select_statement>
[WITH CHECK OPTION];

<full_column_list> ::= (colname [, colname ...])
```

Параметры оператора CREATE OR ALTER VIEW

viewname
Имя представления. Может содержать до 31 байта.

select_statement
Оператор SELECT.

full_column_list
Список столбцов представления.

colname
Имя столбца представления. Дубликаты имён столбцов не позволяются.

Оператор CREATE OR ALTER VIEW создаёт представление, если оно не существует. В противном случае он изменит представление с сохранением существующих зависимостей.

Примеры

Пример 5.66. Создание нового или изменение существующего представления

```
CREATE OR ALTER VIEW PRICE_WITH_MARKUP (
  CODE_PRICE,
  COST,
  COST_WITH_MARKUP
```

```
) AS
SELECT
  CODE_PRICE,
  COST,
  COST * 1.15
FROM PRICE;
```

См. также: CREATE VIEW, ALTER VIEW, RECREATE VIEW.

DROP VIEW

Назначение: Удаление существующего представления.

Доступно в: DSQL.

Синтаксис:

```
DROP VIEW viewname;
```

Параметры оператора DROP VIEW

viewname
Имя представления.

Оператор DROP VIEW удаляет существующее представление. Если представление имеет зависимости, то удаление не будет произведено.

Кто может удалить представление?

Удалить представление могут:

Удалить представление могут:

- SYSDBA;
- Владелец базы данных;
- Владелец представления;
- Любой пользователь с привилегией на удаление любого представления (GRANT DROP ANY VIEW);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.67. Удаление представления

```
DROP VIEW PRICE_WITH_MARKUP;
```

См. также: CREATE VIEW, RECREATE VIEW.

RECREATE VIEW

Назначение: Создание нового или пересоздание существующего представления.

Доступно в: DSQL.

Синтаксис:

```
RECREATE VIEW viewname [<full_column_list>]
AS <select_statement>
[WITH CHECK OPTION];

<full_column_list> ::= (colname [, colname ...])
```

Параметры оператора RECREATE VIEW

viewname
Имя представления. Может содержать до 31 байта.

select_statement
Оператор SELECT.

full_column_list
Список столбцов представления.

colname
Имя столбца представления. Дубликаты имён столбцов не позволяются.

Создаёт или пересоздаёт представление. Если представление с таким именем уже существует, то оператор RECREATE VIEW попытается удалить его и создать новое. Оператор RECREATE VIEW не выполнится, если существующее представление имеет зависимости.

Примеры:

Пример 5.68. Создание нового или пересоздание существующего представления

```
RECREATE VIEW PRICE_WITH_MARKUP (
  CODE_PRICE,
  COST,
  COST_WITH_MARKUP
) AS
SELECT
  CODE_PRICE,
  COST,
  COST * 1.15
```

```
FROM PRICE;
```

См. также: CREATE VIEW, CREATE OR VIEW, DROP VIEW.

TRIGGER

Триггер (trigger) — это хранимая процедура особого типа, которая не вызывается непосредственно, а исполнение которой обусловлено наступлением одного из событий, относящегося к одной конкретной таблице (представлению), или наступлению одного из событий базы данных.

CREATE TRIGGER

Назначение: Создание нового триггера.

Доступно в: DSQL, ESQL.

Синтаксис:

```
CREATE TRIGGER trigname {
    <relation_trigger_legacy>
  | <relation_trigger_sql2003>
  | <database_trigger>
  | <ddl_trigger> }
  {EXTERNAL NAME '<extname>' ENGINE <engine>} |
  {
    AS
      [<declarations>]
    BEGIN
      [<PSQL_statements>]
    END
  }


<relation_trigger_legacy> ::=
  FOR {tablename | viewname}
  [ACTIVE | INACTIVE]
  {BEFORE | AFTER} <mutation_list>
  [POSITION number]

<relation_trigger_sql2003> ::=
  [ACTIVE | INACTIVE]
  {BEFORE | AFTER} <mutation_list>
  [POSITION number]
  ON {tablename | viewname}

<database_trigger> ::=
  [ACTIVE | INACTIVE]
  ON db_event
  [POSITION number]
```

```
<ddl_trigger> ::=
  [ACTIVE | INACTIVE]
  {BEFORE | AFTER} <ddl_events>
  [POSITION number]

<mutation_list> ::= <mutation> [OR <mutation> [OR <mutation>]]

<mutation> ::= { INSERT | UPDATE | DELETE }

<db_event> ::= {
    CONNECT
  | DISCONNECT
  | TRANSACTION START
  | TRANSACTION COMMIT
  | TRANSACTION ROLLBACK
}

<ddl_events> ::= {
    ANY DDL STATEMENT
  | <ddl_event_item> [{OR <ddl_event_item>} ...]
}

<ddl_event_item> ::=
    CREATE TABLE | ALTER TABLE | DROP TABLE
  | CREATE PROCEDURE | ALTER PROCEDURE | DROP PROCEDURE
  | CREATE FUNCTION | ALTER FUNCTION | DROP FUNCTION
  | CREATE TRIGGER | ALTER TRIGGER | DROP TRIGGER
  | CREATE EXCEPTION | ALTER EXCEPTION | DROP EXCEPTION
  | CREATE VIEW | ALTER VIEW | DROP VIEW
  | CREATE DOMAIN | ALTER DOMAIN | DROP DOMAIN
  | CREATE ROLE | ALTER ROLE | DROP ROLE
  | CREATE SEQUENCE | ALTER SEQUENCE | DROP SEQUENCE
  | CREATE USER | ALTER USER | DROP USER
  | CREATE INDEX | ALTER INDEX | DROP INDEX
  | CREATE COLLATION | DROP COLLATION
  | ALTER CHARACTER SET
  | CREATE PACKAGE | ALTER PACKAGE | DROP PACKAGE
  | CREATE PACKAGE BODY | DROP PACKAGE BODY
  | CREATE MAPPING | ALTER MAPPING | DROP MAPPING
```

Параметры оператора CREATE TRIGGER

trigname
: Имя триггера. Может содержать до 31 байта.

relation_trigger_legacy
: Объявление табличного триггера (унаследованное).

relation_trigger_sql2003
: Объявление табличного триггера согласно стандарту SQL-2003.

database_trigger
: Объявление триггера базы данных.

ddl_trigger
: Объявление DDL триггера.

tablename
Имя таблицы.

viewname
Имя представления.

mutation_list
Список событий таблицы.

mutation
Одно из событий таблицы.

db_event
Событие соединения или транзакции.

ddl_events
Список событий изменения метаданных.

ddl_event_item
Одно из событий изменения метаданных.

number
Порядок срабатывания триггера. От 0 до 32767.

declarations
Секция объявления локальных переменных и именованных курсоров.

PSQL_statments
Операторы языка PSQL.

Оператор CREATE TRIGGER создаёт новый триггер. Триггер может быть создан для события (или событий) таблицы (представления), для события (событий) изменения метаданных или для одного из событий базы данных.

Оператор CREATE TRIGGER как и его родственники ALTER TRIGGER, CREATE OR ALTER TRIGGER и RECREATE TRIGGER являются составными операторами, содержащими заголовок и тело.

Заголовок определяет имя триггера, а также содержит имя отношения (для табличных триггеров), фазу триггера, событие (или события) на которые срабатывает триггер и позицию. Имя триггера должно быть уникальным среди имён других триггеров.

Тело триггера состоит из секции объявления локальных переменных, именованных курсоров и подпрограмм (подробнее в см. в разделе DECLARE), за которой следует один или несколько операторов, или блоков операторов, заключённых во внешнем блоке, который начинается с ключевого слова BEGIN, и завершается ключевым словом END. Объявления локальных переменных и именованных курсоров, а также внутренние операторы должны завершаться точкой с запятой (;).

Табличные триггеры

Табличные триггеры выполняются на уровне строки (записи) каждый раз, когда изменяется образ строки.

Форма объявления

Объявление табличного триггера существует в двух вариантах:

- своеобразная, унаследованная форма;
- SQL-2003 совместимая.

В настоящее время рекомендуется использовать SQL-2003 совместимую форму.

Табличные триггеры указать фазу и одно или несколько событий.

Состояние триггера

Триггер может быть в одном из двух состояний активном (ACTIVE) или неактивном (INACTIVE). Запускаются только активные триггеры. По умолчанию триггеры создаются в активном состоянии.

Фаза

Триггер может выполняться в одной из двух фаз, связанных с запрошенными изменениями состояния данных. Ключевое слово BEFORE означает, что триггер вызывается до наступления соответствующего события (событий, если их указано несколько), AFTER — после наступления события (событий).

События таблицы

Для табличного триггера может быть указано одно из событий таблицы (представления) — INSERT (добавление), UPDATE (изменение), DELETE (удаление) — или несколько событий, разделённых ключевым словом OR, при которых вызывается триггер. При создании триггера каждое событие (INSERT, UPDATE или DELETE) не должен упоминаться более одного раза.

Порядок срабатывания

Ключевое слово POSITION позволяет задать порядок, в котором будут выполняться триггеры с одинаковой фазой и событием (или группы событий). По умолчанию позиция равна 0. Если позиции для триггеров не заданы или несколько триггеров имеют одно и то же значение позиции, то такие триггеры будут выполняться в алфавитном порядке их имен.

Тело табличного триггера

Внутри тела табличных триггеров доступны специфические контекстные переменные OLD и NEW. Более правильное название этих ключевых слов — префиксы имён столбцов. В триггерах можно обращаться к значению любого столбца таблицы (представления) до его изменения в клиентской программе (для этого перед именем столбца помещается ключевое слово OLD и точка) и после изменения (перед именем столбца помещается NEW и точка).

Контекстная переменная OLD (префикс имени столбца) для всех видов триггеров является переменной только для чтения. Она недоступна в триггерах, вызываемых при добавлении данных, независимо от фазы события.

Контекстная переменная NEW в триггерах для фазы события после (AFTER) также является переменной только для чтения. Она недоступна в триггерах для события удаления данных.

Внешние триггеры

Триггер может быть расположена во внешнем модуле. В этом случае вместо тела триггера указывается место его расположения во внешнем модуле с помощью предложения EXTERNAL NAME. Аргументом этого предложения является строка, в которой через разделитель указано имя внешнего модуля, имя процедуры внутри модуля и определённая пользователем информация. В предложении ENGINE указывается имя движка для обработки подключения внешних модулей. В Firebird для работы с внешними модулями используется движок UDR.

Кто может создать табличный триггер?

Триггеры для событий таблицы (представления) могут создать:

- SYSDBA;
- Владелец базы данных;
- Владельцем таблицы (представления);
- Любой пользователь с привилегией на изменение любой таблицы/представления (GRANT ALTER ANY {TABLE | VIEW});
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.69. Создание табличного триггера в Legacy стиле

```
CREATE TRIGGER SET_CUST_NO FOR CUSTOMER
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.CUST_NO IS NULL) THEN
    NEW.CUST_NO = GEN_ID(CUST_NO_GEN, 1);
END
```

Пример 5.70. Создание табличного триггера согласно стандарту SQL-2003

```
CREATE TRIGGER set_cust_no
ACTIVE BEFORE INSERT POSITION 0 ON customer
AS
BEGIN
  IF (NEW.cust_no IS NULL) THEN
    NEW.cust_no = GEN_ID(cust_no_gen, 1);
END
```

Пример 5.71. Создание табличного триггера на несколько событий

```
CREATE TRIGGER TR_CUST_LOG
```

```
ACTIVE AFTER INSERT OR UPDATE OR DELETE POSITION 10
ON CUSTOMER
AS
BEGIN
  INSERT INTO CHANGE_LOG (LOG_ID,
                          ID_TABLE,
                          TABLE_NAME,
                          MUTATION)
  VALUES (NEXT VALUE FOR SEQ_CHANGE_LOG,
          OLD.CUST_NO,
          'CUSTOMER',
          CASE
            WHEN INSERTING THEN 'INSERT'
            WHEN UPDATING  THEN 'UPDATE'
            WHEN DELETING  THEN 'DELETE'
          END);
END
```

См. также: ALTER TRIGGER, DROP TRIGGER.

Триггеры на событие базы данных

Триггер может быть создан для одного из событий базы данных:

- CONNECT (соединение с базой данных);
- DISCONNECT (отсоединение от базы данных);
- TRANSACTION START (старт транзакции);
- TRANSACTION COMMIT (подтверждение транзакции);
- TRANSACTION ROLLBACK (откат транзакции).

Указать для триггера несколько событий базы данных невозможно.

Выполнение триггеров на событие базы данных и обработка исключений

Триггера на события CONNECT и DISCONNECT выполняются в специально созданной для этого транзакции. Если при обработке триггера не было вызвано исключение, то транзакция подтверждается. Не перехваченные исключения откатят транзакцию и:

- в случае триггера на событие CONNECT соединение разрывается, а исключения возвращается клиенту;
- для триггера на событие DISCONNECT соединение разрывается, как это и предусмотрено, но исключения не возвращается клиенту.

Триггера на событие TRANSACTION срабатывают при старте транзакции, её подтверждении или отмене. Не перехваченные исключения обрабатываются в зависимости от типа события TRANSACTION:

- для события START исключение возвращается клиенту, а транзакция отменяется;
- для события COMMIT исключение возвращается клиенту, действия, выполненные триггером, и транзакция отменяются;
- для события ROLLBACK исключение не возвращается клиенту, а транзакция, как и предусмотрено, отменяется.

Ловушки

Из вышеизложенного следует, что нет прямого способа узнать, какой триггер (DISCONNECT или ROLLBACK) вызвал исключение. Также ясно, что вы не сможете подключиться к базе данных в случае исключения в триггере на событие CONNECT, а также отменяется старт транзакции при исключении в триггере на событие TRANSACTION START. В обоих случаях база данных эффективно блокируется, в то время как вы собирались работать с ней.

Двухфазное подтверждение транзакций

В случае двухфазных транзакций триггеры на событие TRANSACTION START срабатывают в фазе подготовки (prepare), а не в фазе commit.

Отключение триггеров

В некоторые утилиты командной строки Firebird были добавлены новые ключи для отключения триггеров на базу данных:

```
gbak -nodbtriggers
isql -nodbtriggers
nbackup -T
```

Эти ключи могут использоваться только SYSDBA или владельцем базы данных.

Предостережения

1. Триггеры для событий базы данных DISCONNECT и ROLLBACK не будут вызваны при отключении клиентов через таблицы мониторинга (DELETE FROM MON$ATTACHMENTS).

2. Использование оператора IN AUTONOMOUS TRANSACTION DO в триггерах на событие базы данных связанные с транзакциями (COMMIT, ROLLBACK, START) может привести к его зацикливанию.

Кто может создать триггеры на события базы данных?

Триггеры для событий базы данных могут создать:

Триггеры для событий базы данных могут создать:

- SYSDBA;
- Владелец базы данных;
- Пользователь, имеющий привилегию ALTER DATABASE;
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.72. Создание триггера на событие подключения к БД для логирования события

```
CREATE TRIGGER tr_log_connect
INACTIVE ON CONNECT POSITION 0
AS
BEGIN
  INSERT INTO LOG_CONNECT (ID,
                           USERNAME,
                           ATIME)
  VALUES (NEXT VALUE FOR SEQ_LOG_CONNECT,
          CURRENT_USER,
          CURRENT_TIMESTAMP);
END
```

Пример 5.73. Создание триггера на событие подключения к БД для контроля доступа

```
CREATE EXCEPTION E_INCORRECT_WORKTIME 'Рабочий день ещё не начался';

CREATE TRIGGER TR_LIMIT_WORKTIME ACTIVE
ON CONNECT POSITION 1
AS
BEGIN
  IF ((CURRENT_USER <> 'SYSDBA') AND
      NOT (CURRENT_TIME BETWEEN time '9:00' AND time '17:00')) THEN
     EXCEPTION E_INCORRECT_WORKTIME;
END
```

См. также: ALTER TRIGGER, DROP TRIGGER.

Триггеры на события изменения метаданных

Триггеры на события изменения метаданных (DDL триггеры) предназначены для обеспечения ограничений, которые будут распространены на пользователей, которые пытаются создать, изменить или удалить DDL объект. Другое их назначение — ведение журнала изменений метаданных.

Триггеры на события изменения метаданных являются одним из подвидов триггеров на события базы данных.

Особенности:

1. BEFORE триггеры запускаются до изменений в системных таблицах. AFTER триггеры запускаются после изменений в системных таблицах.

2. Когда оператор DDL запускает триггер, в котором возбуждается исключение (BEFORE или AFTER, преднамеренно или неумышленно), оператор не будет фиксирован. Т.е. исключения могут использоваться, чтобы гарантировать, что оператор DDL будет отменен, если некоторые условия не будут соблюдены.

3. Действия DDL триггеров выполняются только при фиксации транзакции, в которой работает затронутая DDL команда. Никогда не забывайте о том, что в AFTER триггере, возможно

сделать только то, что возможно сделать после DDL команды без автоматической фиксации транзакций. Вы не можете, например, создать таблицу в триггере и использовать её там.

4. Для операторов "CREATE OR ALTER" триггер срабатывает один раз для события CREATE или события ALTER, в зависимости от того существовал ли ранее объект. Для операторов RECREATE триггер вызывается для события DROP, если объект существовал, и после этого для события CREATE.

5. Если объект метаданных не существует, то обычно триггеры на события ALTER и DROP не запускаются. Исключения описаны в пункте 6.

6. Исключением из правила 5 являются BEFORE ALTER/DROP USER триггеры, которые будут вызваны, даже если имя пользователя не существует. Это вызвано тем, что эти команды выполняются для базы данных безопасности, для которой не делается проверка существования пользователей перед их выполнением. Данное поведение, вероятно, будет отличаться для встроенных пользователей, поэтому не пишите код, который зависит от этого.

7. Если некоторое исключение возбуждено после того как начала выполняться DDL команда и до того как запущен AFTER триггер, то AFTER триггер не запускается.

8. Для процедур и функций в составе пакетов не запускаются индивидуальные триггеры {CREATE | ALTER | DROP} {PROCEDURE | FUNCTION}.

9. Оператор ALTER DOMAIN *<old name>* TO *<new name>* устанавливает контекстные переменные OLD_OBJECT_NAME и NEW_OBJECT_NAME в обоих триггерах BEFORE и AFTER. Контекстная переменная OBJECT_NAME будет содержать старое имя объекта метаданных в триггере BEFORE, и новое — в триггере AFTER.

Если в качестве события указано предложение ANY DDL STATEMENT, то триггер будет вызван при наступлении любого из DDL событий.

Пространство имён DDL_TRIGGER

Во время работы DDL триггера доступно пространство имён DDL_TRIGGER для использования в функции RDB$GET_CONTEXT. Его использование также допустимо в хранимых процедурах и функциях, вызванных триггерами DDL.

Контекст DDL_TRIGGER работает как стек. Перед возбуждением DDL триггера, значения, относящиеся к выполняемой команде, помещаются в этот стек. После завершения работы триггера значения выталкиваются. Таким образом. В случае каскадных DDL операторов, когда каждая пользовательская DDL команда возбуждает DDL триггер, и этот триггер запускает другие DDL команды, с помощью EXECUTE STATEMENT, значения переменных в пространстве имён DDL_TRIGGER будут соответствовать команде, которая вызвала последний DDL триггер в стеке вызовов.

Переменные доступные в пространстве имён DDL_TRIGGER

- EVENT_TYPE – тип события (CREATE, ALTER, DROP)
- OBJECT_TYPE – тип объекта (TABLE, VIEW и д.р.)
- DDL_EVENT – имя события (*<ddl event item>*),

  где *<ddl event item>* = EVENT_TYPE || ' ' || OBJECT_TYPE

- OBJECT_NAME – имя объекта метаданных
- OLD_OBJECT_NAME – имя объекта метаданных до переименования
- NEW_OBJECT_NAME – имя объекта метаданных после переименования
- SQL_TEXT – текст SQL запроса

Отключение триггеров

В некоторые утилиты командной строки Firebird были добавлены новые ключи для отключения триггеров на базу данных:

```
gbak -nodbtriggers
isql -nodbtriggers
nbackup -T
```

Эти ключи могут использоваться только SYSDBA или владельцем базы данных.

Кто может создать триггеры на события базы данных?

Триггеры для событий базы данных могут создать:

- SYSDBA;
- Владелец базы данных;
- Пользователь, имеющий привилегию ALTER DATABASE;
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.74. Контроль наименования объектов базы данных с помощью DDL триггера

```
CREATE EXCEPTION e_invalid_sp_name
  'Неверное имя хранимой процедуры (должно начинаться с SP_)';

SET TERM !;

CREATE TRIGGER trig_ddl_sp BEFORE CREATE PROCEDURE
AS
BEGIN
  IF (rdb$get_context('DDL_TRIGGER', 'OBJECT_NAME')
      NOT STARTING 'SP_') THEN
    EXCEPTION e_invalid_sp_name;
END!

-- Test
CREATE PROCEDURE sp_test
AS
```

```
BEGIN
END!

CREATE PROCEDURE test
AS
BEGIN
END!
```

```
-- Statement failed, SQLSTATE = 42000
-- exception 1
-- -E_INVALID_SP_NAME
-- -Неверное имя хранимой процедуры (должно начинаться с SP_)
-- -At trigger 'TRIG_DDL_SP' line: 4, col: 5
```

```
SET TERM ;!
```

Пример 5.75. Контроль безопасности DDL операторов

```
CREATE EXCEPTION e_access_denied 'Access denied';

SET TERM !;

CREATE TRIGGER trig_ddl BEFORE ANY DDL STATEMENT
AS
BEGIN
  IF (current_user <> 'SUPER_USER') THEN
    EXCEPTION e_access_denied;
END!

-- Test
CREATE PROCEDURE sp_test
AS
BEGIN
END!
```

```
-- The last command raises this exception and procedure SP_TEST is not created
-- Statement failed, SQLSTATE = 42000
-- exception 1
-- -E_ACCESS_DENIED
-- -Access denied
-- -At trigger 'TRIG_DDL' line: 4, col: 5
```

```
SET TERM ;!
```

Примечание

В Firebird существуют привилегии на DDL операторы, поэтому прибегать к написанию DDL триггера нужно только в случае, если того же самого эффекта невозможно достичь стандартными методами.

Пример 5.76. Использование DDL триггеров для регистрации событий изменения метаданных

```
CREATE SEQUENCE ddl_seq;

CREATE TABLE ddl_log (
  id BIGINT NOT NULL PRIMARY KEY,
  moment TIMESTAMP NOT NULL,
  user_name VARCHAR(31) NOT NULL,
  event_type VARCHAR(25) NOT NULL,
  object_type VARCHAR(25) NOT NULL,
  ddl_event VARCHAR(25) NOT NULL,
  object_name VARCHAR(31) NOT NULL,
  old_object_name VARCHAR(31),
  new_object_name VARCHAR(31),
  sql_text BLOB sub_type text NOT NULL,
  ok CHAR(1) NOT NULL
);

SET TERM !;

CREATE TRIGGER trig_ddl_log_before BEFORE ANY DDL STATEMENT
AS
  DECLARE id TYPE OF COLUMN ddl_log.id;
BEGIN
  -- Мы должны производить изменения в AUTONOMOUS TRANSACTION,
  -- таким образом, если произойдёт исключение и команда
  -- не будет запущена, она всё равно будет зарегистрирована.
  IN AUTONOMOUS TRANSACTION DO
  BEGIN
    INSERT INTO ddl_log (
      id, moment, user_name, event_type, object_type, ddl_event,
      object_name, old_object_name, new_object_name, sql_text, ok)
    VALUES (NEXT VALUE FOR ddl_seq,
            current_timestamp, current_user,
            rdb$get_context('DDL_TRIGGER', 'EVENT_TYPE'),
            rdb$get_context('DDL_TRIGGER', 'OBJECT_TYPE'),
            rdb$get_context('DDL_TRIGGER', 'DDL_EVENT'),
            rdb$get_context('DDL_TRIGGER', 'OBJECT_NAME'),
            rdb$get_context('DDL_TRIGGER', 'OLD_OBJECT_NAME'),
            rdb$get_context('DDL_TRIGGER', 'NEW_OBJECT_NAME'),
            rdb$get_context('DDL_TRIGGER', 'SQL_TEXT'),
            'N')
    RETURNING id INTO id;
    rdb$set_context('USER_SESSION', 'trig_ddl_log_id', id);
  END
END!

-- Примечание:
-- созданный выше триггер будет запущен для этой DDL.
-- Хорошей идеей является использование -nodbtriggers
-- при работе с ним
CREATE TRIGGER trig_ddl_log_after AFTER ANY DDL STATEMENT
AS
BEGIN
  -- Здесь нам требуется автономная транзакция,
```

```
  -- потому что в оригинальной транзакции
  -- мы не увидим запись, вставленную в
  -- BEFORE триггере в автономной транзакции,
  -- если пользовательская транзакции не запущена
  -- с режимом изоляции  READ COMMITTED.
  IN AUTONOMOUS TRANSACTION DO
    UPDATE ddl_log SET ok = 'Y'
    WHERE
      id = rdb$get_context('USER_SESSION', 'trig_ddl_log_id');
END!

COMMIT!

SET TERM ;!

-- Удаляем запись о создании trig_ddl_log_after.
DELETE FROM ddl_log;
COMMIT;

-- Тест

-- Эта команда будет зарегистрирована единожды
-- (т.к. T1 не существует, RECREATE вызовет событие CREATE)
-- с OK = Y.
RECREATE TABLE t1 (
  n1 INTEGER,
  n2 INTEGER
);

-- Оператор не выполнится, т.к. T1 уже существует,
-- таким образом OK будет иметь значение N.
CREATE TABLE t1 (
  n1 INTEGER,
  n2 INTEGER
);

-- T2 не существует. Это действие не будет зарегистрировано.
DROP TABLE t2;

-- Это действие будет зарегистрировано дважды
-- (т.к. T1 существует, действие RECREATE рассматривается
-- как DROP и CREATE) с полем OK = Y.
RECREATE TABLE t1 (
  n INTEGER
);

CREATE DOMAIN dom1 AS INTEGER;

ALTER DOMAIN dom1 TYPE BIGINT;

ALTER DOMAIN dom1 TO dom2;

COMMIT;

SELECT
  id,
  ddl_event,
```

```
  object_name as name,
  old_object_name as old_name,
  new_object_name as new_name,
  sql_text,
  ok
FROM ddl_log
ORDER BY id;
```

ID	DDL_EVENT	NAME	OLD_NAME	NEW_NAME	SQL_TEXT	OK

2	CREATE TABLE	T1	<null>	<null>	`RECREATE TABLE t1 (`
					`  n1 INTEGER,`
					`  n2 INTEGER`
					`)`
						Y

3	CREATE TABLE	T1	<null>	<null>	`CREATE TABLE t1 (`
					`  n1 INTEGER,`
					`  n2 INTEGER`
					`)`
						N

4	DROP TABLE	T1	<null>	<null>	`RECREATE TABLE t1 (`
					`  n INTEGER`
					`)`
						Y

5	CREATE TABLE	T1	<null>	<null>	`RECREATE TABLE t1 (`
					`  n INTEGER`
					`)`
						Y

6	CREATE DOMAIN	DOM1	<null>	<null>	`CREATE DOMAIN dom1` `  AS INTEGER`	Y
7	ALTER DOMAIN	DOM1	<null>	<null>	`ALTER DOMAIN dom1` `  TYPE BIGINT`	Y
8	ALTER DOMAIN	DOM1	DOM1	DOM2	`ALTER DOMAIN dom1` `  TO dom2`	Y

См. также: ALTER TRIGGER, DROP TRIGGER.

ALTER TRIGGER

Назначение: Изменение существующего триггера.

Доступно в: DSQL, ESQL.

Синтаксис:

```
ALTER TRIGGER trigname {
[ACTIVE | INACTIVE]
[{BEFORE | AFTER} <mutation_list>]
[POSITION number]
[
  {EXTERNAL NAME '<extname>' ENGINE <engine>} |
  {
    AS
      [<declarations>]
    BEGIN
      [<PSQL_statements>]
    END
  }
]

<mutation_list> ::= <mutation> [OR <mutation> [OR <mutation>]]

<mutation> ::= { INSERT | UPDATE | DELETE }
```

Параметры оператора ALTER TRIGGER

trigname
Имя триггера.

mutation_list
Список событий таблицы.

mutation
Одно из событий таблицы.

number
Порядок срабатывания триггера. От 0 до 32767.

declarations
Секция объявления локальных переменных и именованных курсоров.

PSQL_statments
Операторы языка PSQL.

Оператор ALTER TRIGGER позволяет изменять заголовок и/или тело триггера.

Допустимые изменения

В операторе изменения триггера можно изменить:

- Состояние активности (ACTIVE | INACTIVE);
- Фазу (BEFORE | AFTER);
- Событие(я);
- Позицию срабатывания;
- Код тела триггера.

Если какой-либо элемент не указан, то он остаётся без изменений.

Примечание

Табличный триггер невозможно изменить в триггер на событие базы данных и наоборот.

Событие в триггере базы данных невозможно изменить.

Помните

Триггер с ключевым словом BEFORE наступает до соответствующего события, с ключевым словом AFTER — после соответствующего события.

Один табличный триггер может содержать более одного события (INSERT, UPDATE, DELETE). События должны быть разделены ключевым словом OR. Каждое из событий может быть указано не более одного раза.

Ключевое слово POSITION позволяет задать дополнительный порядок выполнения с одинаковыми фазой и событием. По умолчанию позиция равна 0. Если позиция не задана, или если несколько триггеров имеют один и тот же номер позиции, то триггеры будут выполнены в алфавитном порядке их наименований.

Кто может изменить триггеры?

Триггеры для событий таблицы (представления) могут изменить:

- SYSDBA;
- Владельцем таблицы (представления);
- Любой пользователь с привилегией на изменение любой таблицы/представления (GRANT ALTER ANY {TABLE | VIEW});
- Владелец базы данных;
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Триггеры для событий базы данных и триггеры событий на изменение метаданных могут изменить:

- SYSDBA;
- Владелец базы данных;
- Пользователь, имеющий привилегию ALTER DATABASE;
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.77. Отключение (перевод в неактивное состояние) триггера

```
ALTER TRIGGER set_cust_no INACTIVE;
```

Пример 5.78. Изменение позиции триггера

```
ALTER TRIGGER set_cust_no POSITION 14;
```

Пример 5.79. Перевод триггера в неактивное состояние и изменение списка событий

```
ALTER TRIGGER TR_CUST_LOG
INACTIVE AFTER INSERT OR UPDATE;
```

Пример 5.80. Перевод триггера в активное состояние, изменение его позиции и его тела

```
ALTER TRIGGER tr_log_connect
ACTIVE POSITION 1
AS
BEGIN
  INSERT INTO LOG_CONNECT (ID,
                           USERNAME,
                           ROLENAME,
                           ATIME)
  VALUES (NEXT VALUE FOR SEQ_LOG_CONNECT,
          CURRENT_USER,
          CURRENT_ROLE,
          CURRENT_TIMESTAMP);
END
```

См. также: CREATE TRIGGER, CREATE OR ALTER TRIGGER, RECREATE TRIGGER.

CREATE OR ALTER TRIGGER

Назначение: Создание нового или изменение существующего триггера.

Доступно в: DSQL, ESQL.

Синтаксис:

```
CREATE OR ALTER TRIGGER trigname {
    <relation_trigger_legacy>
  | <relation_trigger_sql2003>
  | <database_trigger>
  | <ddl_trigger> }
  {EXTERNAL NAME '<extname>' ENGINE <engine>} |
  {
    AS
      [<declarations>]
```

```
    BEGIN
      [<PSQL_statements>]
    END
  }
```

Полное описание оператора см. CREATE TRIGGER.

Оператор CREATE OR ALTER TRIGGER создаёт новый триггер, если он не существует, или изменяет и перекомпилирует его в противном случае, при этом существующие права и зависимости сохраняются.

Примеры

Пример 5.81. Создание нового или изменение существующего триггера

```
CREATE OR ALTER TRIGGER set_cust_no
ACTIVE BEFORE INSERT POSITION 0 ON customer
AS
BEGIN
  IF (NEW.cust_no IS NULL) THEN
    NEW.cust_no = GEN_ID(cust_no_gen, 1);
END
```

См. также: CREATE TRIGGER, ALTER TRIGGER, RECREATE TRIGGER.

DROP TRIGGER

Назначение: Удаление существующего триггера.

Доступно в: DSQL, ESQL.

Синтаксис:

```
DROP TRIGGER trigname
```

Параметры оператора DROP TRIGGER

trigname
Имя триггера.

Оператор DROP TRIGGER удаляет существующий триггер.

Кто может удалить триггеры?

Триггеры для событий таблицы (представления) могут удалить:

- SYSDBA;
- Владельцем таблицы (представления);

- Любой пользователь с привилегией на изменение любой таблицы/представления (GRANT ALTER ANY {TABLE | VIEW});
- Владелец базы данных;
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Триггеры для событий базы данных и триггеры событий на изменение метаданных могут удалить:

- SYSDBA;
- Владелец базы данных;
- Пользователь, имеющий привилегию ALTER DATABASE;
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.82. Удаление триггера

```
DROP TRIGGER set_cust_no;
```

См. также: CREATE TRIGGER, ALTER TRIGGER.

RECREATE TRIGGER

Назначение: Создание нового или пересоздание существующего триггера.

Доступно в: DSQL, ESQL.

Синтаксис:

```
RECREATE TRIGGER trigname {
    <relation_trigger_legacy>
  | <relation_trigger_sql2003>
  | <database_trigger>
  | <ddl_trigger> }
  {EXTERNAL NAME '<extname>' ENGINE <engine>} |
  {
    AS
      [<declarations>]
    BEGIN
      [<PSQL_statements>]
    END
```

```
  }
```

Полное описание оператора см. CREATE TRIGGER.

Оператор RECREATE TRIGGER создаёт новый триггер, если триггер с указанным именем не существует, в противном случае оператор RECREATE TRIGGER попытается удалить его и создать новый.

Примеры

Пример 5.83. Создание или пересоздание триггера

```
RECREATE TRIGGER set_cust_no
ACTIVE BEFORE INSERT POSITION 0 ON customer
AS
BEGIN
  IF (NEW.cust_no IS NULL) THEN
    NEW.cust_no = GEN_ID(cust_no_gen, 1);
END
```

См. также: CREATE TRIGGER, DROP TRIGGER, CREATE OR ALTER TRIGGER.

PROCEDURE

Хранимая процедура (ХП) — это программный модуль, который может быть вызван с клиента, из другой процедуры, функции, выполнимого блока (executable block) или триггера. Хранимые процедуры, хранимые функции, исполняемые блоки и триггеры пишутся на процедурном языке SQL (PSQL). Большинство операторов SQL доступно и в PSQL, иногда с ограничениями или расширениями. Заметными исключениями являются DDL и операторы управления транзакциями.

Хранимые процедуры могут принимать и возвращать множество параметров.

CREATE PROCEDURE

Назначение: Создание новой хранимой функции.

Доступно в: DSQL, ESQL.

Синтаксис:

```
CREATE PROCEDURE procname [(<inparam> [, <inparam> ...])]
RETURNS (<outparam> [, <outparam> ...])
  { EXTERNAL NAME '<extname>' ENGINE <engine> }  |
  { AS
      [<declarations>]
    BEGIN
      [<PSQL_statements>]
```

```
    END
  }

<inparam> ::= <param_decl> [{= | DEFAULT} <value>]

<outparam>  ::=  <param_decl>

<value> ::=  {literal | NULL | context_var}

<param_decl> ::= paramname <type> [NOT NULL] [COLLATE collation]

<extname> ::= '<module name>!<routine name>[!<misc info>]'

<type> ::= <datatype> | [TYPE OF] domain | TYPE OF COLUMN rel.col

<datatype> ::=
    {SMALLINT | INTEGER | BIGINT}
  | BOOLEAN
  | {FLOAT | DOUBLE PRECISION}
  | {DATE | TIME | TIMESTAMP}
  | {DECIMAL | NUMERIC} [(precision [, scale])]
  | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(size)]
    [CHARACTER SET charset]
  | {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR} [VARYING] [(size)]
  | BLOB [SUB_TYPE {subtype_num | subtype_name}]
    [SEGMENT SIZE seglen] [CHARACTER SET charset]
  | BLOB [(seglen [, subtype_num])]

<declarations> ::= <declare_item> [<declare_item> ...]

<declare_item> ::=
  <declare_var>; |
  <declare_cursor>; |
  <declare_subfunc> |
  <declare_subproc>
```

Параметры оператора CREATE PROCEDURE

procname
Имя хранимой процедуры. Может содержать до 31 байта.

inparam
Описание входного параметра.

outparam
Описание выходного параметра.

declarations
Секция объявления локальных переменных, именованных курсоров, подпроцедур и подфункций.

declare_var
Объявление локальной переменной.

declare_cursor
Объявление именованного курсора.

declare_subfunc
Объявление подпрограммы – функции.

declare_subproc
Объявление подпрограммы – процедуры.

literal
Литерал.

context_var
Любая контекстная переменная, тип которой совместим с типом параметра.

paramname
Имя входного или выходного параметра процедуры. Может содержать до 31 байта. Имя параметра должно быть уникальным среди входных и выходных параметров процедуры, а также её локальных переменных.

module name
Имя внешнего модуля, в котором расположена функция.

routine name
Внутреннее имя функции внутри внешнего модуля.

misc info
Определяемая пользователем информация для передачи в функцию внешнего модуля.

engine
Имя движка для использования внешних функций. Обычно указывается имя UDR.

datatype
Тип данных SQL.

collation
Порядок сортировки.

domain
Домен.

rel
Имя таблицы или представления.

col
Имя столбца таблицы или представления.

precision
Точность. От 1 до 18.

scale
Масштаб. От 0 до 18, должно быть меньше или равно *precision*.

size
Максимальный размер строки в символах.

charset
Набор символов.

subtype_num
Номер подтипа BLOB.

subtype_name
Мнемоника подтипа BLOB.

seqlen
Размер сегмента, не может превышать 65535.

Оператор CREATE PROCEDURE создаёт новую хранимую процедуру. Имя хранимой процедуры должно быть уникальным среди имён всех хранимых процедур, таблиц и представлений базы данных.

> **Примечание**
>
> Желательно также, чтобы имя хранимой процедуры было уникальным и среди имён процедур расположенных в PSQL пакетах (package), хотя это и допустимо. Дело в том, что в настоящее время вы не сможете вызвать функцию/процедуру из глобального пространства имён внутри пакета, если в пакете объявлена одноименная функция/процедура. В этом случае всегда будет вызвана процедура/функция пакета.

CREATE PROCEDURE является составным оператором, состоящий из заголовка и тела. Заголовок определяет имя хранимой процедуры и объявляет входные и выходные параметры, если они должны быть возвращены процедурой.

Тело процедуры состоит из секции объявления локальных переменных, именованных курсоров и подпрограмм (подробнее в см. в разделе DECLARE), за которой следует один или несколько операторов, или блоков операторов, заключённых во внешнем блоке, который начинается с ключевого слова BEGIN, и завершается ключевым словом END. Объявления локальных переменных и именованных курсоров, а также внутренние операторы должны завершаться точкой с запятой (;).

Параметры

У каждого параметра указывается тип данных. Кроме того, для параметра можно указать ограничение NOT NULL, тем самым запретив передавать в него значение NULL.

Для параметра строкового типа существует возможность задать порядок сортировки с помощью предложения COLLATE.

Входные параметры

Входные параметры заключаются в скобки после имени хранимой процедуры. Они передаются в процедуру по значению, то есть любые изменения входных параметров внутри процедуры никак не повлияет на значения этих параметров в вызывающей программе.

Входные параметры могут иметь значение по умолчанию. Параметры, для которых заданы значения, должны располагаться в конце списка параметров.

Выходные параметры

Необязательное предложение RETURNS позволяет задать список выходных параметров хранимой процедуры.

Использование доменов при объявлении параметров

В качестве типа параметра можно указать имя домена. В этом случае, параметр будет наследовать все характеристики домена.

Если перед названием домена дополнительно используется предложение "TYPE OF", то используется только тип данных домена — не проверяется (не используется) его ограничение (если оно есть в домене) на NOT NULL, CHECK ограничения и/или значения по умолчанию. Если домен текстового типа, то всегда используется его набор символов и порядок сортировки.

Использование типа столбца при объявлении параметров

Входные и выходные параметры можно объявлять, используя тип данных столбцов существующих таблиц и представлений. Для этого используется предложение TYPE OF COLUMN, после которого указывается имя таблицы или представления и через точку имя столбца.

При использовании TYPE OF COLUMN наследуется только тип данных, а в случае строковых типов ещё и набор символов, и порядок сортировки. Ограничения и значения по умолчанию столбца никогда не используются.

Объявление локальных переменных, курсоров и подпрограмм

В необязательной секции `declarations` описаны локальные переменные процедуры, именованные курсоры и подпрограммы (подпроцедуры и подфункции). Локальные переменные подчиняются тем же правилам что и входные и выходные параметры процедуры в отношении спецификации типа данных. Подробнее см. в разделе DECLARE.

После заголовка следует тело хранимой процедуры, состоящее из одного или нескольких PSQL операторов, заключенных между ключевыми словами BEGIN и END. Внутри тела процедуры может быть множество BEGIN...END блоков, представляющих собой законченный оператор.

Внешние хранимые процедуры

Хранимая процедура может быть расположена во внешнем модуле. В этом случае вместо тела процедуры указывается место её расположения во внешнем модуле с помощью предложения EXTERNAL NAME. Аргументом этого предложения является строка, в которой через разделитель указано имя внешнего модуля, имя процедуры внутри модуля и определённая пользователем информация. В предложении ENGINE указывается имя движка для обработки подключения внешних модулей. В Firebird для работы с внешними модулями используется движок UDR.

Кто может создать хранимую процедуру?

Создать новую хранимую процедуру могут:

- SYSDBA;
- Владелец базы данных;
- Любой пользователь с привилегией на создание процедуры (GRANT CREATE PROCEDURE);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;

- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Пользователь, создавший хранимую процедуру, становится её владельцем.

Примеры

Пример 5.84. Создание хранимой процедуры

```
CREATE PROCEDURE ADD_BREED (
  NAME D_BREEDNAME, /* Наследуются характеристики домена */
  NAME_EN TYPE OF D_BREEDNAME, /* Наследуется только тип домена */
  SHORTNAME TYPE OF COLUMN BREED.SHORTNAME, /* Наследуется тип столбца таблицы */
  REMARK VARCHAR(120) CHARACTER SET WIN1251 COLLATE PXW_CYRL,
  CODE_ANIMAL INT NOT NULL DEFAULT 1
)
RETURNS (
  CODE_BREED INT
)
AS
BEGIN
  INSERT INTO BREED (
    CODE_ANIMAL, NAME, NAME_EN, SHORTNAME, REMARK)
  VALUES (
    :CODE_ANIMAL, :NAME, :NAME_EN, :SHORTNAME, :REMARK)
  RETURNING CODE_BREED INTO CODE_BREED;
END
```

Пример 5.85. Создание внешней хранимой процедуры

Создание процедуры находящейся во внешнем модуле (UDR). Реализация процедуры расположена во внешнем модуле udrcpp_example. Имя процедуры внутри модуля — gen_rows.

```
CREATE PROCEDURE gen_rows (
    start_n INTEGER NOT NULL,
    end_n INTEGER NOT NULL
) RETURNS (
    n INTEGER NOT NULL
)
    EXTERNAL NAME 'udrcpp_example!gen_rows'
    ENGINE udr;
```

См. также: CREATE OR ALTER PROCEDURE, ALTER PROCEDURE, RECREATE PROCEDURE, DROP PROCEDURE.

ALTER PROCEDURE

Назначение: Изменение существующей хранимой процедуры.

Доступно в: DSQL, ESQL.

Синтаксис:

```
ALTER PROCEDURE procname [(<inparam> [, <inparam> ...])]
RETURNS (<outparam> [, <outparam> ...])
  { EXTERNAL NAME '<extname>' ENGINE <engine> }  |
  { AS
      [<declarations>]
    BEGIN
      [<PSQL_statements>]
    END
  }
```

Подробнее см. CREATE PROCEDURE.

Оператор ALTER PROCEDURE позволяет изменять состав и характеристики входных и выходных параметров, локальных переменных, именованных курсоров и тело хранимой процедуры. Для внешних процедур (UDR) вы можете изменить точку входа и имя движка. После выполнения существующие привилегии и зависимости сохраняются.

> **Предупреждение**
>
> Будьте осторожны при изменении количества и типов входных и выходных параметров хранимых процедур. Существующий код приложения может стать неработоспособным из-за того, что формат вызова процедуры несовместим с новым описанием параметров. Кроме того, PSQL модули, использующие изменённую хранимую процедуру, могут стать некорректными. Информация о том, как это обнаружить, находится в приложении Поле RDB$VALID_BLR.

Кто может изменить хранимую процедуру?

Изменить хранимую процедуру могут:

- SYSDBA;
- Владелец базы данных;
- Владелец процедуры;
- Любой пользователь с привилегией на изменение любой процедуры (GRANT ALTER ANY PROCEDURE);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.86. Изменение хранимой процедуры

```
ALTER PROCEDURE GET_EMP_PROJ (
    EMP_NO SMALLINT)
RETURNS (
```

```
    PROJ_ID VARCHAR(20))
AS
BEGIN
  FOR SELECT
          PROJ_ID
      FROM
          EMPLOYEE_PROJECT
      WHERE
          EMP_NO = :emp_no
      INTO :proj_id
  DO
    SUSPEND;
END
```

См. также: CREATE PROCEDURE, CREATE OR ALTER PROCEDURE, RECREATE PROCEDURE, DROP PROCEDURE.

CREATE OR ALTER PROCEDURE

Назначение: Создание новой или изменение существующей хранимой процедуры.

Доступно в: DSQL, ESQL.

Синтаксис:

```
CREATE OR ALTER PROCEDURE procname [(<inparam> [, <inparam> ...])]
RETURNS (<outparam> [, <outparam> ...])
  { EXTERNAL NAME '<extname>' ENGINE <engine> }  |
  { AS
      [<declarations>]
    BEGIN
      [<PSQL_statements>]
    END
  }
```

Подробнее см. CREATE PROCEDURE.

Оператор CREATE OR ALTER PROCEDURE создаёт новую или изменяет существующую хранимую процедуру. Если хранимая процедура не существует, то она будет создана с использованием предложения CREATE PROCEDURE. Если она уже существует, то она будет изменена и откомпилирована, при этом существующие привилегии и зависимости сохраняются.

Примеры

Пример 5.87. Создание или изменение хранимой процедуры

```
CREATE OR ALTER PROCEDURE GET_EMP_PROJ (
    EMP_NO SMALLINT)
RETURNS (
    PROJ_ID VARCHAR(20))
```

```
AS
BEGIN
  FOR SELECT
          PROJ_ID
      FROM
          EMPLOYEE_PROJECT
      WHERE
          EMP_NO = :emp_no
      INTO :proj_id
  DO
    SUSPEND;
END
```

См. также: CREATE PROCEDURE, ALTER PROCEDURE, RECREATE PROCEDURE, DROP PROCEDURE.

DROP PROCEDURE

Назначение: Удаление существующей хранимой процедуры.

Доступно в: DSQL, ESQL.

Синтаксис:

```
DROP PROCEDURE procname
```

Параметры оператора DROP PROCEDURE

procname
Имя хранимой процедуры.

Оператор DROP PROCEDURE удаляет существующую хранимую процедуру. Если от хранимой процедуры существуют зависимости, то при попытке удаления такой процедуру будет выдана соответствующая ошибка.

Кто может удалить хранимую процедуру?

Удалить хранимую процедуру могут:

- SYSDBA;
- Владелец базы данных;
- Владелец процедуры;
- Любой пользователь с привилегией на удаление любой процедуры (GRANT DROP ANY PROCEDURE);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.88. Удаление хранимой процедуры

```
DROP PROCEDURE GET_EMP_PROJ;
```

См. также: CREATE PROCEDURE, RECREATE PROCEDURE.

RECREATE PROCEDURE

Назначение: Создание новой или пересоздание существующей хранимой процедуры.

Доступно в: DSQL, ESQL.

Синтаксис:

```
RECREATE PROCEDURE procname [(<inparam> [, <inparam> ...])]
RETURNS (<outparam> [, <outparam> ...])
  { EXTERNAL NAME '<extname>' ENGINE <engine> }  |
  { AS
      [<declarations>]
    BEGIN
      [<PSQL_statements>]
    END
  }
```

Подробнее см. CREATE PROCEDURE.

Оператор RECREATE PROCEDURE создаёт новую или пересоздаёт существующую хранимую процедуру. Если процедура с таким именем уже существует, то оператор попытается удалить её и создать новую процедуру. Пересоздать процедуру невозможно, если у существующей процедуры имеются зависимости. После пересоздания процедуры привилегии на выполнение хранимой процедуры и привилегии самой хранимой процедуры не сохраняются.

Примеры

Пример 5.89. Создание новой или пересоздание существующей хранимой процедуры

```
RECREATE PROCEDURE GET_EMP_PROJ (
    EMP_NO SMALLINT)
RETURNS (
    PROJ_ID VARCHAR(20))
AS
BEGIN
  FOR SELECT
          PROJ_ID
      FROM
          EMPLOYEE_PROJECT
      WHERE
          EMP_NO = :emp_no
```

```
      INTO :proj_id
  DO
    SUSPEND;
END
```

См. также: CREATE PROCEDURE, CREATE OR ALTER PROCEDURE, DROP PROCEDURE.

FUNCTION

Хранимая функция является программой, хранящейся в области метаданных базы данных и выполняющейся на стороне сервера. К хранимой функции могут обращаться хранимые процедуры, хранимые функции (в том числе и сама к себе), триггеры и клиентские программы. При обращении хранимой функции самой к себе такая хранимая функция называется рекурсивной.

В отличие от хранимых процедур хранимые функции всегда возвращают одно скалярное значение. Для возврата значения из хранимой функции используется оператор RETURN, который немедленно прекращает выполнение функции.

CREATE FUNCTION

Назначение: Создание новой хранимой функции.

Доступно в: DSQL.

Синтаксис:

```
CREATE FUNCTION funcname [(<inparam> [, <inparam> ...])]
RETURNS <type> [COLLATE collation] [DETERMINISTIC]
  { EXTERNAL NAME '<extname>' ENGINE <engine> }  |
  { AS
      [<declarations>]
    BEGIN
      [<PSQL_statements>]
    END
  }

<inparam> ::= <param_decl> [{= | DEFAULT} <value>]

<value> ::=  {literal | NULL | context_var}

<param_decl> ::= paramname <type> [NOT NULL] [COLLATE collation]

<extname> ::= '<module name>!<routine name>[!<misc info>]'

<type> ::= <datatype> | [TYPE OF] domain | TYPE OF COLUMN rel.col

<datatype> ::=
    {SMALLINT | INTEGER | BIGINT}
  | BOOLEAN
  | {FLOAT | DOUBLE PRECISION}
```

```
  | {DATE | TIME | TIMESTAMP}
  | {DECIMAL | NUMERIC} [(precision [, scale])]
  | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(size)]
    [CHARACTER SET charset]
  | {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR} [VARYING] [(size)]
  | BLOB [SUB_TYPE {subtype_num | subtype_name}]
    [SEGMENT SIZE seglen] [CHARACTER SET charset]
  | BLOB [(seglen [, subtype_num])]

<declarations> ::= <declare_item> [<declare_item> ...]

<declare_item> ::=
  <declare_var>; |
  <declare_cursor>; |
  <declare_subfunc> |
  <declare_subproc>
```

Параметры оператора CREATE FUNCTION

funcname
: Имя хранимой функции. Может содержать до 31 байта.

inparam
: Описание входного параметра.

declarations
: Секция объявления локальных переменных, именованных курсоров, подпроцедур и подфункций.

declare_var
: Объявление локальной переменной.

declare_cursor
: Объявление именованного курсора.

declare_subfunc
: Объявление подпрограммы – функции.

declare_subproc
: Объявление подпрограммы – процедуры.

literal
: Литерал.

context_var
: Любая контекстная переменная, тип которой совместим с типом параметра.

paramname
: Имя входного параметра функции. Может содержать до 31 байта. Имя параметра должно быть уникальным среди входных параметров функции, а также её локальных переменных.

module name
: Имя внешнего модуля, в котором расположена функция.

routine name
: Внутреннее имя функции внутри внешнего модуля.

misc info
Определяемая пользователем информация для передачи в функцию внешнего модуля.

engine
Имя движка для использования внешних функций. Обычно указывается имя UDR.

datatype
Тип данных SQL.

collation
Порядок сортировки.

domain
Домен.

rel
Имя таблицы или представления.

col
Имя столбца таблицы или представления.

precision
Точность. От 1 до 18.

scale
Масштаб. От 0 до 18, должно быть меньше или равно *precision*.

size
Максимальный размер строки в символах.

charset
Набор символов.

subtype_num
Номер подтипа BLOB.

subtype_name
Мнемоника подтипа BLOB.

seglen
Размер сегмента, не может превышать 65535.

Оператор CREATE FUNCTION создаёт новую хранимую функцию. Имя хранимой функции должно быть уникальным среди имён всех хранимых функций и внешних (UDF) функций. Если только это не внутренняя функция («подпрограмма»). Для внутренних функций достаточно уникальности только в рамках модулей, которые их «охватывают».

Примечание

Желательно также, чтобы имя хранимой функции было уникальным и среди имён функций расположенных в PSQL пакетах (package), хотя это и допустимо. Дело в том, что в настоящее время вы не сможете вызвать функцию/процедуру из глобального пространства имён внутри пакета, если в пакете объявлена одноименная функция/процедура. В этом случае всегда будет вызвана процедура/функция пакета.

CREATE FUNCTION является составным оператором, состоящий из заголовка и тела. Заголовок определяет имя хранимой функции, объявляет входные параметры и тип возвращаемого значения.

Тело функции состоит из секции объявления локальных переменных, именованных курсоров и подпрограмм (подробнее в см. в разделе DECLARE), за которой следует один или несколько операторов, или блоков операторов, заключённых во внешнем блоке, который начинается с ключевого слова BEGIN, и завершается ключевым словом END. Объявления локальных переменных и именованных курсоров, а также внутренние операторы должны завершаться точкой с запятой (;).

Входные параметры

Входные параметры заключаются в скобки после имени хранимой функции. Они передаются в функцию по значению, то есть любые изменения входных параметров внутри функции никак не повлияет на значения этих параметров в вызывающей программе.

У каждого параметра указывается тип данных. Кроме того, для параметра можно указать ограничение NOT NULL, тем самым запретив передавать в него значение NULL.

Для параметра строкового типа существует возможность задать порядок сортировки с помощью предложения COLLATE.

Входные параметры могут иметь значение по умолчанию. Параметры, для которых заданы значения, должны располагаться в конце списка параметров.

Использование доменов при объявлении параметров

В качестве типа параметра можно указать имя домена. В этом случае, параметр будет наследовать все характеристики домена.

Если перед названием домена дополнительно используется предложение "TYPE OF", то используется только тип данных домена — не проверяется (не используется) его ограничение (если оно есть в домене) на NOT NULL, CHECK ограничения и/или значения по умолчанию. Если домен текстового типа, то всегда используется его набор символов и порядок сортировки.

Использование типа столбца при объявлении параметров

Входные и выходные параметры можно объявлять, используя тип данных столбцов существующих таблиц и представлений. Для этого используется предложение TYPE OF COLUMN, после которого указывается имя таблицы или представления и через точку имя столбца.

При использовании TYPE OF COLUMN наследуется только тип данных, а в случае строковых типов ещё и набор символов, и порядок сортировки. Ограничения и значения по умолчанию столбца никогда не используются.

Возвращаемое значение

Предложение RETURNS задаёт тип возвращаемого значения хранимой функции. Если функция возвращает значение строкового типа, то существует возможность задать порядок сортировки с помощью предложения COLLATE. В качестве типа выходного значения можно

указать имя домена, ссылку на его тип (с помощью предложения TYPE OF) или ссылку на тип столбца таблицы (с помощью предложения TYPE OF COLUMN).

Детерминированные функции

Необязательное предложение DETERMINISTIC указывает, что функция детерминированная. Детерминированные функции каждый раз возвращают один и тот же результат, если предоставлять им один и тот же набор входных значений. Недетерминированные функции могут возвращать каждый раз разные результаты, даже если предоставлять им один и тот же набор входных значений. Если для функции указано, что она является детерминированной, то такая функция не вычисляется заново, если она уже была вычислена однажды с данным набором входных аргументов, а берет свои значения из кэша метаданных (если они там есть).

Примечание

На самом деле в текущей версии Firebird, не существует кэша хранимых функций с маппингом входных аргументов на выходные значения.

Указание инструкции "DETERMINISTIC" на самом деле нечто вроде «обещания», что код функции будет возвращать одно и то же. В данный момент детерминистическая функция считается инвариантом и работает по тем же принципам, что и другие инварианты. Т.е. вычисляется и кэшируется на уровне текущего выполнения данного запроса.

Это легко демонстрируется таким примером:

```
CREATE FUNCTION FN_T
RETURNS DOUBLE PRECISION DETERMINISTIC
AS
BEGIN
  RETURN rand();
END

-- функция будет вычислена дважды и вернёт 2 разных значения
SELECT fn_t() FROM rdb$database
UNION ALL
SELECT fn_t() FROM rdb$database

-- функция будет вычислена единожды и вернёт 2 одинаковых значения
WITH t(n) AS (
  SELECT 1 FROM rdb$database
  UNION ALL
  SELECT 2 FROM rdb$database
)
SELECT n, fn_t() FROM t
```

Объявление локальных переменных, курсоров и подпрограмм

В необязательной секции *declarations* описаны локальные переменные процедуры, именованные курсоры и подпрограммы (подпроцедуры и подфункции). Локальные переменные подчиняются тем же правилам что и входные и выходные параметры процедуры в отношении спецификации типа данных. Подробнее см. в разделе DECLARE.

После заголовка следует тело хранимой функции, состоящее из одного или нескольких PSQL операторов, заключенных между ключевыми словами BEGIN и END. Внутри тела процедуры может быть множество BEGIN...END блоков, представляющих собой законченный оператор.

Внешние функции

Хранимая функция может быть расположена во внешнем модуле. В этом случае вместо тела функции указывается место расположения функции во внешнем модуле с помощью предложения EXTERNAL NAME. Аргументом этого предложения является строка, в которой через разделитель указано имя внешнего модуля, имя функции внутри модуля и определённая пользователем информация. В предложении ENGINE указывается имя движка для обработки подключения внешних модулей. В Firebird для работы с внешними модулями используется движок UDR.

> **Предупреждение**
>
> Не следует путать внешние функции, объявленные как DECLARE EXTERNAL FUNCTION, так же известные как UDF, с функциями расположенными во внешних модулях объявленных как CREATE FUNCTION ... EXTERNAL NAME, называемых UDR (User Defined Routine). Первые являются унаследованными (Legacy) из предыдущих версий Firebird. Их возможности существенно уступают возможностям нового типа внешних функций.

Кто может создать функцию?

Создать новую хранимую функцию могут:

- SYSDBA;
- Владелец базы данных;
- Любой пользователь с привилегией на создание функции (GRANT CREATE FUNCTION);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Пользователь, создавший хранимую функцию, становится её владельцем.

Примеры

Пример 5.90. Создание хранимой функции

```
CREATE FUNCTION ADD_INT(A INT, B INT DEFAULT 0)
RETURNS INT
AS
BEGIN
  RETURN A+B;
END
```

Вызов в запросе:

```
SELECT ADD_INT(2, 3) AS R FROM RDB$DATABASE
```

Вызов внутри PSQL кода, второй необязательный параметр не указан:

```
MY_VAR = ADD_INT(A);
```

Пример 5.91. Создание детерминистической хранимой функции

```
CREATE FUNCTION FN_E()
RETURNS DOUBLE PRECISION DETERMINISTIC
AS
BEGIN
  RETURN EXP(1);
END
```

Пример 5.92. Создание хранимой функции с параметрами типа столбца таблицы

Функция, возвращающая имя мнемоники по имени столбца и значения мнемоники.

```
CREATE FUNCTION GET_MNEMONIC (
    AFIELD_NAME TYPE OF COLUMN RDB$TYPES.RDB$FIELD_NAME,
    ATYPE TYPE OF COLUMN RDB$TYPES.RDB$TYPE)
RETURNS TYPE OF COLUMN RDB$TYPES.RDB$TYPE_NAME
AS
BEGIN
  RETURN (SELECT RDB$TYPE_NAME
          FROM RDB$TYPES
          WHERE RDB$FIELD_NAME = :AFIELD_NAME
            AND RDB$TYPE = :ATYPE);
END
```

Пример 5.93. Создание внешней хранимой функции

Создание функции находящейся во внешнем модуле (UDR). Реализация функции расположена во внешнем модуле udrcpp_example. Имя функции внутри модуля — wait_event.

```
CREATE FUNCTION wait_event (
   event_name varchar(31) CHARACTER SET ascii
) RETURNS INTEGER
EXTERNAL NAME 'udrcpp_example!wait_event'
ENGINE udr
```

Пример 5.94. Создание хранимой функции содержащую подфункцию

Создание функции для перевода числа в 16-ричный формат.

```
CREATE FUNCTION INT_TO_HEX (
    ANumber BIGINT,
    AByte_Per_Number SMALLINT = 8)
RETURNS CHAR(66)
```

```
AS
DECLARE VARIABLE xMod SMALLINT;
DECLARE VARIABLE xResult VARCHAR(64);
DECLARE FUNCTION TO_HEX(ANum SMALLINT) RETURNS CHAR
AS
BEGIN
  RETURN CASE ANum
           WHEN 0 THEN '0'
           WHEN 1 THEN '1'
           WHEN 2 THEN '2'
           WHEN 3 THEN '3'
           WHEN 4 THEN '4'
           WHEN 5 THEN '5'
           WHEN 6 THEN '6'
           WHEN 7 THEN '7'
           WHEN 8 THEN '8'
           WHEN 9 THEN '9'
           WHEN 10 THEN 'A'
           WHEN 11 THEN 'B'
           WHEN 12 THEN 'C'
           WHEN 13 THEN 'D'
           WHEN 14 THEN 'E'
           WHEN 15 THEN 'F'
           ELSE NULL
         END;
END
BEGIN
  xMod = MOD(ANumber, 16);
  ANumber = ANumber / 16;
  xResult = TO_HEX(xMod);
  WHILE (ANUMBER > 0) DO
  BEGIN
    xMod = MOD(ANumber, 16);
    ANumber = ANumber / 16;
    xResult = TO_HEX(xMod) || xResult;
  END
  RETURN '0x' || LPAD(xResult, AByte_Per_Number * 2, '0');
END
```

См. также: CREATE OR ALTER FUNCTION, ALTER FUNCTION, RECREATE FUNCTION, DROP FUNCTION.

ALTER FUNCTION

Назначение: Изменение существующей хранимой функции.

Доступно в: DSQL.

Синтаксис:

```
ALTER FUNCTION funcname [(<inparam> [, <inparam> ...])]
RETURNS <type> [COLLATE collation] [DETERMINISTIC]
  { EXTERNAL NAME '<extname>' ENGINE <engine> }  |
```

```
  { AS
      [<declarations>]
    BEGIN
      [<PSQL_statements>]
    END
  }
```

Подробнее см. CREATE FUNCTION.

Оператор ALTER FUNCTION позволяет изменять состав и характеристики входных параметров, типа выходного значения, локальных переменных, именованных курсоров, подпрограмм и тело хранимой функции. Для внешних функций (UDR) вы можете изменить точку входа и имя движка. Внешние функции, объявленные как DECLARE EXTERNAL FUNCTION, так же известные как UDF, невозможно преобразовать в PSQL функции и наоборот. После выполнения существующие привилегии и зависимости сохраняются.

> **Примечание**
>
> Будьте осторожны при изменении количества и типов входных параметров хранимых функций. Существующий код приложения может стать неработоспособным из-за того, что формат вызова функции несовместим с новым описанием параметров. Кроме того, PSQL модули, использующие изменённую хранимую функцию, могут стать некорректными. Информация о том, как это обнаружить, находится в приложении Поле RDB$VALID_BLR.

Кто может изменить функцию?

Изменить хранимую функцию могут:

- SYSDBA;
- Владелец базы данных;
- Владелец функции;
- Любой пользователь с привилегией на изменение любой функции (GRANT ALTER ANY FUNCTION);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.95. Изменение хранимой функции

```
ALTER FUNCTION ADD_INT(A INT, B INT, C INT)
RETURNS INT
AS
BEGIN
  RETURN A+B+C;
END
```

См. также: CREATE FUNCTION, CREATE OR ALTER FUNCTION, DROP FUNCTION.

CREATE OR ALTER FUNCTION

Назначение: Создание новой или изменение существующей хранимой функции.

Доступно в: DSQL.

Синтаксис:

```
CREATE OR ALTER FUNCTION funcname [(<inparam> [, <inparam> ...])]
RETURNS <type> [COLLATE collation] [DETERMINISTIC]
  { EXTERNAL NAME '<extname>' ENGINE <engine> }  |
  { AS
      [<declarations>]
    BEGIN
      [<PSQL_statements>]
    END
  }
```

Подробнее см. CREATE FUNCTION.

Оператор CREATE OR ALTER FUNCTION создаёт новую или изменяет существующую хранимую функцию. Если хранимая функция не существует, то она будет создана с использованием предложения CREATE FUNCTION. Если она уже существует, то она будет изменена и перекомпилирована, при этом существующие привилегии и зависимости сохраняются.

Примеры

Пример 5.96. Создание новой или изменение существующей хранимой функции

```
CREATE OR ALTER FUNCTION ADD_INT(A INT, B INT DEFAULT 0)
RETURNS INT
AS
BEGIN
  RETURN A+B;
END
```

См. также: CREATE FUNCTION, ALTER FUNCTION.

DROP FUNCTION

Назначение: Удаление хранимой функции.

Доступно в: DSQL.

Синтаксис:

```
DROP FUNCTION funcname
```

Параметры оператора DROP FUNCTION

funcname
Имя хранимой функции.

Оператор DROP FUNCTION удаляет существующую хранимую функцию. Если от хранимой функции существуют зависимости, то при попытке удаления такой функции будет выдана соответствующая ошибка.

Кто может удалить функцию?

Удалить хранимую функцию могут:

- SYSDBA;
- Владелец базы данных;
- Владелец функции;
- Любой пользователь с привилегией на удаление любой функции (GRANT DROP ANY FUNCTION);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.97. Удаление хранимой функции

```
DROP FUNCTION ADD_INT;
```

См. также: CREATE FUNCTION.

RECREATE FUNCTION

Назначение: Создание новой или пересоздание существующей хранимой функции.

Доступно в: DSQL.

Синтаксис:

```
RECREATE FUNCTION funcname [(<inparam> [, <inparam> ...])]
RETURNS <type> [COLLATE collation] [DETERMINISTIC]
  { EXTERNAL NAME '<extname>' ENGINE <engine> }  |
  { AS
```

```
      [<declarations>]
    BEGIN
      [<PSQL_statements>]
    END
  }
```

Подробнее см. CREATE FUNCTION.

Описание:

Оператор RECREATE FUNCTION создаёт новую или пересоздаёт существующую хранимую функцию. Если функция с таким именем уже существует, то оператор попытается удалить её и создать новую функцию. Пересоздать функцию невозможно, если у существующей функции имеются зависимости. После пересоздания функции привилегии на выполнение хранимой функции и привилегии самой хранимой функции не сохраняются.

Примеры

Пример 5.98. Создание или пересоздание хранимой функции

```
RECREATE FUNCTION ADD_INT(A INT, B INT DEFAULT 0)
RETURNS INT
AS
BEGIN
  RETURN A+B;
END
```

См. также: CREATE FUNCTION, DROP FUNCTION.

PACKAGE

Пакет — группа процедур и функций, которая представляет собой один объект базы данных.

Пакеты Firebird состоят из двух частей: заголовка (ключевое слово PACKAGE) и тела (ключевые слова PACKAGE BODY). Такое разделение очень сильно напоминает модули Delphi, заголовок соответствует интерфейсной части, а тело — части реализации.

Сначала создаётся заголовок (CREATE PACKAGE), а затем — тело (CREATE PACKAGE BODY).

PACKAGE

CREATE PACKAGE

Назначение: Создание заголовка пакета.

Доступно в: DSQL.

Синтаксис:

```
CREATE PACKAGE package_name
AS
BEGIN
  [<package_item> ...]
END

<package_item> ::=
    <function_decl>;
  | <procedure_decl>;

<function_decl> ::=
  FUNCTION func_name [(<in_params>)]
    RETURNS <type> [COLLATE collation] [DETERMINISTIC]

<procedure_decl> ::=
  PROCEDURE proc_name [(<in_params>)] [RETURNS (<out_params>)]

<in_params> ::= <inparam> [, <inparam> ...]

<inparam> ::= <param_decl> [{= | DEFAULT} <value>]

<value> ::=  {literal | NULL | context_var}

<out_params> ::= <outparam> [, <outparam> ...]

<outparam>  ::=  <param_decl>

<param_decl> ::= paramname <type> [NOT NULL] [COLLATE collation]

<type> ::= <datatype> | [TYPE OF] domain | TYPE OF COLUMN rel.col

<datatype> ::=
    {SMALLINT | INTEGER | BIGINT}
  | BOOLEAN
  | {FLOAT | DOUBLE PRECISION}
  | {DATE | TIME | TIMESTAMP}
  | {DECIMAL | NUMERIC} [(precision [, scale])]
  | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(size)]
    [CHARACTER SET charset]
  | {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR} [VARYING] [(size)]
  | BLOB [SUB_TYPE {subtype_num | subtype_name}]
    [SEGMENT SIZE seglen] [CHARACTER SET charset]
  | BLOB [(seglen [, subtype_num])]
```

Параметры оператора CREATE PACKAGE

package_name
Имя пакета. Может содержать до 31 байта.

function_decl
Объявление функции.

procedure_decl
Объявление процедуры.

proc_name
Имя процедуры. Может содержать до 31 байта.

func_name
Имя функции. Может содержать до 31 байта.

inparam
Описание входного параметра.

outparam
Описание выходного параметра.

literal
Литерал.

context_var
Любая контекстная переменная, тип которой совместим с типом параметра.

paramname
Имя входного или выходного параметра процедуры/функции. Может содержать до 31 символа. Имя параметра должно быть уникальным среди входных и выходных параметров процедуры/функции, а также её локальных переменных.

datatype
Тип данных SQL.

collation
Порядок сортировки.

domain
Домен.

rel
Имя таблицы или представления.

col
Имя столбца таблицы или представления.

precision
Точность. От 1 до 18.

scale
Масштаб. От 0 до 18, должно быть меньше или равно *precision*.

size
Максимальный размер строки в символах.

charset
Набор символов.

subtype_num
Номер подтипа BLOB.

subtype_name
Мнемоника подтипа BLOB.

seqlen
Размер сегмента, не может превышать 65535.

Оператор CREATE PACKAGE создаёт новый заголовок пакета. Имя пакета должно быть уникальным среди имён всех пакетов.

Процедуры и функции, объявленные в заголовке пакета, доступны вне тела пакета через полный идентификатор имён процедур и функций (*package_name.procedure_name* и *package_name.function_name*). Процедуры и функции, определенные в теле пакета, но не объявленные в заголовке пакета, не видны вне тела пакета.

Имена процедур и функций, объявленные в заголовке пакета, должны быть уникальны среди имён процедур и функций, объявленных в заголовке и теле пакета.

> **Примечание**
>
> Желательно чтобы имена хранимых процедур и функций пакета не пересекались с именами хранимых процедур и функций из глобального пространства имен, хотя это и допустимо. Дело в том, что в настоящее время вы не сможете вызвать функцию/процедуру из глобального пространства имён внутри пакета, если в пакете объявлена одноименная функция/процедура. В этом случае всегда будет вызвана процедура/функция пакета.

Параметры процедур и функций

У каждого параметра указывается тип данных. Кроме того, для параметра можно указать ограничение NOT NULL, тем самым запретив передавать в него значение NULL.

Для параметра строкового типа существует возможность задать порядок сортировки с помощью предложения COLLATE.

Входные параметры

Входные параметры заключаются в скобки после имени хранимой процедуры. Они передаются в процедуру по значению, то есть любые изменения входных параметров внутри процедуры никак не повлияет на значения этих параметров в вызывающей программе.

Входные параметры могут иметь значение по умолчанию. Параметры, для которых заданы значения, должны располагаться в конце списка параметров.

Выходные параметры

Для хранимых процедур список выходных параметров задаётся в необязательное предложение RETURNS.

Для хранимых функций в обязательном предложении RETURNS задаётся тип возвращаемого значения.

Использование доменов при объявлении параметров

В качестве типа параметра можно указать имя домена. В этом случае, параметр будет наследовать все характеристики домена.

Если перед названием домена дополнительно используется предложение "TYPE OF", то используется только тип данных домена — не проверяется (не используется) его ограничение

(если оно есть в домене) на NOT NULL, CHECK ограничения и/или значения по умолчанию. Если домен текстового типа, то всегда используется его набор символов и порядок сортировки.

Использование типа столбца при объявлении параметров

Входные и выходные параметры можно объявлять, используя тип данных столбцов существующих таблиц и представлений. Для этого используется предложение TYPE OF COLUMN, после которого указывается имя таблицы или представления и через точку имя столбца.

При использовании TYPE OF COLUMN наследуется только тип данных, а в случае строковых типов ещё и набор символов, и порядок сортировки. Ограничения и значения по умолчанию столбца никогда не используются.

Детерминированные функции

Необязательное предложение DETERMINISTIC в объявлении функции указывает, что функция детерминированная. Детерминированные функции каждый раз возвращают один и тот же результат, если предоставлять им один и тот же набор входных значений. Недетерминированные функции могут возвращать каждый раз разные результаты, даже если предоставлять им один и тот же набор входных значений. Если для функции указано, что она является детерминированной, то такая функция не вычисляется заново, если она уже была вычислена однажды с данным набором входных аргументов, а берет свои значения из кэша метаданных (если они там есть).

> **Примечание**
>
> На самом деле в текущей версии Firebird, не существует кэша хранимых функций с маппингом входных аргументов на выходные значения.
>
> Указание инструкции "deterministic" на самом деле нечто вроде "обещания", что код функции будет возвращать одно и то же. В данный момент детерминистическая функция считается инвариантом и работает по тем же принципам, что и другие инварианты. Т.е. вычисляется и кэшируется на уровне текущего выполнения данного запроса.

Кто может создать пакет?

Создать новый заголовок пакета могут:

- SYSDBA;
- Владелец базы данных;
- Любой пользователь с привилегией на создание пакета (GRANT CREATE PACKAGE);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Пользователь, создавший заголовок пакета становится владельцем пакета.

Примеры

Пример 5.99. Создание заголовка пакета

```
CREATE PACKAGE APP_VAR
AS
BEGIN
  FUNCTION GET_DATEBEGIN() RETURNS DATE DETERMINISTIC;
  FUNCTION GET_DATEEND() RETURNS DATE DETERMINISTIC;
  PROCEDURE SET_DATERANGE(ADATEBEGIN DATE, ADATEEND DATE DEFAULT CURRENT_DATE);
END
```

См. также: CREATE PACKAGE BODY, ALTER PACKAGE, DROP PACKAGE.

ALTER PACKAGE

Назначение: Изменение заголовка пакета.

Доступно в: DSQL.

Синтаксис:

```
ALTER PACKAGE package_name
AS
BEGIN
  [<package_item> ...]
END

<package_item> ::=
    <function_decl>;
  | <procedure_decl>;

<function_decl> ::=
  FUNCTION func_name [(<in_params>)]
    RETURNS <type> [COLLATE collation] [DETERMINISTIC]

<procedure_decl> ::=
  PROCEDURE proc_name [(<in_params>)] [RETURNS (<out_params>)]
```

Подробнее см. CREATE PACKAGE.

Оператор ALTER PACKAGE изменяет заголовок пакета. Позволяется изменять количество и состав процедур и функций, их входных и выходных параметров. При этом исходный код тела пакета сохраняется. Состояние соответствия тела пакета его заголовку отображается в столбце RDB$PACKAGES.RDB$VALID_BODY_FLAG.

Кто может изменить заголовок пакета?

Изменить заголовок пакета могут:

- SYSDBA;
- Владелец базы данных;
- Владелец пакета;
- Любой пользователь с привилегией на изменение любого пакета (GRANT ALTER ANY PACKAGE);

- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.100. Изменение заголовка пакета

```
ALTER PACKAGE APP_VAR
AS
BEGIN
  FUNCTION GET_DATEBEGIN() RETURNS DATE DETERMINISTIC;
  FUNCTION GET_DATEEND() RETURNS DATE DETERMINISTIC;
  PROCEDURE SET_DATERANGE(ADATEBEGIN DATE, ADATEEND DATE DEFAULT CURRENT_DATE);
END
```

См. также: CREATE PACKAGE, DROP PACKAGE, ALTER PACKAGE BODY.

CREATE OR ALTER PACKAGE

Назначение: Создание нового или изменение существующего заголовка пакета.

Доступно в: DSQL.

Синтаксис:

```
CREATE OR ALTER PACKAGE package_name
AS
BEGIN
  [<package_item> ...]
END

<package_item> ::=
    <function_decl>;
  | <procedure_decl>;

<function_decl> ::=
  FUNCTION func_name [(<in_params>)]
    RETURNS <type> [COLLATE collation] [DETERMINISTIC]

<procedure_decl> ::=
  PROCEDURE proc_name [(<in_params>)] [RETURNS (<out_params>)]
```

Подробнее см. CREATE PACKAGE.

Оператор CREATE OR ALTER PACKAGE создаёт новый или изменяет существующий заголовок пакета. Если заголовок пакета не существует, то он будет создан с использованием предложения CREATE PACKAGE. Если он уже существует, то он будет изменен и перекомпилирован, при этом существующие привилегии и зависимости сохраняются.

Примеры

Пример 5.101. Создание нового или изменение существующего заголовка пакета

```
CREATE OR ALTER PACKAGE APP_VAR
AS
BEGIN
  FUNCTION GET_DATEBEGIN() RETURNS DATE DETERMINISTIC;
  FUNCTION GET_DATEEND() RETURNS DATE DETERMINISTIC;
  PROCEDURE SET_DATERANGE(ADATEBEGIN DATE, ADATEEND DATE DEFAULT CURRENT_DATE);
END
```

См. также: CREATE PACKAGE, ALTER PACKAGE, ALTER PACKAGE BODY.

DROP PACKAGE

Назначение: Удаление заголовка пакета.

Доступно в: DSQL.

Синтаксис:

```
DROP PACKAGE package_name
```

Параметры оператора DROP PACKAGE

package_name
Имя пакета.

Оператор DROP PACKAGE удаляет существующий заголовок пакета. Перед удалением заголовка пакета (DROP PACKAGE), необходимо выполнить удаление тела пакета (DROP PACKAGE BODY), иначе будет выдана ошибка. Если от заголовка пакета существуют зависимости, то при попытке удаления такого заголовка будет выдана соответствующая ошибка.

Кто может удалить заголовок пакета?

Удалить заголовок пакета могут:

- SYSDBA;
- Владелец базы данных;
- Владелец пакета;
- Любой пользователь с привилегией на удаление любого пакета (GRANT DROP ANY PACKAGE);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.102. Удаление заголовка пакета

```
DROP PACKAGE APP_VAR;
```

См. также: CREATE PACKAGE, ALTER PACKAGE, DROP PACKAGE BODY.

RECREATE PACKAGE

Назначение: Создание нового или пересоздание существующего заголовка пакета.

Доступно в: DSQL.

Синтаксис:

```
RECREATE PACKAGE package_name
AS
BEGIN
  [<package_item> ...]
END

<package_item> ::=
    <function_decl>;
  | <procedure_decl>;

<function_decl> ::=
  FUNCTION func_name [(<in_params>)]
    RETURNS <type> [COLLATE collation] [DETERMINISTIC]

<procedure_decl> ::=
  PROCEDURE proc_name [(<in_params>)] [RETURNS (<out_params>)]
```

Подробнее см. CREATE PACKAGE.

Оператор RECREATE PACKAGE создаёт новый или пересоздаёт существующий заголовок пакета. Если заголовок пакета с таким именем уже существует, то оператор попытается удалить его и создать новый заголовок пакета. Пересоздать заголовок пакета невозможно, если у существующей заголовка пакета имеются зависимости или существует тело этого пакета. После пересоздания заголовка пакета привилегии на выполнение подпрограмм пакета и привилегии самого пакета не сохраняются.

Примеры

Пример 5.103. Создание нового или пересоздание существующего заголовка пакета

```
RECREATE PACKAGE APP_VAR
AS
BEGIN
  FUNCTION GET_DATEBEGIN() RETURNS DATE DETERMINISTIC;
```

```
  FUNCTION GET_DATEEND() RETURNS DATE DETERMINISTIC;
  PROCEDURE SET_DATERANGE(ADATEBEGIN DATE, ADATEEND DATE DEFAULT CURRENT_DATE);
END
```

См. также: CREATE PACKAGE, DROP PACKAGE, RECREATE PACKAGE BODY.

PACKAGE BODY

CREATE PACKAGE BODY

Назначение: Создание тела пакета.

Доступно в: DSQL.

Синтаксис:

```
CREATE PACKAGE BODY package_name
AS
BEGIN
  [<package_item> ...]
  [<package_body_item> ...]
END

<package_item> ::=
    <function_decl>;
  | <procedure_decl>;

<function_decl> ::=
  FUNCTION func_name [(<in_params>)]
    RETURNS <type> [COLLATE collation] [DETERMINISTIC]

<procedure_decl> ::=
  PROCEDURE proc_name [(<in_params>)] [RETURNS (<out_params>)]

<package_body_item> ::=
    <function_impl>
  | <procedure_impl>

<function_impl> ::=
  FUNCTION func_name [(<in_impl_params>)]
    RETURNS <type> [COLLATE collation] [DETERMINISTIC]
      { EXTERNAL NAME '<extname>' ENGINE <engine> }  |
      { AS
          [<declarations>]
        BEGIN
          [<PSQL_statements>]
        END
      }

<procedure_impl> ::=
  PROCEDURE proc_name [(<in_impl_params>)] [RETURNS (<out_params>)]
      { EXTERNAL NAME '<extname>' ENGINE <engine> }  |
      { AS
          [<declarations>]
```

```
        BEGIN
          [<PSQL_statements>]
        END
      }

<in_params> ::= <inparam> [, <inparam> ...]

<inparam> ::= <param_decl> [{= | DEFAULT} <value>]

<in_impl_params> ::= <param_decl> [, <param_decl> ...]

<value> ::=  {literal | NULL | context_var}

<out_params> ::= <outparam> [, <outparam> ...]

<outparam>  ::=  <param_decl>

<param_decl> ::= paramname <type> [NOT NULL] [COLLATE collation]

<type> ::= <datatype> | [TYPE OF] domain | TYPE OF COLUMN rel.col

<datatype> ::=
    {SMALLINT | INTEGER | BIGINT}
  | BOOLEAN
  | {FLOAT | DOUBLE PRECISION}
  | {DATE | TIME | TIMESTAMP}
  | {DECIMAL | NUMERIC} [(precision [, scale])]
  | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(size)]
    [CHARACTER SET charset]
  | {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR} [VARYING] [(size)]
  | BLOB [SUB_TYPE {subtype_num | subtype_name}]
    [SEGMENT SIZE seglen] [CHARACTER SET charset]
  | BLOB [(seglen [, subtype_num])]

<extname> ::= '<module name>!<routine name>[!<misc info>]'

<declarations> ::= <declare_item> [<declare_item> ...]

<declare_item> ::=
  <declare_var>; |
  <declare_cursor>; |
  <declare_subfunc> |
  <declare_subproc>
```

Параметры оператора CREATE PACKAGE BODY

package_name
Имя пакета. Может содержать до 31 байта.

function_decl
Объявление функции.

procedure_decl
Объявление процедуры.

function_impl
Реализация функции.

procedure_impl
Реализация процедуры.

proc_name
Имя процедуры. Может содержать до 31 байта.

func_name
Имя функции. Может содержать до 31 байта.

inparam
Описание входного параметра.

outparam
Описание выходного параметра.

declarations
Секция объявления локальных переменных, именованных курсоров, подпроцедур и подфункций.

declare_var
Объявление локальной переменной.

declare_cursor
Объявление именованного курсора.

declare_subfunc
Объявление подпрограммы–функции.

declare_subproc
Объявление подпрограммы–процедуры.

module name
Имя внешнего модуля, в котором расположена процедура/функция.

routine name
Внутреннее имя процедуры/функции внутри внешнего модуля.

misc info
Определяемая пользователем информация для передачи в функцию внешнего модуля.

engine
Имя движка для использования внешних функций. Обычно указывается имя UDR.

literal
Литерал.

context_var
Любая контекстная переменная, тип которой совместим с типом параметра.

paramname
Имя входного или выходного параметра процедуры/функции. Может содержать до 31 символа. Имя параметра должно быть уникальным среди входных и выходных параметров процедуры/функции, а также её локальных переменных.

datatype
Тип данных SQL.

collation
Порядок сортировки.

domain
Домен.

rel
Имя таблицы или представления.

col
Имя столбца таблицы или представления.

precision
Точность. От 1 до 18.

scale
Масштаб. От 0 до 18, должно быть меньше или равно *precision*.

size
Максимальный размер строки в символах.

charset
Набор символов.

subtype_num
Номер подтипа BLOB.

subtype_name
Мнемоника подтипа BLOB.

seqlen
Размер сегмента, не может превышать 65535.

Оператор CREATE PACKAGE BODY создаёт новое тело пакета. Тело пакета может быть создано только после того как будет создан заголовок пакета. Если заголовка пакета с именем *package_name* не существует, то будет выдана соответствующая ошибка.

Все процедуры и функции, объявленные в заголовке пакета, должны быть реализованы в теле пакета. Кроме того, должны быть реализованы и все процедуры и функции, объявленные в теле пакета. Процедуры и функции, определенные в теле пакета, но не объявленные в заголовке пакета, не видны вне тела пакета.

Имена процедур и функций, объявленные в теле пакета, должны быть уникальны среди имён процедур и функций, объявленных в заголовке и теле пакета.

> **Примечание**
>
> Желательно чтобы имена хранимых процедур и функций пакета не пересекались с именами хранимых процедур и функций из глобального пространства имен, хотя это и допустимо. Дело в том, что в настоящее время вы не сможете вызвать функцию/процедуру из глобального пространства имён внутри пакета, если в пакете объявлена одноименная функция/процедура. В этом случае всегда будет вызвана процедура/функция пакета.

Правила:

- В теле пакеты должны быть реализованы все подпрограммы, стой же сигнатурой, что и объявленные в заголовке и в начале тела пакета.

- Значения по умолчанию для параметров процедур не могут быть переопределены (которые указываются в *<package_item>*). Это означает, что они могут быть в *<package_body_item>* только для частных процедур, которые не были объявлены.

> **Примечание**
>
> UDF деклараций (DECLARE внешняя функция) в настоящее время не поддерживается внутри пакетов.

Кто может создать тело пакета?

Создать тело пакета могут:

- SYSDBA;
- Владелец базы данных;
- Владелец пакета;
- Любой пользователь с привилегией на изменение любого пакета (GRANT ALTER ANY PACKAGE);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.104. Создание тела пакета

```
CREATE PACKAGE BODY APP_VAR
AS
BEGIN
  -- Возвращает дату начала периода
  FUNCTION GET_DATEBEGIN() RETURNS DATE
  AS
  BEGIN
    RETURN RDB$GET_CONTEXT('USER_SESSION', 'DATEBEGIN');
  END
  -- Возвращает дату окончания периода
  FUNCTION GET_DATEEND() RETURNS DATE
  AS
  BEGIN
    RETURN RDB$GET_CONTEXT('USER_SESSION', 'DATEEND');
  END
  -- Устанавливает диапазон дат рабочего периода
  PROCEDURE SET_DATERANGE(ADATEBEGIN DATE, ADATEEND DATE)
  AS
  BEGIN
    RDB$SET_CONTEXT('USER_SESSION', 'DATEBEGIN', ADATEBEGIN);
    RDB$SET_CONTEXT('USER_SESSION', 'DATEEND', ADATEEND);
  END
```

```
END
```

См. также: ALTER PACKAGE BODY, DROP PACKAGE BODY, CREATE PACKAGE.

ALTER PACKAGE BODY

Назначение: Изменение тела пакета.

Доступно в: DSQL.

Синтаксис:

```
ALTER PACKAGE BODY package_name
AS
BEGIN
  [<package_item> ...]
  [<package_body_item> ...]
END

<package_item> ::=
    <function_decl>;
  | <procedure_decl>;

<function_decl> ::=
  FUNCTION func_name [(<in_params>)]
    RETURNS <type> [COLLATE collation] [DETERMINISTIC]

<procedure_decl> ::=
  PROCEDURE proc_name [(<in_params>)] [RETURNS (<out_params>)]

<package_body_item> ::=
    <function_impl>
  | <procedure_impl>

<function_impl> ::=
  FUNCTION func_name [(<in_impl_params>)]
    RETURNS <type> [COLLATE collation] [DETERMINISTIC]
      { EXTERNAL NAME '<extname>' ENGINE <engine> }  |
      { AS
          [<declarations>]
        BEGIN
          [<PSQL_statements>]
        END
      }

<procedure_impl> ::=
  PROCEDURE proc_name [(<in_impl_params>)] [RETURNS (<out_params>)]
      { EXTERNAL NAME '<extname>' ENGINE <engine> }  |
      { AS
          [<declarations>]
        BEGIN
          [<PSQL_statements>]
        END
      }
```

Подробнее см. CREATE PACKAGE BODY.

Оператор ALTER PACKAGE BODY изменяет тело пакета. Позволяется изменять количество и состав процедур и функций тела пакета.

Все процедуры и функции, объявленные в заголовке пакета, должны быть реализованы в теле пакета. Кроме того, должны быть реализованы и все процедуры и функции, объявленные в теле пакета.

Имена процедур и функций, объявленные в теле пакета, должны быть уникальны среди имён процедур и функций, объявленных в заголовке и теле пакета.

> **Примечание**
>
> Желательно чтобы имена хранимых процедур и функций пакета не пересекались с именами хранимых процедур и функций из глобального пространства имен, хотя это и допустимо. Дело в том, что в настоящее время вы не сможете вызвать функцию/процедуру из глобального пространства имён внутри пакета, если в пакете объявлена одноименная функция/процедура. В этом случае всегда будет вызвана процедура/функция пакета.

Правила:

- В теле пакеты должны быть реализованы все подпрограммы, стой же сигнатурой, что и объявленные в заголовке и в начале тела пакета.
- Значения по умолчанию для параметров процедур не могут быть переопределены (которые указываются в *<package_item>*). Это означает, что они могут быть в *<package_body_item>* только для частных процедур, которые не были объявлены.

Кто может изменить тело пакета?

Изменить тело пакета могут:

- SYSDBA;
- Владелец базы данных;
- Владелец пакета;
- Любой пользователь с привилегией на изменение любого пакета (GRANT ALTER ANY PACKAGE);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.105. Изменение тела пакета

```
ALTER PACKAGE BODY APP_VAR
AS
BEGIN
  -- Возвращает дату начала периода
  FUNCTION GET_DATEBEGIN() RETURNS DATE
```

```
  AS
  BEGIN
    RETURN RDB$GET_CONTEXT('USER_SESSION', 'DATEBEGIN');
  END
  -- Возвращает дату окончания периода
  FUNCTION GET_DATEEND() RETURNS DATE
  AS
  BEGIN
    RETURN RDB$GET_CONTEXT('USER_SESSION', 'DATEEND');
  END
  -- Устанавливает диапазон дат рабочего периода
  PROCEDURE SET_DATERANGE(ADATEBEGIN DATE, ADATEEND DATE)
  AS
  BEGIN
    RDB$SET_CONTEXT('USER_SESSION', 'DATEBEGIN', ADATEBEGIN);
    RDB$SET_CONTEXT('USER_SESSION', 'DATEEND', ADATEEND);
  END
END
```

См. также: CREATE PACKAGE BODY, DROP PACKAGE BODY, ALTER PACKAGE.

DROP PACKAGE BODY

Назначение: Удаление тела пакета.

Доступно в: DSQL.

Синтаксис:

```
DROP PACKAGE BODY package_name
```

Параметры оператора DROP PACKAGE BODY

package_name
Имя пакета.

Оператор DROP PACKAGE BODY удаляет тело пакета.

Кто может удалить тело пакета?

Удалить тело пакета могут:

- SYSDBA;
- Владелец базы данных;
- Владелец пакета;
- Любой пользователь с привилегией на изменение любого пакета (GRANT ALTER ANY PACKAGE);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.106. Удаление тела пакета

```
DROP PACKAGE BODY APP_VAR;
```

См. также: CREATE PACKAGE BODY, ALTER PACKAGE BODY, DROP PACKAGE.

RECREATE PACKAGE BODY

Назначение: Создание нового и пересоздание существующего тела пакета.

Доступно в: DSQL.

Синтаксис:

```
RECREATE PACKAGE BODY package_name
AS
BEGIN
  [<package_item> ...]
  [<package_body_item> ...]
END

<package_item> ::=
    <function_decl>;
  | <procedure_decl>;

<function_decl> ::=
  FUNCTION func_name [(<in_params>)]
    RETURNS <type> [COLLATE collation] [DETERMINISTIC]

<procedure_decl> ::=
  PROCEDURE proc_name [(<in_params>)] [RETURNS (<out_params>)]

<package_body_item> ::=
    <function_impl>
  | <procedure_impl>

<function_impl> ::=
  FUNCTION func_name [(<in_impl_params>)]
    RETURNS <type> [COLLATE collation] [DETERMINISTIC]
      { EXTERNAL NAME '<extname>' ENGINE <engine> }  |
      { AS
          [<declarations>]
        BEGIN
          [<PSQL_statements>]
        END
      }

<procedure_impl> ::=
  PROCEDURE proc_name [(<in_impl_params>)] [RETURNS (<out_params>)]
      { EXTERNAL NAME '<extname>' ENGINE <engine> }  |
      { AS
          [<declarations>]
        BEGIN
```

```
          [<PSQL_statements>]
        END
      }
```

Подробнее см. CREATE PACKAGE BODY.

Оператор RECREATE PACKAGE BODY создаёт новое или пересоздаёт существующее тело пакета. Если тело пакета с таким именем уже существует, то оператор попытается удалить его и создать новое тело пакета. После пересоздания тела пакета привилегии на выполнение подпрограмм пакета и привилегии самого пакета сохраняются.

Примеры

Пример 5.107. Пересоздание тела пакета

```
RECREATE PACKAGE BODY APP_VAR
AS
BEGIN
  -- Возвращает дату начала периода
  FUNCTION GET_DATEBEGIN() RETURNS DATE
  AS
  BEGIN
    RETURN RDB$GET_CONTEXT('USER_SESSION', 'DATEBEGIN');
  END
  -- Возвращает дату окончания периода
  FUNCTION GET_DATEEND() RETURNS DATE
  AS
  BEGIN
    RETURN RDB$GET_CONTEXT('USER_SESSION', 'DATEEND');
  END
  -- Устанавливает диапазон дат рабочего периода
  PROCEDURE SET_DATERANGE(ADATEBEGIN DATE, ADATEEND DATE)
  AS
  BEGIN
    RDB$SET_CONTEXT('USER_SESSION', 'DATEBEGIN', ADATEBEGIN);
    RDB$SET_CONTEXT('USER_SESSION', 'DATEEND', ADATEEND);
  END
END
```

См. также: CREATE PACKAGE BODY, DROP PACKAGE BODY, ALTER PACKAGE.

EXTERNAL FUNCTION

Внешние функции, также известные как функции определяемые пользователем (User Defined Function) — это программы, написанные на любом языке программирования, и хранящиеся в динамических библиотеках. После того как функция объявлена в базе данных, она становится в динамических и процедурных операторах, как будто они реализованы внутри языка SQL.

Внешние функции существенно расширяют возможности SQL по обработке данных. Для того чтобы функции были доступны в базе данных, их необходимо объявить с помощью оператора DECLARE EXTERNAL FUNCTON.

После объявления функции, содержащая её библиотека будет загружаться при первом обращении к любой из функций, включённой в библиотеку.

> **Примечание**
>
> Внешние функции, объявленные как DECLARE EXTERNAL FUNCTION, являются унаследованными (Legacy) из предыдущих версий Firebird. Их возможности существенно уступают возможностям нового типа внешних функций UDR (User Defined Routine). Такие функции объявляются как CREATE FUNCTION ... EXTERNAL NAME. Подробнее см. CREATE FUNCTION.

DECLARE EXTERNAL FUNCTION

Назначение: Объявление в базе данных функции определённой пользователем (UDF).

Доступно в: DSQL, ESQL.

Синтаксис:

```
DECLARE EXTERNAL FUNCTION funcname
[<arg_type_decl> [, <arg_type_decl> ...]]
RETURNS {
  sqltype [BY {DESCRIPTOR | VALUE}] |
  CSTRING(length) |
  PARAMETER param_num }
[FREE_IT]
ENTRY_POINT 'entry_point' MODULE_NAME 'library_name';

<arg_type_decl> ::=
  sqltype [{BY DESCRIPTOR} | NULL] |
  CSTRING(length) [NULL]
```

Параметры оператора DECLARE EXTERNAL FUNCTION

funcname
: Имя внешней функции. Может содержать до 31 байта.

entry_point
: Имя экспортируемой функции (точка входа).

library_name
: Имя модуля, в котором расположена функция.

sqltype
: Тип данных SQL. Не может быть массивом или элементом массива.

length
: Максимальная длина нуль терминальной строки. Указывается в байтах.

param_num
: Номер входного параметра, который будет возвращён функцией.

Оператор DECLARE EXTERNAL FUNCTION делает доступным функцию, определенную пользователем (UDF), в базе данных. Внешние функции должны быть объявлены в каждой

базе данных, которая собирается их использовать. Не нужно объявлять UDF, если вы никогда не будете её использовать.

Имя внешней функции должно быть уникальным среди всех имён функций. Оно может отличаться от имени функции указанной в аргументе ENTRY_POINT.

Входные параметры функции перечисляются через запятую сразу после имени функции. Для каждого параметра указывается SQL тип данных. Массивы не могут использоваться в качестве параметров функций. Помимо SQL типов можно указать тип CSTRING. В этом случае параметр является нуль терминальной строкой с максимальной длиной *`length`* байт.

По умолчанию входные параметры передаются по ссылке. При передаче NULL значения по ссылке оно преобразовывается в эквивалент нуля, например, число 0 или пустую строку. Если после указанного параметра указано ключевое слово NULL, то при передаче значение NULL оно попадёт в функцию в виде нулевого указателя (null).

> **Примечание**
>
> Обратите внимание на то, что объявление функции с ключевым словом NULL не гарантирует вам, что эта функция правильно обработает входной параметр со значением NULL. Любая функция должна быть написана или переписана таким образом, чтобы правильно обрабатывать значения NULL. Всегда смотрите и используйте объявления функции, предоставленные её разработчиком.

Если указано предложение BY DESCRIPTOR, то входной параметр передаётся по дескриптору. Передача параметра по дескриптору облегчает обработку значений NULL. Отметим, что это объявление работает только в том случае, если внешняя функция поддерживает его. Простое добавление "BY DESCRIPTOR" к существующей декларации не заставит его работать. Всегда используйте объявление блока, обеспеченное функционалом внешней функции.

> **Предупреждение**
>
> При передаче параметра функции по дескриптору передаваемое значение не приводится к задекларированному типу данных.

Обязательное предложение RETURNS описывает выходной параметр возвращаемый функцией. Функция всегда возвращает только один параметр. Выходной параметр может быть любым SQL типом (кроме массива и элемента массива) или нуль терминальной строкой (CSTRING). Выходной параметр может быть передан по ссылке, по дескриптору или по значению. По умолчанию выходной параметр передаётся по ссылке. Если указано предложение BY DESCRIPTOR, то выходной параметр передаётся по дескриптору. Если указано предложение BY VALUE, то выходной параметр передаётся по значению.

Ключевое слово PARAMETER указывает, что функция возвращает значение из параметра с номером *`param_num`*. Такая необходимость возникает, если необходимо возвращать значение типа BLOB.

Ключевое слово FREE_IT означает, что память, выделенная для хранения возвращаемого значения, будет освобождена после завершения выполнения функции. Применяется только в том случае, если эта память в UDF выделялась динамически. В такой UDF память должна выделяться при помощи функции *ib_util_malloc* из модуля *ib_util*. Это необходимо для совместимости функций выделения и освобождения памяти используемого в коде Firebird и коде UDF.

Предложение ENTRY_POINT указывает имя точки входа (имя экспортируемой функции) в модуле.

Предложение MODULE_NAME задаёт имя модуля, в котором находится экспортируемая функция. В ссылке на модуль может отсутствовать полный путь и расширение файла. Это позволяет легче переносить базу данных между различными платформами. По умолчанию динамические библиотеки пользовательских функций должны располагаться в папке UDF корневого каталога сервера. Параметр *UDFAccess* в файле `firebird.conf` позволяет изменить ограничения доступа к библиотекам внешних функций.

Кто может объявить внешнюю функцию?

Объявить внешнюю функцию могут:

- SYSDBA;
- Владелец базы данных;
- Любой пользователь с привилегией на создание функции (GRANT CREATE FUNCTION);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Пользователь, объявивший внешнюю функцию, становится её владельцем.

Примеры

Пример 5.108. Объявление внешней функции с передачей входных и выходных параметров по ссылке

```
DECLARE EXTERNAL FUNCTION addDay
TIMESTAMP, INT
RETURNS TIMESTAMP
ENTRY_POINT 'addDay' MODULE_NAME 'fbudf';
```

Пример 5.109. Объявление внешней функции с передачей входных и выходных параметров по дескриптору

```
DECLARE EXTERNAL FUNCTION invl
INT BY DESCRIPTOR, INT BY DESCRIPTOR
RETURNS INT BY DESCRIPTOR
ENTRY_POINT 'idNvl' MODULE_NAME 'fbudf';
```

Пример 5.110. Объявление внешней функции с передачей входных параметров по ссылке, выходных по значению

```
DECLARE EXTERNAL FUNCTION isLeapYear
TIMESTAMP
RETURNS INT BY VALUE
ENTRY_POINT 'isLeapYear' MODULE_NAME 'fbudf';
```

Пример 5.111. Объявление внешней функции с передачей входных и выходных параметров по дескриптору. В качестве выходного параметра используется второй параметр функции.

```
DECLARE EXTERNAL FUNCTION i64Truncate
NUMERIC(18) BY DESCRIPTOR, NUMERIC(18) BY DESCRIPTOR
RETURNS PARAMETER 2
ENTRY_POINT 'fbtruncate' MODULE_NAME 'fbudf';
```

См. также: ALTER EXTERNAL FUNCTION, DROP EXTERNAL FUNCTION, CREATE FUNCTION.

ALTER EXTERNAL FUNCTION

Назначение: Изменение точки входа и/или имени модуля для функции определённой пользователем (UDF).

Доступно в: DSQL.

Синтаксис:

```
ALTER EXTERNAL FUNCTION funcname
[ENTRY_POINT 'new_entry_point']
[MODULE_NAME 'new_library_name'];
```

Параметры оператора ALTER EXTERNAL FUNCTION

funcname
Имя внешней функции.

new_entry_point
Новое имя экспортируемой функции (точки входа).

new_library_name
Новое имя модуля, в котором расположена функция.

Оператор ALTER EXTERNAL FUNCTION изменяет точку вход и/или имя модуля для функции определённой пользователем (UDF). При этом существующие зависимости сохраняются.

Предложение ENTRY_POINT позволяет указать новую точку входа (имя экспортируемой функции).

Предложение MODULE_NAME позволяет указать новое имя модуля, в котором расположена экспортируемая функция.

Кто может изменить внешнюю функцию?

Изменить точку входа и имя модуля внешней функции могут:

- SYSDBA;

- Владелец базы данных;
- Владелец функции;
- Любой пользователь с привилегией на изменение любой функции (GRANT ALTER ANY FUNCTION);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.112. Изменение точки входа для внешней функции

```
ALTER EXTERNAL FUNCTION invl ENTRY_POINT 'intNvl';
```

Пример 5.113. Изменение имени модуля для внешней функции

```
ALTER EXTERNAL FUNCTION invl MODULE_NAME 'fbudf2';
```

См. также: DECLARE EXTERNAL FUNCTION, DROP EXTERNAL FUNCTION.

DROP EXTERNAL FUNCTION

Назначение: Удаление объявления функции определённой пользователем (UDF) из базы данных.

Доступно в: DSQL, ESQL.

Синтаксис:

```
DROP EXTERNAL FUNCTION funcname;
```

Параметры оператора DROP EXTERNAL FUNCTION

funcname
Имя внешней функции.

Оператор DROP EXTERNAL FUNCTION удаляет объявление функции определённой пользователем из базы данных. Если есть зависимости от внешней функции, то удаления не произойдёт и будет выдана соответствующая ошибка.

Кто может удалить внешнюю функцию?

Удалить внешнюю функцию могут:

- SYSDBA;
- Владелец базы данных;
- Владелец функции;

- Любой пользователь с привилегией на удаление любой функции (GRANT DROP ANY FUNCTION);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.114. Удаление внешней функции

```
DROP EXTERNAL FUNCTION addDay;
```

См. также: DECLARE EXTERNAL FUNCTION.

FILTER

BLOB фильтр — объект базы данных, являющийся, по сути, специальным видом внешних функций с единственным назначением: получение объекта BLOB одного формата и преобразования его в объект BLOB другого формата. Форматы объектов BLOB задаются с помощью подтипов BLOB.

Внешние функции для преобразования BLOB типов хранятся в динамических библиотеках и загружаются по необходимости.

Подробнее о подтипах BLOB см. в разделе Бинарные типы данных.

DECLARE FILTER

Назначение: Объявление в базе данных BLOB фильтра.

Доступно в: DSQL, ESQL.

Синтаксис:

```
DECLARE FILTER filtername
INPUT_TYPE <sub_type> OUTPUT_TYPE <sub_type>
ENTRY_POINT 'function_name' MODULE_NAME 'library_name';

<sub_type> ::= number | <mnemonic>

<mnemonic> ::= binary | text | blr | acl | ranges | summary |
               format | transaction_description |
               external_file_description | user_defined
```

Параметры оператора DECLARE FILTER

filtername
Имя фильтра. Может содержать до 31 байта.

sub_type
Подтип BLOB.

number
Номер подтипа BLOB.

mnemonic
Мнемоника подтипа BLOB.

function_name
Имя экспортируемой функции (точка входа).

library_name
Имя модуля, в котором расположен фильтр.

user_defined
Определяемая пользователем мнемоника подтипа BLOB.

Оператор DECLARE FILTER делает доступным BLOB фильтр в базе данных. Имя BLOB фильтра должно быть уникальным среди имён BLOB фильтров.

Задание подтипов

Подтип может быть задан в виде номера подтипа или мнемоники подтипа. Пользовательские подтипы должны быть представлены отрицательными числами (от -1 до -32768). Не допускается создание двух и более фильтров BLOB с одинаковыми комбинациями входных и выходных типов. Объявление фильтра с уже существующими комбинациями входных и выходных типов BLOB приведёт к ошибке.

Предложение INPUT_TYPE идентифицирует тип преобразуемого объекта (подтип BLOB).

Предложение OUTPUT_TYPE идентифицирует тип создаваемого объекта.

Примечание

Если вы хотите определить мнемоники для собственных подтипов BLOB, вы можете добавить их в системную таблицу RDB$TYPES, как показано ниже. После подтверждения транзакции мнемоники могут быть использованы для декларации при создании новых фильтров.

```
INSERT INTO RDB$TYPES (RDB$FIELD_NAME, RDB$TYPE, RDB$TYPE_NAME)
VALUES ('RDB$FIELD_SUB_TYPE', -33, 'MIDI');
```

Значение поля rdb$field_name всегда должно быть 'RDB$FIELD_SUB_TYPE'. Если вы определяете мнемоники в верхнем регистре, то можете использовать их без учёта регистра и без кавычек при объявлении фильтра.

Параметры DECLARE FILTER

Предложение ENTRY_POINT указывает имя точки входа (имя экспортируемой функции) в модуле.

Предложение MODULE_NAME задаёт имя модуля, в котором находится экспортируемая функция. По умолчанию модули должны располагаться в папке UDF корневого каталога

сервера. Параметр *UDFAccess* в файле `firebird.conf` позволяет изменить ограничения доступа к библиотекам фильтров.

Кто может создать BLOB фильтр?

Создать BLOB фильтр могут:

- SYSDBA;
- Владелец базы данных;
- Любой пользователь с привилегией на создание фильтра (GRANT CREATE FILTER);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Пользователь, создавший BLOB фильтр, становится его владельцем.

Примеры

Пример 5.115. Создание BLOB фильтра с использованием номеров подтипов

```
DECLARE FILTER DESC_FILTER
INPUT_TYPE 1
OUTPUT_TYPE -4
ENTRY_POINT 'desc_filter'
MODULE_NAME 'FILTERLIB';
```

Пример 5.116. Создание BLOB фильтра с использованием мнемоник подтипов

```
DECLARE FILTER FUNNEL
INPUT_TYPE blr OUTPUT_TYPE text
ENTRY_POINT 'blr2asc' MODULE_NAME 'myfilterlib';
```

См. также: DROP FILTER.

DROP FILTER

Назначение: Удаление объявления BLOB фильтра.

Доступно в: DSQL, ESQL.

Синтаксис:

```
DROP FILTER filtername;
```

Параметры оператора DROP FILTER

filtername
Имя BLOB фильтра.

Оператор DROP FILTER удаляет объявление BLOB фильтра из базы данных. Удаление BLOB фильтра из базы данных делает его не доступным из базы данных, при этом динамическая библиотека, в которой расположена функция преобразования, остаётся не тронутой.

Кто может удалить BLOB фильтр?

Удалить BLOB фильтр могут:

- SYSDBA;
- Владелец базы данных;
- Владелец фильтра;
- Любой пользователь с привилегией на удаление любого фильтра (GRANT DROP ANY FILTER);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.117. Удаление BLOB фильтра

```
DROP FILTER DESC_FILTER;
```

См. также: DECLARE FILTER.

SEQUENCE (GENERATOR)

Последовательность (sequence) или генератор (generator) — объект базы данных, предназначенный для получения уникального числового значения. Термин последовательность является SQL совместимым. Ранее в Interbase и Firebird последовательности называли генераторами.

Независимо от диалекта базы данных последовательности (или генераторы) всегда хранятся как 64-битные целые значения.

> **Внимание**
>
> Если клиент использует 1 диалект, то сервер передаёт ему значения последовательности, усечённые до 32-битного значения. Если значение последовательности передаются в 32-разрядное поле или переменную, то до тех пор, пока текущее значение последовательности не вышло за границы для 32-битного числа, ошибок не будет. В момент выхода значения последовательности за этот диапазон база данных 3-го диалекта выдаст сообщение об ошибке, а база данных 1-го диалекта будет молча обрезать значения (что также может привести к ошибке — например, если поле, заполняемое генератором, является первичным или уникальным).

В данном разделе описываются вопросы создания, модификации (установка значения последовательности) и удаления последовательностей.

CREATE {SEQUENCE | GENERATOR}

Назначение: Создание новой последовательности (генератора).

Доступно в: DSQL, ESQL.

Синтаксис:

```
CREATE {SEQUENCE | GENERATOR} seq_name
[START WITH value] [INCREMENT [BY] increment];
```

Параметры оператора CREATE SEQUENCE

seq_name
Имя последовательности (генератора). Может содержать до 31 байта.

value
Начальное значение последовательности (генератора).

increment
Шаг приращения. 4 байтное целое число.

Оператор CREATE SEQUENCE создаёт новую последовательность. Слова SEQUENCE и GENERATOR являются синонимами. Вы можете использовать любое из них, но рекомендуется использовать SEQUENCE.

В момент создания последовательности ей устанавливается значение, указанное в необязательном предложении START WITH. Если предложение STARTING WITH отсутствует, то последовательности устанавливается значение равное 0.

Необязательное предложение INCREMENT [BY] позволяет задать шаг приращения для оператора NEXT VALUES FOR. По умолчанию шаг приращения равен единице. Приращение не может быть установлено в ноль для пользовательских последовательностей. Значение последовательности изменяется также при обращении к функции GEN_ID, где в качестве параметра указывается имя последовательности и значение приращения, которое может быть отлично от указанного в предложении INCREMENT BY.

Кто может создать последовательность?

Создать новую последовательность (генератор) могут:

- SYSDBA;
- Владелец базы данных;
- Любой пользователь с привилегией на создание последовательности (генератора) (GRANT CREATE SEQUENCE);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Пользователь, создавший последовательность, становится её владельцем.

Примеры

Пример 5.118. Создание последовательности

Создание последовательности EMP_NO_GEN с начальным значением 0 и шагом приращения равным единице.

```
CREATE SEQUENCE EMP_NO_GEN;
```

Пример 5.119. Создание последовательности

Создание последовательности EMP_NO_GEN с начальным значением 5 и шагом приращения равным единице.

```
CREATE SEQUENCE EMP_NO_GEN START WITH 5;
```

Пример 5.120. Создание последовательности

Создание последовательности EMP_NO_GEN с начальным значением 0 и шагом приращения равным 10.

```
CREATE SEQUENCE EMP_NO_GEN INCREMENT BY 10;
```

Пример 5.121. Создание последовательности

Создание последовательности EMP_NO_GEN с начальным значением 5 и шагом приращения равным 10.

```
CREATE SEQUENCE EMP_NO_GEN START WITH 5 INCREMENT BY 10;
```

См. также: ALTER SEQUENCE, SET GENERATOR, DROP SEQUENCE, NEXT VALUE FOR, GEN_ID.

ALTER {SEQUENCE | GENERATOR}

Назначение: Изменение последовательности (генератора).

Доступно в: DSQL, ESQL.

Синтаксис:

```
ALTER {SEQUENCE | GENERATOR} seq_name
[RESTART [WITH new_val]]
[INCREMENT [BY] increment];
```

Параметры оператора ALTER SEQUENCE

seq_name
Имя последовательности (генератора).

new_val
Новое значение последовательности (генератора). 64 битное целое в диапазоне от -2^{63} .. $2^{63} - 1$

increment
Шаг приращения.

Оператор ALTER SEQUENCE устанавливает значение последовательности или генератора в заданное значение и/или изменяет значение приращения.

Предложение RESTART WITH позволяет установить значение последовательности. Предложение RESTART может быть использовано самостоятельно (без WITH) для перезапуска значения последовательности с того значения с которого был начат старт генерации значений или предыдущий рестарт.

> **Предупреждение**
>
> Неосторожное использование оператора ALTER SEQUENCE (изменение значения последовательности или генератора) может привести к нарушению логической целостности данных.

Предложение INCREMENT [BY] позволяет изменить шаг приращения последовательности для оператора NEXT VALUES FOR.

> **Примечание**
>
> Изменение значения приращения — это возможность, которая вступает в силу для каждого запроса, который запускается после фиксаций изменения. Процедуры, которые вызваны впервые после изменения приращения, будут использовать новое значение, если они будут содержать операторы NEXT VALUE FOR. Процедуры, которые уже работают, не будут затронуты, потому что они кэшируются. Процедуры, использующие NEXT VALUE FOR, не должны быть перекомпилированы, чтобы видеть новое приращение, но если они уже работают или загружены, то никакого эффекта не будет. Конечно процедуры, использующие gen_id(gen, expression), не затронут при изменении приращения.

Кто может изменить последовательность?

Изменить последовательность (генератор) могут:

- SYSDBA;
- Владелец базы данных;
- Владелец последовательности (генератора);
- Любой пользователь с привилегией на изменение любой последовательности (GRANT ALTER ANY SEQUENCE);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.122. Изменение последовательности

Установка для последовательности EMP_NO_GEN значения 145.

```
ALTER SEQUENCE EMP_NO_GEN RESTART WITH 145;
```

Пример 5.123. Изменение последовательности

Сброс значения последовательности в то, которое было установлено при создании последовательности (или при предыдущей установке значения).

```
ALTER SEQUENCE EMP_NO_GEN RESTART;
```

Пример 5.124. Изменение последовательности

Изменение значения приращения последовательности EMP_NO_GEN.

```
ALTER SEQUENCE EMP_NO_GEN INCREMENT BY 10;
```

См. также: SET GENERATOR, CREATE SEQUENCE, DROP SEQUENCE, NEXT VALUE FOR, GEN_ID.

CREATE OR ALTER {SEQUENCE | GENERATOR}

Назначение: Создание новой или изменение существующей последовательности (генератора).

Доступно в: DSQL, ESQL.

Синтаксис:

```
CREATE OR ALTER {SEQUENCE | GENERATOR} seq_name
[{START WITH value | RESTART}]
[INCREMENT [BY] increment];
```

Параметры оператора CREATE OR ALTER SEQUENCE

seq_name
Имя последовательности (генератора). Может содержать до 31 байта.

value
Начальное значение последовательности (генератора).

increment
Шаг приращения.

Если последовательности не существует, то она будет создана. Уже существующая последовательность будет изменена, при этом существующие зависимости последовательности будут сохранены.

Примеры

Пример 5.125. Создание новой или изменение существующей последовательности

```
CREATE OR ALTER SEQUENCE EMP_NO_GEN
START WITH 10
INCREMENT BY 1;
```

См. также: CREATE SEQUENCE, ALTER SEQUENCE, SET GENERATOR.

DROP {SEQUENCE | GENERATOR}

Назначение: Удаление последовательности (генератора).

Доступно в: DSQL, ESQL.

Синтаксис:

```
DROP {SEQUENCE | GENERATOR} seq_name;
```

Параметры оператора DROP SEQUENCE

seq_name
Имя последовательности (генератора).

Оператор DROP SEQUENCE удаляет существующую последовательность (генератор). Слова SEQUENCE и GENERATOR являются синонимами. Вы можете использовать любое из них, но рекомендуется использовать SEQUENCE. При наличии зависимостей для существующей последовательности (генератора) удаления не будет выполнено.

Кто может удалить генератор?

Удалить последовательность (генератор) могут:

- SYSDBA;
- Владелец базы данных;
- Владелец последовательности (генератора);
- Любой пользователь с привилегией на удаление любой последовательности (GRANT DROP ANY SEQUENCE);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.126. Удаление последовательности

```
DROP SEQUENCE EMP_NO_GEN;
```

См. также: CREATE SEQUENCE, ALTER SEQUENCE, RECREATE SEQUENCE.

RECREATE {SEQUENCE | GENERATOR}

Назначение: Создание или пересоздание последовательности (генератора).

Доступно в: DSQL, ESQL.

Синтаксис:

```
RECREATE {SEQUENCE | GENERATOR} seq_name
[START WITH value] [INCREMENT [BY] increment];
```

Параметры оператора RECREATE SEQUENCE

seq_name
Имя последовательности (генератора). Может содержать до 31 байта.

value
Начальное значение последовательности (генератора).

increment
Шаг приращения.

Оператор RECREATE SEQUENCE создаёт или пересоздаёт последовательность (генератор). Если последовательность с таким именем уже существует, то оператор RECREATE SEQUENCE попытается удалить её и создать новую последовательность. При наличии зависимостей для существующей последовательности оператор RECREATE SEQUENCE не выполнится.

Примеры

Пример 5.127. Пересоздание последовательности

```
RECREATE SEQUENCE EMP_NO_GEN
START WITH 10
INCREMENT BY 1;
```

SET GENERATOR

Назначение: Устанавливает значение последовательности или генератора в заданное значение.

Доступно в: DSQL, ESQL.

Синтаксис:

```
SET GENERATOR seq_name TO new_val;
```

Параметры оператора SET GENERATOR

seq_name
Имя последовательности (генератора).

new_val
Новое значение последовательности (генератора). 64 битное целое в диапазоне от $-2^{63}..2^{63} - 1$

Оператор SET GENERATOR устанавливает значение последовательности или генератора в заданное значение.

> **Примечание**
>
> Оператор SET GENERATOR считается устаревшим и оставлен ради обратной совместимости. В настоящее время вместо него рекомендуется использовать стандарт-совместимый оператор ALTER SEQUENCE.
>
> Неосторожное использование оператора SET GENERATOR (изменение значения последовательности или генератора) может привести к потере логической целостности данных.

Кто может изменить значение генератора?

Пример 5.128. Установка значения для последовательности

```
SET GENERATOR EMP_NO_GEN TO 145;
```

> **Примечание**
>
> То же самое можно сделать, используя оператор ALTER SEQUENCE
>
> ```
> ALTER SEQUENCE EMP_NO_GEN RESTART WITH 145;
> ```

См. также: ALTER SEQUENCE, NEXT VALUE FOR, GEN_ID.

EXCEPTION

Пользовательское исключение (exception) — объект базы данных, описывающий сообщение об ошибке. Исключение можно вызывать и обрабатывать в PSQL коде (см. EXCEPTION, WHEN … DO).

В данном разделе описываются операторы создания, модификации и удаления исключений.

CREATE EXCEPTION

Назначение: Создание пользовательского исключения.

Доступно в: DSQL, ESQL.

Синтаксис:

```
CREATE EXCEPTION exception_name '<message>';

<message> ::= {txt | @n} <message>
```

Параметры оператора CREATE EXCEPTION

exception_name
Имя исключения. Максимальная длина 31 байта.

message
Сообщение об ошибке. Максимальная длина ограничена 1021 символом.

txt
Текст.

n
Номер слота для параметра. Нумерация начинается с 1. Максимальный номер слота равен 9.

Оператор CREATE EXCEPTION создаёт новое пользовательское исключение. Исключение должно отсутствовать в базе данных, иначе будет выдана соответствующая ошибка. Сообщение об ошибке может содержать слоты для параметров, которые заполняются при возбуждении исключения.

> **Внимание!**
>
> Если в тексте сообщения, встретится номер слота параметра больше 9, то второй и последующий символ будут восприняты как литералы. Например, @10 будет воспринято как @1, после которого следует литерал 0.

Кто может создать исключение?

Создать новое пользовательское исключение могут:

- SYSDBA;
- Владелец базы данных;
- Любой пользователь с привилегией на создание исключения (GRANT CREATE EXCEPTION);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Пользователь, создавший исключение, становится его владельцем.

Примеры

Пример 5.129. Создание пользовательского исключения

```
CREATE EXCEPTION E_LARGE_VALUE 'Значение превышает предельно допустимое';
```

Пример 5.130. Создание параметризованного исключения

```
CREATE EXCEPTION E_INVALID_VALUE
'Неверное значение @1 для поля @2';
```

См. также: ALTER EXCEPTION, CREATE OR ALTER EXCEPTION, DROP EXCEPTION, RECREATE EXCEPTION, EXCEPTION.

ALTER EXCEPTION

Назначение: Изменение текста сообщения пользовательского исключения.

Доступно в: DSQL, ESQL.

Синтаксис:

```
ALTER EXCEPTION exception_name '<message>';

<message> ::= {txt | @n} <message>
```

Параметры оператора ALTER EXCEPTION

exception_name
Имя исключения.

message
Сообщение об ошибке. Максимальная длина ограничена 1021 символом.

txt
Текст.

n
Номер слота для параметра.

Оператор ALTER EXCEPTION изменяет текст сообщения пользовательского исключения.

Кто может изменить исключение?

Изменить пользовательское исключение могут:

- SYSDBA;
- Владелец базы данных;
- Владелец исключения;
- Любой пользователь с привилегией на изменение любого исключения (GRANT ALTER ANY EXCEPTION);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.131. Изменение текста сообщения пользовательского исключения

```
ALTER EXCEPTION E_LARGE_VALUE 'Значение превышает максимально допустимое';
```

См. также: CREATE EXCEPTION, CREATE OR ALTER EXCEPTION, RECREATE EXCEPTION.

CREATE OR ALTER EXCEPTION

Назначение: Создание нового или изменение существующего исключения.

Доступно в: DSQL.

Синтаксис:

```
CREATE OR ALTER EXCEPTION exception_name '<message>';

<message> ::= {txt | @n} <message>
```

Параметры оператора CREATE OR ALTER EXCEPTION

exception_name
Имя исключения. Максимальная длина 31 байта.

message
Сообщение об ошибке. Максимальная длина ограничена 1021 символом.

txt
Текст.

n
Номер слота для параметра.

Если исключения не существует, то оно будет создано. Уже существующее исключение будет изменено, при этом существующие зависимости исключения будут сохранены.

Примеры

Пример 5.132. Создание или изменение пользовательского исключения

```
CREATE OR ALTER EXCEPTION E_LARGE_VALUE
'Значение превышает максимально допустимое';
```

См. также: CREATE EXCEPTION, ALTER EXCEPTION, RECREATE EXCEPTION.

DROP EXCEPTION

Назначение: Удаление пользовательского исключения.

Доступно в: DSQL, ESQL.

Синтаксис:

```
DROP EXCEPTION exception_name
```

Параметры оператора DROP EXCEPTION

exception_name
Имя исключения.

Оператор DROP EXCEPTION удаляет пользовательское исключение. При наличии зависимостей для существующего исключения удаления не будет выполнено.

Кто может удалить исключение?

Удалить пользовательское исключение могут:

- SYSDBA;
- Владелец базы данных;
- Владелец исключения;
- Любой пользователь с привилегией на удаление любого исключения (GRANT DROP ANY EXCEPTION);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.133. Удаление пользовательского исключения

```
DROP EXCEPTION E_LARGE_VALUE;
```

См. также: CREATE EXCEPTION, RECREATE EXCEPTION.

RECREATE EXCEPTION

Назначение: Создание или пересоздание пользовательского исключения.

Доступно в: DSQL.

Синтаксис:

```
RECREATE EXCEPTION exception_name '<message>';

<message> ::= {txt | @n} <message>
```

Параметры оператора RECREATE EXCEPTION

exception_name
Имя исключения. Максимальная длина 31 байта.

message
Сообщение об ошибке. Максимальная длина ограничена 1021 символом.

txt
Текст.

n
Номер слота для параметра.

Оператор RECREATE EXCEPTION создаёт или пересоздаёт пользовательское исключение. Если исключение с таким именем уже существует, то оператор RECREATE EXCEPTION попытается удалить его и создать новое исключение. При наличии зависимостей для существующего исключения оператор RECREATE EXCEPTION не выполнится.

Примеры

Пример 5.134. Создание или пересоздание пользовательского исключения

```
RECREATE EXCEPTION E_LARGE_VALUE
'Значение превышает максимально допустимое';
```

См. также: CREATE EXCEPTION, ALTER EXCEPTION, CREATE OR ALTER EXCEPTION.

COLLATION

В SQL текстовые строки принадлежат к сортируемым объектам. Это означает, что они подчиняются своим внутренним правилам упорядочения, например, алфавитному порядку.

К таким текстовым строкам можно применять операции сравнения (например, "меньше чем" или "больше чем"), при этом значения выражения должны вычисляться согласно определённой последовательности сортировки. Например, выражение 'a'<'b' означает, что 'a' предшествует 'b' в последовательности сортировки. Под выражением 'c'>'b' имеется в виду, что в последовательности сортировки 'c' определено после 'b'. Текстовые строки, включающие больше одного символа, сортируются путём последовательного сравнения символов: сначала сравниваются первые символы двух строк, затем вторые символы и так далее, до тех пор, пока не будет найдено различие между двумя строками. Такое различие управляет порядком сортировки.

Под сравнением (сортировкой) (COLLATION) принято понимать такой объект схемы, который определяет упорядочивающую последовательность (или последовательность сортировки).

CREATE COLLATION

Назначение: Добавление новой сортировки (сравнения) для набора символов поддерживаемого в базе данных.

Доступно в: DSQL.

Синтаксис:

```
CREATE COLLATION collname
FOR charset
[FROM basecoll | FROM EXTERNAL ('extname')]
[NO PAD | PAD SPACE]
[CASE [IN]SENSITIVE]
[ACCENT [IN]SENSITIVE]
['<specific-attributes>'];

<specific-attributes> ::= <attribute> [; <attribute> ...]

<attribute> ::= attrname=attrvalue
```

Параметры оператора CREATE COLLATION

collname
Имя сортировки (сравнения). Максимальная длина 31 байта.

charset
Набор символов.

basecoll
Базовая сортировка (сравнение).

extname
Имя сортировки из конфигурационного файла. Чувствительно к регистру.

Оператор CREATE COLLATION ничего не "создаёт", его целью является сделать сортировку известной для базы данных. Сортировка уже должна присутствовать в системе, как правило в файле библиотеки, и должна быть зарегистрирована в файле `fbintl.conf` подкаталога `intl` корневой директории Firebird.

Необязательное предложение FROM указывает сортировку, на основе которой будет создана новая сортировка. Такая сортировка должна уже присутствовать в базе данных. Если указано ключевое слово EXTERNAL, то будет осуществлён поиск сортировки из файла `$fbroot/intl/fbintl.conf`, при этом *`extname`* должно в точности соответствовать имени в конфигурационном файле (чувствительно к регистру).

Если предложение FROM отсутвует, то Firebird ищет в конфигурационном файле `fbintl.conf` подкаталога `intl` корневой директории сервера сортировку с именем, указанным сразу после CREATE COLLATION. Другими словами, отсутствие предложения FROM *`basecoll`* эквивалентно заданию FROM EXTERNAL ('*`collname`*').

При создании сортировки можно указать учитываются ли конечные пробелы при сравнении. Если указана опция NO PAD, то конечные пробелы при сравнении учитываются. Если указана опция PAD SPACE, то конечные пробелы при сравнении не учитываются.

Необязательное предложение CASE позволяет указать будет ли сравнение чувствительно к регистру.

Необязательное предложение ACCENT позволяет указать будет ли сравнение чувствительно к акцентированным буквам (например «е» и «ё»).

Специфичные атрибуты

В операторе CREATE COLLATION можно также указать специфичные атрибуты для сортировки. Ниже в таблице приведён список доступных специфичных атрибутов. Не все атрибуты применимы ко всем сортировкам. Если атрибут не применим к сортировке, но указан при её создании, то это не вызовет ошибки.

> **Важно**
>
> Имена специфичных атрибутов чувствительны к регистру.

«1 bpc» в таблице указывает на то, что атрибут действителен для сортировок наборов символов, использующих 1 байт на символ (так называемый узкий набор символов), а «UNI» — для юникодных сортировок.

Таблица 5.3. Список доступных специфичных атрибутов COLLATION

Имя	Значение	Валидность	Описание
DISABLE-COMPRESSIONS	0, 1	1 bpc	Отключает сжатия (иначе сокращения). Сжатия заставляют определённые символьные последовательности быть сортированными как атомарные модули, например, испанские c + h как единственный символ ch.
DISABLE-EXPANSIONS	0, 1	1 bpc	Отключение расширений. Расширения позволяют рассматривать определённые символы (например, лигатуры или гласные умляуты) как

Имя	Значение	Валидность	Описание
			последовательности символов и соответственно сортировать.
ICU-VERSION	*default* или *M.m*	UNI	Задаёт версию библиотеки ICU для использования. Допустимые значения определены в соответствующих элементах <intl_module> в файле `intl/fbintl.conf`. Формат: либо строка «default» или основной и дополнительный номер версии, как «3.0» (оба без кавычек).
LOCALE	*xx_YY*	UNI	Задаёт параметры сортировки языкового стандарта. Требуется полная версия библиотеки ICU. Формат строки: «du_NL» (без кавычек).
MULTI-LEVEL	0, 1	1 bpc	Использование нескольких уровней сортировки.
NUMERIC-SORT	0, 1	UNI	Обрабатывает непрерывные группы десятичных цифр в строке как атомарные модули и сортирует их в числовой последовательности (известна как естественная сортировка).
SPECIALS-FIRST	0, 1	1 bpc	Сортирует специальные символы (пробелы и т.д.) до буквенно-цифровых символов.

Подсказка

Если вы хотите добавить в базу данных новый набор символов с его умалчиваемой сортировкой, то зарегистрируйте и выполните хранимую процедуру *sp_register_character_name(name, max_bytes_per_character)* из подкаталога `misc/intl.sql` установки Firebird. Для нормальной работы с набором символов, он должен присутствовать в вашей операционной системе, и зарегистрирован в файле `fbintl.conf` поддиректории `intl`.

Кто может создать сортировку?

Создать новую сортировку могут:

- SYSDBA;
- Владелец базы данных;
- Любой пользователь с привилегией на создание сортировок (GRANT CREATE COLLATION);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Пользователь, создавший сортировку, становится её владельцем.

Примеры

Пример 5.135. Создание сортировки с использованием имени, найденном в файле fbintl.conf (регистро-чувствительно).

```
CREATE COLLATION ISO8859_1_UNICODE FOR ISO8859_1;
```

Пример 5.136. Создание сортировки с использованием специального (заданного пользователем) названия («external» имя должно в точности соответствовать имени в файле fbintl.conf).

```
CREATE COLLATION LAT_UNI
FOR ISO8859_1
FROM EXTERNAL ('ISO8859_1_UNICODE');
```

Пример 5.137. Создание регистронезависимой сортировки на основе уже присутствующей в базе данных.

```
CREATE COLLATION ES_ES_NOPAD_CI
FOR ISO8859_1
FROM ES_ES
NO PAD
CASE INSENSITIVE;
```

Пример 5.138. Создание регистронезависимой сортировки на основе уже присутствующей в базе данных со специфичными атрибутами.

```
CREATE COLLATION ES_ES_CI_COMPR
FOR ISO8859_1
FROM ES_ES
CASE INSENSITIVE
'DISABLE-COMPRESSIONS=0';
```

Пример 5.139. Создание регистронезависимой сортировки по значению чисел (так называемой натуральной сортировки).

```
CREATE COLLATION nums_coll FOR UTF8
FROM UNICODE
CASE INSENSITIVE 'NUMERIC-SORT=1';

CREATE DOMAIN dm_nums AS varchar(20)
CHARACTER SET UTF8 COLLATE nums_coll; -- original (manufacturer) numbers

CREATE TABLE wares(id int primary key, articul dm_nums ...);
```

См. также: DROP COLLATION.

DROP COLLATION

Назначение: Удаление существующей сортировки.

Доступно в: DSQL.

Синтаксис:

```
DROP COLLATION collname;
```

Параметры оператора DROP COLLATION

collname
Имя сортировки.

Оператор DROP COLLATION удаляет указанную сортировку. Сортировка должна присутствовать в базе данных, иначе будет выдана соответствующая ошибка.

> **Подсказка**
>
> Если вы хотите удалить в базе данных набор символов со всеми его сортировками, то зарегистрируйте и выполните хранимую процедуру *sp_unregister_character_set(name)* из подкаталога `misc/intl.sql` установки Firebird.

Кто может удалить сортировку?

Удалить сортировку могут:

- SYSDBA;
- Владелец базы данных;
- Владелец сортировки;
- Любой пользователь с привилегией на удаление любой сортировки (GRANT DROP ANY COLLATION);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.140. Удаление сортировки

```
DROP COLLATION ES_ES_NOPAD_CI;
```

См. также: CREATE COLLATION.

CHARACTER SET

ALTER CHARACTER SET

Назначение: Установка сортировки по умолчанию для набора символов.

Доступно в: DSQL.

Синтаксис:

```
ALTER CHARACTER SET charset
SET DEFAULT COLLATION collation;
```

Параметры оператора ALTER CHARACTER SET

charset
Набор символов.

collation
Сортировка.

Оператор ALTER CHARACTER SET изменяет сортировку по умолчанию для указанного набора символов. Это повлияет на использование набора символов в будущем, кроме случаев, когда явно переопределена сортировка COLLATE. Сортировка существующих доменов, столбцов и переменных PSQL при этом не будет изменена.

> **Примечание**
>
> Если сортировка по умолчанию была изменена для набора символов базы данных (тот, что был указан при создании базы данных), то также изменяется и сортировка по умолчанию для базы данных.
>
> Если сортировка по умолчанию была изменена для набора символов, который был указан при подключении, строковые константы будут интерпретироваться в соответствии с новыми параметрами сортировки (если набор символов и/или сортировка не переопределяются).

Примеры

Пример 5.141. Установка сортировки UNICODE_CI_AI по умолчанию для кодировки UTF8

```
ALTER CHARACTER SET UTF8 SET DEFAULT COLLATION UNICODE_CI_AI;
```

COMMENTS

Объекты базы данных и сама база данных могут содержать примечания. Это удобное средство документирования во время разработки базы данных и её поддержки.

COMMENT ON

Назначение: Документирование метаданных.

Доступно в: DSQL, ESQL.

Синтаксис:

```
COMMENT ON <object> IS {'sometext' | NULL}
<object> ::=
    DATABASE
  | <basic-type> objectname
  | COLUMN relationname.fieldname
  | [PROCEDURE | FUNCTION] PARAMETER
    [package_name.] routinename.paramname
  | {PROCEDURE | [EXTERNAL] FUNCTION}
    [package_name.]routinename

<basic-type> ::=
    CHARACTER SET
  | COLLATION
  | DOMAIN
  | EXCEPTION
  | FILTER
  | GENERATOR
  | INDEX
  | PACKAGE
  | USER
  | ROLE
  | SEQUENCE
  | TABLE
  | TRIGGER
  | VIEW
```

Параметры оператора COMMENT ON

sometext
Текст комментария.

basic-type
Тип объекта метаданных.

objectname
Имя объекта метаданных.

relationname
Имя таблицы или представления.

filedname
Имя поля таблицы или представления.

routinename
Имя хранимой процедуры или функции.

paramname
Имя параметра хранимой процедуры или функции.

package_name
Имя пакета.

Оператор COMMENT ON добавляет комментарии для объектов базы данных (метаданных). Комментарии при этом сохраняются в текстовые поля RDB$DESCRIPTION типа BLOB соответствующей системной таблицы (из этих полей клиентское приложение может просмотреть комментарии).

> **Примечание**
>
> Если вы вводите пустой комментарий (''), то он будет сохранен в базе данных как NULL.

Кто может добавить комментарий?

Добавлять комментарий могут:

- SYSDBA;
- Владелец базы данных;
- Владелец объекта, для которого добавляется комментарий;
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры

Пример 5.142. Добавление комментария для текущей базы данных.

```
COMMENT ON DATABASE IS 'Это тестовая (''my.fdb'') БД';
```

Пример 5.143. Добавление комментария для таблицы.

```
COMMENT ON TABLE METALS IS 'Справочник металлов';
```

Пример 5.144. Добавление комментария для поля таблицы.

```
COMMENT ON COLUMN METALS.ISALLOY
IS '0 = чистый металл, 1 = сплав';
```

Пример 5.145. Добавление комментария для параметра процедуры.

```
COMMENT ON PARAMETER ADD_EMP_PROJ.EMP_NO
IS 'Код сотрудника';
```

Пример 5.146. Добавление комментария для пакета, его процедур и функций, и их параметров.

```
COMMENT ON PACKAGE APP_VAR IS 'Переменные приложения';

COMMENT ON FUNCTION APP_VAR.GET_DATEBEGIN
IS 'Возвращает дату начала периода';

COMMENT ON PROCEDURE APP_VAR.SET_DATERANGE
IS 'Установка диапазона дат';

COMMENT ON
PROCEDURE PARAMETER APP_VAR.SET_DATERANGE.ADATEBEGIN
IS 'Дата начала';
```

Глава 6

Операторы DML

SELECT

Назначение: Выборка данных.

Доступно в: DSQL, ESQL, PSQL.

Синтаксис:

```
[WITH [RECURSIVE] <cte> [, <cte> ...]]
SELECT
  [FIRST m] [SKIP n]
  [DISTINCT | ALL] <columns>
FROM
  source [[AS] alias]
  [<joins>]
[WHERE <condition>]
[GROUP BY <grouping-list>
[HAVING <aggregate-condition>]]
[PLAN <plan-expr>]
[UNION [DISTINCT | ALL] <other-select>]
[ORDER BY <ordering-list>]
[   {ROWS m [TO n]}
  | {[OFFSET n {ROW | ROWS}]
     [FETCH {FIRST | NEXT} [m] {ROW | ROWS} ONLY]}
]
[FOR UPDATE [OF <columns>]]
[WITH LOCK]
[INTO <variables>]

<variables> ::= [:]varname [, [:]varname ...]
```

Описание:

Оператор (команда) SELECT извлекает данные из базы данных и передаёт их в приложение или в вызывающую SQL команду. Данные возвращаются в виде набора строк (которых может быть 0 или больше), каждая строка содержит один или более столбцов или полей. Совокупность возвращаемых строк является результирующим набором данных команды.

Следующие части команды SELECT являются обязательными:

- Ключевое слово SELECT, за которым следует список полей. Эта часть определяет, что запрашивается из базы данных;
- Ключевое слово FROM, за которым следует объект выборки (например, таблица). Эта часть сообщает серверу, где следует искать запрашиваемые данные.

В простейшей форме SELECT извлекает ряд полей из единственной таблицы, например:

```
SELECT id, name, address
FROM contacts
```

Или, для того чтобы извлечь все поля таблицы:

```
SELECT * FROM contacts
```

На практике команда SELECT обычно выполняется с выражением WHERE, которое ограничивает возвращаемый набор данных. Также, полученный набор данных обычно сортируется с помощью выражения ORDER BY, дополнительно ограничивается (с целью организации постраничного просмотра данных) выражениями FIRST ... SKIP, OFFSET ... FETCH или ROWS.

Список полей может содержать различные типы выражений вместо имён полей, а источник необязательно должен быть таблицей или представлением, он так же может быть производной таблицей (derived table), общим табличным выражением (CTE) или селективной хранимой процедурой.

Несколько источников данных могут быть соединены с помощью выражения JOIN, и несколько результирующих наборов данных могут быть скомбинированы с использованием выражения UNION.

В следующих секциях мы подробно рассмотрим все выражения для команды SELECT и их использование.

FIRST, SKIP

Назначение: Получение части строк из упорядоченного набора.

Синтаксис:

```
SELECT [FIRST <m>] [SKIP <n>]
FROM ...
...

<m>, <n> ::=
    integer literal
  | query parameter
  | (integer-expression)
```

Параметры предложений FIRST и SKIP

integer literal
: Целочисленный литерал.

query parameter
: Параметр запроса. ? — в DSQL и :*paramname* — в PSQL.

integer-expression
Выражение, возвращающее целочисленное значение.

> **Важно**
>
> FIRST и SKIP используются только в Firebird, они не включены в стандарт SQL. Рекомендуется использовать OFFSET, FETCH везде, где это возможно.

Описание

Выражение FIRST *<m>* ограничивает результирующий набор данным указанным числом записей.

Выражение SKIP *<n>* пропускает указанное число записей перед выдачей результирующего набора данных.

Когда эти выражения используются совместно, например FIRST *m* SKIP *n*, то в результате *n* записей будет пропущено и, из оставшихся, *m* записей будет возвращено в результирующем наборе данных.

FIRST и SKIP являются необязательными выражениями.

Особенности использования

- Разрешается использовать SKIP 0 – в этом случае 0 записей будет пропущено;
- В случае использования FIRST 0 будет возвращён пустой набор записей;
- Отрицательные значения FIRST и SKIP вызовут ошибку;
- Если указанное в SKIP значение превышает размер результирующего набора данных, то вернётся пустой набор данных;
- Если число записей в наборе данных (или остаток после применения SKIP) меньше, чем заданное в FIRST значение, то соответственно меньшее количество записей будет возвращено;
- Любой аргумент FIRST или SKIP, который не является целым числом или параметром SQL должен был заключён в круглые скобки. Это, означает, что в случае использования вложенной команды SELECT в качестве параметра для FIRST или SKIP, он должен быть вложен в две пары скобок.

Примеры

Следующий запрос вернёт первые 10 имён из таблицы PEOPLE (имена также будут отсортированы, см. ниже раздел ORDER BY):

```
SELECT FIRST 10 id, name
FROM People
ORDER BY name ASC
```

Следующий запрос вернёт все записи из таблицы PEOPLE, за исключением первых 10 имён:

```
SELECT SKIP 10 id, name
FROM People
ORDER BY name ASC
```

А этот запрос вернёт последние 10 записей (обратите внимание на двойные скобки):

```
SELECT SKIP ((SELECT COUNT(*) - 10 FROM People))
  id, name
FROM People
ORDER BY name ASC
```

Этот запрос вернёт строки 81-100 из таблицы PEOPLE:

```
SELECT FIRST 20 SKIP 80 id, name
FROM People
ORDER BY name ASC
```

См. также: FETCH, OFFSET, ROWS.

Список полей SELECT

Список полей содержит одно или более выражений, разделённых запятыми. Результатом каждого выражения является значение соответствующего поля в наборе данных команды SELECT. Исключением является выражение * («звёздочка»), которое возвращает все поля отношения.

Синтаксис:

```
SELECT   [...]
[DISTINCT | ALL] <output-column> [, <output-column> ...]
[...]
FROM ...

<output-column> ::=
    [qualifier.]*
  | <value-expression> [COLLATE collation] [[AS] alias]

<value-expression> ::=
    [qualifier.]table-column
  | [qualifier.]view-column
  | [qualifier.]selectable-SP-outparm
  | constant
  | NULL
  | context-variable
  | function-call
  | single-value-subselect
  | CASE-construct
  | other-single-value-expr
```

Параметры списка полей оператора SELECT

qualifier
: Имя таблицы (представления) или псевдоним таблицы (представления, хранимой процедуры, производной таблицы).

collation
Существующее имя сортировки (только для столбцов символьных типов).

alias
Псевдоним поля.

table-column
Столбец таблицы.

view-column
Столбец представления.

selectable-SP-outparm
Выходной параметр селективной хранимой процедуры.

constant
Константа.

context-variable
Контекстная переменная.

function-call
Вызов скалярной, агрегатной или оконной функции.

single-value-subselect
Подзапрос, возвращающий единственное скалярное значение (сингелтон).

CASE-construct
Конструкция CASE.

other-single-value-expr
Любое другое выражение, возвращающее единственное значение типа данных Firebird или NULL.

Описание

Хорошим тоном является указание полного имени поля вместе с именем алиаса или таблицы/представления/хранимой процедуры, к которой это поле принадлежит.

Указание полного имени становится **обязательным** в случае, если поле с одним и тем же именем находится в более чем одной таблице, участвующей в объединении.

> **Обратите внимание**
>
> Алиасы (псевдонимы) заменяют оригинальное имя таблицы/ представления/ хранимой процедуры: как только определён алиас для соответствующего отношения, использовать оригинальное имя нельзя.

В начало списка полей могут быть добавлены ключевые слова DISTINCT или ALL:

- DISTINCT удаляет дубликаты строк: то есть, если две или более записей содержат одинаковые значения во всех соответствующих полях, только одна из этих строк будет включена в результирующий набор данных.

- ALL включает все строки в результирующий набор данных. ALL включено по умолчанию и поэтому редко используется: явное указание поддерживается для совместимости со стандартом SQL.

Выражение COLLATE не изменяет содержимое поля, однако, если указать COLLATE для определённого поля, то это может изменить чувствительность к регистру символов или к акцентам (accent sensitivity), что, в свою очередь, может повлиять на:

- Порядок сортировки, в случае если это поле указано в выражении ORDER BY;
- Группировку, в случае если это поле указано в выражении GROUP BY;
- Количество возвращаемых строк, если используется DISTINCT.

Примеры операторов SELECT с различными типами полей

Простой SELECT использующий только имена полей:

```
SELECT cust_id, cust_name, phone
FROM customers
WHERE city = 'London'
```

Запрос с конкатенацией и вызовом функции в списке полей:

```
SELECT
  'Mr./Mrs. ' || lastname,
  street,
  zip,
  upper(city)
FROM contacts
WHERE date_last_purchase(id) = current_date
```

Запрос с двумя подзапросами:

```
SELECT
  p.fullname,
  (SELECT name FROM classes c
   WHERE c.id = p.class) AS class,
  (SELECT name FROM mentors m
   WHERE m.id = p.mentor) AS mentor
FROM pupils p
```

Следующий запрос делает то же самое, что и предыдущий, только с использованием соединения таблиц (JOIN) вместо подзапросов:

```
SELECT
  p.fullname,
  c.name AS class,
  m.name AS mentor
FROM pupils p
```

```
  JOIN classes c ON c.id = p.class
  JOIN mentors m ON m.id = p.mentor
```

Этот запрос использует конструкцию CASE для определения корректного обращения, например, при рассылке сообщений конкретному человеку:

```
SELECT
  CASE upper(sex)
    WHEN 'F' THEN 'Mrs.'
    WHEN 'M' THEN 'Mr.'
    ELSE ''
  END AS title,
  lastname,
  address
FROM employees
```

Запрос с использованием оконной функции. Выводит сотрудников отранжированных по заработной плате.

```
SELECT
    id,
    salary,
    name,
    DENSE_RANK() OVER(ORDER BY salary) AS EMP_RANK
FROM employees
ORDER BY salary;
```

Запрос к хранимой процедуре:

```
SELECT *
FROM interesting_transactions(2010, 3, 'S')
ORDER BY amount
```

Выборка полей производной таблицы. Производная таблица – это заключённый в скобки оператор SELECT, результат которого используется в запросе уровнем выше, как будто является обычной таблицей или представлением.

```
SELECT
  fieldcount,
  COUNT(relation) AS num_tables
FROM
  (SELECT
     r.rdb$relation_name AS relation,
     COUNT(*) AS fieldcount
   FROM rdb$relations r
     JOIN rdb$relation_fields rf
       ON rf.rdb$relation_name = r.rdb$relation_name
   GROUP BY relation)
```

```
GROUP BY fieldcount
```

Запрос к контекстной переменной (CURRENT_TIME):

```
SELECT current_time FROM rdb$database
```

Для тех, кто не знаком с RDB$DATABASE: это системная таблица, которая всегда существует во всех базах данных Firebird и всегда содержит только одну строку. И, хотя эта таблица не была создана специально для этой цели, стало распространённой практикой среди разработчиков Firebird выполнять запросы к этой таблице в случае, если нужно выполнить запрос, не привязанный ни к какой таблице, в котором результат получается из выражений, указанных в списке полей оператора SELECT. Например:

```
SELECT
  power(12, 2) AS twelve_squared,
  power(12, 3) AS twelve_cubed
FROM rdb$database
```

И, наконец, пример запроса к самой таблице RDB$DATABASE, с помощью которого можно получить кодировку по умолчанию данной БД:

```
SELECT rdb$character_set_name FROM rdb$database
```

См. также: Скалярные функции, Агрегатные функции, Оконные (аналитические) функции, Контекстные переменные, CASE, Подзапросы.

FROM

Выражение FROM определяет источники, из которых будут отобраны данные. В простейшей форме, это может быть единственная таблица или представление. Однако источниками также могут быть хранимая процедура, производная таблица или общее табличное выражение (CTE). Различные виды источников могут комбинироваться с использованием разнообразных видов соединений (JOIN).

Этот раздел посвящён запрос из единственного источника. Соединения рассматриваются в следующем разделе.

Синтаксис:

```
SELECT
...
FROM <source>
[<joins>]
[...]

<source> ::= {
```

```
    table
  | view
  | selectable-stored-procedure [(args)]
  | <derived-table>
  | <common-table-expression>
} [[AS] alias]

<derived-table> ::= (select-statement) [[AS] alias] [(<column-aliases>)]

<common-table-expression> ::= WITH [RECURSIVE]
  <cte-def> [, <cte-def> ...]
  select-statement

<cte-def> ::= name [(<column-aliases>)] AS (select-statement)

<column-aliases> ::= column-alias [, column-alias ...]
```

Параметры предложения FROM

table
: Таблица.

view
: Представление.

selectable-stored-procedure
: Селективная хранимая процедура.

args
: Аргументы селективной хранимой процедуры.

derived-table
: Производная таблица.

cte-def
: Общее табличное выражение (CTE).

select-statement
: Произвольный SELECT запрос.

column-alias
: Алиас столбца CTE или производной таблицы.

name
: Имя CTE.

alias
: Псевдоним (алиас) для одного из источников данных (таблицы, представления, процедуры, CTE, производной таблицы).

Выборка из таблицы или представления

При выборке из таблицы или представления предложение FROM не требует ничего кроме его имени. Псевдоним (алиас) может быть полезен или даже необходим при использовании

подзапросов, которые соотнесены с главным запросом (обычно подзапросы являются коррелированными).

Примеры

```
SELECT id, name, sex, age
FROM actors
WHERE state = 'Ohio'

SELECT *
FROM birds
WHERE type = 'flightless'
ORDER BY family, genus, species

SELECT
  firstname,
  middlename,
  lastname,
  date_of_birth,
  (SELECT name FROM schools s WHERE p.school = s.id) schoolname
FROM pupils p
WHERE year_started = 2012
ORDER BY schoolname, date_of_birth
```

Важно

Если вы дадите таблице или представлению псевдоним (алиас), то вы должны везде использовать этот псевдоним, а не имя таблицы, при обращении к именам столбцов.

Корректное использование:

```
SELECT PEARS
FROM FRUIT

SELECT FRUIT.PEARS
FROM FRUIT

SELECT PEARS
FROM FRUIT F

SELECT F.PEARS
FROM FRUIT F
```

Некорректное использование:

```
SELECT FRUIT.PEARS
FROM FRUIT F
```

Выборка из селективной хранимой процедуры

Селективная хранимая процедура (т.е. с возможностью выборки) должна удовлетворять следующим условиям:

- Содержать, по крайней мере, один выходной параметр;
- Использовать ключевое слово SUSPEND таким образом, чтобы вызывающий запрос могу выбирать выходные строки одну за другой, также как выбираются строки таблицы или представления.

Выходные параметры селективной хранимой процедуры с точки зрения команды SELECT соответствуют полям обычной таблицы.

Выборка из хранимой процедуры без входных параметров осуществляется точно так же, как обычная выборка из таблицы:

```
SELECT *
FROM suspicious_transactions
WHERE assignee = 'Dmitrii'
```

Если хранимая процедура требует входные параметры, то они должны быть указаны в скобках после имени процедуры:

```
SELECT name, az, alt
FROM visible_stars('Brugge', current_date, '22:30')
WHERE alt >= 20
ORDER BY az, alt
```

Значения для опциональных параметров (то есть, параметров, для которых определены значения по умолчанию) могут быть указаны или опущены.

Однако если параметры задаются частично, то пропущенные параметры должны быть в конце перечисления внутри скобок.

Если предположить, что процедура visible_stars из предыдущего примера имеет два опциональных параметра spectral_class (varchar(12)) и min_magn (numeric(3,1)), то следующие команды будут корректными:

```
SELECT name, az, alt
FROM visible_stars('Brugge', current_date, '22:30')

SELECT name, az, alt
FROM visible_stars('Brugge', current_date, '22:30', 4.0)
```

А вот этот запрос не будет корректным:

```
SELECT name, az, alt
FROM visible_stars('Brugge', current_date, 4.0)
```

Алиас для селективной хранимой процедуры указывается после списка параметров:

```
SELECT
  number,
  (SELECT name FROM contestants c
   WHERE c.number = gw.number)
FROM get_winners('#34517', 'AMS') gw
```

Если вы указываете поле (выходной параметр) с полным именем процедуры, не включайте в это имя список параметров процедуры:

```
SELECT number,
  (SELECT name FROM contestants c
   WHERE c.number = get_winners.number)
FROM get_winners('#34517', 'AMS')
```

См. также: Хранимые процедуры, CREATE PROCEDURE.

Выборка из производной таблицы (derived table)

Производная таблица — это корректная команда SELECT, заключённая в круглые скобки, опционально обозначенная псевдонимом таблицы и псевдонимами полей.

Синтаксис:

```
(select-query) [[AS] derived-table-alias] [(<derived-column-aliases>)]

<derived-column-aliases> := column-alias [, column-alias ...]
```

Возвращаемый набор данных такого оператора представляет собой виртуальную таблицу, к которой можно составлять запросы, так как будто это обычная таблица.

Производная таблица в запросе ниже выводит список имён таблиц в базе данных и количество столбцов в них. Запрос к производной таблице выводит количество полей, и количество таблиц с таким количеством полей.

```
SELECT
  FIELDCOUNT,
  COUNT(RELATION) AS NUM_TABLES
FROM (SELECT
        R.RDB$RELATION_NAME RELATION,
        COUNT(*) AS FIELDCOUNT
      FROM RDB$RELATIONS R
        JOIN RDB$RELATION_FIELDS RF
          ON RF.RDB$RELATION_NAME = R.RDB$RELATION_NAME
      GROUP BY RELATION)
GROUP BY FIELDCOUNT
```

Тривиальный пример, демонстрирующий использование псевдонима производной таблицы и списка псевдонимов столбцов (оба опциональные):

```
SELECT
  DBINFO.DESCR, DBINFO.DEF_CHARSET
FROM (SELECT *
      FROM RDB$DATABASE) DBINFO (DESCR, REL_ID, SEC_CLASS, DEF_CHARSET)
```

Примечания:

- Производные таблицы могут быть вложенными;
- Производные таблицы могут быть объединениями и использоваться в объединениях. Они могут содержать агрегатные функции, подзапросы и соединения, и сами по себе могут быть использованы в агрегатных функциях, подзапросах и соединениях. Они также могут быть хранимыми процедурами или запросами из них. Они могут иметь предложения WHERE, ORDER BY и GROUP BY, указания FIRST, SKIP или ROWS и т.д.;
- Каждый столбец в производной таблице должен иметь имя. Если этого нет по своей природе (например, потому что это — константа), то надо в обычном порядке присвоить псевдоним или добавить список псевдонимов столбцов в спецификации производной таблицы;
- Список псевдонимов столбцов опциональный, но если он присутствует, то должен быть полным (т.е. он должен содержать псевдоним для каждого столбца производной таблицы);
- Оптимизатор может обрабатывать производные таблицы очень эффективно. Однако если производная таблица включена во внутреннее соединение и содержит подзапрос, то никакой порядок соединения не может быть использован оптимизатором.

Приведём пример того, как использование производных таблиц может упростить решение некоторой задачи.

Предположим, что у нас есть таблица COEFFS, которая содержит коэффициенты для ряда квадратных уравнений, которые мы собираемся решить. Она может быть определена примерно так:

```
CREATE TABLE coeffs (
  a DOUBLE PRECISION NOT NULL,
  b DOUBLE PRECISION NOT NULL,
  c DOUBLE PRECISION NOT NULL,
  CONSTRAINT chk_a_not_zero CHECK (a <> 0)
)
```

В зависимости от значений коэффициентов a, b и c, каждое уравнение может иметь ноль, одно или два решения. Мы можем найти эти решения с помощью одноуровневого запроса к таблице COEFFS, однако код такого запроса будет громоздким, а некоторые значения (такие, как дискриминанты) будут вычисляться несколько раз в каждой строке.

Если использовать производную таблицу, то запрос можно сделать гораздо более элегантным:

```
SELECT
  IIF (D >= 0, (-b - sqrt(D)) / denom, NULL) AS sol_1,
  IIF (D > 0, (-b + sqrt(D)) / denom, NULL) AS sol_2
FROM
  (SELECT b, b*b - 4*a*c, 2*a FROM coeffs) (b, D, denom)
```

Если мы захотим показывать коэффициенты рядом с решениями уравнений, то мы можем модифицировать запрос следующим образом:

```
SELECT
  a, b, c,
  IIF (D >= 0, (-b - sqrt(D)) / denom, NULL) sol_1,
  IIF (D > 0, (-b + sqrt(D)) / denom, NULL) sol_2
FROM
  (SELECT a, b, c, b*b - 4*a*c AS D, 2*a AS denom
   FROM coeffs)
```

Обратите внимание, что в первом запросе мы назначили алиасы для всех полей производной таблицы в виде списка после таблицы, а во втором, по мере необходимости, добавляем алиасы внутри запроса производной таблицы. Оба этих метода корректны, так как при правильном применении гарантируют, что каждое поле производной таблицы имеет уникальное имя.

Примечание

На самом деле все столбцы, вычисляемые в производной таблице, будут перевычислены столько раз, сколько раз они указываются в основном запросе. Это важно может привести к неожиданным результатам при использовании недетерминированных функций. Следующий пример показывает сказанное:

```
SELECT
    UUID_TO_CHAR(X) AS C1,
    UUID_TO_CHAR(X) AS C2,
    UUID_TO_CHAR(X) AS C3
FROM (SELECT GEN_UUID() AS X
      FROM RDB$DATABASE) T;
```

результатом этого запроса будет

```
C1                              80AAECED-65CD-4C2F-90AB-5D548C3C7279
C2                              C1214CD3-423C-406D-B5BD-95BF432ED3E3
C3                              EB176C10-F754-4689-8B84-64B666381154
```

Для материализации результата функции GEN_UUID вы можете воспользоваться следующим способом:

```
SELECT
    UUID_TO_CHAR(X) AS C1,
    UUID_TO_CHAR(X) AS C2,
    UUID_TO_CHAR(X) AS C3
FROM (SELECT GEN_UUID() AS X
      FROM RDB$DATABASE
      UNION ALL
      SELECT NULL FROM RDB$DATABASE WHERE 1=0) T;
```

результатом этого запроса будет

```
C1                              80AAECED-65CD-4C2F-90AB-5D548C3C7279
C2                              80AAECED-65CD-4C2F-90AB-5D548C3C7279
C3                              80AAECED-65CD-4C2F-90AB-5D548C3C7279
```

или завернуть функцию GEN_UUID в подзапрос

```
SELECT
    UUID_TO_CHAR(X) AS C1,
    UUID_TO_CHAR(X) AS C2,
    UUID_TO_CHAR(X) AS C3
FROM (SELECT
          (SELECT GEN_UUID() FROM RDB$DATABASE) AS X
      FROM RDB$DATABASE) T;
```

Эта особенность текущей реализации и она может быть изменена в следующих версиях сервера.

Выборка из общих табличных выражений (CTE)

Общие табличные выражения являются более сложной и более мощной вариацией производных таблиц. CTE состоят из преамбулы, начинающейся с ключевого слова WITH, которая определяет одно или более общих табличных выражений (каждое из которых может иметь список алиасов полей). Основной запрос, который следует за преамбулой, может обращаться к CTE так, как будто обычные таблицы. CTE доступны только в рамках основного запроса.

Подробно CTE описываются в разделе Общие табличные выражения CTE (WITH ... AS ... SELECT), а здесь приведены лишь некоторые примеры использования.

Следующий запрос представляет наш пример с производной таблицей в варианте для общих табличных выражений:

```
WITH vars (b, D, denom) AS (
  SELECT b, b*b - 4*a*c, 2*a
  FROM coeffs
)
SELECT
  IIF (D >= 0, (-b - sqrt(D)) / denom, NULL) AS sol_1,
  IIF (D > 0, (-b + sqrt(D)) / denom, NULL) AS sol_2
FROM vars
```

Это не слишком большое улучшение по сравнению с вариантом с производными таблицами (за исключением того, что вычисления проводятся до основного запроса). Мы можем ещё улучшить запрос, исключив двойное вычисление sqrt(D) для каждой строки:

```
WITH vars (b, D, denom) AS (
  SELECT b, b*b - 4*a*c, 2*a
  FROM coeffs
),
vars2 (b, D, denom, sqrtD) AS (
  SELECT
    b, D, denom,
    IIF (D >= 0, sqrt(D), NULL)
  FROM vars
)
SELECT
  IIF (D >= 0, (-b - sqrtD) / denom, NULL) AS sol_1,
  IIF (D > 0, (-b + sqrtD) / denom, NULL) AS sol_2
FROM vars2
```

Текст запроса выглядит более сложным, но он стал более эффективным (предполагая, что исполнение функции SQRT занимает больше времени, чем передача значений переменных b, d и denom через дополнительное CTE).

Примечание

На самом деле все столбцы, вычисляемые в CTE, будут перевычислены столько раз, сколько раз они указываются в основном запросе. Это важно может привести к неожиданным результатам при использовании недетерминированных функций. Следующий пример показывает сказанное:

```
WITH T(X)
AS (SELECT GEN_UUID()
    FROM RDB$DATABASE)
SELECT
    UUID_TO_CHAR(X) as c1,
    UUID_TO_CHAR(X) as c2,
    UUID_TO_CHAR(X) as c3
FROM T
```

результатом этого запроса будет

```
C1                              80AAECED-65CD-4C2F-90AB-5D548C3C7279
C2                              C1214CD3-423C-406D-B5BD-95BF432ED3E3
C3                              EB176C10-F754-4689-8B84-64B666381154
```

Для материализации результата функции GEN_UUID вы можете воспользоваться следующим способом:

```
WITH T(X)
AS (SELECT GEN_UUID()
    FROM RDB$DATABASE
    UNION ALL
    SELECT NULL FROM RDB$DATABASE WHERE 1=0)
SELECT
    UUID_TO_CHAR(X) as c1,
    UUID_TO_CHAR(X) as c2,
    UUID_TO_CHAR(X) as c3
FROM T;
```

результатом этого запроса будет

```
C1                              80AAECED-65CD-4C2F-90AB-5D548C3C7279
C2                              80AAECED-65CD-4C2F-90AB-5D548C3C7279
C3                              80AAECED-65CD-4C2F-90AB-5D548C3C7279
```

или завернуть функцию GEN_UUID в подзапрос

```
WITH T(X)
AS (SELECT (SELECT GEN_UUID() FROM RDB$DATABASE)
    FROM RDB$DATABASE)
SELECT
    UUID_TO_CHAR(X) as c1,
    UUID_TO_CHAR(X) as c2,
    UUID_TO_CHAR(X) as c3
FROM T;
```

Эта особенность текущей реализации и она может быть изменена в следующих версиях сервера.

Конечно, мы могли бы добиться такого результата и с помощью производных таблиц, но это потребовало бы вложить запросы один в другой.

См. также: Общие табличные выражения CTE (WITH ... AS ... SELECT).

Соединения (JOINS)

Соединения объединяют данные из двух источников в один набор данных. Соединение данных осуществляется для каждой строки и обычно включает в себя проверку условия соединения (join condition) для того, чтобы определить, какие строки должны быть объединены и оказаться в результирующем наборе данных.

Результат соединения также может быть соединён с другим набором данных с помощью следующего соединения.

Существует несколько типов (INNER, OUTER) и классов (квалифицированные, натуральные, и др.) соединений, каждый из которых имеет свой синтаксис и правила.

Синтаксис:

```
SELECT
...
FROM <source>
[<joins>]
[...]

<source> ::= {
    table
  | view
  | selectable-stored-procedure [(args)]
  | derived-table
  | common-table-expression
} [[AS] alias]

<joins> ::= <join> [<join> ...]

<join> ::=
    [<join-type>] JOIN <source> <join-condition>
  | NATURAL [<join-type>] JOIN <source>
  | {CROSS JOIN | ,} <source>

<join-type> ::= INNER | {LEFT | RIGHT | FULL} [OUTER]

<join-condition> ::= ON condition | USING (column-list)
```

Параметры предложения JOIN

table
Таблица.

view
Представление.

selectable-stored-procedure
Селективная хранимая процедура.

args
Аргументы селективной хранимой процедуры.

derived-table
Производная таблица.

common-table-expression
Общее табличное выражение (CTE).

alias
Псевдоним (алиас) для одного из источников данных (таблицы, представления, процедуры, CTE, производной таблицы).

condition
Условие соединения.

column-list
Список столбцов по которым происходит эквисоединение.

Внутренние (INNER) и внешние (OUTER) соединения

Соединение всегда соединяет строки из двух наборов данных (которые обычно называются "левый" и "правый"). По умолчанию, только строки, которые удовлетворяет условию соединения (те, которым соответствует хотя бы одна строка из другого набора строк согласно применяемому условию) попадают в результирующий набор данных. Такой тип соединения (который является типом по умолчанию) называется внутренним (INNER JOIN).

Предположим, у нас есть 2 таблицы:

Таблица A:

ID	S
87	Just some text
35	Silence

Таблица B:

CODE	X
-23	56.7735
87	416.0

Если мы соединим эти таблицы с помощью вот такого запроса:

```
SELECT *
FROM A
```

```
JOIN B ON A.id = B.code
```

то результат будет:

ID	S	CODE	X
87	Just some text	87	416.0

То есть, первая строка таблицы A была соединена со второй строкой таблицы B, потому что вместе они удовлетворяют условию соединения "A.id = B.code". Другие строки не имеют соответствия и поэтому не включаются в соединение. Помните, что умолчанию соединение всегда внутреннее (INNER).

Мы можем сделать это явным, указав тип соединения:

```
SELECT *
FROM A
INNER JOIN B ON A.id = B.code
```

но обычно слово INNER опускается.

Разумеется, возможны случаи, когда строке в левом наборе данных соответствует несколько строк в правом наборе данных (или наоборот).

В таких случаях все комбинации включаются в результирующих набор данных, и мы можем получить результат вроде этого:

ID	S	CODE	X
87	Just some text	87	416.0
87	Just some text	87	-1.0
-23	Don't know	-23	56.7735
-23	Still don't know	-23	56.7735
-23	I give up	-23	56.7735

Иногда необходимо включить в результат все записи из левого или правого набора данных, вне зависимости от того, есть ли для них соответствующая запись в парном наборе данных. В этом случае необходимо использовать внешние соединения.

Внешнее левое соединение (LEFT OUTER) включает все записи из левого набора данных, и те записи из правого набора, которые удовлетворяют условию соединения.

Внешнее правое соединение (RIGHT OUTER) включает все записи из правого набора данных и те записи из левого набора данных, которые удовлетворяют условию соединения.

Полное внешнее соединение (FULL OUTER) включает все записи из обоих наборов данных.

Во всех внешних соединениях, "дыры" (то есть поля набора данных, в которых нет соответствующей записи) заполняются NULL.

Для обозначения внешнего соединения используются ключевые слова LEFT, RIGHT или FULL с необязательным ключевым словом OUTER.

Рассмотрим различные внешние соединения на примере запросов с указанными выше таблицами A и B:

```
SELECT *
FROM A
LEFT OUTER JOIN B ON A.id = B.code
```

то же самое

```
SELECT *
FROM A
LEFT JOIN B ON A.id = B.code
```

ID	S	CODE	X
87	Just some text	87	416.0
235	Silence	<null>	<null>

```
SELECT *
FROM A
RIGHT OUTER JOIN B ON A.id = B.code
```

то же самое

```
SELECT *
FROM A
RIGHT JOIN B ON A.id = B.code
```

ID	S	CODE	X
<null>	<null>	-23	56.7735
87	Just some text	87	416.0

```
SELECT *
FROM A
FULL OUTER JOIN B ON A.id = B.code
```

то же самое

```
SELECT *
FROM A
```

```
FULL JOIN B ON A.id = B.code
```

ID	S	CODE	X
<null>	<null>	-23	56.7735
87	Just some text	87	416.0
235	Silence	<null>	<null>

Обычные соединения

Явный синтаксис соединения требует указания условия соединения записей. Это условие указывается явно в предложении ON или неявно при помощи предложения USING.

Синтаксис:

```
<qualified-join> ::= [<join-type>] JOIN <source> <join-condition>

<join-type> ::= INNER | {LEFT | RIGHT | FULL} [OUTER]

<join-condition> ::= ON condition | USING (column-list)
```

Соединения с явными условиями

В синтаксисе явного соединения есть предложение ON, с условием соединения, в котором может быть указано любое логическое выражение, но, как правило, оно содержит условие сравнения между двумя участвующими источниками.

Довольно часто, это условие — проверка на равенство (или ряд проверок на равенство объединённых оператором AND) использующая оператор "=". Такие соединения называются эквисоединениями. (Примеры в главе Внутренние (INNER) и внешние (OUTER) соединения были эквисоединениями).

Примеры соединений с явными условиями:

```
/*
 * Выборка всех заказчиков из города Детройт, которые
 * сделали покупку.
 */
SELECT *
FROM customers c
JOIN sales s ON s.cust_id = c.id
WHERE c.city = 'Detroit'

/*
 * Тоже самое, но включает в выборку заказчиков, которые
 * не совершали покупки.
 */
SELECT *
FROM customers c
```

```
LEFT JOIN sales s ON s.cust_id = c.id
WHERE c.city = 'Detroit'

/*
 * Для каждого мужчины выбрать женщин, которые выше него.
 * Мужчины, для которых такой женщины не существуют,
 * не будут выключены в выборку.
 */
SELECT
    m.fullname AS man,
    f.fullname AS woman
FROM males m
JOIN females f ON f.height > m.height

/*
 * Выборка всех учеников, их класса и наставника.
 * Ученики без наставника буду включены в выборку.
 * Ученики без класса не будут включены в выборку.
 */
SELECT
    p.firstname,
    p.middlename,
    p.lastname,
    c.name,
    m.name
FROM pupils p
JOIN classes c ON c.id = p.class
LEFT JOIN mentors m ON m.id = p.mentor
```

Соединения именованными столбцами

Эквисоединения часто сравнивают столбцы, которые имеют одно и то же имя в обеих таблицах. Для таких соединений мы можем использовать второй тип явных соединений, называемый соединением именованными столбцами (Named Columns Joins). Соединение именованными столбцами осуществляются с помощью предложения USING, в котором перечисляются только имена столбцов.

> **Примечание**
>
> Соединения именованными столбцами доступны только в диалекте 3.

Таким образом, следующий пример:

```
SELECT *
FROM flotsam f
  JOIN jetsam j
    ON f.sea = j.sea AND f.ship = j.ship
```

можно переписать так:

```
SELECT *
FROM flotsam
JOIN jetsam USING (sea, ship)
```

что значительно короче. Результирующий набор несколько отличается, по крайней мере, при использовании "SELECT *":

- Результат соединения с явным условием соединения в предложении ON будет содержать каждый из столбцов SEA и SHIP дважды: один раз для таблицы FLOTSAM и один раз для таблицы JETSAM. Очевидно, что они будут иметь они и те же значения;
- Результат соединения именованными столбцами, с помощью предложения USING, будет содержать эти столбцы один раз.

Если вы хотите получить в результате соединения именованными столбцами все столбцы, перепишите запрос следующим образом:

```
SELECT f.*, j.*
FROM flotsam f
JOIN jetsam j USING (sea, ship)
```

Для внешних (OUTER) соединений именованными столбцами, существуют дополнительные нюансы, при использовании "SELECT *" или неполного имени столбца. Если столбец строки из одного источника не имеет совпадений со столбцом строки из другого источника, но все равно должен быть включён результат из-за инструкций LEFT, RIGHT или FULL, то объединяемый столбец получит не NULL значение. Это достаточно справедливо, но теперь вы не можете сказать из какого набора левого, правого или обоих пришло это значение. Это особенно обманывает, когда значения пришли из правой части набора данных, потому что "*" всегда отображает для комбинированных столбцов значения из левой части набора данных, даже если используется RIGHT соединение.

Является ли это проблемой, зависит от ситуации. Если это так, используйте "f.*, j.*" подход, продемонстрированный выше, где f и j имена или алиасы двух источников. Или лучше вообще избегать "*" в серьёзных запросах и перечислять все имена столбцов для соединяемых множеств. Такой подход имеет дополнительное преимущество, заставляя вас думать, о том какие данные вы хотите получить и откуда.

Вся ответственность за совместимость типов столбцов между соединяемыми источниками, имена которых перечислены в предложении USING, лежит на вас. Если типы совместимы, но не равны, то Firebird преобразует их в тип с более широким диапазоном значений перед сравнением. Кроме того, это будет типом данных объединённого столбца, который появится в результирующем наборе, если используются "SELECT *" или неполное имя столбца. Полные имена столбцов всегда будут сохранять свой первоначальный тип данных.

Естественные соединения (Natural Joins)

Взяв за основу соединения именованными столбцами, следующим шагом будет естественное соединение, которое выполняет эквисоединение по всем одноименным столбцам правой и левой таблицы. Типы данных этих столбцов должны быть совместимыми.

> **Примечание**
>
> Естественные соединения доступны только в диалекте 3.

Синтаксис:

```
<natural-join> ::= NATURAL [<join-type>] JOIN <source>

<join-type> ::= INNER | {LEFT | RIGHT | FULL} [OUTER]
```

Даны две таблицы:

```
CREATE TABLE TA (
    a BIGINT,
    s VARCHAR(12),
    ins_date DATE
);

CREATE TABLE TB (
    a BIGINT,
    descr VARCHAR(12),
    x FLOAT,
    ins_date DATE
);
```

Естественное соединение таблиц TA и TB будет происходить по столбцам a и ins_date и два следующих оператора дадут один и тот же результат:

```
SELECT *
FROM TA
NATURAL JOIN TB;

SELECT *
FROM TA
JOIN TB USING (a, ins_date);
```

Как и все соединения, естественные соединения являются внутренними соединениями по умолчанию, но вы можете превратить их во внешние соединения, указав LEFT, RIGHT или FULL перед ключевым словом JOIN.

> **Внимание**
>
> Если в двух исходных таблицах не будут найдены одноименные столбцы, то будет выполнен CROSS JOIN.

Неявные соединения

В стандарте SQL-89 таблицы, участвующие в соединении, задаются списком с разделяющими запятыми в предложении FROM. Условия соединения задаются в предложении WHERE среди других условий поиска. Такие соединения называются неявными.

Синтаксис неявного соединения может осуществлять только внутренние соединения.

Пример неявного соединения:

```
/*
 * Выборка всех заказчиков из города Детройт, которые
 * сделали покупку.
 */
SELECT *
FROM customers c, sales s
WHERE s.cust_id = c.id AND c.city = 'Detroit'
```

Важно

В настоящее время синтаксис неявных соединений не рекомендуется к использованию.

Смешивание явного и неявного соединения

Смешивание явных и неявных соединений не рекомендуется, но позволяется. Некоторые виды смешивания запрещены в Firebird.

Например, такой запрос вызовет ошибку "Column does not belong to referenced table"

```
SELECT *
FROM
TA, TB
JOIN TC ON TA.COL1 = TC.COL1
WHERE TA.COL2 = TB.COL2
```

Это происходит потому, что явный JOIN не может видеть таблицу TA. Однако следующий запрос будет выполнен без ошибок, поскольку изоляция не нарушена.

```
SELECT *
FROM
TA, TB
JOIN TC ON TB.COL1 = TC.COL1
WHERE TA.COL2 = TB.COL2
```

Перекрёстное соединение (CROSS JOIN)

Перекрёстное соединение или декартово произведение. Каждая строка левой таблицы соединяется с каждой строкой правой таблицы.

Синтаксис:

```
<cross-join> ::= {CROSS JOIN | , } <source>
```

Обратите внимание, что синтаксис с использованием запятой является устаревшим. Он поддерживается только для поддержания работоспособности унаследованного программного кода и может быть удалён в будущих версиях.

Перекрёстное соединение двух наборов эквивалентно их соединению по условию тавтологии (условие, которое всегда верно).

Следующие два запроса дадут один и тот же результат:

```
SELECT *
FROM TA
CROSS JOIN TB;

SELECT *
FROM TA
JOIN TB ON 1 = 1;
```

Перекрёстные соединения являются внутренними соединениями, потому что они отбирают строки, для которых есть соответствие — так уж случилось, что каждая строка соответствует! Внешнее перекрёстное соединение, если бы оно существовало, ничего не добавило бы к результату, потому что внешние соединения добавляют записи, по которым нет соответствия, а они не существуют в перекрёстном соединении.

Перекрёстные соединения редко полезны, кроме случаев, когда вы хотите получить список всех возможных комбинаций двух или более переменных. Предположим, вы продаёте продукт, который поставляется в различных размерах, различных цветов и из различных материалов. Если для каждой переменной значения перечислены в собственной таблице, то этот запрос будет возвращать все комбинации:

```
SELECT
    m.name,
    s.size,
    c.name
FROM materials m
CROSS JOIN sizes s
CROSS JOIN colors c
```

Неоднозначные имена полей в соединениях

Firebird отвергает неполные имена полей в запросе, если эти имена полей существуют в более чем одном наборе данных, участвующих в объединении. Это также верно для внутренних эквисоединений, в которых имена полей фигурируют в предложении ON:

```
SELECT a, b, c
FROM TA
JOIN TB ON TA.a = TB.a
```

Существует одно исключение из этого правила: соединения по именованным столбцам и естественные соединения, которые используют неполное имя поля в процессе подбора, могут использоваться законно. Это же относится и к одноименным объединяемым столбцам. Для соединений по именованным столбцам эти столбцы должны быть перечислены в предложении USING. Для естественных соединений это столбцы, имена которых присутствуют в обеих

таблицах. Но снова замечу, что, особенно во внешних соединениях, плоское имя colname является не всегда тем же самым что left.colname или right.colname. Типы данных могут отличаться, и один из полных столбцов может иметь значение NULL, в то время как другой нет. В этом случае значение в объединённом, неполном столбце может замаскировать тот факт, что одно из исходных значений отсутствует.

Соединения с хранимыми процедурами

Если соединение происходит с хранимой процедурой, которая не коррелирована с другими потоками данных через входные параметры, то нет никаких особенностей.

В противном случае, есть одна особенность: потоки, используемые во входных параметрах, должны быть описаны раньше соединения с хранимой процедурой:

```
SELECT *
FROM MY_TAB
JOIN MY_PROC(MY_TAB.F) ON 1 = 1
```

Запрос же написанный следующим образом вызовет ошибку

```
SELECT *
FROM MY_PROC(MY_TAB.F)
JOIN MY_TAB ON 1 = 1
```

WHERE

Предложение WHERE предназначено для ограничения количества возвращаемых строк, теми которые нас интересуют. Условие после ключевого слова WHERE может быть простым, как проверка "AMOUNT = 3", так и сложным, запутанным выражением, содержащим подзапросы, предикаты, вызовы функций, математические и логические операторы, контекстные переменные и многое другое.

Условие в предложении WHERE часто называют условием поиска, выражением поиска или просто поиск.

В DSQL и ESQL, выражение поиска могут содержать параметры. Это полезно, если запрос должен быть повторен несколько раз с разными значениями входных параметров. В строке SQL запроса, передаваемого на сервер, вопросительные знаки используются как заполнители для параметров. Их называют позиционными параметрами, потому что они не могут сказать ничего кроме как о позиции в строке. Библиотеки доступа часто поддерживают именованные параметры в виде `:id`, `:amount`, `:a` и т.д. Это более удобно для пользователя, библиотека заботится о трансляции именованных параметров в позиционные параметры, прежде чем передать запрос на сервер.

Условие поиска может также содержать локальные (PSQL) или хост (ESQL) имена переменных, предваряемых двоеточием.

Синтаксис:

```
```

```
SELECT ...
FROM ...
[...]
WHERE <search-condition>
[...]
```

Параметры предложения WHERE

search-condition
Логическое выражение возвращающее TRUE, FALSE и возможно UNKNOWN (NULL).

Только те строки, для которых условие поиска истинно будут включены в результирующий набор. Будьте осторожны с возможными получаемыми значениями NULL: если вы отрицаете выражение, дающее NULL с помощью NOT, то результат такого выражения все равно будет NULL и строка не пройдёт. Это демонстрируется в одном из ниже приведённых примеров.

Примеры

```
SELECT genus, species
FROM mammals
WHERE family = 'Felidae'
ORDER BY genus;

SELECT *
FROM persons
WHERE birthyear IN (1880, 1881)
   OR birthyear BETWEEN 1891 AND 1898;

SELECT name, street, borough, phone
FROM schools s
WHERE EXISTS (SELECT * FROM pupils p WHERE p.school = s.id)
ORDER BY borough, street;

SELECT *
FROM employees
WHERE salary >= 10000 AND position <> 'Manager';

SELECT name
FROM wrestlers
WHERE region = 'Europe'
  AND weight > ALL (SELECT weight FROM shot_putters
                    WHERE region = 'Africa');

SELECT id, name
FROM players
WHERE team_id = (SELECT id FROM teams
                 WHERE name = 'Buffaloes');

SELECT SUM (population)
FROM towns
WHERE name LIKE '%dam'
  AND province CONTAINING 'land';

SELECT pass
FROM usertable
```

```
WHERE username = current_user;
```

Следующий пример показывает, что может быть, если условие поиска вычисляется как NULL.

Предположим у вас есть таблица, в которой находятся несколько детских имён и количество шариков, которыми они обладают.

CHILD	MARBLES
Anita	23
Bob E.	12
Chris	<null>
Deirdre	1
Eve	17
Fritz	0
Gerry	21
Hadassah	<null>
Isaac	6

Первое, обратите внимание на разницу между NULL и 0. Известно, что Fritz не имеет шариков вовсе, однако неизвестно количество шариков у Chris и Hadassah.

Теперь, если ввести этот SQL оператор:

```
SELECT LIST(child) FROM marbletable WHERE marbles > 10
```

вы получите имена Anita, Bob E., Eve и Gerry. Все эти дети имеют более чем 10 шариков.

Если вы отрицаете выражение:

```
SELECT LIST(child) FROM marbletable WHERE NOT marbles > 10
```

запрос вернёт Deirdre, Fritz и Isaac. Chris и Hadassah не будут включены в выборку, так как не известно 10 у них шариков или меньше. Если вы измените последний запрос так:

```
SELECT LIST(child) FROM  marbletable WHERE marbles <= 100
```

результат будет тем же самым, поскольку выражение NULL <= 10 даёт UNKNOWN. Это не то же самое что TRUE, поэтому Chris и Hadassah не отображены. Если вы хотите что бы в списке были перечислены все "бедные" дети, то измените запрос следующим образом:

```
SELECT LIST(child)
FROM marbletable
WHERE marbles <= 10 OR marbles IS NULL
```

Теперь условие поиска становится истинным для Chris и Hadassah, потому что условие "marbles is null" возвращает TRUE в этом случае. Фактически, условие поиска не может быть NULL ни для одного из них.

Наконец, следующие два примера SELECT запросов с параметрами в условии поиска. Как определяются параметры запроса и возможно ли это, зависит от приложения. Обратите внимание, что запросы подобные этим не могут быть выполнены немедленно, они должны быть предварительно подготовлены. После того как параметризированный запрос был подготовлен, пользователь (или вызывающий код) может подставить значения параметров и выполнить его многократно, подставляя перед каждым вызовом новые значения параметров. Как вводятся значения параметров, и проходят ли они предобработку зависит от приложения. В GUI средах пользователь, как правило, вводит значения параметров через одно и более текстовых полей, и щелкает на кнопку "Execute", "Run" или "Refresh".

```
SELECT name, address, phone
FROM stores
WHERE city = ? AND class = ?

SELECT *
FROM pants
WHERE model = :model AND size = :size AND color = :col
```

Последний запрос не может быть передан непосредственно к движку сервера, приложение должно преобразовать его в другой формат, отображая именованные параметры на позиционные параметры.

GROUP BY

Предложение GROUP BY соединяет записи, имеющие одинаковую комбинацию значений полей, указанных в его списке, в одну запись. Агрегатные функции в списке выбора применяются к каждой группе индивидуально, а не для всего набора в целом.

Если список выборки содержит только агрегатные столбцы или столбцы, значения которых не зависит от отдельных строк основного множества, то предложение GROUP BY необязательно. Когда предложение GROUP BY опущено, результирующее множество будет состоять из одной строки (при условии, что хотя бы один агрегатный столбец присутствует).

Если в списке выборки содержатся как агрегатные столбцы, так и столбцы, чьи значения зависит от выбираемых строк, то предложение GROUP BY становится обязательным.

Синтаксис:

```
SELECT ...
FROM ...
GROUP BY <grouping-item> [, <grouping-item> ...]
[HAVING <grouped-row-condition>] ...

<grouping-item> ::= <non-aggr-select-item> | <non-aggr-expression>
```

```
<non-aggr-select-item> ::= column-copy | column-alias | column-position
```

Параметры предложения GROUP BY

non-aggr-expression
Любое не агрегатное выражение, которое не включено в список выборки, т.е. невыбираемые столбцы из набора источника или выражения, которые не зависит от набора данных вообще.

column-copy
Дословная копия выражения из списка выбора, не содержащего агрегатной функции.

column-alias
Псевдоним выражения (столбца) из списка выбора, не содержащего агрегатной функции.

column-position
Номер позиции выражения (столбца) из списка выбора, не содержащего агрегатной функции.

Общее правило гласит, что каждый не агрегированный столбец в SELECT списке, должен быть так же включён в GROUP BY список. Вы можете это сделать тремя способами:

1. Копировать выражение дословно из списка выбора, например "class" или "'D:' || upper(doccode)";
2. Указать псевдоним, если он существует;
3. Задать положение столбца в виде целого числа, которое находится в диапазоне от 1 до количества столбцов в списке SELECT. Целые значения, полученные из выражений, параметров или просто инварианты будут использоваться в качестве таковых в группировке. Они не будут иметь никакого эффекта, поскольку их значение одинаково для каждой строки.

> **Важно**
>
> Если вы группируете по позиции столбца или алиасу, то выражение соответствующее этой позиции (алиасу) будет скопировано из списка выборки SELECT. Это касается и подзапросов, таким образом, подзапрос будет выполняться, по крайней мере, два раза.

В дополнении к требуемым элементам, список группировки так же может содержать:

- Столбцы исходной таблицы, которые не включены в список выборки SELECT, или неагрегатные выражения, основанные на таких столбцах. Добавление таких столбцов может дополнительно разбить группы. Но так как эти столбцы не в списке выборки SELECT, вы не можете сказать, какому значению столбца соответствует значение агрегированной строки. Таким образом, если вы заинтересованы в этой информации, вы так же должны включить этот столбец или выражение в список выборки SELECT, что возвращает вас к правилу "каждый не агрегированный столбце в списке выборки SELECT должен быть включён в список группировки GROUP BY";
- Выражения, которые не зависят от данных из основного набора, т.е. константы, контекстные переменные, некоррелированные подзапросы, возвращающие единственное значение и т.д. Это упоминается только для полноты картины, т.к. добавление этих элементов является абсолютно бессмысленным, поскольку они не повлияют на группировку вообще. "Безвредные, но бесполезные" элементы так же могут фигурировать в списке выбора SELECT без их копирования в список группировки GROUP BY.

Примеры

Когда в списке выбора SELECT содержатся только агрегатные столбцы, предложение GROUP BY необязательно:

```
SELECT COUNT(*), AVG(age)
FROM students
WHERE sex = 'M'
```

Этот запрос вернёт одну строку с указанием количества студентов мужского пола и их средний возраст. Добавление выражения, которое не зависит от строк таблицы STUDENTS, ничего не меняет:

```
SELECT COUNT(*), AVG(age), current_date
FROM students
WHERE sex = 'M'
```

Теперь строка результата будет иметь дополнительный столбец, отображающий текущую дату, но кроме этого, ничего фундаментального не изменилось. Группировка по-прежнему не требуется.

Тем не менее, в обоих приведённых выше примерах это разрешено. Это совершенно справедливо и для запроса:

```
SELECT COUNT(*), AVG(age)
FROM students
WHERE sex = 'M'
GROUP BY class
```

и вернёт результат для каждого класса, в котором есть мальчики, перечисляя количество мальчиков и их средний возраст в этой конкретном классе. Если вы также оставите поле CURRENT_DATE, то это значение будет повторяться на каждой строке, что не интересно.

Этот запрос имеет существенный недостаток, хотя он даёт вам информацию о различных классах, но не говорит вам, какая строка к какому классу относится. Для того чтобы получить эту дополнительную часть информации, не агрегатный столбец CLASS должен быть добавлен в список выборки SELECT:

```
SELECT class, COUNT(*), AVG(age)
FROM students
WHERE sex = 'M'
GROUP BY class
```

Теперь у нас есть полезный запрос. Обратите внимание, что добавление столбца CLASS делает предложение GROUP BY обязательным. Мы не можем удалить это предложение, так же мы не можем удалить столбец CLASS из списка столбцов.

Результат последнего запроса будет выглядеть примерно так:

CLASS	COUNT	AVG
2A	12	13.5
2B	9	13.9
3A	11	14.6
3B	12	14.4
...	...	...

Заголовки "COUNT" и "AVG" не очень информативны. В простейшем случае вы можете обойти это, но лучше, если мы дадим им значимые имена с помощью псевдонимов:

```
SELECT
    class,
    COUNT(*) AS num_boys,
    AVG(age) AS boys_avg_age
FROM students
WHERE sex = 'M'
GROUP BY class
```

Как вы помните из формального синтаксиса списка столбцов, ключевое слово AS не является обязательным.

Добавление большего не агрегированных (или точнее строчно зависимых) столбцов требуется добавления их в предложения GROUP BY тоже. Например, вы хотите видеть вышеуказанную информацию о девочках то же, и хотите видеть разницу между интернатами и студентами дневного отделения:

```
SELECT
    class,
    sex,
    boarding_type,
    COUNT(*) AS anumber,
    AVG(age) AS avg_age
FROM students
GROUP BY class, sex, boarding_type
```

CLASS	SEX	BOARDING_TYPE	ANUMBER	AVG_AGE
2A	F	BOARDING	9	13.3
2A	F	DAY	6	13.5
2A	M	BOARDING	7	13.6
2A	M	DAY	5	13.4
2B	F	BOARDING	11	13.7
2B	F	DAY	5	13.7

CLASS	SEX	BOARDING_TYPE	ANUMBER	AVG_AGE
2B	M	BOARDING	6	13.8
...	...	...	...	...

Каждая строка в результирующем наборе соответствует одной конкретной комбинации переменных CLASS, SEX и BOARDING_TYPE. Агрегированные результаты — количество и средний возраст — приведены для каждой из конкретизированной группы отдельно. В результате запроса вы не можете увидеть обобщённые результаты для мальчиков отдельно или для студентов дневного отделения отдельно. Таким образом, вы должны найти компромисс. Чем больше вы добавляете неагрегатных столбцов, тем больше вы конкретизируете группы, и тем больше вы упускаете общую картину из виду. Конечно, вы все ещё можете получить "большие" агрегаты, с помощью отдельных запросов.

HAVING

Так же, как и предложение WHERE ограничивает строки в наборе данных, теми которые удовлетворяют условию поиска, с той разницей, что предложение HAVING накладывает ограничения на агрегированные строки сгруппированного набора. Предложение HAVING не является обязательным и может быть использовано только в сочетании с предложением GROUP BY.

Условие(я) в предложении HAVING может ссылаться на:

- Любой агрегированный столбец в списке выбора SELECT. Это наиболее широко используемый случай;
- Любое агрегированное выражение, которое не находится в списке выбора SELECT, но разрешено в контексте запроса. Иногда это полезно;
- Любой столбец в списке GROUP BY. Однако более эффективно фильтровать не агрегированные данные на более ранней стадии в предложении WHERE;
- Любое выражение, значение которого не зависит от содержимого набора данных (например, константа или контекстная переменная). Это допустимо, но совершенно бессмысленно, потому что такое условие, не имеющее никакого отношения к самому набору данных, либо подавит весь набор, либо оставит его не тронутым.

Предложение HAVING не может содержать:

- Не агрегированные выражения столбца, которые не находятся в списке GROUP BY;
- Позицию столбца. Целое число в предложении HAVING – просто целое число;
- Псевдонимы столбца – даже, если они появляются в предложении GROUP BY.

Примеры

Перестроим наши ранние примеры. Мы можем использовать предложение HAVING для исключения малых групп студентов:

```
SELECT
    class,
    COUNT(*) AS num_boys,
    AVG(age) AS boys_avg_age
FROM students
```

```
WHERE sex = 'M'
GROUP BY class
HAVING COUNT(*) >= 5
```

Выберем только группы, которые имеют минимальный разброс по возрасту 1,2 года:

```
SELECT
    class,
    COUNT(*) AS num_boys,
    AVG(age) AS boys_avg_age
FROM students
WHERE sex = 'M'
GROUP BY class
HAVING MAX(age) - MIN(age) > 1.2
```

Обратите внимание, что если вас действительно интересует эта информация, то неплохо бы включить в список выбора min(age) и max(age) или выражение max(age) – min(age).

Следующий запрос отбирает только учеников 3 класса:

```
SELECT
    class,
    COUNT(*) AS num_boys,
    AVG(age) AS boys_avg_age
FROM students
WHERE sex = 'M'
GROUP BY class
HAVING class STARTING WITH '3'
```

Однако гораздо лучше переместить это условие в предложение WHERE:

```
SELECT
    class,
    COUNT(*) AS num_boys,
    AVG(age) AS boys_avg_age
FROM students
WHERE sex = 'M' AND class STARTING WITH '3'
GROUP BY class
```

PLAN

Предложение PLAN позволяет пользователю указать свой план выполнения запроса, который перекрывает тот план, который оптимизатор сгенерировал автоматически.

Синтаксис:

```
```

```
PLAN <plan-expr>

<plan-expr> ::=
    (<plan-item> [, <plan-item> ...])
  | <sorted-item>
  | <joined-item>
  | <merged-item>
  | <hash-item>

<sorted-item> ::= SORT (<plan-item>)

<joined-item> ::= JOIN (<plan-item>, <plan-item> [, <plan-item> ...])

<merged-item> ::=
  [SORT] MERGE (<sorted-item>, <sorted-item> [, <sorted-item> ...])

<hash-item> ::= HASH (<plan-item>, <plan-item> [, <plan-item> ...])

<plan-item> ::= <basic-item> | <plan-expr>

<basic-item> ::= <relation> {
    NATURAL
  | INDEX (<indexlist>)
  | ORDER index [INDEX (<indexlist>)]
}

<relation> ::= table | view [table]

<indexlist> ::= index [, index ...]
```

Параметры предложения PLAN

table
: Имя таблицы или её алиас.

view
: Имя представления.

index
: Имя индекса.

Каждый раз, когда пользователь отправляет запрос ядру Firebird, оптимизатор вычисляет стратегию извлечения данных. Большинство клиентов Firebird имеют возможность отобразить пользователю план извлечения данных. В собственном инструменте isql это делается с помощью команды SET PLAN ON. Если вы хотите только изучить план запроса без его выполнения, то вам необходимо ввести команду SET PLANONLY ON, после чего будут извлекаться планы запросов без их выполнения. Для возврата isql в режим выполнения запросов введите команду SET PLANONLY OFF.

> **Примечание**
>
> Более подробный план можно получить при включении расширенного плана. В isql это делается с помощью команды SET EXPLAIN ON. Этот план выводит более подробную информацию о методах доступа используемых оптимизатором, однако его нельзя включить в запрос. Описание расширенного плана выходит за рамки данного руководства.

В большинстве случаев, вы можете доверять тому, что Firebird выберет наиболее оптимальный план запроса. Однако если ваши запросы очень сложны и вам кажется, что они выполняются не эффективно, вам необходимо посмотреть план запроса и подумать можете ли вы улучшить его.

Простые планы

Простейшие планы состоят только из имени таблицы и следующим за ним метода извлечения. Например, для неотсортированной выборки из единственной таблицы без предложения WHERE:

```
SELECT * FROM students
PLAN (students NATURAL)
```

План в EXPLAIN форме:

```
Select Expression
  -> Table "STUDENTS" Full Scan
```

Если есть предложение WHERE вы можете указать индекс, который будет использоваться при нахождении совпадений:

```
SELECT *
FROM students
WHERE class = '3C'
PLAN (students INDEX (ix_stud_class))
```

План в EXPLAIN форме:

```
Select Expression
  -> Filter
      -> Table "STUDENTS" Access By ID
          -> Bitmap
              -> Index "IX_STUD_CLASS" Range Scan (full match)
```

Директива INDEX может использоваться также для условий соединения (которые будут обсуждаться чуть позже). Она содержит список индексов, разделённых запятыми.

Директива ORDER определяет индекс, который используется при сортировке набора данных, если присутствуют предложения ORDER BY или GROUP BY:

```
SELECT *
FROM students
PLAN (students ORDER pk_students)
ORDER BY id
```

План в EXPLAIN форме:

```
Select Expression
  -> Table "STUDENTS" Access By ID
      -> Index "PK_STUDENTS" Full Scan
```

Инструкции ORDER и INDEX могут быть объединены:

```
SELECT *
FROM students
WHERE class >= '3'
PLAN (students ORDER pk_students INDEX (ix_stud_class))
ORDER BY id
```

План в EXPLAIN форме:

```
Select Expression
  -> Filter
      -> Table "STUDENTS" Access By ID
          -> Index "PK_STUDENTS" Full Scan
              -> Bitmap
                  -> Index "IX_STUD_CLASS" Range Scan (lower bound: 1/1)
```

Так же совершено нормально, если указать для инструкций ORDER и INDEX один и тот же индекс:

```
SELECT *
FROM students
WHERE class >= '3'
PLAN (students ORDER ix_stud_class INDEX (ix_stud_class))
ORDER BY class
```

План в EXPLAIN форме:

```
Select Expression
  -> Filter
      -> Table "STUDENTS" Access By ID
          -> Index "IX_STUD_CLASS" Range Scan (lower bound: 1/1)
              -> Bitmap
                  -> Index "IX_STUD_CLASS" Range Scan (lower bound: 1/1)
```

Для сортировки наборов данных, когда невозможно использовать индекс (или вы хотите подавить его использование), уберите инструкцию ORDER и предварите выражение плана инструкцией SORT:

```
SELECT *
FROM students
PLAN SORT (students NATURAL)
ORDER BY name
```

План в EXPLAIN форме:

```
Select Expression
  -> Sort (record length: 128, key length: 56)
      -> Table "STUDENTS" Full Scan
```

Или когда индекс используется для поиска:

```
SELECT *
FROM students
WHERE class >= '3'
PLAN SORT (students INDEX (ix_stud_class))
ORDER BY name
```

План в EXPLAIN форме:

```
Select Expression
  -> Sort (record length: 136, key length: 56)
      -> Filter
          -> Table "STUDENTS" Access By ID
              -> Bitmap
                  -> Index "IX_STUD_CLASS" Range Scan (lower bound: 1/1)
```

Обратите внимание, что инструкция SORT, в отличие от ORDER, находится за пределами скобок. Это отражает тот факт, что строки данных извлекаются неотсортированными и сортируются впоследствии.

При выборке из представления указывается само представление и участвующее в нем таблица. Например, если у вас есть представление FRESHMEN, которое выбирает только студентов первокурсников:

```
SELECT *
FROM freshmen
PLAN (freshmen students NATURAL)
```

План в EXPLAIN форме:

```
Select Expression
  -> Table "STUDENTS" as "FRESHMEN" Full Scan
```

Или, например:

```
SELECT *
FROM freshmen
WHERE id > 10
PLAN SORT (freshmen students INDEX (pk_students))
ORDER BY name DESC
```

План в EXPLAIN форме:

```
Select Expression
  -> Sort (record length: 144, key length: 24)
      -> Filter
          -> Table "STUDENTS" as "FRESHMEN" Access By ID
              -> Bitmap
                  -> Index "PK_STUDENTS" Range Scan (lower bound: 1/1)
```

Обратите внимание: если вы назначили псевдоним таблице или представлению, то в предложении PLAN необходимо использовать псевдоним, а не оригинальное имя.

Составные планы

Если вы делаете соединение, то вы можете указать индекс, который будет использоваться для сопоставления. Кроме того, вы должны использовать директиву JOIN для двух потоков в плане:

```
SELECT s.id, s.name, s.class, c.mentor
FROM students s
JOIN classes c ON c.name = s.class
PLAN JOIN (s NATURAL, c INDEX (pk_classes))
```

План в EXPLAIN форме:

```
Select Expression
  ->  Nested Loop Join (inner)
      -> Table "STUDENTS" as "S" Full Scan
      -> Filter
          -> Table "CLASSES" as "C" Access By ID
              -> Bitmap
                  -> Index "PK_CLASSES" Unique Scan
```

То же самое соединение, отсортированное по индексированному столбцу:

```
SELECT s.id, s.name, s.class, c.mentor
FROM students s
JOIN classes c ON c.name = s.class
PLAN JOIN (s ORDER pk_students, c INDEX (pk_classes))
ORDER BY s.id
```

План в EXPLAIN форме:

```
Select Expression
  ->  Nested Loop Join (inner)
      -> Table "STUDENTS" as "S" Access By ID
          -> Index "PK_STUDENTS" Full Scan
      -> Filter
```

```
          -> Table "CLASSES" as "C" Access By ID
              -> Bitmap
                  -> Index "PK_CLASSES" Unique Scan
```

И соединение, отсортированное не по индексированному столбцу:

```
SELECT s.id, s.name, s.class, c.mentor
FROM students s
JOIN classes c ON c.name = s.class
PLAN SORT (JOIN (S NATURAL, c INDEX (pk_classes))))
ORDER BY s.name
```

План в EXPLAIN форме:

```
Select Expression
  -> Sort (record length: 152, key length: 12)
      ->  Nested Loop Join (inner)
          -> Table "STUDENTS" as "S" Full Scan
          -> Filter
              -> Table "CLASSES" as "C" Access By ID
                  -> Bitmap
                      -> Index "PK_CLASSES" Unique Scan
```

Соединение с добавленным условием поиска:

```
SELECT s.id, s.name, s.class, c.mentor
FROM students s
JOIN classes c ON c.name = s.class
WHERE s.class <= '2'
PLAN SORT (JOIN (s INDEX (fk_student_class), c INDEX (pk_classes)))
ORDER BY s.name
```

План в EXPLAIN форме:

```
Select Expression
  -> Sort (record length: 152, key length: 12)
     ->  Nested Loop Join (inner)
         -> Filter
            -> Table "STUDENTS" as "S" Access By ID
               -> Bitmap
                  -> Index "FK_STUDENT_CLASS" Range Scan (lower bound: 1/1)
         -> Filter
            -> Table "CLASSES" as "C" Access By ID
               -> Bitmap
                  -> Index "PK_CLASSES" Unique Scan
```

То же самое, но используется левое внешнее соединение:

```
SELECT s.id, s.name, s.class, c.mentor
```

```
FROM classes c
LEFT JOIN students s ON c.name = s.class
WHERE s.class <= '2'
PLAN SORT (JOIN (c NATURAL, s INDEX (fk_student_class)))
ORDER BY s.name
```

План в EXPLAIN форме:

```
Select Expression
  -> Sort (record length: 192, key length: 56)
      -> Filter
         ->  Nested Loop Join (outer)
             -> Table "CLASSES" as "C" Full Scan
             -> Filter
                -> Table "STUDENTS" as "S" Access By ID
                   -> Bitmap
                      -> Index "FK_STUDENT_CLASS" Range Scan (full match)
```

Если нет доступных индексов для условия соединения (или вы не хотите его использовать), то возможно соединение потоков с помощью метода HASH или MERGE.

Для соединения методом HASH в плане вместо директивы JOIN используется директива HASH. В этом случае меньший (ведомый) поток целиком вычитывается во внутренний буфер. В процессе чтения к каждому ключу связи применяется хеш-функция и пара *{хеш, указатель в буфере}* записывается в хеш-таблицу. После чего читается ведущий поток и его ключ связи апробируется в хеш-таблице.

```
SELECT *
FROM students s
JOIN classes c ON c.cookie = s.cookie
PLAN HASH (c NATURAL, s NATURAL)
```

План в EXPLAIN форме:

```
Select Expression
    -> Filter
        -> Hash Join (inner)
            -> Table "STUDENTS" as "S" Full Scan
            -> Record Buffer (record length: 145)
                -> Table "CLASSES" as "C" Full Scan
```

При выполнении соединения методом MERGE план должен сначала отсортировать оба потока по соединяемым столбцам и затем произвести слияние. Это достигается с помощью директив SORT (которую вы уже встречали) и MERGE используемую вместо JOIN.

```
SELECT *
FROM students s
JOIN classes c ON c.cookie = s.cookie
PLAN MERGE (SORT (c NATURAL), SORT (s NATURAL))
```

Добавление предложения ORDER BY означает, что результат слияния также должен быть отсортирован:

```
SELECT *
FROM students s
JOIN classes c ON c.cookie = s.cookie
PLAN SORT (MERGE (SORT (c NATURAL), SORT (s NATURAL)))
ORDER BY c.name, s.id
```

И наконец, мы добавляем условие поиска на двух индексированных столбцах таблицы STUDENTS:

```
SELECT *
FROM students s
JOIN classes c ON c.cookie = s.cookie
WHERE s.id < 10 AND s.class <= '2'
PLAN SORT (MERGE (SORT (c NATURAL),
                 SORT (s INDEX (pk_students, fk_student_class))))
ORDER BY c.name, s.id
```

Как следует из формального определения синтаксиса, JOIN и MERGE могут объединять в плане более двух потоков. Кроме того, каждое выражение плана может использоваться в качестве элемента в охватывающем плане. Это означает, что планы некоторых сложных запросов могут иметь различные уровни вложенности.

Наконец, вместо MERGE вы можете писать SORT MERGE. Поскольку это не имеет абсолютно никакого значения и может создать путаницу с "настоящей" директивой SORT (которая действительно имеет значение), то вероятно лучше придерживаться простой директивы MERGE.

UNION

Предложение UNION объединяет два и более набора данных, тем самым увеличивая общее количество строк, но не столбцов. Наборы данных, принимающие участие в UNION, должны иметь одинаковое количество столбцов. Однако столбцы в соответствующих позициях не обязаны иметь один и тот же тип данных, они могут быть абсолютно не связанными.

По умолчанию, объединение подавляет дубликаты строк. UNION ALL отображает все строки, включая дубликаты. Необязательное ключевое слово DISTINCT делает поведение по умолчанию явным.

Синтаксис:

```
<union> ::=
  <individual-select>
  UNION [DISTINCT | ALL]
  <individual-select>
  [UNION [DISTINCT | ALL]
  <individual-select>
  [...]
```

```
  [<union-wide-clauses>]

<individual-select> ::=
  SELECT
  [FIRST m] [SKIP n]
  [DISTINCT | ALL] <columns>
  FROM source [[AS] alias]
  [<joins>]
  [WHERE <condition>]
  [GROUP BY <grouping-list>
  [HAVING <aggregate-condition>]]
  [PLAN <plan-expr>]

<union-wide-clauses> ::=
  [ORDER BY <ordering-list>]
  [ROWS m [TO n]]
  [FOR UPDATE [OF <forupdate-columns>]]
  [WITH LOCK]
  [INTO <PSQL-varlist>]
```

Объединения получают имена столбцов из первого запроса на выборку. Если вы хотите дать псевдонимы объединяемым столбцам, то делайте это для списка столбцов в самом верхнем запросе на выборку. Псевдонимы в других участвующих в объединении выборках разрешены, и могут быть даже полезными, но они не будут распространяться на уровне объединения.

Если объединение имеет предложение ORDER BY, то единственно возможными элементами сортировки являются целочисленные литералы, указывающие на позиции столбцов, необязательно сопровождаемые ASC/DESC и/или NULLS FIRST/LAST директивами. Это так же означает, что вы не можете упорядочить объединение ничем, что не является столбцом объединения. (Однако вы можете завернуть его в производную таблицу, которая даст вам все обычные параметры сортировки.)

Объединения позволены в подзапросах любого вида и могут самостоятельно содержать подзапросы. Они также могут содержать соединения (joins), и могут принимать участие в соединениях, если завёрнуты в производную таблицу.

Примеры

Этот запрос представляет информацию из различных музыкальных коллекций в одном наборе данных с помощью объединений:

```
SELECT id, title, artist, len, 'CD' AS medium
FROM cds
UNION
SELECT id, title, artist, len, 'LP'
FROM records
UNION
SELECT id, title, artist, len, 'MC'
FROM cassettes
ORDER BY 3, 2 -- artist, title
```

Если id, title, artist и length – единственные поля во всех участвующих таблицах, то запрос может быть записан так:

```
SELECT c.*, 'CD' AS medium
FROM cds c
UNION
SELECT r.*, 'LP'
FROM records r
UNION
SELECT c.*, 'MC'
FROM cassettes c
ORDER BY 3, 2 -- artist, title
```

Квалификация "звёзд" необходима здесь, потому что они не являются единственным элементом в списке столбцов. Заметьте, что псевдонимы "c" в первой и третьей выборке не кусают друг друга. Они не имеют контекста объединения, а лишь применяются к отдельным запросам на выборку.

Следующий запрос получает имена и телефонные номера переводчиков и корректоров. Те переводчики, которые также работают корректорами, будут отображены только один раз в результирующем наборе, если номера их телефонов одинаковые в обеих таблицах. Тот же результат может быть получен без ключевого слова DISTINCT. Если вместо ключевого слова DISTINCT, будет указано ключевое слово ALL, эти люди будут отображены дважды.

```
SELECT name, phone
FROM translators
UNION DISTINCT
SELECT name, telephone
FROM proofreaders
```

Пример использования UNION в подзапросе:

```
SELECT name, phone, hourly_rate
FROM clowns
WHERE hourly_rate < ALL
  (SELECT hourly_rate FROM jugglers
   UNION
   SELECT hourly_rate FROM acrobats)
ORDER BY hourly_rate
```

ORDER BY

Результат выборки данных при выполнении оператора SELECT по умолчанию никак не упорядочивается (хотя довольно часто происходит упорядочение в хронологическом порядке помещёния строк в таблицу операторами INSERT). Предложение ORDER BY позволяет задать необходимый порядок при выборке данных.

Синтаксис:

```
SELECT ... FROM ...
```

```
...
ORDER BY <ordering-item> [, <ordering-item> ...]

<ordering-item> ::=
  {col-name | col-alias | col-position | expression}
  [COLLATE collation-name]
  [ASC[ENDING] | DESC[ENDING]]
  [NULLS {FIRST | LAST}]
```

Параметры предложения ORDER BY

col-name
: Полное имя столбца.

col-alias
: Алиас (псевдоним) столбца.

col-position
: Позиция столбца.

expression
: Произвольное выражение.

collation-name
: Имя сопоставления (порядка сортировки).

В предложении через запятую перечисляются столбцы, по которым нужно упорядочить результирующий набор данных. Можно задавать имя столбца, псевдоним, присвоенный столбцу в списке выбора при помощи ключевого слова AS, или порядковый номер столбца в списке выбора. В одном предложении можно для разных столбцов смешивать форму записи. Например, один столбец в списке сортировки может быть задан своим именем, а другой порядковым номером.

> **Важно**
>
> Если вы сортируете по позиции столбца или его алиасу, то выражение соответствующее этой позиции (алиасу) будет скопировано из списка выборки SELECT. Это касается и подзапросов, таким образом, подзапрос будет выполняться, по крайней мере, два раза.

> **Примечание**
>
> В случае сортировки по номеру столбца для запроса вида "SELECT *" сервер раскрывает звёздочку (*) для определения сортируемых столбцов. Однако использование данной особенности в ваших запросах является плохой практикой.

Направление сортировки

Ключевое слово ASCENDING задаёт упорядочение по возрастанию значений. Допустимо сокращение ASC. Применяется по умолчанию.

Ключевое слово DESCENDING задаёт упорядочение по убыванию значений. Допустимо сокращение DESC.

В одном предложении упорядочение по одному столбцу может идти по возрастанию значений, а по другому — по убыванию.

Порядок сравнения

Ключевое слово COLLATE позволяет задать порядок сортировки строкового столбца, если нужен порядок, отличный от того, который был установлен для этого столбца (явно при описании столбца или по умолчанию, принятому для соответствующего набора символов).

Расположение NULL

Ключевое слово NULLS определяет, где в отсортированном наборе данных будут находиться значения NULL соответствующего столбца – в начале выборки (FIRST) или в конце (LAST). По умолчанию принимается NULLS FIRST.

Сортировка частей UNION

Части выборок SELECT, участвующих в объединении UNION, не могут быть отсортированы с использованием предложения ORDER BY. Однако вы можете достичь желаемого результата с использованием производных таблиц или общих табличных выражений. Предложение ORDER BY, записанное последним в объединении, будет применено ко всей выборке в целом, а не к последней его части. Для объединений, единственно возможными элементами сортировки являются целочисленные литералы, указывающие на позиции столбцов, необязательно сопровождаемые ASC / DESC и/или NULLS FIRST / LAST директивами.

Примеры

В описанном ниже запросе выборка будет отсортирована по возрастанию по столбцам RDB $CHARACTER_SET_ID, RDB$COLLATION_ID таблицы DB$COLLATIONS:

```
SELECT
    RDB$CHARACTER_SET_ID AS CHARSET_ID,
    RDB$COLLATION_ID AS COLL_ID,
    RDB$COLLATION_NAME AS NAME
FROM RDB$COLLATIONS
ORDER BY RDB$CHARACTER_SET_ID, RDB$COLLATION_ID
```

То же самое, но сортировка производится по псевдонимам столбцов:

```
SELECT
    RDB$CHARACTER_SET_ID AS CHARSET_ID,
    RDB$COLLATION_ID AS COLL_ID,
    RDB$COLLATION_NAME AS NAME
FROM RDB$COLLATIONS
ORDER BY CHARSET_ID, COLL_ID
```

В следующем запросе производится сортировка, по номерам столбцов:

```
SELECT
    RDB$CHARACTER_SET_ID AS CHARSET_ID,
    RDB$COLLATION_ID AS COLL_ID,
    RDB$COLLATION_NAME AS NAME
FROM RDB$COLLATIONS
ORDER BY 1, 2
```

Как было выше сказано, такая сортировка тоже допустима, но не рекомендуется:

```
SELECT *
FROM RDB$COLLATIONS
ORDER BY 3, 2
```

В данном запросе сортировка происходит по второму столбцу таблицы BOOKS:

```
SELECT
    BOOKS.*,
    FILMS.DIRECTOR
FROM BOOKS, FILMS
ORDER BY 2
```

Предупреждение

Обратите внимание на то, что выражения, результатом вычисления которых должны быть целые неотрицательные числа, будут интерпретироваться как номер столбца и вызовут исключение, если они не будут в диапазоне от 1 до числа столбцов.

Пример:

```
SELECT
  X, Y, NOTE
FROM PAIRS
ORDER BY X+Y DESC
```

Примечания:

- Число, возвращаемое функцией или процедурой из UDF или хранимой процедуры, непредсказуемо, независимо от того, определена сортировка самим выражением или номером столбца;
- Только неотрицательные целые числа интерпретируются как номер столбца. Целое число, полученное однократным вычислением выражения или заменой параметра, запоминается как целочисленная постоянная величина, так как это значение одинаково для всех строк.

Сортировка по убыванию значений столбца PROCESS_TIME с размещёнием значений NULL в начале выборки:

```
SELECT *
FROM MSG
ORDER BY PROCESS_TIME DESC NULLS FIRST
```

Сортировка выборки полученной объединением выборок из двух запросов. Выборка сортируется по убыванию значений второго столбца с размещённием NULL значений в конце списка и возрастанием значений первого столбца с размещённием NULL значений в начале списка.

```
SELECT
  DOC_NUMBER, DOC_DATE
FROM PAYORDER
UNION ALL
SELECT
  DOC_NUMBER, DOC_DATE
FROM BUDGORDER
ORDER BY 2 DESC NULLS LAST, 1 ASC NULLS FIRST
```

ROWS

Назначение: Получение части строк из упорядоченного набора.

Синтаксис:

```
SELECT
...
FROM ...
...
ROWS m [TO n]
```

Параметры предложения ROWS

m, *n*
: Любые целочисленные выражения.

Предложение ROWS было введено для совместимости с Interbase 6.5 и выше.

В отличие от FIRST и SKIP, выражение ROWS принимает все типы целочисленных (integer) выражений в качестве аргумента – без скобок! Конечно, скобки могут требоваться для правильных вычислений внутри выражения, и вложенный запрос также должен быть обернут в скобки.

Важно

- Нумерация записей в наборе данных начинается с 1.
- И FIRST/SKIP, и ROWS могут быть использованы без выражения ORDER BY, хотя это редко имеет смысл, за исключением случая, когда необходимо быстро взглянуть на данные таблицы – получаемые строки при этом будут чаще всего в случайном порядке. В этом случае запрос вроде "SELECT * FROM TABLE1 ROWS 20" вернёт 20 первых записей, а не целую таблицу (которая может очень большой).

Вызов ROWS m приведёт к возвращению первых m записей из набора данных.

Особенности при использовании ROWS с одним аргументом

- Если m больше общего числа записей в возвращаемом наборе данных, то будет возвращён весь набор данных;
- Если m = 0, то будет возвращён пустой набор данных;
- Если m < 0, выдаётся ошибка.

В случае указания ROWS m TO n, то будут возвращены записи с m по n из набора данных.

Особенности при использовании ROWS с двумя аргументами

- Если m больше общего количества строк в наборе данных и n >= m, то будет возвращён пустой набор данных;
- Если число m не превышает общего количества строк в наборе данных, а n превышает, то выборка ограничивается строками, начиная с m до конца набора данных;
- Если m < 1 и n < 1, то оператор SELECT выдаст ошибку;
- Если n = m -1, то будет возвращён пустой набор данных;
- Если n < m -1, то оператор SELECT выдаст ошибку.

Замена FIRST..SKIP

В сущности, ROWS заменяет собой нестандартные выражения FIRST и SKIP, за исключением единственного случая, когда указывается только SKIP, т.е. когда возвращается весь набор данных за исключением пропуска указанного числа записей с начала.

Для того, что реализовать такое поведение с помощью ROWS, необходимо указать второй аргумент, заведомо больший, чем размер возвращаемого набора данных. Или запросить число записей в возвращаемом наборе через подзапрос.

Совместное использование FIRST..SKIP и ROWS

Нельзя использовать ROWS вместе с FIRST/SKIP в одном и том же операторе SELECT, но можно использовать разный синтаксис в разных подзапросах.

Использование ROWS в UNION

При использовании ROWS с выражением UNION, он будет применяться к объединённому набору данных, и должен быть помещён после последнего SELECT.

При необходимости ограничить возвращаемые наборы данных одного или нескольких операторов SELECT внутри UNION, можно воспользоваться следующими вариантами:

1. Использовать FIRST/SKIP в этих операторах SELECT. Необходимо помнить, что нельзя локально использовать выражение ORDER BY в SELECT внутри UNION – только глобально, ко всему суммарному набору данных;

2. Преобразовать SELECT в производные таблицы с выражениями ROWS.

Примеры

Ниже приведены примеры, ранее использованные для демонстрации FIRST/SKIP.

Следующий запрос вернёт первые 10 имён из таблицы PEOPLE (имена также будут отсортированы, см. ORDER BY).

```
SELECT id, name
FROM People
ORDER BY name ASC
ROWS 1 TO 10
```

или его эквивалент

```
SELECT id, name
FROM People
ORDER BY name ASC
ROWS 10
```

Следующий запрос вернёт все записи из таблицы PEOPLE, за исключением первых 10 имён:

```
SELECT id, name
FROM People
ORDER BY name ASC
ROWS 11 TO (SELECT COUNT(*) FROM People)
```

А этот запрос вернёт последние 10 записей (обратите внимание на скобки):

```
SELECT id, name
FROM People
ORDER BY name ASC
ROWS (SELECT COUNT(*) - 9 FROM People)
TO (SELECT COUNT(*) FROM People)
```

Этот запрос вернёт строки 81-100 из таблицы PEOPLE:

```
SELECT id, name
FROM People
ORDER BY name ASC
ROWS 81 TO 100
```

См. также: FETCH, OFFSET, FIRST, SKIP.

FETCH, OFFSET

Предложения FETCH и OFFSET являются SQL:2008 совместимым эквивалентом предложениям FIRST/SKIP и альтернативой предложению ROWS. Предложение OFFSET указывает, какое количество строк необходимо пропустить. Предложение FETCH указывает, какое количество строк необходимо получить.

Предложения OFFSET и FETCH могут применяться независимо уровня вложенности выражений запросов.

Синтаксис:

```
SELECT ...
FROM ...
[...]
[ORDER BY <expr_order_list>]
[OFFSET <simple_value_expr> {ROW | ROWS}]
[FETCH {FIRST | NEXT} [<simple_value_expr>] {ROW | ROWS} ONLY]
```

Параметры предложений OFFSET и FETCH

expr_order_list
: Список выражений сортировки.

simple_value_expr
: Числовой литерал, SQL параметр (?) или PSQL параметр (:param).

Примечания:

- Firebird не поддерживает указание FETCH в процентах, определённое в стандарте.
- Firebird не поддерживает предложение FETCH с опцией WITH TIES, которая определена в стандарте.
- FIRST ... SKIP и ROWS являются нестандартными альтернативами.
- Предложения OFFSET и/или FETCH не могут быть объединены с предложениями ROWS или FIRST/SKIP в одном выражении запроса.
- Выражения, ссылки на столбцы и т.д. недопустимы в любом из предложений.
- В отличие от предложения ROWS, предложения OFFSET и FETCH допустимы только в операторе SELECT.

Примеры:

Следующий запрос возвращает все строки кроме первых 10, упорядоченных по столбцу COL1:

```
SELECT *
FROM T1
ORDER BY COL1
OFFSET 10 ROWS
```

В этом примере возвращается первые 10 строк, упорядоченных по столбцу COL1:

```
SELECT *
```

```
FROM T1
ORDER BY COL1
OFFSET 10 ROWS
```

Использование предложений OFFSET и FETCH в производной таблице, результат которой ограничивается ещё раз во внешнем запросе.

```
SELECT *
FROM (
  SELECT *
  FROM T1
  ORDER BY COL1 DESC
  OFFSET 1 ROW
  FETCH NEXT 10 ROWS ONLY
) a
ORDER BY a.COL1
FETCH FIRST ROW ONLY
```

См. также: ROWS, FIRST, SKIP.

WITH LOCK

Назначение: Пессимистическая блокировка.

Опция WITH LOCK, обеспечивает возможность ограниченной явной пессимистической блокировки для осторожного использования в затронутых наборах строк:

- крайне малой выборки (в идеале из одной строки) и
- при контроле из приложения.

Пессимистическая блокировка редко требуется при работе с Firebird. Эту функцию можно использовать только хорошо понимая её последствия. Требуется хорошее знание различных уровней изоляции транзакции. Предложение WITH LOCK доступно для использования в DSQL и PSQL и только для выборки данных (SELECT) из одной таблицы. Предложение WITH LOCK нельзя использовать:

- в подзапросах;
- в запросах с объединением нескольких таблиц (JOIN);
- с оператором DISTINCT, предложением GROUP BY и при использовании любых агрегатных функций;
- при работе с представлениями;
- при выборке данных из селективных хранимых процедур;
- при работе с внешними таблицами.

Синтаксис:

```
SELECT ...
FROM single_table
[WHERE ...]
[FOR UPDATE [OF <column-names>]]
```

```
[WITH LOCK]
```

При успешном выполнении предложения WITH LOCK будут заблокированы выбранные строки данных и таким образом запрещён доступ на их изменение в рамках других транзакций до момента завершения Вашей транзакции.

При выборке с использованием предложения FOR UPDATE блокировка применяется к каждой строке, одна за другой, по мере попадания выборки в кэш сервера. Это делает возможным ситуацию, в которой успешная блокировка данных *перестаёт работать* при достижении в выборке строки, заблокированной другой транзакцией.

Сервер, в свою очередь, для каждой записи, подпадающей под явную блокировку, возвращает версию записи, которая является в настоящее время подтверждённой (актуальной), независимо от состояния базы данных, когда был выполнен оператор выборки данных, или исключение при попытке обновления заблокированной записи.

Ожидаемое поведение и сообщения о конфликте зависят от параметров транзакции, определённых в TPB (Transaction Parameters Block):

Таблица 6.1. Влияние параметров TPB на явную блокировку

Режим TPB	**Поведение**
isc_tpb_consistency	Явные блокировки переопределяются неявными или явными блокировками табличного уровня и игнорируются.
isc_tpb_concurrency + isc_tpb_nowait	При подтверждении изменения записи в транзакции, стартовавшей после транзакции, запустившей явную блокировку, немедленно возникает исключение конфликта обновления.
isc_tpb_concurrency + isc_tpb_wait	При подтверждении изменения записи в транзакции, стартовавшей после транзакции, запустившей явную блокировку, немедленно возникает исключение конфликта обновления. Если в активной транзакции идёт редактирование записи (с использованием явной блокировки или нормальной оптимистической блокировкой записи), то транзакция, делающая попытку явной блокировки, ожидает окончания транзакции блокирования и, после её завершения, снова пытается получить блокировку записи. Это означает, что при изменении версии записи и подтверждении транзакции с блокировкой возникает исключение конфликта обновления.
isc_tpb_read_committed + isc_tpb_nowait	Если есть активная транзакция, редактирующая запись (с использованием явной блокировки или нормальной оптимистической блокировкой записи), то сразу же возникает исключение конфликта обновления.
isc_tpb_read_committed + isc_tpb_wait	Если в активной транзакции идёт редактирование записи (с использованием явной блокировки или нормальной оптимистической блокировкой записи), то транзакция, делающая попытку явной блокировки, ожидает окончания транзакции блокирования и, после её завершения, снова пытается получить блокировку записи. Для этого режима TPB никогда не возникает конфликта обновления.

Как сервер работает с WITH LOCK

Попытка редактирования записи с помощью оператора UPDATE, заблокированной другой транзакцией, приводит к вызову исключения конфликта обновления или ожиданию завершения блокирующей транзакции – в зависимости от режима TPB. Поведение сервера здесь такое же, как если бы эта запись уже была изменена блокирующей транзакцией.

Нет никаких специальных кодов gdscode, возвращаемых для конфликтов обновления, связанных с пессимистической блокировкой.

Сервер гарантирует, что все записи, возвращённые явным оператором блокировки, фактически заблокированы и действительно соответствуют условиям поиска, заданным в операторе WHERE, если эти условия не зависят ни от каких других таблиц, не имеется операторов соединения, подзапросов и т.п. Это также гарантирует то, что строки, не попадающие под условия поиска, не будут заблокированы. Это не даёт гарантии, что нет строк, которые попадают под условия поиска, и не заблокированы.

> **Примечание**
>
> Такая ситуация может возникнуть, если в другой, параллельной транзакции подтверждаются изменения в процессе выполнения текущего оператора блокировки.

Сервер блокирует строки по мере их выборки. Это имеет важные последствия, если вы блокируете сразу несколько строк. Многие методы доступа к базам данных Firebird по умолчанию используют для выборки данных пакеты из нескольких сотен строк (так называемый "буфер выборки"). Большинство компонентов доступа к данным не выделяют строки, содержащиеся в последнем принятом пакете, и для которых произошёл конфликт обновления.

Опциональное предложение "OF <column-names>"

Предложение FOR UPDATE обеспечивает метод, позволяющий предотвратить использование буферизованных выборок, с помощью подпункта "OF <column-names>", что включает позиционированное обновление.

> **Подсказка**
>
> Кроме того, некоторые компоненты доступа позволяют установить размер буфера выборки и уменьшить его до 1 записи. Это позволяет вам заблокировать и редактировать строку до выборки и блокировки следующей или обрабатывать ошибки, не отменяя действий вашей транзакции.

Предостережения при использовании WITH LOCK

- Откат неявной или явной точки сохранения отменяет блокировку записей, которые изменялись в рамках её действий, но ожидающие окончания блокировки транзакции при этом не уведомляются. Приложения не должны зависеть от такого поведения, поскольку в будущем оно может быть изменено;
- Хотя явные блокировки могут использоваться для предотвращения и/или обработки необычных ошибок конфликтов обновления, объем ошибок обновления (deadlock) вырастет, если вы тщательно не разработаете свою стратегию блокировки и не будете ей строго управлять;

- Большинство приложений не требуют явной блокировки записей. Основными целями явной блокировки являются: 1) предотвращение дорогостоящей обработки ошибок конфликта обновления в сильно загруженных приложениях и 2) для поддержания целостности объектов, отображаемых из реляционной базы данных в кластеризируемой среде. Если использование вами явной блокировки не подпадает под одну из этих двух категорий, то это является неправильным способом решения задач в Firebird;
- Явная блокировка — это расширенная функция; не злоупотребляйте её использованием! В то время как явная блокировка может быть очень важной для веб-сайтов, обрабатывающих тысячи параллельных пишущих транзакций или для систем типа ERP/CRM, работающих в крупных корпорациях, большинство прикладных программ не требуют её использования.

Примеры использования явной блокировки

1. Одиночная:

```
SELECT *
FROM DOCUMENT
WHERE DOCUMENT_ID=? WITH LOCK
```

2. Несколько строк с последовательной их обработкой с курсором DSQL:

```
SELECT *
FROM DOCUMENT
WHERE PARENT_ID=?
FOR UPDATE WITH LOCK
```

INTO

Назначение: Передача результатов SELECT в переменные.

Доступно в: PSQL.

В PSQL (хранимых процедурах, триггерах и др.) результаты выборки команды SELECT могут быть построчно загружены в локальные переменные (число, порядок и типы локальных переменных должны соответствовать полям SELECT). Часто такая загрузка – единственный способ что-то сделать с возвращаемыми значениями.

Простой оператор SELECT может быть использован в PSQL, только если он возвращает единственную строку, то есть, если это запрос типа синглтон (singleton). Для запросов, возвращающих несколько строк, PSQL предлагает использовать оператор FOR SELECT.

Также, PSQL поддерживает оператор DECLARE CURSOR, который связывает именованный курсор с определенной командой SELECT — и этот курсор впоследствии может быть использован для навигации по возвращаемому набору данных.

В PSQL выражение INTO должно появляться в самом конце команды SELECT.

Синтаксис:

```
SELECT [...] <column-list>
FROM ...
[...]
[INTO <variable-list>]

<variable-list> ::= [:]psqlvar [, [:]psqlvar ...]
```

Обратите внимание.

В PSQL двоеточие перед именами переменных является опциональным.

Примеры

В PSQL, можно присвоить значения min_amt, avg_amt и max_amt заранее объявленным переменным или выходным параметрам:

```
SELECT
  MIN(amount),
  AVG(CAST(amount AS float)),
  MAX(amount)
FROM orders
WHERE artno = 372218
INTO min_amt,
     avg_amt,
     max_amt;
```

В данном запросе CAST служит для корректного вычисления среднего значения. Если не привести значение к float, то среднее значение будет обрезано до ближайшего целого значения.

В триггере:

```
SELECT LIST(name, ', ')
FROM persons p
WHERE p.id IN (new.father, new.mother)
INTO new.parentnames;
```

Общие табличные выражения CTE ("WITH ... AS ... SELECT")

Общие табличные выражения (Common Table Expressions), сокращённо CTE, описаны как виртуальные таблицы или представления, определённые в преамбуле основного запроса, которые участвуют в основном запросе. Основной запрос может ссылаться на любое CTE из определённых в преамбуле, как и при выборке данных из обычных таблиц или представлений. CTE могут быть рекурсивными, т.е. ссылающимися сами на себя, но не могут быть вложенными.

Синтаксис:

```
<cte-construct> ::=
  <cte-defs>
```

```
  <main-query>

<cte-defs> ::= WITH [RECURSIVE] <cte> [, <cte> ...]

<cte> ::= name [(<column-list>)] AS (<cte-stmt>)

<column-list> ::= column-alias [, column-alias ...]
```

Параметры CTE

cte-stmt
Любой оператор SELECT или UNION.

main-query
Основной оператор SELECT, который может ссылаться на любое CTE из найденных в преамбуле.

name
Алиас табличного выражения.

column-alias
Алиас столбца табличного выражения.

Примеры:

Пример 6.1. Запрос с использованием CTE

```
WITH DEPT_YEAR_BUDGET AS (
  SELECT
      FISCAL_YEAR,
      DEPT_NO,
      SUM(PROJECTED_BUDGET) BUDGET
  FROM PROJ_DEPT_BUDGET
  GROUP BY FISCAL_YEAR, DEPT_NO
)
SELECT
    D.DEPT_NO,
    D.DEPARTMENT,
    DYB_2008.BUDGET BUDGET_08,
    DYB_2009.BUDGET AS BUDGET_09
FROM DEPARTMENT D
    LEFT JOIN DEPT_YEAR_BUDGET DYB_2008
      ON D.DEPT_NO = DYB_2008.DEPT_NO AND
         DYB_2008.FISCAL_YEAR = 2008
    LEFT JOIN DEPT_YEAR_BUDGET DYB_2009
      ON D.DEPT_NO = DYB_2009.DEPT_NO AND
         DYB_2009.FISCAL_YEAR = 2009
WHERE EXISTS (SELECT *
              FROM PROJ_DEPT_BUDGET B
              WHERE D.DEPT_NO = B.DEPT_NO)
```

Примечания

- Определение CTE может содержать любой правильный оператор SELECT, если он не содержит преамбулы "WITH ..." (операторы WITH не могут быть вложенными);

- CTE могут использовать друг друга, но ссылки не должны иметь циклов;
- CTE могут быть использованы в любой части главного запроса или другого табличного выражения и сколько угодно раз;
- Основной запрос может ссылаться на CTE несколько раз, но с разными алиасами;
- CTE могут быть использованы в операторах INSERT, UPDATE и DELETE как подзапросы;
- Если CTE объявлен, то он должен быть обязательно использован;
- CTE могут быть использованы и в PSQL (FOR WITH ... SELECT ... INTO ...):

```
FOR
  WITH MY_RIVERS AS (
      SELECT *
      FROM RIVERS
      WHERE OWNER = 'me')
  SELECT
      NAME,
      LENGTH
  FROM MY_RIVERS
  INTO :RNAME,
       :RLEN
DO
BEGIN
  ...
END
```

Рекурсивные CTE

Рекурсивное (ссылающееся само на себя) CTE это ОБЪЕДИНЕНИЕ, у которого должен быть, по крайней мере, один не рекурсивный элемент, к которому привязываются остальные элементы объединения. Не рекурсивный элемент помещается в CTE первым. Рекурсивные члены отделяются от не рекурсивных и друг от друга с помощью UNION ALL. Объединение не рекурсивных элементов может быть любого типа.

Рекурсивное CTE требует наличия ключевого слова RECURSIVE справа от WITH. Каждый рекурсивный член объединения может сослаться на себя только один раз и это должно быть сделано в предложении FROM.

Главным преимуществом рекурсивных CTE является то, что они используют гораздо меньше памяти и процессорного времени, чем эквивалентные рекурсивные хранимые процедуры.

Выполнение рекурсивного CTE

Выполнение рекурсивного CTE с точки зрения сервера Firebird можно описать следующим образом:

- Сервер начинает выполнение с не рекурсивного члена;
- Для каждой выбранной строки из нерекурсивного части выполняется каждый рекурсивный член один за другим, используя текущие значения из предыдущей итерации как параметры;
- Если во время выполнения экземпляр рекурсивного элемента не выдаёт строк, цикл выполнения переходит на предыдущий уровень и получает следующую строку от внешнего для него набора данных.

Важно отметить, что если CTE объявлена, то где-то ниже она обязательно должна быть использована, иначе будет ошибка вида «CTE "AAA" is not used in query».

Примеры:

Пример 6.2. Рекурсивное CTE

```
WITH RECURSIVE DEPT_YEAR_BUDGET AS (
  SELECT
      FISCAL_YEAR,
      DEPT_NO,
      SUM(PROJECTED_BUDGET) BUDGET
  FROM PROJ_DEPT_BUDGET
  GROUP BY FISCAL_YEAR, DEPT_NO
),
DEPT_TREE AS (
  SELECT
      DEPT_NO,
      HEAD_DEPT,
      DEPARTMENT,
      CAST('' AS VARCHAR(255)) AS INDENT
  FROM DEPARTMENT
  WHERE HEAD_DEPT IS NULL
  UNION ALL
  SELECT
      D.DEPT_NO,
      D.HEAD_DEPT,
      D.DEPARTMENT,
      H.INDENT || ' '
  FROM DEPARTMENT D
    JOIN DEPT_TREE H ON H.HEAD_DEPT = D.DEPT_NO
)
SELECT
    D.DEPT_NO,
    D.INDENT || D.DEPARTMENT DEPARTMENT,
    DYB_2008.BUDGET AS BUDGET_08,
    DYB_2009.BUDGET AS BUDGET_09
FROM DEPT_TREE D
    LEFT JOIN DEPT_YEAR_BUDGET DYB_2008 ON
      (D.DEPT_NO = DYB_2008.DEPT_NO) AND
      (DYB_2008.FISCAL_YEAR = 2008)
    LEFT JOIN DEPT_YEAR_BUDGET DYB_2009 ON
      (D.DEPT_NO = DYB_2009.DEPT_NO) AND
      (DYB_2009.FISCAL_YEAR = 2009)
```

Следующий пример выводит родословную лошади. Основное отличие состоит в том, что рекурсия идёт сразу по двум веткам дерева родословной.

```
WITH RECURSIVE PEDIGREE (
    CODE_HORSE,
    CODE_FATHER,
    CODE_MOTHER,
    NAME,
    MARK,
    DEPTH)
AS (SELECT
        HORSE.CODE_HORSE,
```

```
        HORSE.CODE_FATHER,
        HORSE.CODE_MOTHER,
        HORSE.NAME,
        CAST('' AS VARCHAR(80)),
        0
    FROM HORSE
    WHERE
        HORSE.CODE_HORSE = :CODE_HORSE
    UNION ALL
    SELECT
        HORSE.CODE_HORSE,
        HORSE.CODE_FATHER,
        HORSE.CODE_MOTHER,
        HORSE.NAME,
        'F' || PEDIGREE.MARK,
        PEDIGREE.DEPTH + 1
    FROM HORSE
        JOIN PEDIGREE
          ON HORSE.CODE_HORSE = PEDIGREE.CODE_FATHER
    WHERE
        -- ограничение глубины рекурсии
        PEDIGREE.DEPTH < :MAX_DEPTH
    UNION ALL
    SELECT
        HORSE.CODE_HORSE,
        HORSE.CODE_FATHER,
        HORSE.CODE_MOTHER,
        HORSE.NAME,
        'M' || PEDIGREE.MARK,
        PEDIGREE.DEPTH + 1
    FROM HORSE
        JOIN PEDIGREE
          ON HORSE.CODE_HORSE = PEDIGREE.CODE_MOTHER
    WHERE
        -- ограничение глубины рекурсии
        PEDIGREE.DEPTH < :MAX_DEPTH
)
SELECT
    CODE_HORSE,
    NAME,
    MARK,
    DEPTH
FROM
    PEDIGREE
```

Примечания для рекурсивного CTE:

- В рекурсивных членах объединения не разрешается использовать агрегаты (DISTINCT, GROUP BY, HAVING) и агрегатные функции (SUM, COUNT, MAX и т.п.);
- Рекурсивная ссылка не может быть участником внешнего объединения OUTER JOIN;
- Максимальная глубина рекурсии составляет 1024;
- Рекурсивный член не может быть представлен в виде производной таблицы.

INSERT

Назначение: Вставка данных в таблицу.

Доступно в: DSQL, ESQL, PSQL.

Синтаксис:

```
INSERT INTO target
{DEFAULT VALUES | [(<column_list>)] <value_source>}
[RETURNING <returning_list> [INTO <variables>]]

<column_list> ::= colname [, colname ...]

<value_source> ::= VALUES (<value_list>) | <select_stmt>

<value_list> ::= value [, value ...]

<returning_list> ::=
  ret_value [[AS] ret_alias] [, ret_value [[AS] ret_alias] ...]

<variables> ::= [:]varname [, [:]varname ...]
```

Параметры оператора INSERT

target
Имя таблицы или представления, в которую происходит вставка новой записи или записей.

colname
Столбец таблицы или представления.

value
Выражение, значение которого используется для вставки в таблицу.

ret_value
Выражение, возвращаемое в предложении RETURNING.

ret_alias
Псевдоним для выражения, возвращаемого в предложении RETURNING.

varname
Имя PSQL переменной.

Оператор INSERT добавляет строки в таблицу, или в одну или более таблиц представления. Если значения столбцов указаны в разделе VALUES, то будет вставлена одна строка. Значения столбцов также могут быть получены из оператора SELECT, в этом случае может быть вставлено от нуля и более строк. В случае DEFAULT VALUES, значения можно не указывать, и вставлена будет одна строка.

Примечание

- Один столбец не может быть указан более одного раза в списке столбцов;
- При возвращении значений столбцов вставленной записи с помощью INTO в контекстные переменные NEW.*columnname* в триггерах не должен использоваться префикс двоеточия (":").

INSERT ... VALUES

В списке VALUES должны быть указаны значения для всех столбцов в списке столбцов в том же порядке и совместимые по типу. Если список столбцов отсутствует, то значения должны быть указаны для каждого столбца таблицы или представления (исключая вычисляемые столбцы).

Указание набора символов для строковых литералов

Строковым литералам может предшествовать указание набора символов, используя упрощённый синтаксис, для того чтобы сообщить системе каким образом интерпретировать входные данные.

Примеры:

```
INSERT INTO cars (make, model, byyear)
VALUES ('Ford', 'T', 1908);

INSERT INTO cars
VALUES ('Ford', 'T', 1908, 'USA', 850);

-- обратите внимание на префикс '_'
INSERT INTO People
VALUES (_ISO8859_1 'Hans-Jörg Schäfer');
```

INSERT ... SELECT

В этом случае выходные столбцы оператора SELECT, должны предоставить значения для каждого целевого столбца в списке столбцов, в том же порядке и совместимого типа. Если список столбцов отсутствует, то значения должны быть предоставлены для каждого столбца таблицы или представления (исключая вычисляемые столбцы).

Примеры:

```
INSERT INTO cars (make, model, byyear)
SELECT make, model, byyear
FROM new_cars;

INSERT INTO cars
SELECT *
FROM new_cars;

INSERT INTO Members (number, name)
SELECT number, name
FROM NewMembers
WHERE Accepted = 1
UNION ALL
SELECT number, name
FROM SuspendedMembers
WHERE Vindicated = 1

INSERT INTO numbers(num)
```

```
WITH RECURSIVE r(n) AS (
  SELECT 1 FROM rdb$database
  UNION ALL
  SELECT n+1 FROM r where n < 100
)
SELECT n FROM r
```

Конечно, имена столбцов в таблице источнике не обязательно должны быть такими же, как и в таблице приёмнике.

Любой тип оператора SELECT разрешён, пока его выходные столбцы точно соответствуют столбцам вставки по числу и типу. Типы не должны быть точно такими же, но они должны быть совместимыми по присваиванию.

INSERT ... DEFAULT VALUES

Предложение DEFAULT VALUES позволяет вставлять записи без указания значений вообще, ни непосредственно (в предложении VALUES), ни из оператора SELECT. Это возможно, только если для каждого NOT NULL поля и полей, на которые наложены другие ограничения, или имеются допустимые объявленные значения по умолчанию, или эти значения устанавливаются в BEFORE INSERT триггере.

Примеры:

```
INSERT INTO journal
DEFAULT VALUES
RETURNING entry_id
```

RETURNING

Оператор INSERT вставляющий только одну строку может включать необязательное предложение RETURNING для возврата значений из вставленной строки. Если предложение указано, то оно может содержать любые столбцы, указанные в операторе, или другие столбцы и выражения. Возвращаемые значения содержат все изменения, произведённые в триггерах BEFORE.

> **Важно**
>
> В DSQL, оператор с RETURNING всегда возвращает только одну строку. Если указано предложение RETURNING, и обнаружено более одной совпадающей строки, выдаётся сообщение об ошибке. Это поведение может быть изменено в последующих версиях Firebird.

Примеры:

Пример 6.3. Использование предложения RETURNING в операторе INSERT

```
INSERT INTO Scholars (
  firstname,
  lastname,
  address,
  phone,
```

```
  email)
VALUES (
  'Henry',
  'Higgins',
  '27A Wimpole Street',
  '3231212',
  NULL)
RETURNING lastname, fullname, id;

INSERT INTO Dumbbells (first_name, last_name, iq)
SELECT fname, lname, iq
FROM Friends
ORDER BY iq ROWS 1
RETURNING id, first_name, iq
INTO :id, :fname, :iq;
```

Замечания:

- Предложение RETURNING поддерживается только для INSERT .. VALUES и одиночных INSERT .. SELECT.
- В DSQL оператор с использованием RETURNING всегда возвращает ровно одну строку. Если запись не была вставлена на самом деле, то все поля в возвращаемой строке будут меть значения NULL. Это поведение может быть изменено позже. В PSQL, если ни одна строка не вставлена, то ничего не возвращается и все целевые переменные сохраняют свои прежние значения.

Вставка столбцов BLOB

Вставка в столбцы BLOB только возможна при следующих обстоятельствах:

1. Клиентское приложение вставляет BLOB посредством Firebird API. В этом случае все зависит от приложения, и не рассматривается в этом руководстве;
2. Длина строкового литерала не может превышать 32767 байт.
3. Если источником данных является столбец BLOB или выражение, возвращающее BLOB.

UPDATE

Назначение: Обновление данных в таблице.

Доступно в: DSQL, ESQL, PSQL.

Синтаксис:

```
UPDATE target [[AS] alias]
SET col = newval [, col = newval ...]
[WHERE {search-conditions | CURRENT OF cursorname}]
[PLAN plan_items]
[ORDER BY sort_items]
[ROWS m [TO n]]
```

```
[RETURNING <returning_list> [INTO <variables>]]

<returning_list> ::=
  ret_value [[AS] ret_alias] [, ret_value [[AS] ret_alias] ...]

<variables> ::= [:]varname [, [:]varname ...]
```

Параметры оператора UPDATE

target
Имя таблицы или представления, в которой происходит обновление записей.

alias
Псевдоним таблицы или представления.

col
Столбец таблицы или представления.

newval
Новые значения обновляемых столбцов в таблице (представлении).

search-conditions
Условие поиска, ограничивающее набор обновляемых строк.

cursorname
Имя курсора, по которому позиционируется обновляемая запись.

plan_items
Предложение плана.

sort_items
Предложение сортировки.

m, *n*
Целочисленные выражения для ограничения количества обновляемых строк.

ret_value
Выражение, возвращаемое в предложении RETURNING.

ret_alias
Псевдоним для выражения, возвращаемого в предложении RETURNING.

varname
Имя PSQL переменной.

Изменяет значения в одной таблице или в большем количестве таблиц в представлении.

Если вы назначаете псевдоним таблице или представлению, вы обязаны использовать псевдоним для уточнения столбцов таблицы.

SET

Изменяемые столбцы указываются в предложении SET. Столбцы и их значения перечисляются через запятую. Слева имя столбца, и справа значение или выражение.

Разрешено использовать имена столбцов в выражениях справа. При этом использоваться будет всегда старое значение столбца, даже если присваивание этому столбцу уже произошло ранее в перечислении SET. Один столбец может быть использован только один раз в конструкции SET.

Пример:

Данные в таблице TSET:

```
A B
----
1 0
2 0
```

После выполнения оператора

```
update tset set a = 5, b = a
```

```
A B
----
5 1
5 2
```

Обратите внимание, что старые значения (1 и 2) используются для обновления столбца b, даже после того как столбцу a были назначено новое значение (5).

WHERE

Предложение WHERE ограничивает набор обновляемых записей заданным условием, или – в PSQL – текущей строкой именованного курсора, если указано предложение WHERE CURRENT OF.

> **Примечание**
>
> Предложение WHERE CURRENT OF используется только в PSQL, т.к. в DSQL нет оператора DSQL для создания курсора.

Строковые литералы могут предваряться именем набора символов, для того чтобы Firebird понимал, как интерпретировать данные.

Примеры:

```
UPDATE addresses
SET city = 'Saint Petersburg', citycode = 'PET'
WHERE city = 'Leningrad';

UPDATE employees
SET salary = 2.5 * salary
WHERE title = 'CEO';
```

```
UPDATE People
SET name = _ISO8859_1 'Hans-Jörg Schäfer'
-- обратите внимание на префикс '_'
WHERE id = 53662;

UPDATE employee e
SET salary = salary * 1.05
WHERE EXISTS(
  SELECT *
  FROM employee_project ep
  WHERE e.emp_no = ep.emp_no);
```

ORDER BY и ROWS

Предложения ORDER BY и ROWS имеют смысл только вместе. Однако их можно использовать по отдельности.

При одном аргументе *m*, ROWS ограничивает update первыми *m* строками. Особенности:

- Если *m* > количества обрабатываемых строк, то обновляется весь набор строк;
- Если *m* = 0, ни одна строка не обновляется;
- Если *m* < 0, выдаётся ошибка.

При двух аргументах *m* и *n*, ROWS ограничивает update до строк от *m* до *n* включительно. Оба аргумента – целочисленные, и начинаются с 1. Особенности:

- Если *m* > количества обрабатываемых строк, ни одна строка не обновляется;
- Если *n* больше количества строк, то обновляются строки от *m* до конца набора;
- Если *m* < 1 или *n* < 1, выдаётся ошибка;
- Если *n* = *m* - 1, ни одна строка не обновляется;
- Если *n* < *m* - 1, выдаётся ошибка.

Примеры:

Пример 6.4. Использование предложения ROWS в операторе UPDATE

```
-- дать надбавку 20ти сотрудникам с наименьшей зарплатой
UPDATE employees
SET salary = salary + 50
ORDER BY salary ASC
ROWS 20;
```

RETURNING

Оператор UPDATE, обновляющий максимум одну строку, может включать RETURNING для возврата значений из обновляемой строки. В RETURNING могут включаться любые строки, необязательно только те, которые обновляются.

Возвращаемые значения содержат изменения, произведённые в триггерах BEFORE, но не в триггерах AFTER. OLD.*fieldname* и NEW.*fieldname* могут быть использованы в качестве

имён столбцов. Если OLD. Или NEW. не указано, возвращаются новые значения столбцов (NEW.).

> **Примечание**
>
> В DSQL оператор с RETURNING всегда возвращает только одну строку. Если записи не были обновлены оператором, то возвращаемые значения содержат NULL. Это поведение может быть изменено в последующих версиях Firebird.

INTO

Предложение INTO предназначено для передачи значений в локальные переменные. Оно доступно только в PSQL. Если записи не было обновлены, ничего не возвращается, и переменные, указанные в RETURING, сохранят свои прежние значения.

Пример 6.5. Использование предложения RETURNING в операторе UPDATE

```
UPDATE Scholars
SET first_name = 'Hugh', last_name = 'Pickering'
WHERE first_name = 'Henry' AND last_name = 'Higgins'
RETURNING id, old.last_name, new.last_name;
```

Обновление столбцов BLOB

Обновление столбцов BLOB всегда полностью меняет их содержимое. Даже идентификатор BLOB (ID), который является ссылкой на данные BLOB и хранится в столбце, меняется. BLOB могут быть изменены, если:

1. Клиентское приложение меняет BLOB посредством Firebird API. В этом случае все зависит от приложения, и не рассматривается в этом руководстве;
2. Длина строкового литерала не может превышать 65535 байт;
3. Если источником данных является столбец BLOB или выражение, возвращающее BLOB.

UPDATE OR INSERT

Назначение: Добавление новой или обновление существующей записи в таблице.

Доступно в: DSQL, PSQL.

Синтаксис:

```
UPDATE OR INSERT  target [(<column_list>)]
VALUES (<value_list>)
[MATCHING (<column_list>)]
[RETURNING <returning_list> [INTO <variables>]]

<column_list> ::= colname [, colname ...]
```

```
<value_list> ::= value [, value ...]

<returning_list> ::=
  ret_value [[AS] ret_alias] [, ret_value [[AS] ret_alias] ...]

<variables> ::= [:]varname [, [:]varname ...]
```

Параметры оператора UPDATE OR INSERT

target
: Имя таблицы или представления, запись в которой будет обновлена или произойдет вставка новой записи.

colname
: Столбец таблицы или представления.

value
: Выражение, значение которого используется для вставки или обновления таблицы.

ret_value
: Выражение, возвращаемое в предложении RETURNING.

ret_alias
: Псевдоним для выражения, возвращаемого в предложении RETURNING.

varname
: Имя PSQL переменной.

Оператор UPDATE OR INSERT вставляет или обновляет одну или более существующих записей. Производимое действие зависит от значений столбцов в предложении MATCHING (или, если оно не указано, то от значений столбцов первичного ключа — PK). Если найдены записи, совпадающие с указанными значениями, то они обновляются. Если нет, то вставляется новая запись.

Совпадением считается полное совпадение значений столбцов MATCHING или PK. Совпадение проверяется с использованием IS NOT DISTINCT, поэтому NULL совпадает с NULL.

Ограничения

- Если у таблицы нет первичного ключа, то указание MATCHING считается обязательным;
- В списке MATCHING, так же как и в списке столбцов update/insert, каждый столбец может быть упомянут только один раз;
- Предложение INTO доступно только в PSQL.
- У столбцов, возвращаемых в контекстные переменные NEW в триггере, перед именем не должно использоваться двоеточие.

RETURNING

Предложение RETURNING может содержать любые столбцы, указанные в операторе, или другие столбцы и выражения. Возвращаемые значения содержат все изменения, произведённые в триггерах BEFORE, но не в триггерах AFTER. OLD.*fieldname* и

NEW.*fieldname* могут быть использованы в качестве возвращаемых значений. Для обычных имён столбцов возвращаются новые значения.

В DSQL, оператор с RETURNING всегда возвращает только одну строку. Если присутствует предложение RETURNING, и обнаружено более одной совпадающей строки, выдаётся сообщение об ошибке. Это поведение может быть изменено в последующих версиях Firebird.

Примеры: Использование оператора UPDATE OR INSERT

```
UPDATE OR INSERT INTO Cows (Name, Number, Location)
VALUES ('Suzy Creamcheese', 3278823, 'Green Pastures')
MATCHING (Number)
RETURNING rec_id
INTO :id;
```

DELETE

Назначение: Удаление данных из таблицы.

Доступно в: DSQL, ESQL, PSQL.

Синтаксис:

```
DELETE
FROM target [[AS] alias]
[WHERE {search-conditions | CURRENT OF cursorname}]
[PLAN plan_items]
[ORDER BY sort_items]
[ROWS m [TO n]]
[RETURNING <returning_list> [INTO <variables>]]

<returning_list> ::= ret_value [[AS] alias_val] [, ret_value [[AS] alias_val] ...]

<variables> ::= [:]varname [, [:]varname ...]
```

Параметры оператора DELETE

target
Имя таблицы или представления, из которой удаляются записи.

alias
Псевдоним таблицы или представления.

search-conditions
Условие поиска, ограничивающее набор удаляемых строк.

cursorname
Имя курсора, по которому позиционируется удаляемая запись.

plan_items
Предложение плана.

sort_items
Предложение сортировки.

m, *n*
Целочисленные выражения для ограничения количества удаляемых строк.

ret_value
Выражение, возвращаемое в предложении RETURNING.

alias_val
Псевдоним для выражения, возвращаемого в предложении RETURNING.

varname
Имя PSQL переменной.

Оператор DELETE удаляет строки из таблицы или из одной и более таблиц представления.

Если для таблицы указан псевдоним, то он должен использоваться для всех столбцов таблицы.

WHERE

Условие в предложении WHERE ограничивает набор удаляемых строк. Удаляются только те строки, которые удовлетворяют условию поиска, или только текущей строке именованного курсора.

Удаление с помощью WHERE CURRENT OF называется *позиционированным удалением* (positioned delete), потому что удаляется запись в текущей позиции. Удаление при помощи "WHERE условие" называется *поисковым удалением* (searched delete), поскольку Firebird ищет записи, соответствующие условию.

> **Примечание**
>
> В чистом DSQL выражение WHERE CURRENT OF не имеет смысла, т.к. в DSQL нет оператора для создания курсора.

Примеры:

Пример 6.6. Использование предложения WHERE в операторе DELETE

```
DELETE FROM People
WHERE first_name <> 'Boris' AND last_name <> 'Johnson';
```

```
DELETE FROM employee e
WHERE NOT EXISTS(
  SELECT *
  FROM employee_project ep
  WHERE e.emp_no = ep.emp_no);
```

```
DELETE FROM Cities
WHERE CURRENT OF Cur_Cities; -- только в PSQL
```

PLAN

Предложение PLAN позволяет вручную указать план для оптимизатора.

Примеры:

```
DELETE FROM Submissions
WHERE date_entered < '1-Jan-2002'
PLAN (Submissions INDEX ix_subm_date)
```

ORDER BY и ROWS

Предложение ORDER BY упорядочивает набор перед его удалением. Имеет смысл только в комбинации с предложением ROWS, но допустимо и без ROWS.

Предложение ROWS позволяет ограничить количество удаляемых строк.

Синтаксис:

```
ROWS <m> [TO <n>]
```

В качестве *m* и *n* могут выступать любые целочисленные выражения.

При одном аргументе *m*, удаляются первые *m* строк. Порядок строк без ORDER BY не определён (случаен).

Замечания:

- Если *m* больше общего числа строк в наборе, то весь набор удаляется;
- Если *m* = 0, то удаление не происходит;
- Если *m* < 0, то выдаётся сообщение об ошибке.

Если указаны аргументы *m* и *n*, удаление ограничено количеством строк от *m* до *n*, включительно. Нумерация строк начинается с 1.

Замечания по использованию двух аргументов:

- Если *m* > общего числа строк в наборе, ни одна строка не удаляется;
- Если *m* > 0 и <= числа строк в наборе, а *n* вне этих значений, то удаляются строки от m до конца набора;
- Если *m* < 1 или n < 1, выдаётся сообщение об ошибке;
- Если *n* = *m* – 1, ни одна строка не удаляется;
- Если *n* < *m* – 1, выдаётся сообщение об ошибке.

Примеры:

Удаление самой старой покупки

```
DELETE FROM Purchases ORDER BY ByDate ROWS 1
```

Удаление заказов для 10 клиентов с самыми большими номерами

```
DELETE FROM Sales ORDER BY custno DESC ROWS 1 TO 10
```

Удаляет все записи из sales, поскольку не указано ROWS

```
DELETE FROM Sales ORDER BY custno DESC
```

Удаляет одну запись "с конца", т.е. от Z…

```
DELETE FROM popgroups ORDER BY name DESC ROWS 1
```

Удаляет пять самых старых групп

```
DELETE FROM popgroups ORDER BY formed ROWS 5
```

Сортировка (ORDER BY) не указана, поэтому будут удалены 8 обнаруженных записей, начиная с пятой.

```
DELETE FROM popgroups ROWS 5 TO 12
```

RETURNING

Оператор DELETE, удаляющий не более одной строки, может содержать конструкцию RETURNING для возвращения значений удаляемой строки. В RETURNING могут быть указаны любые столбцы, не обязательно все, а также другие столбцы и выражения.

Примечание

- В DSQL, оператор с RETURNING всегда возвращает только одну строку. Если записи не были удалены, то возвращаемые столбцы содержат NULL. Это поведение может быть изменено в последующих версиях Firebird;
- В PSQL, если строка не была удалена, ничего не возвращается, и целевые переменные сохраняют свои значения;
- Предложение INTO доступно только в PSQL.

Примеры:

```
DELETE FROM Scholars
WHERE first_name = 'Henry' AND last_name = 'Higgins'
RETURNING last_name, fullname, id

DELETE FROM Dumbbells
ORDER BY iq DESC
ROWS 1
RETURNING last_name, iq
INTO :lname, :iq;

DELETE FRMO TempSales ts
WHERE ts.id = tempid
RETURNING ts.qty
INTO new.qty;
```

MERGE

Назначение: Слияние записей источника в целевую таблицу (или обновляемое представление).

Доступно в: DSQL, PSQL.

Синтаксис:

```
MERGE INTO target [[AS] target_alias]
USING <source> [[AS] source_alias]
ON <join condition>
<merge when> [<merge when> ...]
[RETURNING <returning_list> [INTO <variable_list>]]

<merge when> ::= <merge when matched> | <merge when not matched>

<merge when matched> ::=
  WHEN MATCHED [ AND <condition> ]
  THEN { UPDATE SET <assignment_list> | DELETE }

<merge when not matched> ::=
  WHEN NOT MATCHED [ AND <condition> ]
  THEN INSERT [ (<column_list>) ]
  VALUES (<value_list>)

<assignment_list> ::= colname = value [, colname = value ...]

<column_list> ::= colname [, colname ...]

<value_list> ::= value [, value ...]

<returning_list> ::=
  ret_value [[AS] ret_alias] [, ret_value [[AS] ret_alias] ...]

<variable_list> ::= [:]varname [, [:]varname ...]
```

Параметры оператора MERGE

target
: Целевая таблица или обновляемое представление.

source
: Источник данных. Может быть таблицей, представлением, хранимой процедурой или производной таблицей.

target_alias
: Псевдоним целевой таблицы или представления.

source_alias
: Псевдоним источника.

join condition
: Условие соединения целевой таблицы и источника.

condition
: Дополнительные условия проверки в предложениях WHEN MATCHED или WHEN NOT MATCHED.

colname
Столбец целевой таблицы или представления.

value
Значение, присваиваемое столбцу целевой таблицы. Выражение, которое может содержать литералы, PSQL переменные, столбцы из источника.

ret_value
Выражение, возвращаемое в предложении RETURNING.

ret_alias
Псевдоним для выражения, возвращаемого в предложении RETURNING.

varname
Имя PSQL переменной.

Оператор MERGE производит слияние записей источника в целевую таблицу (или обновляемое представление). Источником может быть таблица, представление, хранимой процедурой или производной таблицей. Каждая запись источника используется для обновления (предложение UPDATE) или удаления (предложение DELETE) одной или более записей цели, или вставки (предложение INSERT) записи в целевую таблицу, или ни для того, ни для другого. Условие обычно содержит сравнение столбцов в таблицах источника и цели.

Допускается указывать несколько предложений WHEN MATCHED и WHEN NOT MATCHED.

Предложения WHEN проверяются в указанном порядке. Если условие в предложении WHEN не выполняется, то мы пропускаем его и переходим к следующему предложению. Так будет происходить до тех пор, пока условие для одного из предложений WHEN не будет выполнено. В этом случае выполняется действие, связанное с предложением WHEN, и осуществляется переход на следующую строку источника. Для каждой строки источника выполняется только одно действие.

> **Примечание**
>
> WHEN NOT MATCHED должно быть видимо с точки зрения источника (таблицы, которая указана в USING). Так сделано потому, что если запись источника не имеет совпадения с записью цели, то выполняется INSERT. Разумеется, если записи в цели не соответствует запись в источнике, то никакие действия не производятся.
>
> На данный момент переменная ROW_COUNT возвращает значение 1, даже если было модифицировано или вставлено более 1 записи. См. CORE-4400.

> **Предупреждение**
>
> Если условие WHEN MATCHED присутствует, и несколько записей совпадают с записями в целевой таблице, UPDATE выполнится для всех совпадающих записей источника, и каждое последующее обновление перезапишет предыдущее. Это нестандартное поведение: стандарт SQL-2003 требует, чтобы в этой ситуации выдавалось исключение (ошибка).

Примеры:

```
MERGE INTO books b
USING purchases p
```

```
ON p.title = b.title AND p.booktype = 'bk'
WHEN MATCHED THEN
  UPDATE SET b.descr = b.descr || '; ' || p.descr
WHEN NOT MATCHED THEN
  INSERT (title, descr, bought)
  VALUES (p.title, p.descr, p.bought);

-- с использованием производной таблицы
MERGE INTO customers c
USING (SELECT * FROM customers_delta WHERE id > 10) cd
ON (c.id = cd.id)
WHEN MATCHED THEN
  UPDATE SET name = cd.name
WHEN NOT MATCHED THEN
  INSERT (id, name) VALUES (cd.id, cd.name);

-- совместно с рекурсивным CTE
MERGE INTO numbers
USING (
  WITH RECURSIVE r(n) AS (
    SELECT 1 FROM rdb$database
    UNION ALL
    SELECT n+1 FROM r WHERE n < 200
  )
  SELECT n FROM r
) t
ON numbers.num = t.n
WHEN NOT MATCHED THEN
  INSERT(num) VALUES(t.n);

-- с использованием предложения DELETE
MERGE INTO SALARY_HISTORY
USING (
  SELECT EMP_NO
  FROM EMPLOYEE
  WHERE DEPT_NO = 120) EMP
ON SALARY_HISTORY.EMP_NO = EMP.EMP_NO
WHEN MATCHED THEN DELETE
```

В следующем примере происходит ежедневное обновление таблицы PRODUCT_IVENTORY на основе заказов, обработанных в таблице SALES_ORDER_LINE. Если количество заказов на продукт таково, что уровень запасов продукта опускается до нуля или становится ещё ниже, то строка этого продукта удаляется из таблицы PRODUCT_IVENTORY.

```
MERGE INTO PRODUCT_IVENTORY AS TARGET
USING (
  SELECT
    SL.ID_PRODUCT,
    SUM(SL.QUANTITY)
  FROM SALES_ORDER_LINE SL
    JOIN SALES_ORDER S ON S.ID = SL.ID_SALES_ORDER
  WHERE S.BYDATE = CURRENT_DATE
  GROUP BY 1
) AS SRC(ID_PRODUCT, QUANTITY)
```

```
ON TARGET.ID_PRODUCT = SRC.ID_PRODUCT
WHEN MATCHED AND TARGET.QUANTITY - SRC.QUANTITY <= 0 THEN
  DELETE
WHEN MATCHED THEN
  UPDATE SET
    TARGET.QUANTITY = TARGET.QUANTITY - SRC.QUANTITY,
    TARGET.BYDATE = CURRENT_DATE
```

См. также: SELECT, INSERT, UPDATE, DELETE.

RETURNING

Оператор MERGE, затрагивающий не более одной строки, может содержать конструкцию RETURNING для возвращения значений добавленной, модифицируемой или удаляемой строки. В RETURNING могут быть указаны любые столбцы из целевой таблицы (обновляемого представления), не обязательно все, а также другие столбцы и выражения.

Возвращаемые значения содержат изменения, произведённые в триггерах BEFORE.

Имена столбцов могут быть уточнены с помощью префиксов NEW и OLD для определения, какое именно значение вы хотите столбца вы хотите получить до модификации или после.

Для предложений WHEN MATCHED UPDATE и MERGE WHEN NOT MATCHED неуточнённые имена столбцов или уточнённые именами таблиц или их псевдонимами понимаются как столбцы с префиксом NEW, для предложений MERGE WHEN MATCHED DELETE – с префиксом OLD.

Пример 6.7. Использование оператора MERGE с предложением RETURNING

Немного модифицируем наш предыдущий пример, чтобы он затрагивал только одну строку, и добавим в него инструкцию RETURNING возвращающего старое и новое количество товара и разницу между этими значениями.

```
MERGE INTO PRODUCT_IVENTORY AS TARGET
USING (
  SELECT
    SL.ID_PRODUCT,
    SUM(SL.QUANTITY)
  FROM SALES_ORDER_LINE SL
    JOIN SALES_ORDER S ON S.ID = SL.ID_SALES_ORDER
  WHERE S.BYDATE = CURRENT_DATE
    AND SL.ID_PRODUCT = :ID_PRODUCT
  GROUP BY 1
) AS SRC(ID_PRODUCT, QUANTITY)
ON TARGET.ID_PRODUCT = SRC.ID_PRODUCT
WHEN MATCHED AND TARGET.QUANTITY - SRC.QUANTITY <= 0 THEN
  DELETE
WHEN MATCHED THEN
  UPDATE SET
    TARGET.QUANTITY = TARGET.QUANTITY - SRC.QUANTITY,
    TARGET.BYDATE = CURRENT_DATE
RETURNING OLD.QUANTITY, NEW.QUANTITY, SRC.QUANTITY
```

```
INTO :OLD_QUANTITY, :NEW_QUANTITY, :DIFF_QUANTITY
```

EXECUTE PROCEDURE

Назначение: Выполнение хранимой процедуры.

Доступно в: DSQL, ESQL, PSQL.

Синтаксис:

```
EXECUTE PROCEDURE procname
    [<inparam> [, <inparam> ...]]
  | ([<inparam> [, <inparam> ...]])
[RETURNING_VALUES <outvar> [, <outvar> ...]]

<outvar> ::= [:]varname
```

Параметры оператора EXECUTE PROCEDURE

procname
Имя хранимой процедуры.

inparam
Выражение совместимое по типу с входным параметром хранимой процедуры.

varname
PSQL переменная, в которую возвращается значение выходного параметра процедуры.

Оператор EXECUTE PROCEDURE выполняет хранимую процедуру, получая список из одного или нескольких входных параметров, если они определены, и возвращает однострочный набор значений, если он определён.

"Выполняемые" хранимые процедуры

Оператор EXECUTE PROCEDURE является наиболее часто используемым стилем вызова хранимой процедуры, которая написана для модификации некоторых данных, код которой не содержит оператора SUSPEND. Такие хранимые процедуры могут возвратить набор данных, состоящий не более чем из одной строки. Этот набор может быть передан в переменные другой (вызывающей) процедуры с помощью предложения RETURNING_VALUES. Клиентские интерфейсы, как правило, имеют обертку API, которые могут извлекать выходные значения в однострочный буфер при вызове процедуры через EXECUTE PROCEDURE в DSQL.

При вызове с помощью EXECUTE PROCEDURE процедур другого типа (селективных процедур) будет возвращена только первая запись из результирующего набора, несмотря на то, что эта процедура скорее всего должна возвращать многострочный результат. "Селективные" хранимые процедуры должны вызываться с помощью оператора SELECT, в этом случае они ведут себя как виртуальные таблицы.

Примечание

- В PSQL И DSQL входными параметрами могут быть любые совместимые по типу выражения;
- Несмотря на то, что скобки для отделения списка передаваемых параметров необязательны после имени хранимой процедуры, желательно их использовать;
- В DSQL приложениях, использующих Firebird API или иную обёртку, вызов процедуры через EXECUTE PROCEDURE не требует указания предложения RETURNING_VALUES для получения выходных значений в однострочный буфер.

Примеры:

В PSQL (с опциональными двоеточиями):

```
EXECUTE PROCEDURE MakeFullName(:First_Name, :Middle_Name, :Last_Name)
RETURNING_VALUES :FullName;
```

В утилите командной строки isql (с литералами в качестве параметров):

```
EXECUTE PROCEDURE MakeFullName
  'J', 'Edgar', 'Hoover';
```

С выражениями в качестве параметров:

```
EXECUTE PROCEDURE MakeFullName
   ('Mr./Mrs. ' || First_Name, Middle_Name, upper(Last_Name))
RETURNING_VALUES FullName;
```

EXECUTE BLOCK

Назначение: Выполнение анонимного PSQL блока.

Доступно в: DSQL.

Синтаксис:

```
EXECUTE BLOCK [(<inparams>)] [RETURNS (<outparams>)]
AS
  [<declarations>]
```

```
BEGIN
  [<PSQL statements>]
END

<inparams> ::= <param_decl> = ? [, <inparams> ]

<outparams> ::= <param_decl> [, <outparams>]

<param_decl> ::= paramname <type> [NOT NULL] [COLLATE collation]

<type> ::=
    <datatype>
  | [TYPE OF] domain
  | TYPE OF COLUMN rel.col

<datatype> ::=
    {SMALLINT | INTEGER | BIGINT}
  | BOOLEAN
  | {FLOAT | DOUBLE PRECISION}
  | {DATE | TIME | TIMESTAMP}
  | {DECIMAL | NUMERIC} [(precision [, scale])]
  | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(size)]
    [CHARACTER SET charset]
  | {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR} [VARYING] [(size)]
  | BLOB [SUB_TYPE {subtype_num | subtype_name}]
    [SEGMENT SIZE seglen] [CHARACTER SET charset]
  | BLOB [(seglen [, subtype_num])]


<declarations> ::= <declare_item> [<declare_item> ...]

<declare_item> ::=
  <declare_var>; |
  <declare_cursor>; |
  <declare_subfunc> |
  <declare_subproc>
```

Параметры оператора EXECUTE BLOCK

param_decl
Описание входного или выходного параметра.

declarations
Секция объявления локальных переменных, именованных курсоров, подпроцедур и подфункций.

declare_var
Объявление локальной переменной.

declare_cursor
Объявление именованного курсора.

declare_subfunc
Объявление подпрограммы – функции.

declare_subproc
Объявление подпрограммы – процедуры.

paramname
Имя входного или выходного параметра процедуры. Может содержать до 31 символа. Имя параметра должно быть уникальным среди входных и выходных параметров процедуры, а также её локальных переменных.

datatype
Тип данных SQL.

collation
Порядок сортировки.

domain
Домен.

rel
Имя таблицы или представления.

col
Имя столбца таблицы или представления.

precision
Точность. От 1 до 18.

scale
Масштаб. От 0 до 18, должно быть меньше или равно *precision*.

size
Максимальный размер строки в символах.

charset
Набор символов.

subtype_num
Номер подтипа BLOB.

subtype_name
Мнемоника подтипа BLOB.

seqlen
Размер сегмента, не может превышать 65535.

Выполняет блок PSQL кода, так как будто это хранимая процедура, возможно с входными и выходными параметрами и локальными переменными. Это позволяет пользователю выполнять "на лету" PSQL в контексте DSQL.

Примеры:

Этот пример вводит цифры от 0 до 127 и соответствующие им ASCII символов в таблицу ASCIITABLE:

```
EXECUTE BLOCK
AS
  DECLARE i INT = 0;
```

```
BEGIN
  WHILE (i < 128) DO
  BEGIN
    INSERT INTO AsciiTable VALUES (:i, ascii_char(:i));
    i = i + 1;
  END
END
```

Следующий пример вычисляет среднее геометрическое двух чисел и возвращает его пользователю:

```
EXECUTE BLOCK (
  x DOUBLE PRECISION = ?,
  y DOUBLE PRECISION = ?)
RETURNS (gmean DOUBLE PRECISION)
AS
BEGIN
  gmean = sqrt(x*y);
  SUSPEND;
END
```

Поскольку этот блок имеет входные параметры, он должен быть предварительно подготовлен. После чего можно установить параметры и выполнить блок. Как это будет сделано, и вообще возможно ли это сделать, зависит от клиентского программного обеспечения. Смотрите примечания ниже.

Наш последний пример принимает два целочисленных значений, smallest и largest. Для всех чисел в диапазоне smallest..largest, блок выводит само число, его квадрат, куб и четвертую степень.

```
EXECUTE BLOCK (smallest INT = ?, largest INT = ?)
RETURNS (
  number INT,
  square BIGINT,
  cube BIGINT,
  fourth BIGINT)
AS
BEGIN
  number = smallest;
  WHILE (number <= largest) DO
  BEGIN
    square = number * number;
    cube = number * square;
    fourth = number * cube;
    SUSPEND;
    number = number + 1;
  END
END
```

Опять же, как вы можете установить значения параметров, зависит от программного обеспечения клиента.

См. также: Операторы языка PSQL.

Входные и выходные параметры

Выполнение блока без входных параметров должно быть возможным с любым клиентом Firebird, который позволяет пользователю вводить свои собственные DSQL операторы. Если есть входные параметры, все становится сложнее: эти параметры должны получить свои значения после подготовки оператора, но перед его выполнением. Это требует специальных возможностей, которыми располагает не каждое клиентское приложение (Например, isql такой возможности не предлагает).

Сервер принимает только вопросительные знаки ("?") в качестве заполнителей для входных значений, а не ":*a*", ":*MyParam*" и т.д., или литеральные значения. Клиентское программное обеспечение может поддерживать форму ":*xxx*", в этом случае будет произведена предварительная обработка запроса перед отправкой его на сервер.

Если блок имеет выходные параметры, вы должны использовать SUSPEND, иначе ничего не будет возвращено.

Выходные данные всегда возвращаются в виде набора данных, так же как и в случае с оператором SELECT. Вы не можете использовать RETURNING_VALUES или выполнить блок, вернув значения в некоторые переменные, используя INTO, даже если возвращается всего одна строка.

Для получения дополнительной информации о параметрах и объявлениях переменных, [TYPE OF] *domain*, TYPE OF COLUMN и т.д. обратитесь к главе DECLARE.

Терминатор оператора

Некоторые клиенты, особенно те, что позволяет пользователю отослать несколько операторов сразу, могут потребовать от вас, окружить оператор EXECUTE BLOCK строками SET TERM, как в этом примере:

```
SET TERM #;
EXECUTE BLOCK (...)
AS
BEGIN
  statement1;
  statement2;
END
#
SET TERM ;#
```

В качестве примера, в клиенте isql СУБД Firebird необходимо установить терминатор отличный от ";", прежде чем ввести оператор EXECUTE BLOCK. Если этого не сделать, isql попытается

выполнить часть которая была напечатана до того момента, как вы нажали клавишу Enter после строки с точкой запятой ";".

Глава 7

Процедурный язык PSQL

Procedural SQL (PSQL) — процедурное расширение языка SQL. Это подмножество языка используется для написания хранимых процедур, хранимых функций, пакетов, триггеров и PSQL блоков.

Это расширение содержит все основные конструкции классических языков программирования. Кроме того, в него входят немного модифицированные DML операторы (SELECT, INSERT, UPDATE, DELETE и др.).

Элементы PSQL

Процедурное расширение может содержать объявления локальных переменных и курсоров, операторы присваивания, условные операторы, операторы циклов, выброса пользовательского исключений, средства для обработки ошибок, отправки сообщений (событий) клиентским программам. Кроме того, в триггерах доступны специфичные контекстные переменные, такие как NEW и OLD.

В PSQL не допустимы операторы модификации метаданных (DDL операторы).

DML операторы с параметрами

В DML (SELECT, INSERT, UPDATE, DELETE и др.) операторах допустимы только именованные параметры. Если DML операторы содержат именованные параметры, то они должны быть предварительно объявлены как локальные переменные в операторе DECLARE [VARIABLE] заголовка модуля или доступны во входных или выходных параметрах PSQL модуля.

При использовании именованных параметров в DML операторах необходим префикс двоеточия «:», однако в предложении INTO символ двоеточия не обязателен. Префикс двоеточия является необязательным в операторах специфичных для PSQL, таких как операторы ветвления или присваивания. Префикс двоеточия не требуется также при вызове хранимой процедуры с помощью оператора EXECUTE PROCEDURE из другого PSQL модуля.

Транзакции

Хранимые процедуры и функции (в том числе содержащиеся в пакетах) выполняются в контексте той транзакции, в которой они были запущены. Триггеры выполняются в контексте транзакции, в которой выполнялся DML оператор, вызвавший запуск триггера. Для триггеров на событие базы данных запускается отдельная транзакция.

В PSQL не допустимы операторы старта и завершения транзакций, но существует возможность запуска оператора или блока операторов в автономной транзакции.

Структура модуля

В синтаксисе PSQL модулей можно выделить заголовок и тело. DDL операторы для их объявления являются сложными операторами, т.е. состоят из единственного оператора, который включает в себя блоки нескольких операторов. Такие операторы начинаются с глагола (CREATE, ALTER, DROP, RECREATE, CREATE OR ALTER) и завершаются последним оператором END тела модуля.

Заголовок модуля

Заголовок содержит имя модуля и описание локальных переменных. Для хранимых процедур и PSQL блоков заголовок может содержать описание входных и выходных параметров. Заголовок триггеров не может содержать входных и выходных параметров.

В заголовке триггера обязательно указывается событие (или комбинация событий), при котором триггер будет вызван автоматически.

Тело модуля

Тело PSQL модуля представляет собой блок операторов, содержащий описание выполняемых программой действий. Блок операторов заключается в операторные скобки BEGIN и END. В самих программах возможно присутствие произвольного количества блоков, как последовательных, так и вложенных друг в друга. Максимальная глубина ограничена 512 уровнями вложенности блоков. Все операторы за исключением блоков BEGIN ... END отделяются друг от друга точкой с запятой (;). Никакой другой символ не является допустимым терминатором операторов PSQL.

Изменение терминатора в isql

Здесь мы немного отвлечёмся для того, чтобы объяснить как переключить терминатор в утилите isql. Это необходимо чтобы иметь возможность определять в ней PSQL модули, не конфликтуя с самим isql, который использует тот же самый символ, точку с запятой (;), как разделитель операторов.

isql кодманда SET TERM:

Назначение: Изменение символа(ов) терминатора, чтобы избежать конфликта с терминатором в PSQL операторах.

Доступно в: ISQL.

Синтаксис:

```
SET TERM <new_terminator><old_terminator>
```

Параметры оператора SET TERM

new_terminator
Новый терминатор.

old_terminator
Старый терминатор.

При написании триггеров и хранимых процедур в текстах скриптов, создающих требуемые программные объекты базы данных, во избежание двусмысленности относительно использования символа завершения операторов (по нормам SQL это точка с запятой) применяется оператор SET TERM, который, строго говоря, не является оператором SQL, а является командой интерактивного инструмента isql. При помощи этого оператора перед началом создания триггера или хранимой процедуры задаётся символ или строка символов, являющийся завершающим в конце текста триггера или хранимой процедуры. После описания текста соответствующего программного объекта при помощи того же оператора SET TERM значение терминатора возвращается к обычному варианту — точка с запятой.

Альтернативный терминатор может быть любой произвольной строкой символов за исключением точки с запятой, пробела и апострофа. Если вы используете буквенный символ, то он будет чувствителен к регистру.

Пример 7.1. Задание альтернативного терминатора

```
SET TERM ^;

CREATE OR ALTER PROCEDURE SHIP_ORDER (
    PO_NUM CHAR(8))
AS
BEGIN
  /* Тело хранимой процедуры */
END^

/* Другие хранимые процедуры и триггеры */

SET TERM ;^

/* Другие операторы DDL */
```

Хранимые процедуры

Хранимая процедура является программой, хранящейся в области метаданных базы данных и выполняющейся на стороне сервера. К хранимой процедуре могут обращаться хранимые процедуры (в том числе и сама к себе), триггеры и клиентские программы. Если хранимая процедура вызывает саму себя, то такая хранимая процедура называется рекурсивной.

Преимущества хранимых процедур

Хранимые процедуры имеют следующие преимущества:

1. *Модульность:* Приложения, работающие с одной и той же базой данных, могут использовать одну и ту же хранимую процедуру, тем самым уменьшив размер кода приложения и устранив дублирование кода.

2. *Упрощение поддержки приложений:* При изменении хранимой процедуры, изменения отражаются сразу во всех приложениях, без необходимости их перекомпиляции.

3. *Увеличение производительности:* Поскольку хранимые процедуры выполняются на стороне сервера, а не клиента, то это уменьшает сетевой трафик, что повышает производительность.

Типы хранимых процедур

Существуют два вида хранимых процедур — выполняемые хранимые процедуры (executable stored procedures) и селективные процедуры (selectable stored procedures).

Выполняемые хранимые процедуры

Выполняемые хранимые процедуры, осуществляют обработку данных, находящихся в базе данных. Эти процедуры могут получать входные параметры и возвращать одиночный набор выходных (RETURNS) параметров. Такие процедуры выполняются с помощью оператора EXECUTE PROCEDURE. См. пример создания выполняемой хранимой процедуры в конце раздела CREATE PROCEDURE главы 5.

Хранимые процедуры выбора

Хранимые процедуры выбора обычно осуществляют выборку данных из базы данных, возвращает при этом произвольное количество строк.

Такие процедуры позволяют получать довольно сложные наборы данных, которые зачастую невозможно или весьма затруднительно получить с помощью обычных DSQL SELECT запросов. Обычно такие процедуры выполняют циклический процесс извлечения данных, возможно преобразуя их, прежде чем заполнить выходные переменные (параметры) новыми данными на каждой итерации цикла. Оператор SUSPEND, обычно расположенный в конце каждой итерации, заполняет буфер и ожидает пока вызывающая сторона не выберет (fetch) строку.

Процедуры выбора могут иметь входные параметры и выходное множество, заданное в предложении RETURNS заголовка процедуры.

Обращение к хранимой процедуре выбора осуществляется при помощи оператора SELECT (см. Выборка из селективной хранимой процедуры). См. пример создания хранимой процедуры выбора в конце раздела CREATE PROCEDURE главы 5.

Создание хранимой процедуры

Синтаксис создания выполняемых хранимых процедур и процедур выбора ничем не отличается.

Синтаксис (не полный):

```
CREATE PROCEDURE procname [(<inparam> [, <inparam> ...])]
RETURNS (<outparam> [, <outparam> ...])
  { EXTERNAL NAME '<extname>' ENGINE <engine> }  |
  { AS
      [<declarations>]
    BEGIN
      [<PSQL_statements>]
    END
  }
```

Заголовок хранимой процедуры обязательно содержит имя процедуры, которое должно быть уникальным среди имён хранимых процедур, таблиц и представлений. В нем так же может быть описано некоторое количество входных и выходных параметров. Входные параметры перечисляются после имени процедуры внутри пары скобок. Выходные параметры, которые являются обязательными для процедур выбора, перечисляются внутри пары скобок в предложении RETURNS.

Тело хранимой процедуры может содержать объявление локальных переменных, курсоров и подпрограмм. После секции объявления следует основной блок BEGIN ... END, в который заключается PSQL код процедуры. В этом блоке могут содержаться DML и PSQL операторы, а также вложенные BEGIN ... END блоки. Любой из BEGIN ... END блоков может быть пустым, в том числе и главный блок. Это позволяет разрабатывать процедуры пошагово, методом сверху вниз.

Хранимая процедура может быть расположена во внешнем модуле. В этом случае вместо тела процедуры указывается место её расположения во внешнем модуле с помощью предложения EXTERNAL NAME. Аргументом этого предложения является строка, в которой через разделитель указано имя внешнего модуля, имя процедуры внутри модуля и определённая пользователем информация. В предложении ENGINE указывается имя движка для обработки подключения внешних модулей. В Firebird для работы с внешними модулями используется движок UDR.

Более подробная информация приведена в главе Операторы DDL (CREATE PROCEDURE).

Изменение хранимой процедуры

В существующих хранимых процедурах можно изменять набор входных и выходных параметров и тело процедуры.

Синтаксис (не полный):

```
ALTER PROCEDURE procname [(<inparam> [, <inparam> ...])]
RETURNS (<outparam> [, <outparam> ...])
  { EXTERNAL NAME '<extname>' ENGINE <engine> }  |
  { AS
      [<declarations>]
    BEGIN
      [<PSQL_statements>]
    END
  }
```

Более подробная информация приведена в главе Операторы DDL (ALTER PROCEDURE).

Удаление хранимой процедуры

Для удаления хранимых процедур используется оператор DROP PROCEDURE.

Синтаксис (полный):

```
DROP PROCEDURE procname;
```

Более подробная информация приведена в главе Операторы DDL (DROP PROCEDURE).

Хранимые функции

Хранимая функция является программой, хранящейся в области метаданных базы данных и выполняющейся на стороне сервера. К хранимой функции могут обращаться хранимые процедуры, хранимые функции (в том числе и сама к себе), триггеры и клиентские программы. При обращении хранимой функции самой к себе такая хранимая функция называется рекурсивной.

В отличие от хранимых процедур хранимые функции всегда возвращают одно скалярное значение. Для возврата значения из хранимой функции используется оператор RETURN, который немедленно прекращает выполнение функции.

Создание хранимой функции

Синтаксис (не полный):

```
CREATE FUNCTION funcname [(<inparam> [, <inparam> ...])]
RETURNS <type> [COLLATE collation] [DETERMINISTIC]
  { EXTERNAL NAME '<extname>' ENGINE <engine> }  |
  { AS
      [<declarations>]
    BEGIN
      [<PSQL_statements>]
    END
  }
```

Заголовок хранимой функции обязательно содержит имя функции, которое должно быть уникальным среди имён хранимых функции. В нем так же может быть описано некоторое количество входных параметров и тип выходного результата. Входные параметры перечисляются после имени процедуры внутри пары скобок. Тип выходного результата указывается в предложении RETURNS.

Тело хранимой функции может содержать объявление локальных переменных, курсоров и подпрограмм. После секции объявления следует основной блок BEGIN ... END, в который заключается PSQL код функции. В этом блоке могут содержаться DML и PSQL операторы, а также вложенные BEGIN ... END блоки. Любой из BEGIN ... END блоков может быть пустым, в том числе и главный блок. Это позволяет разрабатывать функции пошагово, методом сверху вниз.

Хранимая функция может быть расположена во внешнем модуле. В этом случае вместо тела функции указывается место её расположения во внешнем модуле с помощью предложения EXTERNAL NAME. Аргументом этого предложения является строка, в которой через разделитель указано имя внешнего модуля, имя процедуры внутри модуля и определённая пользователем информация. В предложении ENGINE указывается имя движка для обработки подключения внешних модулей. В Firebird для работы с внешними модулями используется движок UDR.

Более подробная информация приведена в главе Операторы DDL (CREATE FUNCTION).

Изменение хранимой функции

В существующих хранимых функциях можно изменять набор входных параметров, тип результата и тело функции.

Синтаксис (не полный):

```
ALTER FUNCTION funcname [(<inparam> [, <inparam> ...])]
RETURNS <type> [COLLATE collation] [DETERMINISTIC]
  { EXTERNAL NAME '<extname>' ENGINE <engine> }  |
  { AS
      [<declarations>]
    BEGIN
      [<PSQL_statements>]
    END
  }
```

Более подробная информация приведена в главе Операторы DDL (ALTER FUNCTION).

Удаление хранимой функции

Для удаления хранимых функций используется оператор DROP FUNCTION.

Синтаксис (полный):

```
DROP FUNCTION funcname;
```

Более подробная информация приведена в главе Операторы DDL (DROP FUNCTION).

PSQL блоки

Для выполнения из декларативного SQL (DSQL) некоторых императивных действий используются анонимные (безымянные) PSQL блоки. Заголовок анонимного PSQL блока опциально может содержать входные и выходные параметры. Тело анонимного PSQL блока может содержать объявление локальных переменных, курсоров, подпрограмм и блок PSQL операторов.

Анонимный PSQL блок не определяется и сохраняется как объект метаданных, в отличии от хранимых процедур и триггеров. Он не может обращаться сам к себе.

Как и хранимые процедуры анонимные PSQL блоки могут использоваться для обработки данных или для осуществления выборки из базы данных.

Синтаксис (полный):

```
EXECUTE BLOCK
[(<inparam> = ? [, <inparam> = ? ...])]
[RETURNS (<outparam> [, <outparam> ...])]
AS
[<declarations>]
BEGIN
[<PSQL_statements>]
END
```

Параметры оператора EXECUTE BLOCK

inparam
: Описание входного параметра.

outparam
: Описание выходного параметра.

declarations
: Секция объявления локальных переменных и именованных курсоров.

PSQL_statments
: Операторы языка PSQL.

См. также: EXECUTE BLOCK.

Пакеты

Пакет — группа процедур и функций, которая представляет собой один объект базы данных.

Пакеты Firebird состоят из двух частей: заголовка (ключевое слово PACKAGE) и тела (ключевые слова PACKAGE BODY). Такое разделение очень сильно напоминает модули Delphi, заголовок соответствует интерфейсной части, а тело — части реализации.

Преимущества пакетов

Пакеты обладают следующими преимуществами:

- *Модульность*.

  Блоки взаимозависимого кода выделены в логические модули, как это сделано в других языках программирования. В программировании существует множество способов для группировки кода, например с помощью пространств имен (namespaces), модулей (units) и классов. Со стандартными процедурами и функциями базы данных это не возможно.

- *Упрощение отслеживания зависимостей*.

  Пакеты упрощают механизм отслеживания зависимостей между набором связанных процедур, а также между этим набором и другими процедурами, как упакованными, так и неупакованными.

  Каждый раз, когда упакованная подпрограмма определяет, что используется некоторый объект базы данных, информации о зависимости от этого объекта регистрируется в системных таблицах Firebird. После этого, для того чтобы удалить или изменить этот объект, вы сначала должны удалить, то что зависит от него. Поскольку зависимости от других объектов существуют только для тела пакета, это тело пакета может быть легко удалено, даже если какой-нибудь другой объект зависит от этого пакета. Когда тело удаляется, заголовок остаётся, что позволяет пересоздать это тело после того, как сделаны изменения связанные с удалённым объектом.

- *Упрощение управления разрешениями*.

  Поскольку Firebird выполняет подпрограммы с полномочиями вызывающей стороны, то каждой вызывающей подпрограмме необходимо предоставить полномочия на использования ресурсов, если эти ресурсы не являются непосредственно доступными вызывающей стороне. Использование каждой подпрограммы требует предоставления привилегий на её выполнение для пользователей и/или ролей.

  У упакованных подпрограмм нет отдельных привилегий. Привилегии действуют на пакет в целом. Привилегии, предоставленные пакетам, действительны для всех подпрограмм тела пакета, в том числе частных, и сохраняются для заголовка пакета.

- *Частные области видимости*.

  Некоторые процедуры и функции могут быть частными (private), а именно их использование разрешено только внутри определения пакета.

  Все языки программирования имеют понятие области видимости подпрограмм, которое невозможно без какой-либо формы группировки. Пакеты Firebird в этом отношении подобны модулям Delphi. Если подпрограмма не объявлена в заголовке пакета (interface), но реализована в теле (implementation), то такая подпрограмма становится частной (private). Частную подпрограмму возможно вызвать только из её пакета.

Создание пакета

Синтаксис:

```
CREATE PACKAGE package_name
AS
BEGIN
  [<package_item> ...]
END

CREATE PACKAGE BODY package_name
AS
BEGIN
  [<package_item> ...]
  [<package_body_item> ...]
END

<package_item> ::=
    <function_decl>;
  | <procedure_decl>;

<function_decl> ::=
  FUNCTION func_name [(<in_params>)]
    RETURNS <type> [COLLATE collation] [DETERMINISTIC]

<procedure_decl> ::=
  PROCEDURE proc_name [(<in_params>)] [RETURNS (<out_params>)]

<package_body_item> ::=
    <function_impl>
  | <procedure_impl>

<function_impl> ::=
  FUNCTION func_name [(<in_impl_params>)]
    RETURNS <type> [COLLATE collation] [DETERMINISTIC]
      { EXTERNAL NAME '<extname>' ENGINE <engine> }  |
      { AS
          [<declarations>]
        BEGIN
          [<PSQL_statements>]
        END
      }

<procedure_impl> ::=
  PROCEDURE proc_name [(<in_impl_params>)] [RETURNS (<out_params>)]
      { EXTERNAL NAME '<extname>' ENGINE <engine> }  |
      { AS
          [<declarations>]
        BEGIN
          [<PSQL_statements>]
        END
      }
```

Сначала необходимо создать заголовок пакета (CREATE PACKAGE), а затем — его тело (CREATE PACKAGE BODY). В теле пакеты должны быть реализованы все подпрограммы, с той же сигнатурой, что и объявленные в заголовке и в начале тела пакета. Значения по умолчанию для параметров подпрограмм не могут быть переопределены. Это означает, что они могут быть определены в теле пакета только для частных подпрограмм, которые не были объявлены.

Примеры:

Пример 7.2. Простой пакет

```
SET TERM ^;
-- заголовок пакета, только объявления подпрограмм
CREATE OR ALTER PACKAGE TEST
AS
BEGIN
  PROCEDURE P1(I INT) RETURNS (O INT); -- публичная процедура
END^

-- package body, implementation
RECREATE PACKAGE BODY TEST
AS
BEGIN
  FUNCTION F1(I INT) RETURNS INT; -- частная функция

  PROCEDURE P1(I INT) RETURNS (O INT)
  AS
  BEGIN
  END

  FUNCTION F1(I INT) RETURNS INT
  AS
  BEGIN
    RETURN 0;
  END
END^
```

Пример 7.3. Пакет для работы с сессионными переменными

```
SET TERM ^;

CREATE PACKAGE APP_VAR
AS
BEGIN
  -- Возвращает дату начала периода
  -- Функция помечена как детерминированная, что позволяет
  -- рассматривать её как инвариант в запросах
  FUNCTION GetDateBegin() RETURNS DATE DETERMINISTIC;

  -- Возвращает дату окончания периода
  -- Функция помечена как детерминированная, что позволяет
  -- рассматривать её как инвариант в запросах
  FUNCTION GetDateEnd() RETURNS DATE DETERMINISTIC;

  -- Устанавливает диапазон дат рабочего периода
  PROCEDURE SetDateRange(ADateBegin DATE, ADateEnd DATE);
END^

CREATE PACKAGE BODY APP_VAR
AS
BEGIN
  -- Возвращает дату начала периода
  FUNCTION GetDateBegin() RETURNS DATE
```

```
  AS
  BEGIN
    RETURN RDB$GET_CONTEXT('USER_SESSION', 'DATEBEGIN');
  END

  -- Возвращает дату окончания периода
  FUNCTION GetDateEnd() RETURNS DATE
  AS
  BEGIN
    RETURN RDB$GET_CONTEXT('USER_SESSION', 'DATEEND');
  END

  -- Устанавливает диапазон дат рабочего периода
  PROCEDURE SetDateRange(ADateBegin DATE, ADateEnd DATE)
  AS
  BEGIN
    RDB$SET_CONTEXT('USER_SESSION', 'DATEBEGIN', ADateBegin);
    RDB$SET_CONTEXT('USER_SESSION', 'DATEEND', ADateEnd);
  END
END^

SET TERM ;^
```

Использование:

```
-- Установка рабочего периода
EXECUTE PROCEDURE APP_VAR.SetDateRange(
  date '01.01.2000', date '31.12.2000');

-- Использование в запросах
SELECT *
FROM SALES_ORDER
WHERE bydate BETWEEN APP_VAR.GetDateBegin()
                 AND APP_VAR.GetDateEnd();
```

Пример 7.4. Пакет объединяющий внешние подпрограммы

```
SET TERM ^;

CREATE OR ALTER PACKAGE REGEXP
AS
BEGIN
  PROCEDURE preg_match(
      APattern VARCHAR(8192), ASubject VARCHAR(8192))
    RETURNS (Matches VARCHAR(8192));

  FUNCTION preg_is_match(
      APattern VARCHAR(8192), ASubject VARCHAR(8192))
    RETURNS BOOLEAN;

  FUNCTION preg_replace(
      APattern VARCHAR(8192),
      AReplacement VARCHAR(8192),
      ASubject VARCHAR(8192))
```

```
    RETURNS VARCHAR(8192);

  PROCEDURE preg_split(
      APattern VARCHAR(8192),
      ASubject VARCHAR(8192))
    RETURNS (Lines VARCHAR(8192));

  FUNCTION preg_quote(
      AStr VARCHAR(8192),
      ADelimiter CHAR(10) DEFAULT NULL)
    RETURNS VARCHAR(8192);
END^

RECREATE PACKAGE BODY REGEXP
AS
BEGIN
  PROCEDURE preg_match(
      APattern VARCHAR(8192),
      ASubject VARCHAR(8192))
    RETURNS (Matches VARCHAR(8192))
    EXTERNAL NAME 'PCRE!preg_match' ENGINE UDR;

  FUNCTION preg_is_match(
      APattern VARCHAR(8192),
      ASubject VARCHAR(8192))
    RETURNS BOOLEAN
  AS
  BEGIN
    RETURN EXISTS(
      SELECT * FROM preg_match(:APattern, :ASubject));
  END

  FUNCTION preg_replace(
      APattern VARCHAR(8192),
      AReplacement VARCHAR(8192),
      ASubject VARCHAR(8192))
    RETURNS VARCHAR(8192)
    EXTERNAL NAME 'PCRE!preg_replace' ENGINE UDR;

  PROCEDURE preg_split(
      APattern VARCHAR(8192),
      ASubject VARCHAR(8192))
    RETURNS (Lines VARCHAR(8192))
    EXTERNAL NAME 'PCRE!preg_split' ENGINE UDR;

  FUNCTION preg_quote(
      AStr VARCHAR(8192),
      ADelimiter CHAR(10))
    RETURNS VARCHAR(8192)
    EXTERNAL NAME 'PCRE!preg_quote' ENGINE UDR;
END^

SET TERM ;^
```

См. также: CREATE PACKAGE, CREATE PACKAGE BODY.

Модификация пакета

Синтаксис:

```
ALTER PACKAGE package_name
AS
BEGIN
  [<package_item> ...]
END

{ALTER | RECREATE} PACKAGE BODY package_name
AS
BEGIN
  [<package_item> ...]
  [<package_body_item> ...]
END

<package_item> ::=
    <function_decl>;
  | <procedure_decl>;

<function_decl> ::=
  FUNCTION func_name [(<in_params>)]
    RETURNS <type> [COLLATE collation] [DETERMINISTIC]

<procedure_decl> ::=
  PROCEDURE proc_name [(<in_params>)] [RETURNS (<out_params>)]

<package_body_item> ::=
    <function_impl>
  | <procedure_impl>

<function_impl> ::=
  FUNCTION func_name [(<in_impl_params>)]
    RETURNS <type> [COLLATE collation] [DETERMINISTIC]
      { EXTERNAL NAME '<extname>' ENGINE <engine> }  |
      { AS
          [<declarations>]
        BEGIN
          [<PSQL_statements>]
        END
      }

<procedure_impl> ::=
  PROCEDURE proc_name [(<in_impl_params>)] [RETURNS (<out_params>)]
      { EXTERNAL NAME '<extname>' ENGINE <engine> }  |
      { AS
          [<declarations>]
        BEGIN
          [<PSQL_statements>]
        END
      }
```

После модификации/пересоздания заголовка пакета, исходный код тела пакета сохраняется. При этом код тела пакета может стать невалидным. Столбец RDB$PACKAGES.RDB $VALID_BODY_FLAG отображает состояние тела пакета.

См. также: ALTER PACKAGE, ALTER PACKAGE BODY.

Удаление пакета

Синтаксис:

```
DROP PACKAGE package_name

DROP PACKAGE BODY package_name
```

Перед удалением заголовка пакета необходимо удалить тело пакета.

См. также: DROP PACKAGE, DROP PACKAGE BODY.

Триггеры

Триггер является программой, которая хранится в области метаданных базы данных и выполняется на стороне сервера. Напрямую обращение к триггеру невозможно. Он вызывается автоматически при наступлении одного или нескольких событий, относящихся к одной конкретной таблице (к представлению), или при наступлении одного из событий базы данных.

Триггер, вызываемый при наступлении события таблицы, связан с одной таблицей или представлением, с одним или более событиями для этой таблицы или представления (INSERT, UPDATE, DELETE) и ровно с одной фазой такого события (BEFORE или AFTER).

Триггер выполняется в той транзакции, в контексте которой выполнялась программа, вызвавшая соответствующее событие. Исключением являются триггеры, реагирующие на события базы данных. Для некоторых из них запускается транзакция по умолчанию.

Порядок срабатывания

Для каждой комбинации фаза-событие может быть определено более одного триггера. Порядок, в котором они выполняются, может быть указан явно с помощью дополнительного аргумента POSITION в определении триггера. Максимальная позиция равна 32767. Триггеры с меньшей позицией вызываются первыми.

Если предложение POSITION опущено или несколько триггеров с одинаковыми фазой и событием имеют одну и ту же позицию, то такие триггеры будут выполняться в алфавитном порядке их имен.

DML триггеры

DML триггеры вызываются при изменении состояния данных DML операциями: редактирование, добавление или удаление строк. Они могут быть определены и для таблиц и для представлений.

Варианты триггеров

Существует шесть основных вариантов соотношения событие-фаза для таблицы (представления):

до добавления новой строки	(BEFORE INSERT)
после добавления новой строки	(AFTER INSERT)
до изменения строки	(BEFORE UPDATE)
после изменения строки	(AFTER UPDATE)
до удаления строки	(BEFORE DELETE)
после удаления строки	(AFTER DELETE)

Помимо базовых форм с единственной фазой и событием Firebird поддерживает также формы с одной фазой и множеством событий, например BEFORE INSERT OR UPDATE OR DELETE или AFTER UPDATE OR DELETE или любая другая комбинация на ваш выбор.

> **Примечание**
>
> Триггеры с несколькими фазами, такие как BEFORE OR AFTER ... не поддерживаются.

Контекстные переменные NEW и OLD

В DML триггерах Firebird обеспечивает доступ к множеству контекстных переменных NEW.* и OLD.* Каждое множество является массивом всей строки: OLD.* — значение строки до изменения данных и NEW.* — требуемое ("новое") значение строки. Операторы могут ссылаться на них использую следующие формы NEW.*columname* и OLD.*columnname*. *columnname* может быть любым столбцом определённым в таблице(представлении). а не только тем что был изменён.

Контекстные переменные NEW и OLD подчиняются следующим правилам:

- Во всех триггерах контекстные переменные OLD доступны только для чтения;
- В триггерах BEFORE UPDATE и BEFORE INSERT переменные NEW доступны для чтения и записи, за исключением COMPUTED BY столбцов;
- В INSERT триггерах ссылка на переменные OLD не допускается и вызовет исключение;
- В DELETE триггерах ссылка на переменные NEW не допускается и вызовет исключение;
- Во всех AFTER триггерах переменные NEW доступны только для чтения.

Триггеры на события базы данных

Триггер, связанный с событиями базы данных, может вызываться при следующих событиях:

При соединении с базой данных	(ON CONNECT)	Перед выполнением триггера автоматически запускается транзакция по умолчанию
При отсоединении от базы данных	(ON DISCONNECT)	Перед выполнением триггера автоматически запускается транзакция по умолчанию
При старте транзакции	(ON TRANSACTION START)	Триггер выполняется в контексте текущей транзакции
При подтверждении транзакции	(ON TRANSACTION COMMIT)	Триггер выполняется в контексте текущей транзакции

При отмене транзакции	(ON TRANSACTION ROLLBACK)	Триггер выполняется в контексте текущей транзакции

DDL триггеры

DDL триггеры срабатывают на указанные события изменения метаданных в одной из фаз события. BEFORE триггеры запускаются до изменений в системных таблицах. AFTER триггеры запускаются после изменений в системных таблицах.

Переменные доступные в пространстве имён DDL_TRIGGER

Во время работы DDL триггера доступно пространство имён DDL_TRIGGER для использования в функции RDB$GET_CONTEXT. Его использование также допустимо в хранимых процедурах и функциях, вызванных DDL триггерами.

Контекст DDL_TRIGGER работает как стек. Перед возбуждением DDL триггера, значения, относящиеся к выполняемой команде, помещаются в этот стек. После завершения работы триггера значения выталкиваются. Таким образом, в случае каскадных DDL операторов, когда каждая пользовательская DDL команда возбуждает DDL триггер, и этот триггер запускает другие DDL команды, с помощью EXECUTE STATEMENT, значения переменных в пространстве имен DDL_TRIGGER будут соответствовать команде, которая вызвала последний DDL триггер в стеке вызовов.

Переменные доступные в пространстве имён DDL_TRIGGER:

- EVENT_TYPE — тип события (CREATE, ALTER, DROP)
- OBJECT_TYPE — тип объекта (TABLE, VIEW и д.р.)
- DDL_EVENT — имя события (*<ddl event item>*),

  где *<ddl event item>* = EVENT_TYPE || ' ' || OBJECT_TYPE
- OBJECT_NAME — имя объекта метаданных
- SQL_TEXT — текст SQL запроса

Создание триггера

Синтаксис (не полный):

```
CREATE TRIGGER trigname {
    <relation_trigger_legacy>
  | <relation_trigger_sql2003>
  | <database_trigger>
  | <ddl_trigger> }
  {EXTERNAL NAME '<extname>' ENGINE <engine>} |
  {
    AS
      [<declarations>]
    BEGIN
      [<PSQL_statements>]
    END
  }
```

```
<relation_trigger_legacy> ::=
  FOR {tablename | viewname}
  [ACTIVE | INACTIVE]
  {BEFORE | AFTER} <mutation_list>
  [POSITION number]

<relation_trigger_sql2003> ::=
  [ACTIVE | INACTIVE]
  {BEFORE | AFTER} <mutation_list>
  [POSITION number]
  ON {tablename | viewname}

<database_trigger> ::=
  [ACTIVE | INACTIVE]
  ON db_event
  [POSITION number]

<ddl_trigger> ::=
  [ACTIVE | INACTIVE]
  {BEFORE | AFTER} <ddl_events>
  [POSITION number]

<mutation_list> ::= <mutation> [OR <mutation> [OR <mutation>]]

<mutation> ::= { INSERT | UPDATE | DELETE }

<db_event> ::= {
    CONNECT
  | DISCONNECT
  | TRANSACTION START
  | TRANSACTION COMMIT
  | TRANSACTION ROLLBACK
}

<ddl_events> ::= {
    ANY DDL STATEMENT
  | <ddl_event_item> [{OR <ddl_event_item>} ...]
}

<ddl_event_item> ::=
    CREATE TABLE | ALTER TABLE | DROP TABLE
  | CREATE PROCEDURE | ALTER PROCEDURE | DROP PROCEDURE
  | CREATE FUNCTION | ALTER FUNCTION | DROP FUNCTION
  | CREATE TRIGGER | ALTER TRIGGER | DROP TRIGGER
  | CREATE EXCEPTION | ALTER EXCEPTION | DROP EXCEPTION
  | CREATE VIEW | ALTER VIEW | DROP VIEW
  | CREATE DOMAIN | ALTER DOMAIN | DROP DOMAIN
  | CREATE ROLE | ALTER ROLE | DROP ROLE
  | CREATE SEQUENCE | ALTER SEQUENCE | DROP SEQUENCE
  | CREATE USER | ALTER USER | DROP USER
  | CREATE INDEX | ALTER INDEX | DROP INDEX
  | CREATE COLLATION | DROP COLLATION
  | ALTER CHARACTER SET
  | CREATE PACKAGE | ALTER PACKAGE | DROP PACKAGE
  | CREATE PACKAGE BODY | DROP PACKAGE BODY
  | CREATE MAPPING | ALTER MAPPING | DROP MAPPING
```

Заголовок триггера обязательно содержит имя триггера, которое должно быть уникальным среди имён триггеров, и событие при котором срабатывает триггер. Если триггер создаётся для события таблицы, то необходимо также указать фазу события и имя таблицы.

Тело триггера может содержать объявление локальных переменных и курсоров. Оно также содержит блок операторов PSQL, который может быть пустым.

Более подробная информация приведена в главе Операторы DDL (CREATE TRIGGER).

Изменение триггера

В существующих триггерах можно изменить состояние активности, фазу события и событие (события) таблицы (представления), позицию триггера и его тело. Триггеры на событие (события) таблицы (представления) не могут быть изменены в триггеры на событие базы данных и наоборот. Если какой-либо элемент не указан, то он остаётся без изменений.

Синтаксис (не полный):

```
ALTER TRIGGER trigname
[ACTIVE | INACTIVE]
[
    {{BEFORE | AFTER} {<mutation_list> | <ddl_events>}}
  | ON db_event
]
[POSITION number]
[
  AS
    [<declarations>]
  BEGIN
    [<PSQL_statements>]
  END
]

<mutation_list> ::= <mutation> [OR <mutation> [OR <mutation>]]

<mutation> ::= { INSERT | UPDATE | DELETE }

<db_event> ::= {
    CONNECT
  | DISCONNECT
  | TRANSACTION START
  | TRANSACTION COMMIT
  | TRANSACTION ROLLBACK
}

<ddl_events> ::= {
    ANY DDL STATEMENT
  | <ddl_event_item> [{OR <ddl_event_item>} ...]
}

<ddl_event_item> ::=
    CREATE TABLE | ALTER TABLE | DROP TABLE
  | CREATE PROCEDURE | ALTER PROCEDURE | DROP PROCEDURE
  | CREATE FUNCTION | ALTER FUNCTION | DROP FUNCTION
  | CREATE TRIGGER | ALTER TRIGGER | DROP TRIGGER
  | CREATE EXCEPTION | ALTER EXCEPTION | DROP EXCEPTION
```

```
  | CREATE VIEW | ALTER VIEW | DROP VIEW
  | CREATE DOMAIN | ALTER DOMAIN | DROP DOMAIN
  | CREATE ROLE | ALTER ROLE | DROP ROLE
  | CREATE SEQUENCE | ALTER SEQUENCE | DROP SEQUENCE
  | CREATE USER | ALTER USER | DROP USER
  | CREATE INDEX | ALTER INDEX | DROP INDEX
  | CREATE COLLATION | DROP COLLATION
  | ALTER CHARACTER SET
  | CREATE PACKAGE | ALTER PACKAGE | DROP PACKAGE
  | CREATE PACKAGE BODY | DROP PACKAGE BODY
```

Более подробная информация приведена в главе Операторы DDL (ALTER TRIGGER).

Удаление триггера

Для удаления триггера используется оператор DROP TRIGGER.

Синтаксис:

```
DROP TRIGGER trigname;
```

Более подробная информация приведена в главе Операторы DDL (DROP TRIGGER).

Написание кода тела модуля

В этом разделе подробно рассматривается процедурные конструкции языка SQL и операторы доступные в теле хранимых процедур, триггеров и анонимных PSQL блоков.

Маркер двоеточия (:)

Маркер двоеточие (:) используется в PSQL, чтобы пометить ссылку на переменную в DML операторе. В остальных случаях маркер двоеточия необязателен перед именами переменных. Никогда не задавайте префикс двоеточия для контекстных переменных.

Оператор присваивания

Назначение: Присваивание переменной значения.

Доступно в: PSQL.

Синтаксис:

```
varname = <value_expr>
```

Параметры оператора присваивания

varname
Имя локальной переменной или параметра процедуры.

value_expr
Выражение, константа или переменная совместимая по типу данных с *varname*.

PSQL использует символ равенства (=) в качестве своего оператора присваивания. Оператор присваивания устанавливает переменной слева от оператора значение SQL выражения справа. Выражением может быть любое правильное выражение SQL. Оно может содержать литералы, имена внутренних переменных, арифметические, логические и строковые операции, обращения к встроенным функциям и к функциям, определённым пользователем.

Примеры:

Пример 7.5. Использование оператора присваивания

```
CREATE PROCEDURE MYPROC (
    a INTEGER,
    b INTEGER,
    name VARCHAR (30)
)
RETURNS (
    c INTEGER,
    str VARCHAR(100))
AS
BEGIN
  -- присваиваем константу
  c = 0;
  str = '';
  SUSPEND;
  -- присваиваем значения выражений
  c = a + b;
  str = name || CAST(b AS VARCHAR(10));
  SUSPEND;
  -- присваиваем значение выражения
  -- построенного с использованием запроса
  c = (SELECT 1 FROM rdb$database);
  -- присваиваем значение из контекстной переменной
  str = CURRENT_USER;
  SUSPEND;
END
```

См. также: DECLARE VARIABLE.

DECLARE

Назначение: Объявление локальной переменной, курсора или подпрограммы.

Доступно в: PSQL.

Синтаксис:

```
DECLARE {<local_var> | <cursor> | <subfunc> | <subproc>}
```

Параметры оператора DECLARE

local_var
Объявление локальной переменной.

cursor
Объявление курсора.

subfunc
Объявление подфункции.

subproc
Объявление подпроцедуры.

Оператор DECLARE предназначен для объявления локальной переменной, курсора или подпрограммы (подпроцедуры или подфункции). Каждый тип объявления будет рассмотрен отдельно.

DECLARE VARIABLE

Назначение: Объявление локальной переменной.

Доступно в: PSQL.

Синтаксис:

```
DECLARE [VARIABLE] varname <type> [NOT NULL] [COLLATE collation]
  [{= | DEFAULT} <value>] }

<value> ::= {literal | NULL | context_var}

<type> ::=
    <datatype>
  | [TYPE OF] domain
  | TYPE OF COLUMN rel.col

<datatype> ::=
    {SMALLINT | INTEGER | BIGINT}
  | BOOLEAN
  | {FLOAT | DOUBLE PRECISION}
  | {DATE | TIME | TIMESTAMP}
  | {DECIMAL | NUMERIC} [(precision [, scale])]
  | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(size)]
    [CHARACTER SET charset]
  | {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR} [VARYING] [(size)]
  | BLOB [SUB_TYPE {subtype_num | subtype_name}]
    [SEGMENT SIZE seglen] [CHARACTER SET charset]
  | BLOB [(seglen [, subtype_num])]
```

Параметры оператора DECLARE VARIABLE

varname
Имя локальной переменной.

literal
Литерал.

context_var
Любая контекстная переменная, тип которой совместим с типом локальной переменной.

datatype
Тип данных SQL.

collation
Порядок сортировки.

domain
Домен.

rel
Имя таблицы или представления.

col
Имя столбца таблицы или представления.

precision
Точность. От 1 до 18.

scale
Масштаб. От 0 до 18, должно быть меньше или равно *precision*.

size
Максимальный размер строки в символах.

charset
Набор символов.

subtype_num
Номер подтипа BLOB.

subtype_name
Мнемоника подтипа BLOB.

seqlen
Размер сегмента, не может превышать 65535.

Оператор DECLARE [VARIBALE] объявляет локальную переменную. Ключевое слово VARIABLE можно опустить. В одном операторе разрешено объявлять только одну переменную. В процедурах и триггерах можно объявить произвольное число локальных переменных, используя при этом каждый раз, новый оператор DECLARE VARAIBLE.

Имя локальной переменной должно быть уникально среди имён локальных переменных, входных и выходных параметров процедуры внутри программного объекта.

Типы данных для переменных

В качестве типа данных локальной переменной может быть любой SQL тип, за исключением массивов.

В качестве типа переменной можно указать имя домена. В этом случае, переменная будет наследовать все характеристики домена. Если перед названием домена дополнительно используется предложение "TYPE OF", то используется только тип данных домена – не проверяется (не используется) его ограничение (если оно есть в домене) на NOT NULL, CHECK ограничения и/или значения по умолчанию. Если домен текстового типа, то всегда используется его набор символов и порядок сортировки.

Локальные переменные можно объявлять, используя тип данных столбцов существующих таблиц и представлений. Для этого используется предложение TYPE OF COLUMN, после которого указывается имя таблиц или представления и через точку имя столбца. При использовании TYPE OF COLUMN наследуется только тип данных, а в случае строковых типов ещё набор символов и порядок сортировки. Ограничения и значения по умолчанию столбца никогда не используются.

Для локальных переменных можно указать ограничение NOT NULL, тем самым запретив передавать в него значение NULL. Для переменной строкового типа существует возможность задать порядок сортировки с помощью предложения COLLATE.

Локальной переменной можно устанавливать инициализирующее (начальное) значение. Это значение устанавливается с помощью предложения DEFAULT или оператора "=". В качестве значения по умолчанию может быть использовано значение NULL, литерал и любая контекстная переменная совместимая по типу данных.

> **Важно**
>
> Обязательно используйте инициализацию начальным значением для любых переменных объявленных с ограничением NOT NULL, если они не получают значение по умолчанию иным способом.

Примеры:

Пример 7.6. Различные способы объявления локальных переменных

```
CREATE OR ALTER PROCEDURE SOME_PROC
AS
  -- Объявление переменной типа INT
  DECLARE I INT;
  -- Объявление переменной типа INT не допускающей значение NULL
  DECLARE VARIABLE J INT NOT NULL;
  -- Объявление переменной типа INT со значением по умолчанию 0
  DECLARE VARIABLE K INT DEFAULT 0;
  -- Объявление переменной типа INT со значением по умолчанию 1
  DECLARE VARIABLE L INT = 1;
  -- Объявление переменной на основе домена COUNTRYNAME
  DECLARE FARM_COUNTRY COUNTRYNAME;
  -- Объявление переменной с типом равным типу домена COUNTRYNAME
  DECLARE FROM_COUNTRY TYPE OF COUNTRYNAME;
  -- Объявление переменной с типом столбца CAPITAL таблицы COUNTRY
  DECLARE CAPITAL TYPE OF COLUMN COUNTRY.CAPITAL;
BEGIN
  /* Операторы PSQL */
END
```

См. также: Типы и подтипы данных, Пользовательские типы данных — домены, CREATE DOMAIN

DECLARE CURSOR

Назначение: Объявление курсора.

Доступно в: PSQL.

Синтаксис:

```
DECLARE [VARIABLE] cursor_name [SCROLL | NO SCROLL]
CURSOR FOR (<select_statement>);
```

Параметры оператора DECLARE CURSOR

cursor_name
Имя курсора.

select_statement
Оператор SELECT.

Оператор DECLARE ... CURSOR FOR объявляет именованный курсор, связывая его с набором данных, полученным в операторе SELECT, указанном в предложении CURSOR FOR. В дальнейшем курсор может быть открыт, использоваться для обхода результирующего набора данных, и снова быть закрытым. Также поддерживаются позиционированные обновления и удаления при использовании WHERE CURRENT OF в операторах UPDATE и DELETE.

Имя курсора можно использовать в качестве ссылки на курсор, как на переменные типа запись. Текущая запись доступна через имя курсора, что делает необязательным предложение INTO в операторе FETCH.

Однонаправленные и прокручиваемые курсоры

Курсор может быть однонаправленными прокручиваемым. Необязательное предложение SCROLL делает курсор двунаправленным (прокручиваемым), предложение NO SCROLL — однонаправленным. По умолчанию курсоры являются однонаправленными.

Однонаправленные курсоры позволяют двигаться по набору данных только вперёд. Двунаправленные курсоры позволяют двигаться по набору данных не только вперёд, но и назад, а также на N позиций относительно текущего положения.

> **Предупреждение**
>
> Прокручиваемые курсоры материализуются внутри как временный набор данных, таким образом, они потребляют дополнительные ресурсы памяти/диска, поэтому пользуйтесь ими только тогда, когда это действительно необходимо.

Особенности использования курсора

- Предложение "FOR UPDATE" разрешено использовать в операторе SELECT, но оно не требуется для успешного выполнения позиционированного обновления или удаления;
- Удостоверьтесь, что объявленные имена курсоров не совпадают, ни с какими именами, определёнными позже в предложениях AS CURSOR;
- Если курсор требуется только для прохода по результирующему набору данных, то практически всегда проще (и менее подвержено ошибкам) использовать оператор FOR SELECT с предложением AS CURSOR. Объявленные курсоры должны быть явно открыты, использованы для выборки данных и закрыты. Кроме того, вы должны проверить контекстную переменную ROW_COUNT после каждой выборки и выйти из цикла, если её значение ноль. Предложение FOR SELECT делает эту проверку автоматически. Однако объявленные курсоры дают большие возможности для контроля над последовательными событиями и позволяют управлять несколькими курсорами параллельно;

- Оператор SELECT может содержать параметры, например: "SELECT NAME || :SFX FROM NAMES WHERE NUMBER = :NUM". Каждый параметр должен быть заранее объявлен как переменная PSQL (это касается также входных и выходных параметров). При открытии курсора параметру присваивается текущее значение переменной;
- Если опция прокрутки опущена, то по умолчанию принимается NO SCROLL (т.е курсор открыт для движения только вперёд). Это означает, что могут быть использованы только команды FETCH [NEXT FROM]. Другие команды будут возвращать ошибки.

> **Предупреждение**
>
> Если значение переменной PSQL, используемой в операторе SELECT, изменяется во время выполнения цикла, то её новое значение может (но не всегда) использоваться при выборке следующих строк. Лучше избегать таких ситуаций. Если вам действительно требуется такое поведение, то необходимо тщательно протестировать код и убедиться, что вы точно знаете, как изменения переменной влияют на результаты выборки. Особо отмечу, что поведение может зависеть от плана запроса, в частности, от используемых индексов. В настоящее время нет строгих правил для таких ситуаций, но в новых версиях Firebird это может измениться.

Примеры использования именованного курсора

Пример 7.7. Объявление именованного курсора

```
CREATE OR ALTER TRIGGER TBU_STOCK
BEFORE UPDATE ON STOCK
AS
  -- Объявление именованного курсора
  DECLARE C_COUNTRY CURSOR FOR (
    SELECT
      COUNTRY,
      CAPITAL
    FROM COUNTRY
  );
BEGIN
  /* Операторы PSQL */
END
```

Пример 7.8. Объявление прокручиваемого курсора

```
EXECUTE BLOCK
RETURNS (
  N INT,
  RNAME CHAR(31))
AS
  -- Объявление прокручиваемого курсора
  DECLARE C SCROLL CURSOR FOR (
    SELECT
      ROW_NUMBER() OVER(ORDER BY RDB$RELATION_NAME) AS N,
      RDB$RELATION_NAME
    FROM RDB$RELATIONS
    ORDER BY RDB$RELATION_NAME);
BEGIN
  /* Операторы PSQL */
END
```

См. также: OPEN, FETCH, CLOSE, FOR SELECT.

DECLARE PROCEDURE

Назначение: Объявление подпроцедуры.

Доступно в: PSQL.

Синтаксис:

```
DECLARE PROCEDURE subprocname [(<inparam> [, <inparam> ...])]
[RETURNS (<outparam> [, <outparam> ...])]
AS
  [<declarations>]
BEGIN
  [<PSQL_statements>]
END
```

Параметры оператора DECLARE PROCEDURE

subprocname
Имя подпроцедуры.

inparam
Описание входного параметра.

outparam
Описание выходного параметра.

declarations
Секция объявления локальных переменных, именованных курсоров.

PSQL_statements
PSQL операторы.

Оператор DECLARE PROCEDURE объявляет подпроцедуру. На подпроцедуру накладываются следующие ограничения:

- Подпрограмма не может быть вложена в другую подпрограмму. Они поддерживаются только в основном модуле (хранимой процедуре, хранимой функции, триггере и анонимном PSQL блоке);
- В настоящее время подпрограмма не имеет прямого доступа для использования переменных, курсоров и других подпрограмм из основного модуля. Кроме того, подпрограмма не может вызывать себя рекурсивно. Это может быть разрешено в будущем.

Примеры:

Пример 7.9. Использование подпроцедуры

```
SET TERM ^;
--
-- Подпроцедуры в EXECUTE BLOCK
--
```

```
EXECUTE BLOCK
RETURNS (
    name VARCHAR(31))
AS
  -- Подпроцедура, возвращающая список таблиц
  DECLARE PROCEDURE get_tables
  RETURNS(table_name VARCHAR(31))
  AS
  BEGIN
    FOR
      SELECT
        rdb$relation_name
      FROM
        rdb$relations
      WHERE
        rdb$view_blr IS NULL
      INTO table_name
    DO SUSPEND;
END

  -- Подпроцедура, возвращающая список представлений
  DECLARE PROCEDURE get_views
  RETURNS(view_name  VARCHAR(31))
  AS
  BEGIN
    FOR
      SELECT
        rdb$relation_name
      FROM
        rdb$relations
      WHERE
        rdb$view_blr IS NOT NULL
      INTO view_name
    DO SUSPEND;
  END

BEGIN
  FOR
    SELECT
        table_name
    FROM
        get_tables
    UNION ALL
    SELECT
        view_name
    FROM
        get_views
    INTO name
  DO SUSPEND;
END^
```

См. также: DECLARE FUNCTION, CREATE PROCEDURE.

DECLARE FUNCTION

Назначение: Объявление подфункции.

Доступно в: PSQL.

Синтаксис:

```
DECLARE FUNCTION subfuncname [(<inparam> [, <inparam> ...])]
RETURNS <type> [COLLATE collation] [DETERMINISTIC]
AS
  [<declarations>]
BEGIN
  [<PSQL_statements>]
END
```

Параметры оператора DECLARE FUNCTION

subfuncname
Имя подфункции.

inparam
Описание входного параметра.

type
Тип выходного результата.

collation
Порядок сортировки.

declarations
Секция объявления локальных переменных, именованных курсоров.

PSQL_statements
PSQL операторы.

Оператор DECLARE FUNCTION объявляет подфункцию. На подфункцию накладываются следующие ограничения:

- Подпрограмма не может быть вложена в другую подпрограмму. Они поддерживаются только в основном модуле (хранимой процедуре, хранимой функции, триггере и анонимном PSQL блоке);
- В настоящее время подпрограмма не имеет прямого доступа для использования переменных, курсоров и других подпрограмм из основного модуля. Кроме того, подпрограмма не может вызывать себя рекурсивно. Это может быть разрешено в будущем.

Примеры:

Пример 7.10. Использование подфункции

```
--
-- Подфункция внутри хранимой функции
--
CREATE OR ALTER FUNCTION FUNC1 (n1 INTEGER, n2 INTEGER)
  RETURNS INTEGER
AS
  -- Подфункция
  DECLARE FUNCTION SUBFUNC (n1 INTEGER, n2 INTEGER)
    RETURNS INTEGER
```

```
  AS
  BEGIN
    RETURN n1 + n2;
  END

BEGIN
  RETURN SUBFUNC(n1, n2);
END ^
```

См. также: DECLARE PROCEDURE, CREATE FUNCTION.

BEGIN ... END

Назначение: Обозначение блока операторов.

Доступно в: PSQL.

Синтаксис:

```
<block> ::=
  BEGIN
    <compound_statement>
    [<compound_statement> ...]
  END

<compound_statement> ::= {<block> | <statement>;}
```

Операторные скобки BEGIN ... END определяют блок операторов, которые выполняются как один оператор. Каждый блок начинается оператором BEGIN и завершается оператором END. Блоки могут быть вложенными, глубина вложения не ограничена. Блоки могут быть пустыми, что позволяет использовать его как заглушку, позволяющую избежать написания фиктивных операторов.

После операторов BEGIN и END точка с запятой не ставится. Однако утилита командной строки isql требует, чтобы после последнего оператора END в определении PSQL модуля следовал символ терминатора, установленного командой SET TERM. Терминатор не является частью синтаксиса PSQL.

Последний оператор END в триггере завершает работу триггера. Последний оператор END в хранимой процедуре работает в зависимости от типа процедуры:

- В селективной процедуре последний оператор END возвращает управление приложению и устанавливает значение SQLCODE равным 100, что означает, что больше нет строк для извлечения;
- В выполняемой процедуре последний оператор END возвращает управление и текущие значения выходных параметров, если таковые имеются, вызывающему приложению.

Примеры:

Пример 7.11. Использование BEGIN ... END

```
SET TERM ^;
```

```
CREATE OR ALTER PROCEDURE DEPT_BUDGET (
    DNO CHAR(3))
RETURNS (
    TOT DECIMAL(12,2))
AS
    DECLARE VARIABLE SUMB DECIMAL(12,2);
    DECLARE VARIABLE RDNO CHAR(3);
    DECLARE VARIABLE CNT  INTEGER;
BEGIN
  TOT = 0;

  SELECT
      BUDGET
  FROM
      DEPARTMENT
  WHERE DEPT_NO = :DNO
  INTO :TOT;

  SELECT
      COUNT(BUDGET)
  FROM
      DEPARTMENT
  WHERE HEAD_DEPT = :DNO
  INTO :CNT;

  IF (CNT = 0) THEN
    SUSPEND;

  FOR
      SELECT
          DEPT_NO
      FROM
          DEPARTMENT
      WHERE HEAD_DEPT = :DNO
      INTO :RDNO
  DO
  BEGIN
    EXECUTE PROCEDURE DEPT_BUDGET(:RDNO)
    RETURNING_VALUES :SUMB;
    TOT = TOT + SUMB;
  END

  SUSPEND;
END^
SET TERM ;^
```

См. также: EXIT, LEAVE, SET TERM.

IF ... THEN ... ELSE

Назначение: Условный переход.

Доступно в: PSQL.

Синтаксис:

```
IF (<condition>)
THEN <compound_statement>
[ELSE <compound_statement>]
```

Параметры оператора IF ... THEN ... ELSE

condition
Логическое условие возвращающее TRUE, FALSE или UNKNOWN.

compound_statement
Оператор или блок операторов.

Оператор условного перехода IF используется для выполнения ветвления процесса обработки данных в PSQL. Если условие возвращает значение TRUE, то выполняется оператор или блок операторов после ключевого слова THEN. Иначе (если условие возвращает FALSE или UNKNOWN) выполняется оператор или блок операторов после ключевого слова ELSE, если оно присутствует. Условие всегда заключается в круглые скобки.

Оператор ветвления

В языке PSQL отсутствует оператор ветвления CASE (SWITCH). Однако в нем доступен поисковый оператор CASE из DSQL.

```
CASE <test_expr>
  WHEN <expr> THEN <result>
  [WHEN <expr> THEN <result> ...]
  [ELSE <defaultresult>]
END

CASE
  WHEN <bool_expr> THEN <result>
  [WHEN <bool_expr> THEN <result> ...]
  [ELSE <defaultresult>]
END
```

Пример использования в PSQL.

```
...
C = CASE
      WHEN A=2 THEN 1
      WHEN A=1 THEN 3
      ELSE 0
    END;
...
```

Примеры:

Пример 7.12. Использование оператора IF

```
...
```

```
IF (FIRST IS NOT NULL) THEN
  LINE2 = FIRST || ' ' || LAST;
ELSE
  LINE2 = LAST;
...
```

См. также: WHILE ... DO, CASE.

WHILE ... DO

Назначение: Циклическое выполнение операторов.

Доступно в: PSQL.

Синтаксис:

```
WHILE (<condition>) DO
  <compound_statement>
```

Параметры оператора WHILE ... DO

condition
Логическое условие возвращающее TRUE, FALSE или UNKNOWN.

compound_statement
Оператор или блок операторов.

Оператор WHILE используется для организации циклов в PSQL. Оператор или блок операторов будут выполняться до тех пор, пока условие истинно (возвращает TRUE). Циклы могут быть вложенными, глубина вложения не ограничена.

Примеры:

Пример 7.13. Использование оператора WHILE ... DO

Процедура расчёта суммы от 1 до I для демонстрации использования цикла:

```
CREATE PROCEDURE SUM_INT (I INTEGER)
RETURNS (S INTEGER)
AS
BEGIN
  s = 0;
  WHILE (i > 0) DO
  BEGIN
    s = s + i;
    i = i - 1;
  END
END
```

При выполнении в isql

```
EXECUTE PROCEDURE SUM_INT(4);
```

результат будет следующий

```
S
==========
10
```

См. также: FOR SELECT, FOR EXECUTE STATEMENT, LEAVE, CONTINUE.

LEAVE

Назначение: Выход из цикла.

Доступно в: PSQL.

Синтаксис:

```
[label:]
{FOR <select_stmt> | WHILE (<condition>)} DO
BEGIN
  ...
  LEAVE [label];
  ...
END
```

Параметры оператора LEAVE

label
Метка.

select_stmt
Оператор SELECT.

condition
Логическое условие возвращающее TRUE, FALSE или UNKNOWN.

Оператор LEAVE моментально прекращает работу внутреннего цикла операторов WHILE или FOR. С использованием опционального параметра метки LEAVE также может выйти и из внешних циклов, при этом выполнение кода продолжается с первого оператора, следующего после прекращения блока внешнего цикла.

Примеры:

Пример 7.14. Использование оператора LEAVE

```
/*
 * Выход из цикла при возникновении ошибки вставки в таблицу NUMBERS.
 * Код продолжается со строки C = 0.
 */
...
WHILE (B < 10) DO
BEGIN
```

```
    INSERT INTO NUMBERS(B)
    VALUES (:B);
    B = B + 1;
    WHEN ANY DO
    BEGIN
        EXECUTE PROCEDURE LOG_ERROR (
             CURRENT_TIMESTAMP,
             'ERROR IN B LOOP');
        LEAVE;
    END
END
C = 0;
...
```

Пример 7.15. Использование оператора LEAVE с меткой

```
/*
 * "LEAVE LOOPA" завершает внешний цикл, а "LEAVE LOOPB" - внутренний.
 * Обратите внимание: простого оператора "LEAVE" также было бы достаточно,
 * чтобы завершить внутренний цикл.
 */
...
STMT1 = 'SELECT NAME FROM FARMS';
LOOPA:
FOR EXECUTE STATEMENT :STMT1
INTO :FARM DO
BEGIN
  STMT2 = 'SELECT NAME ' || 'FROM ANIMALS WHERE FARM = ''';
  LOOPB:
  FOR EXECUTE STATEMENT :STMT2 || :FARM || ''''
  INTO :ANIMAL DO
  BEGIN
    IF (ANIMAL = 'FLUFFY') THEN
      LEAVE LOOPB;
    ELSE IF (ANIMAL = FARM) THEN
      LEAVE LOOPA;
    ELSE
      SUSPEND;
  END
END
...
```

См. также: EXIT, CONTINUE.

CONTINUE

Назначение: Досрочное начало новой итерации цикла.

Доступно в: PSQL.

Синтаксис:

```
[label:]
```

```
{FOR <select_stmt> | WHILE (<condition>)} DO
BEGIN
  ...
  CONTINUE [label];
  ...
END
```

Параметры оператора CONTINUE

label
Метка.

select_stmt
Оператор SELECT.

condition
Логическое условие возвращающее TRUE, FALSE или UNKNOWN.

Оператор CONTINUE моментально начинает новую итерацию внутреннего цикла операторов WHILE или FOR. С использованием опционального параметра метки CONTINUE также может начинать новую итерацию для внешних циклов.

Примеры:

Пример 7.16. Использование оператора CONTINUE

```
FOR
  SELECT A, D FROM ATABLE INTO :achar, :ddate
DO BEGIN
  IF (ddate < current_data - 30) THEN
    CONTINUE;
  ELSE
    /* do stuff */
  ...
END
```

См. также: LEAVE.

EXIT

Назначение: Завершение работы процедуры, функции или триггера.

Доступно в: PSQL.

Синтаксис:

```
EXIT
```

Оператор EXIT позволяет из любой точки триггера или хранимой процедуры перейти на последний оператор END, то есть завершить выполнение программы.

Примеры:

Пример 7.17. Использование оператора EXIT в процедуре выбора.

```
CREATE PROCEDURE GEN_100
RETURNS (
  I INTEGER
)
AS
BEGIN
  I = 1;
  WHILE (1=1) DO
  BEGIN
    SUSPEND;
    IF (I=100) THEN
      EXIT;
    I = I + 1;
  END
END
```

См. также: LEAVE, SUSPEND.

SUSPEND

Передача значений параметров в буфер и приостановка выполнения процедуры (PSQL блока) до тех пор, пока вызывающая сторона не получит результат.

Доступно в: PSQL.

Синтаксис:

```
SUSPEND
```

Оператор SUSPEND передаёт значения выходных параметров в буфер и приостанавливает выполнение хранимой процедуры (PSQL блока). Выполнение остаётся приостановленным до тех пор, пока вызывающая сторона не получит содержимое буфера. Выполнение возобновляется с оператора, следующего непосредственно после оператора SUSPEND. Чаще всего это будет новой итерацией циклического процесса.

> **Примечание**
>
> 1. Приложения, использующие API интерфейсы, обычно делают выборку из хранимых процедур прозрачно.
> 2. Если оператор SUSPEND выдаётся в исполняемой (executable) хранимой процедуре, то это равносильно выполнению оператора EXIT, в результате чего завершается работа процедуры.
> 3. Оператор SUSPEND "разрывает" атомарность блока, внутри которого он находится. В случае возникновения ошибки в селективной процедуре, операторы, выполненные после последнего оператора SUSPEND, будут откачены. Операторы, выполненные до последнего оператора SUSPEND, не будут откачены, если не будет выполнен откат транзакции.

Примеры:

Пример 7.18. Использование оператора SUSPEND в процедуре выбора.

```
CREATE PROCEDURE GEN_100
RETURNS (
  I INTEGER
)
AS
BEGIN
  I = 1;
  WHILE (1=1) DO
  BEGIN
    SUSPEND;
    IF (I=100) THEN
      EXIT;
    I = I + 1;
  END
END
```

См. также: EXIT.

EXECUTE STATEMENT

Назначение: Выполнение динамически созданных SQL операторов.

Доступно в: PSQL.

Синтаксис:

```
<execute_statement> ::=
  EXECUTE STATEMENT <argument>
    [<option> ...]
    [INTO <variables>]

<argument> ::=
    paramless_stmt
  | (paramless_stmt)
  | (<stmt_with_params>) (<param_values>)

<param_values> ::= <named_values> | <positional_values>

<named_values> ::=
  paramname := value_expr
  [, paramname := value_expr ...]

<positional_values> ::= value_expr [, value_expr ...]

<option> ::=
    WITH {AUTONOMOUS | COMMON} TRANSACTION
  | WITH CALLER PRIVILEGES
  | AS USER user
  | PASSWORD password
  | ROLE role
  | ON EXTERNAL [DATA SOURCE] <connect_string>
```

```
<connect_string> ::= [<hostspec>] {filepath | db_alias}

<hostspec> ::= <tcpip_hostspec> | <netbeui_hostspec>

<tcpip_hostspec> ::= hostname:

<netbeui_hostspec> ::= \\hostname\

<variables> ::= [:]varname [, [:]varname ...]
```

Параметры оператора EXECUTE STATEMENT

paramless_stmt
Строки или переменная, содержащая не параметризованный SQL запрос.

stmt_with_params
Строки или переменная, содержащая параметризованный SQL запрос.

paramname
Имя параметра SQL запроса.

value_expr
Выражение.

user
Имя пользователя. Может быть строкой или переменной.

password
Пароль. Может быть строкой или переменной.

role
Роль. Может быть строкой или переменной.

connection_string
Строка соединения. Может быть строкой или переменной.

filepath
Путь к первичному файлу базы данных.

db_alias
Псевдоним базы данных.

hostname
Имя компьютера или IP адрес.

varname
Переменная.

Оператор EXECUTE STATEMENT принимает строковый параметр и выполняет его, как будто это оператор DSQL. Если оператор возвращает данные, то с помощью предложения INTO их можно передать в локальные переменные.

Параметризованные операторы

В DSQL операторе можно использовать параметры. Параметры могут быть именованными и позиционными (безымянные). Значение должно быть присвоено каждому параметру.

Особенности использования параметризованных операторов

1. Одновременное использование именованных и позиционных параметров в одном запросе запрещёно;
2. Если у оператора есть параметры, они должны быть помещёны в круглые скобки при вызове EXECUTE STATEMENT, независимо от вида их представления: непосредственно в виде строки, как имя переменной или как выражение;
3. Именованным параметрам должно предшествовать двоеточие (:) в самом операторе, но не при присвоении значения параметру;
4. Передача значений безымянным параметрам должна происходить в том же порядке, в каком они встречаются в тексте запроса;
5. Присвоение значений параметров должно осуществляться при помощи специального оператора ":=", аналогичного оператору присваивания языка Pascal;
6. Каждый именованный параметр может использоваться в операторе несколько раз, но только один раз при присвоении значения;
7. Для позиционных параметров число подставляемых значений должно точно равняться числу параметров (вопросительных знаков) в операторе.

Примеры:

Пример 7.19. С именованными параметрами:

```
...
DECLARE license_num VARCHAR(15);
DECLARE connect_string VARCHAR (100);
DECLARE stmt VARCHAR (100) =
  'SELECT license
   FROM cars
   WHERE driver = :driver AND location = :loc';
BEGIN
  ...
  SELECT connstr
  FROM databases
  WHERE cust_id = :id
  INTO connect_string;
  ...
  FOR
    SELECT id
    FROM drivers
    INTO current_driver
   DO
   BEGIN
     FOR
       SELECT location
       FROM driver_locations
       WHERE driver_id = :current_driver
       INTO current_location
     DO
     BEGIN
       ...
       EXECUTE STATEMENT (stmt)
       (driver := current_driver,
        loc := current_location)
       ON EXTERNAL connect_string
       INTO license_num;
```

```
      ...
```

Пример 7.20. С позиционными параметрами:

```
DECLARE license_num VARCHAR (15);
DECLARE connect_string VARCHAR (100);
DECLARE stmt VARCHAR (100) =
  'SELECT license
   FROM cars
   WHERE driver = ? AND location = ?';
BEGIN
  ...
  SELECT connstr
  FROM databases
  WHERE cust_id = :id
  INTO connect_string;
  ...
  FOR SELECT id
      FROM drivers
      INTO current_driver
  DO
  BEGIN
    FOR
      SELECT location
      FROM driver_locations
      WHERE driver_id = :current_driver
      INTO current_location
    DO
    BEGIN
      ...
      EXECUTE STATEMENT (stmt)
      (current_driver, current_location)
      ON EXTERNAL connect_string
      INTO license_num;
      ...
```

WITH {AUTONOMOUS | COMMON} TRANSACTION

По умолчанию оператор выполняется в контексте текущей транзакции. При использовании предложения WITH AUTONOMOUS TRANSACTION запускается новая транзакция с такими же параметрами, как и текущая. Она будет подтверждена, если оператор выполнился без ошибок и отменена (откачена) в противном случае. С предложением WITH COMMON TRANSACTION по возможности используется текущая транзакция.

Если оператор должен работать в отдельном соединении, то используется уже запущенная в этом соединении транзакция (если таковая транзакция имеется). В противном случае стартует новая транзакция с параметрами текущей транзакции. Любые новые транзакции, запущенные в режиме "COMMON", подтверждаются или откатываются вместе с текущей транзакцией.

WITH CALLER PRIVILEGES

По умолчанию операторы SQL выполняются с правами текущего пользователя. Спецификация WITH CALLER PRIVILEGES добавляет к ним привилегии для вызова хранимой процедуры или

триггера, так же, как если бы оператор выполнялся непосредственно подпрограммой. WITH CALLER PRIVILEGES не имеет никакого эффекта, если также присутствует предложение ON EXTERNAL.

ON EXTERNAL [DATA SOURCE]

С предложением ON EXTERNAL DATA SOURCE оператор выполняется в отдельном соединении с той же или другой базой данных, возможно даже на другом сервере. Если строка подключения имеет значение NULL или '' (пустая строка), предложение ON EXTERNAL считается отсутствующим и оператор выполняется для текущей базы данных.

При выполнении оператора в отдельном соединении используется пул соединений и пул транзакций.

Особенности пула подключений (Connection pooling)

1. Внешние соединения используют по умолчанию предложение WITH COMMON TRANSACTION и остаются открытыми до закрытия текущей транзакции. Они могут быть снова использованы при последующих вызовах оператора EXECUTE STATEMENT, но только если строка подключения точно такая же;
2. Внешние соединения, созданные с использованием предложения WITH AUTONOMOUS TRANSACTION, закрываются после выполнения оператора;
3. Операторы WITH AUTONOMOUS TRANSACTION могут использовать соединения, которые ранее были открыты операторами WITH COMMON TRANSACTION. В этом случае использованное соединение остаётся открытым и после выполнения оператора, т.к. у этого соединения есть, по крайней мере, одна не закрытая транзакция.

Особенности пула транзакций (Transaction pooling)

1. При использовании предложения WITH COMMON TRANSACTION транзакции будут снова использованы как можно дольше. Они будут подтверждаться или откатываться вместе с текущей транзакцией;
2. При использовании предложения WITH AUTONOMOUS TRANSACTION всегда запускается новая транзакция. Она будет подтверждена или отменена сразу же после выполнения оператора;

Особенности обработки исключений

При использовании предложения ON EXTERNAL дополнительное соединение всегда делается через так называемого внешнего провайдера, даже если это соединение к текущей базе данных. Одним из последствий этого является то, что вы не можете обработать исключение привычными способами. Каждое исключение, вызванное оператором, возвращает eds_connection или eds_statement ошибки. Для обработки исключений в коде PSQL вы должны использовать WHEN GDSCODE eds_connection, WHEN GDSCODE eds_statement или WHEN ANY.

> **Примечание**
>
> Если предложение ON EXTERNAL не используется, то исключения перехватываются в обычном порядке, даже если это дополнительное соединение с текущей базой данных.

Другие замечания

- Набор символов, используемый для внешнего соединения, совпадает с используемым набором для текущего соединения.

- Двухфазные транзакции не поддерживаются.

AS USER, PASSWORD и ROLE

Необязательные предложения AS USER, PASSWORD и ROLE позволяют указывать от имени какого пользователя, и с какой ролью будет выполняться SQL оператор. То, как авторизуется пользователь и открыто ли отдельное соединение, зависит от присутствия и значений параметров ON EXTERNAL [DATA SOURCE], AS USER, PASSWORD и ROLE.

- При использовании предложения ON EXTERNAL открывается новое соединение и:
  - Если присутствует, по крайней мере, один из параметров AS USER, PASSWORD и ROLE, то будет предпринята попытка нативной аутентификации с указанными значениями параметров (в зависимости от строки соединения — локально или удалённо). Для недостающих параметров не используются никаких значений по умолчанию;
  - Если все три параметра отсутствуют, и строка подключения не содержит имени сервера (или IP адреса), то новое соединение устанавливается к локальному серверу с пользователем и ролью текущего соединения. Термин 'локальный' означает 'компьютер, где установлен сервер Firebird'. Это совсем не обязательно компьютер клиента;
  - Если все три параметра отсутствуют, но строка подключения содержит имя сервера (или IP адреса), то будет предпринята попытка доверенной (trusted) авторизации к удалённому серверу. Если авторизация прошла, то удалённая операционная система назначит пользователю имя — обычно это учётная запись, под которой работает сервер Firebird.

- Если предложение ON EXTERNAL отсутствует:
  - Если присутствует, по крайней мере, один из параметров AS USER, PASSWORD и ROLE, то будет открыто соединение к текущей базе данных с указанными значениями параметров. Для недостающих параметров не используются никаких значений по умолчанию;
  - Если все три параметра отсутствуют, то оператор выполняется в текущем соединении.

> **Важно**
>
> Если значение параметра NULL или '', то весь параметр считается отсутствующим. Кроме того, если параметр считается отсутствующим, то AS USER принимает значение CURRENT_USER, а ROLE — CURRENT_ROLE. Сравнение при авторизации сделано чувствительным к регистру: в большинстве случаев это означает, что имена пользователя и роли должны быть написаны в верхнем регистре.

Предостережения

1. Не существует способа проверить синтаксис выполняемого SQL оператора;
2. Нет никаких проверок зависимостей для обнаружения удалённых столбцов в таблице или самой таблицы;
3. Выполнение оператора с помощью оператора EXECUTE STATEMENT значительно медленнее, чем при непосредственном выполнении;
4. Возвращаемые значения строго проверяются на тип данных во избежание непредсказуемых исключений преобразования типа. Например, строка '1234' преобразуется в целое число 1234, а строка 'abc' вызовет ошибку преобразования.

В целом эта функция должна использоваться очень осторожно, а вышеупомянутые факторы всегда должны приниматься во внимание. Если такого же результата можно достичь с использованием PSQL и/или DSQL, то это всегда предпочтительнее.

См. также: FOR EXECUTE STATEMENT.

FOR SELECT

Назначение: Цикл по строкам результата выполнения оператора SELECT.

Доступно в: PSQL.

Синтаксис:

```
FOR
  <select_stmt>
  [AS CURSOR cursorname]
  [INTO <variables>]
DO <compound_statement>

<variables> ::= [:]varname [, [:]varname ...]
```

Параметры оператора FOR SELECT

select_stmt
Оператор SELECT.

cursorname
Имя курсора. Должно быть уникальным среди имён курсоров PSQL модуля (хранимой процедуры, триггера или PSQL блока).

varname
Имя локальной переменной или входного/выходного параметра.

compound_statement
Оператор или блок операторов.

Оператор FOR SELECT выбирает очередную строку из таблицы (представления, хранимой процедуры выбора), после чего выполняется оператор или блок операторов. В каждой итерации цикла значения полей текущей строки копируются в локальные переменные. Добавление предложения AS CURSOR делает возможным позиционное удаление и обновление данных. Операторы FOR SELECT могут быть вложенными.

Оператор FOR SELECT может содержать именованные параметры, которые должны быть предварительно объявлены в операторе DECLARE VARIABLE, или во входных (выходных) параметрах процедуры (PSQL блока).

Оператор FOR SELECT должен содержать предложение INTO, которое располагается в конце этого оператора, или предложение AS CURSOR. На каждой итерации цикла в список переменных указанных в предложении INTO копируются значения полей текущей строки запроса. Цикл повторяется, пока не будут прочитаны все строки. После этого происходит выход из цикла. Цикл также может быть завершён до прочтения всех строк при использовании оператора LEAVE.

Необъявленный курсор

Необязательное предложение AS CURSOR создаёт именованный курсор, на который можно ссылаться (с использованием предложения WHERE CURRENT OF) внутри оператора или блока операторов следующего после предложения DO, для того чтобы удалить или модифицировать текущую строку. Кроме того, позволяется использовать ссылки на курсоры, как на переменные

типа запись. Текущая запись доступна через имя курсора. Использование предложение AS CURSOR делает предложение INTO необязательным.

Правила:

- Для разрешения неоднозначности при доступе к переменной курсора перед именем курсора необходим префикс двоеточие;
- К переменной курсора можно получить доступ без префикса двоеточия, но в этом случае, в зависимости от области видимости контекстов, существующих в запросе, имя может разрешиться как контекст запроса вместо курсора;
- Переменные курсора доступны только для чтения;
- В операторе FOR SELECT без предложения AS CURSOR необходимо использовать предложение INTO. Если указано предложение AS CURSOR, предложение INTO не требуется, но разрешено;
- Чтение из переменной курсора возвращает текущие значения полей. Это означает, что оператор UPDATE (с предложением WHERE CURRENT OF) обновит также и значения полей в переменной курсора для последующих чтений. Выполнение оператора DELETE (с предложением WHERE CURRENT OF) установит NULL для значений полей переменной курсора для последующих чтений.

> **Примечание**
>
> - Над курсором, объявленным с помощью предложения AS CURSOR нельзя выполнять операторы OPEN, FETCH и CLOSE;
> - Убедитесь, что имя курсора, определённое здесь, не совпадает ни с какими именами, созданными ранее оператором DECLARE VARIABLE;
> - Предложение FOR UPDATE, разрешённое для использования в операторе SELECT, не является обязательным для успешного выполнения позиционного обновления или удаления.

Примеры:

Пример 7.21. Использование оператора FOR SELECT

```
CREATE PROCEDURE SHOWNUMS
RETURNS (
  AA INTEGER,
  BB INTEGER,
  SM INTEGER,
  DF INTEGER)
AS
BEGIN
  FOR SELECT DISTINCT A, B
      FROM NUMBERS
    ORDER BY A, B
    INTO AA, BB
  DO
  BEGIN
    SM = AA + BB;
    DF = AA - BB;
    SUSPEND;
  END
```

```
END
```

Пример 7.22. Вложенный FOR SELECT

```
CREATE PROCEDURE RELFIELDS
RETURNS (
  RELATION CHAR(32),
  POS INTEGER,
  FIELD CHAR(32))
AS
BEGIN
  FOR SELECT RDB$RELATION_NAME
      FROM RDB$RELATIONS
      ORDER BY 1
      INTO :RELATION
  DO
  BEGIN
    FOR SELECT
          RDB$FIELD_POSITION + 1,
          RDB$FIELD_NAME
        FROM RDB$RELATION_FIELDS
        WHERE
          RDB$RELATION_NAME = :RELATION
        ORDER BY RDB$FIELD_POSITION
        INTO :POS, :FIELD
    DO
    BEGIN
      IF (POS = 2) THEN
        RELATION = ' "';
      -- Для исключения повтора имён таблиц и представлений
      SUSPEND;
    END
  END
END
```

Пример 7.23. Использование предложения AS CURSOR для позиционного удаления записи

```
CREATE PROCEDURE DELTOWN (
  TOWNTODELETE VARCHAR(24))
RETURNS (
  TOWN VARCHAR(24),
  POP INTEGER)
AS
BEGIN
  FOR SELECT TOWN, POP
      FROM TOWNS
      INTO :TOWN, :POP AS CURSOR TCUR
  DO
  BEGIN
    IF (:TOWN = :TOWNTODELETE) THEN
      -- Позиционное удаление записи
      DELETE FROM TOWNS
```

```
      WHERE CURRENT OF TCUR;
    ELSE
      SUSPEND;
  END
END
```

Пример 7.24. Использование неявно объявленного курсора как курсорной переменной

```
EXECUTE BLOCK
RETURNS (
    o CHAR(31))
AS
BEGIN
  FOR
      SELECT
          rdb$relation_name AS name
      FROM
          rdb$relations AS CURSOR c
  DO
  BEGIN
    o = c.name;
    SUSPEND;
  END
END
```

Пример 7.25. Разрешение неоднозначностей курсорной переменной внутри запросов

```
EXECUTE BLOCK
RETURNS (
    o1 CHAR(31),
    o2 CHAR(31))
AS
BEGIN
  FOR
      SELECT
          rdb$relation_name
      FROM
          rdb$relations
      WHERE
          rdb$relation_name = 'RDB$RELATIONS' AS CURSOR c
  DO
  BEGIN
    FOR
        SELECT
            -- с префиксом разрешается как курсор
            :c.rdb$relation_name x1,
            -- без префикса как псевдоним таблицы rdb$relations
            c.rdb$relation_name x2
        FROM
            rdb$relations c
        WHERE
            rdb$relation_name = 'RDB$DATABASE' AS CURSOR d
    DO
```

```
    BEGIN
      o1 = d.x1;
      o2 = d.x2;
      SUSPEND;
    END
  END
END
```

См. также: SELECT, DECLARE ... CURSOR, OPEN, CLOSE, FETCH.

FOR EXECUTE STATEMENT

Назначение: Выполнение динамически созданных SQL операторов с возвратом нескольких строк данных.

Доступно в: PSQL.

Синтаксис:

```
FOR <execute_statement> DO <compound_statement>
```

Параметры оператора FOR EXECUTE STATEMENT

execute_statement
Оператор EXECUTE STATEMENT.

compound_statement
Оператор или блок операторов.

Оператор FOR EXECUTE STATEMENT используется (по аналогии с конструкцией FOR SELECT) для операторов SELECT или EXECUTE BLOCK, возвращающих более одной строки.

Примеры:

Пример 7.26. Использование оператора EXECUTE STATEMENT.

```
CREATE PROCEDURE DynamicSampleThree (
   Q_FIELD_NAME VARCHAR(100),
   Q_TABLE_NAME VARCHAR(100)
) RETURNS(
  LINE VARCHAR(32000)
)
AS
  DECLARE VARIABLE P_ONE_LINE VARCHAR(100);
BEGIN
  LINE = '';
  FOR
    EXECUTE STATEMENT
      'SELECT T1.' || :Q_FIELD_NAME || ' FROM ' || :Q_TABLE_NAME || ' T1 '
    INTO :P_ONE_LINE
  DO
    IF (:P_ONE_LINE IS NOT NULL) THEN
      LINE = :LINE || :P_ONE_LINE || ' ';
```

```
  SUSPEND;
END
```

См. также: EXECUTE STATEMENT.

OPEN

Назначение: Открытие курсора.

Доступно в: PSQL.

Синтаксис:

```
OPEN cursor_name;
```

Параметры оператора OPEN

cursor_name
Имя курсора. Курсор с таким именем должен быть предварительно объявлен с помощью оператора DECLARE ... CURSOR.

Оператор OPEN открывает ранее объявленный курсор, выполняет объявленный в нем оператор SELECT и получает записи из результирующего набора данных. Оператор OPEN применим только к курсорам, объявленным в операторе DECLARE ... CURSOR.

> **Примечание**
>
> Если в операторе SELECT курсора имеются параметры, то они должны быть объявлены как локальные переменные или входные (выходные) параметры до того как объявлен курсор. При открытии курсора параметру присваивается текущее значение переменной.

Примеры: См. примеры в операторе FETCH.

См. также: FETCH, CLOSE, DECLARE ... CURSOR.

FETCH

Назначение: Чтение записи из набора данных, связанного с курсором.

Доступно в: PSQL.

Синтаксис:

```
FETCH cursor_name [INTO [:]var_name [, [:]var_name ...]];
```

или

```
FETCH {
    NEXT
  | PRIOR
  | FIRST
  | LAST
```

```
  | ABSOLUTE <n>
  | RELATIVE <n>
} FROM cursor_name [INTO [:]var_name [,[:]var_name ...]];
```

Параметры оператора FETCH

cursor_name
Имя курсора. Курсор с таким именем должен быть предварительно объявлен с помощью оператора DECLARE ... CURSOR.

var_name
PSQL переменная.

n
Целое число.

Оператор FETCH выбирает следующую строку данных из результирующего набора данных курсора и присваивает значения столбцов в переменные PSQL. Оператор FETCH применим только к курсорам, объявленным в операторе DECLARE VARIABLE.

Во второй версии оператора FETCH вы можете указывать в каком направлении и на сколько записей продвинется позиция курсора. Предложение NEXT указывает, что указатель курсора должен продвинуться на 1 запись вперёд. Это предложение допустимо использовать как с прокручиваемыми, там и не прокручиваемыми курсорами. Остальные предложения допустимо использовать только с прокручиваемыми курсорами. Предложение PRIOR указывает, что указатель курсора должен продвинуться на 1 запись назад. Предложение FIRST позволяет переместить позицию курсора на первую запись, а предложение LAST – на последнюю. Предложение ABSOLTE позволяет указать номер позиции, на которую будет установлен курсор. Номер позиции должен быть в диапазоне от 1 до максимального количества записей извлекаемых запросом курсора. Предложение RELATIVE позволяет указать, на какое количество записей относительно текущей позиции необходимо переместить указатель курсора. Если указано положительное число, то курсор перемещает вперёд на N позиций, если отрицательное, то назад.

Необязательное предложение INTO помещает данные из текущей строки курсора в PSQL переменные.

Позволяется использовать ссылки на курсоры, как на переменные типа запись. Текущая запись доступна через имя курсора.

Правила:

- Для разрешения неоднозначности при доступе к переменной курсора перед именем курсора необходим префикс двоеточие;
- К переменной курсора можно получить доступ без префикса двоеточия, но в этом случае, в зависимости от области видимости контекстов, существующих в запросе, имя может разрешиться как контекст запроса вместо курсора;
- Переменные курсора доступны только для чтения;
- Чтение из переменной курсора возвращает текущие значения полей. Это означает, что оператор UPDATE (с предложением WHERE CURRENT OF) обновит также и значения полей в переменной курсора для последующих чтений. Выполнение оператора DELETE (с предложением WHERE CURRENT OF) установит NULL для значений полей переменной курсора для последующих чтений.

Для проверки того, что записи набора данных исчерпаны, используется контекстная переменная ROW_COUNT, которая возвращает количество считанных оператором строк. Если произошло чтение очередной записи из набора данных, то ROW_COUNT равняется единице, иначе нулю.

Примеры:

Пример 7.27. Использования оператора FETCH

```
SET TERM ^;

CREATE OR ALTER PROCEDURE GET_RELATIONS_NAMES
RETURNS (
  RNAME CHAR(31)
)
AS
  DECLARE C CURSOR FOR (SELECT RDB$RELATION_NAME FROM RDB$RELATIONS);
BEGIN
  OPEN C;
  WHILE (1 = 1) DO
  BEGIN
    FETCH C INTO :RNAME;
    IF (ROW_COUNT = 0) THEN
      LEAVE;
    SUSPEND;
  END
  CLOSE C;
END^

SET TERM ;^
```

Пример 7.28. Использования оператора FETCH со вложенными курсорами

```
EXECUTE BLOCK
RETURNS (
    SCRIPT BLOB SUB_TYPE TEXT)
AS
DECLARE VARIABLE FIELDS VARCHAR(8191);
DECLARE VARIABLE FIELD_NAME TYPE OF RDB$FIELD_NAME;
DECLARE VARIABLE RELATION RDB$RELATION_NAME;
DECLARE VARIABLE SRC   TYPE OF COLUMN RDB$RELATIONS.RDB$VIEW_SOURCE;
-- Объявление именованного курсора
DECLARE VARIABLE CUR_R      CURSOR FOR (
    SELECT
        RDB$RELATION_NAME,
        RDB$VIEW_SOURCE
    FROM
        RDB$RELATIONS
    WHERE
        RDB$VIEW_SOURCE IS NOT NULL);
-- Объявление именованного курсора, в котором
-- используется локальная переменная
DECLARE CUR_F      CURSOR FOR (
    SELECT
```

```
        RDB$FIELD_NAME
    FROM
        RDB$RELATION_FIELDS
    WHERE
        -- Важно переменная должна быть объявлена ранее
        RDB$RELATION_NAME = :RELATION);
BEGIN
  OPEN CUR_R;
  WHILE (1 = 1) DO
  BEGIN
    FETCH CUR_R
    INTO :RELATION, :SRC;
    IF (ROW_COUNT = 0) THEN
      LEAVE;

    FIELDS = NULL;
    -- Курсор CUR_F будет использовать значение
    -- переменной RELATION инициализированной выше
    OPEN CUR_F;
    WHILE (1 = 1) DO
    BEGIN
      FETCH CUR_F
      INTO :FIELD_NAME;
      IF (ROW_COUNT = 0) THEN
        LEAVE;
      IF (FIELDS IS NULL) THEN
        FIELDS = TRIM(FIELD_NAME);
      ELSE
        FIELDS = FIELDS || ', ' || TRIM(FIELD_NAME);
    END
    CLOSE CUR_F;

    SCRIPT = 'CREATE VIEW ' || RELATION;

    IF (FIELDS IS NOT NULL) THEN
      SCRIPT = SCRIPT || ' (' || FIELDS || ')';

    SCRIPT = SCRIPT || ' AS ' || ASCII_CHAR(13);
    SCRIPT = SCRIPT || SRC;

    SUSPEND;
  END
  CLOSE CUR_R;
END
```

Пример 7.29. Пример использования оператора FETCH с прокручиваемым курсором

```
EXECUTE BLOCK
RETURNS (
  N INT,
  RNAME CHAR(31))
AS
  DECLARE C SCROLL CURSOR FOR (
    SELECT
      ROW_NUMBER() OVER(ORDER BY RDB$RELATION_NAME) AS N,
```

```
      RDB$RELATION_NAME
    FROM RDB$RELATIONS
    ORDER BY RDB$RELATION_NAME);
BEGIN
  OPEN C;
  -- перемещаемся на первую запись (N=1)
  FETCH FIRST FROM C;
  RNAME = C.RDB$RELATION_NAME;
  N = C.N;
  SUSPEND;
  -- перемещаемся на 1 запись вперёд (N=2)
  FETCH NEXT FROM C;
  RNAME = C.RDB$RELATION_NAME;
  N = C.N;
  SUSPEND;
  -- перемещаемся на пятую запись (N=5)
  FETCH ABSOLUTE 5 FROM C;
  RNAME = C.RDB$RELATION_NAME;
  N = C.N;
  SUSPEND;
  -- перемещаемся на 1 запись назад (N=4)
  FETCH PRIOR FROM C;
  RNAME = C.RDB$RELATION_NAME;
  N = C.N;
  SUSPEND;
  -- перемещаемся на 3 записи вперёд (N=7)
  FETCH RELATIVE 3 FROM C;
  RNAME = C.RDB$RELATION_NAME;
  N = C.N;
  SUSPEND;
  -- перемещаемся на 5 записей назад (N=2)
  FETCH RELATIVE -5 FROM C;
  RNAME = C.RDB$RELATION_NAME;
  N = C.N;
  SUSPEND;
  -- перемещаемся на первую запись (N=1)
  FETCH FIRST FROM C;
  RNAME = C.RDB$RELATION_NAME;
  N = C.N;
  SUSPEND;
  -- перемещаемся на последнюю запись
  FETCH LAST FROM C;
  RNAME = C.RDB$RELATION_NAME;
  N = C.N;
  SUSPEND;
  CLOSE C;
END
```

Пример 7.30. Использование неявно объявленного курсора как курсорной переменной

```
EXECUTE BLOCK
RETURNS (
    o CHAR(31))
AS
BEGIN
```

```
  FOR
      SELECT
          rdb$relation_name AS name
      FROM
          rdb$relations
      AS CURSOR c
  DO
  BEGIN
    o = c.name;
    SUSPEND;
  END
END
```

См. также: OPEN, CLOSE, DECLARE ... CURSOR.

CLOSE

Назначение: Закрытие курсора.

Доступно в: PSQL.

Синтаксис:

```
CLOSE cursor_name;
```

Параметры оператора CLOSE

cursor_name
: Имя открытого курсора. Курсор с таким именем должен быть предварительно объявлен с помощью оператора DECLARE ... CURSOR.

Оператор CLOSE закрывает открытый курсор. Любые все ещё открытые курсоры будут автоматически закрыты после выполнения кода триггера, хранимой процедуры или выполнимого блока, в пределах кода которого он был открыт. Оператор CLOSE применим только к курсорам, объявленным в операторе DECLARE ... CURSOR.

Примеры: См. примеры в операторе FETCH.

См. также: FETCH, OPEN, DECLARE ... CURSOR.

IN AUTONOMOUS TRANSACTION

Назначение: Выполнение оператора или блока операторов в автономной транзакции.

Доступно в: PSQL.

Синтаксис:

```
```

```
IN AUTONOMOUS TRANSACTION DO <compound_statement>
```

Параметры оператора IN AUTONOMOUS TRANSACTION

compound_statement
Оператор или блок операторов.

Оператор IN AUTONOMOUS TRANSACTION позволяет выполнить оператор или блок операторов в автономной транзакции. Код, работающий в автономной транзакции, будет подтверждаться сразу же после успешного завершения независимо от состояния родительской транзакции. Это бывает нужно, когда определённые действия не должны быть отменены, даже в случае возникновения ошибки в родительской транзакции.

Автономная транзакция имеет тот же уровень изоляции, что и родительская транзакция. Любое исключение, вызванное или появившееся в блоке кода автономной транзакции, приведёт к откату автономной транзакции и отмене всех внесённых изменений. Если код будет выполнен успешно, то автономная транзакция будет подтверждена.

Примеры:

Пример 7.31. Использование автономных транзакций

```
/**
 * Использование автономной транзакции в триггере на событие подключения к базе
 * данных для регистрации всех попыток соединения, в том числе и неудачных.
 */
CREATE TRIGGER TR_CONNECT ON CONNECT
AS
BEGIN
  -- Все попытки соединения с БД сохраняем в журнал
  IN AUTONOMOUS TRANSACTION DO
    INSERT INTO LOG(MSG)
    VALUES ('USER ' || CURRENT_USER || ' CONNECTS.');
  IF (CURRENT_USER IN (SELECT
                           USERNAME
                       FROM
                           BLOCKED_USERS)) THEN
  BEGIN
    -- Сохраняем в журнал, что попытка соединения
    -- с БД оказалась неудачной
    -- и отправляем сообщение о событии
    IN AUTONOMOUS TRANSACTION DO
    BEGIN
      INSERT INTO LOG(MSG)
      VALUES ('USER ' || CURRENT_USER || ' REFUSED.');
      POST_EVENT 'CONNECTION ATTEMPT' || ' BY BLOCKED USER!';
    END
    -- теперь вызываем исключение
    EXCEPTION EX_BADUSER;
  END
END
```

См. также: Управление транзакциями.

POST_EVENT

Назначение: Посылка события (сообщения) клиентским приложениям.

Доступно в: PSQL.

Синтаксис:

```
POST_EVENT event_name
```

Параметры оператора POST_EVENT

event_name
Имя события, ограничено 64 символами.

Оператор POST_EVENT сообщает о событии менеджеру событий, который сохраняет его в таблице событий. При подтверждении транзакции менеджер событий информирует приложения, ожидающие это событие.

В качестве имени события может быть использован строковый литерал, переменная или любое правильное SQL выражение.

Примеры:

Пример 7.32. Оповещёние приложения о вставке записи в таблицу SALES

```
SET TERM ^;
CREATE TRIGGER POST_NEW_ORDER FOR SALES
ACTIVE AFTER INSERT POSITION 0
AS
BEGIN
  POST_EVENT 'new_order';
END^
SET TERM ;^
```

Обработка ошибок

В Firebird существуют PSQL операторы для обработки ошибок и исключений в модулях. Существует множество встроенных исключений, которые возникают в случае возникновения стандартных ошибок при работе с DML и DDL операторами. Полный список этих ошибок перечислен в приложении Обработка ошибок, коды и сообщения, а именно в разделах Коды ошибок SQLSTATE и их описание и Коды ошибок GDSCODE их описание, и SQLCODE.

В том же приложении описаны DDL операторы для создания пользовательских исключений, которые могут быть определены вашего собственного приложения, вместе с подсказками об использовании их. См. CREATE EXCEPTION.

Пользовательские исключения сохранются в таблице RDB$EXCEPTIONS.

EXCEPTION

Назначение: Возбуждение пользовательского исключения или повторный вызов исключения.

Доступно в: PSQL.

Синтаксис:

```
EXCEPTION exception_name [custom_message | USING (<value_list>)]

<value_list> ::=  <val> [, <val> [, <val> ...]]
```

Параметры оператора EXCEPTION

exception_name
: Имя исключения.

custom_message
: Альтернативный текст сообщения, выдаваемый при возникновении исключения. Максимальная длина текстового сообщения составляет 1021 байт.

val
: Значения, которыми заменяются слоты в тексте сообщения исключения.

Оператор EXCEPTION возбуждает пользовательское исключение с указанным именем. При возбуждении исключения можно также указать альтернативный текст сообщения, который заменит текст сообщения заданным при создании исключения.

Текст сообщения исключения может содержать слоты для параметров, которые заполняются при возбуждении исключения. Для передачи значений параметров в исключение используется предложение USING. Параметры рассматриваются слева направо. Каждый параметр передаётся в оператор возбуждающий исключение как N-ый, N начинается с 1:

- Если N-ый параметр не передан, его слот не заменяется;
- Если передано значение NULL, слот будет заменён на строку '***null***';
- Если количество передаваемых параметров будет больше, чем содержится в сообщении исключения, то лишние будут проигнорированы;
- Максимальный номер параметра равен 9;
- Общая длина сообщения, включая значения параметров, ограничена 1053 байтами.

> **Примечание**
>
> Статус вектор генерируется, используя комбинацию кодов *isc_except*, *<exception number>*, *isc_formatted_exception*, *<formatted exception message>*, *<exception parameters>*.
>
> Поскольку используется новый код ошибки (*isc_formatted_exception*), клиент должен быть версии 3.0 или по крайней мере использовать firebird.msg от версии 3.0 для того чтобы правильно преобразовать статус вектор в строку.

Предупреждение

Если в тексте сообщения, встретится номер слота параметра больше 9, то второй и последующий символ будут восприняты как литералы. Например, @10 будет воспринято как @1 после которого следует литерал 0.

Пример:

```
CREATE EXCEPTION ex1
'something wrong in @1 @2 @3 @4 @5 @6 @7 @8 @9 @10 @11';

EXECUTE BLOCK AS
BEGIN
  EXCEPTION ex1 USING ('a','b','c','d','e','f','g','h','i');
END^
```

```
Statement failed, SQLSTATE = HY000
exception 1
-EX1
-something wrong in a b c d e f g h i a0 a1
```

Исключение может быть обработано в операторе WHEN ... DO. Если пользовательское исключение не было обработано в триггере или в хранимой процедуре, то все выполненные действия отменяются, вызвавшая программа получает текст, заданный при создании исключения или альтернативный текст сообщения.

В блоке обработки исключений (и только в нем), вы можете повторно вызвать пойманное исключение или ошибку, вызывая оператор EXCEPTION без параметров. Вне блока с исключением такой вызов не имеет никакого эффекта.

Примеры:

Пример 7.33. Вызов исключения

```
CREATE OR ALTER PROCEDURE SHIP_ORDER (
    PO_NUM CHAR(8))
AS
DECLARE VARIABLE ord_stat  CHAR(7);
DECLARE VARIABLE hold_stat CHAR(1);
DECLARE VARIABLE cust_no   INTEGER;
DECLARE VARIABLE any_po    CHAR(8);
BEGIN
  SELECT
      s.order_status,
      c.on_hold,
      c.cust_no
  FROM
      sales s, customer c
  WHERE
      po_number = :po_num AND
      s.cust_no = c.cust_no
  INTO :ord_stat,
```

```
      :hold_stat,
      :cust_no;

  /* Этот заказ уже отправлен на поставку. */
  IF (ord_stat = 'shipped') THEN
    EXCEPTION order_already_shipped;
  /* Другие операторы */
END
```

Пример 7.34. Вызов исключения с заменой исходного сообщения альтернативным

```
CREATE OR ALTER PROCEDURE SHIP_ORDER (
    PO_NUM CHAR(8))
AS
DECLARE VARIABLE ord_stat  CHAR(7);
DECLARE VARIABLE hold_stat CHAR(1);
DECLARE VARIABLE cust_no   INTEGER;
DECLARE VARIABLE any_po    CHAR(8);
BEGIN
  SELECT
      s.order_status,
      c.on_hold,
      c.cust_no
  FROM
      sales s, customer c
  WHERE
      po_number = :po_num AND
      s.cust_no = c.cust_no
  INTO :ord_stat,
       :hold_stat,
       :cust_no;

  /* Этот заказ уже отправлен на поставку. */
  IF (ord_stat = 'shipped') THEN
    EXCEPTION order_already_shipped 'Order status is "' || ord_stat || '"';
  /* Другие операторы */
END
```

Пример 7.35. Использование параметризованного исключения

```
CREATE EXCEPTION EX_BAD_SP_NAME
  'Name of procedures must start with ''@1'' : ''@2''';
...
CREATE TRIGGER TRG_SP_CREATE BEFORE CREATE PROCEDURE
AS
  DECLARE SP_NAME VARCHAR(255);
BEGIN
  SP_NAME = RDB$GET_CONTEXT('DDL_TRIGGER', 'OBJECT_NAME');
  IF (SP_NAME NOT STARTING 'SP_') THEN
    EXCEPTION EX_BAD_SP_NAME USING ('SP_', SP_NAME);
END^
```

Пример 7.36. Регистрация ошибке в журнале и повторное её возбуждение в блоке WHEN

```
CREATE PROCEDURE ADD_COUNTRY (
    ACountryName COUNTRYNAME,
    ACurrency VARCHAR(10) )
AS
BEGIN
  INSERT INTO country (country,
                       currency)
  VALUES (:ACountryName,
          :ACurrency);
  WHEN ANY DO
  BEGIN
    -- Записываем ошибку в журнал
    IN AUTONOMOUS TRANSACTION DO
      INSERT INTO ERROR_LOG (PSQL_MODULE,
                             GDS_CODE,
                             SQL_CODE,
                             SQL_STATE)
      VALUES ('ADD_COUNTRY',
              GDSCODE,
              SQLCODE,
              SQLSTATE);
    -- Повторно возбуждаем ошибку
    EXCEPTION;
  END
END
```

См. также: CREATE EXCEPTION, WHEN ... DO.

WHEN ... DO

Назначение: Обработка ошибок.

Доступно в: PSQL.

Синтаксис:

```
WHEN {<error> [, <error> ...] | ANY}
DO <compound_statement>

<error> ::= {
    EXCEPTION exception_name
  | SQLCODE number
  | GDSCODE errcode
  | SQLSTATE 'sqlstate_code'
}
```

Параметры оператора WHEN ... DO

exception_name
Имя исключения.

number
Код ошибки SQLCODE.

errcode
Символическое имя ошибки GDSCODE.

sqlstate_code
Код ошибки SQLSTATE.

compound_statement
Оператор или блок операторов.

Оператор WHEN ... DO используется для обработки ошибочных ситуаций и пользовательских исключений. Оператор перехватывает все ошибки и пользовательские исключения, перечисленные после ключевого слова WHEN. Если после ключевого слова WHEN указано ключевое слово ANY, то оператор перехватывает любые ошибки и пользовательские исключения, даже если они уже были обработаны в вышестоящем WHEN блоке.

Оператор WHEN ... DO должен находиться в самом конце блока операторов перед оператором END.

После ключевого слова DO следует оператор или блок операторов, заключённый в операторные скобки BEGIN и END, которые выполняют некоторую обработку возникшей ситуации. В этом операторе (блоке операторов) доступны контекстные переменные SQLCODE, GDSCODE, SQLSTATE. Там же разрешён оператор повторного вызова ошибки или исключительной ситуации EXCEPTION (без параметров).

Важно

После предложения WHEN GDSCODE вы должны использовать символьные имена - такие, как *grant_obj_notfound* и т.д. Но в операторе или блоке операторов после предложения DO доступна контекстная переменная GDSCODE, которая содержит целое число. Для сравнения его с определённой ошибкой вы должны использовать числовое значение, например, 335544551 для *grant_obj_notfound*.

Оператор WHEN ... DO вызывается только в том случае, если произошло одно из указанных в его условии событий. В случае выполнения оператора (даже если в нем фактически не было выполнено никаких действий) ошибка или пользовательское исключение не прерывает и не отменяет действий триггера или хранимой процедуры, где был выдан этот оператор, работа продолжается, как если бы никаких исключительных ситуаций не было. Однако в этом случае будет отменено действие DML оператора (SELECT, INSERT, UPDATE, DELETE, MERGE), который вызвал ошибку и все ниже находящиеся операторы в том же блоке операторов не будут выполнены.

Важно

Если ошибка вызвана не одним из DML операторов (SELECT, INSERT, UPDATE, DELETE, MERGE), то будет отменен не только оператор вызвавший ошибку, а весь блок операторов. Кроме того, действия в операторе WHEN ... DO так же будут откачены. Это относится также и к оператору выполнения хранимой процедуры EXECUTE PROCEDURE. Подробнее смотри в CORE-4483.

Область действия оператора WHEN ... DO

Оператор перехватывает ошибки и исключения в текущем блоке операторов. Он также перехватывает подобные ситуации во вложенных блоках, если эти ситуации не были в них обработаны.

Оператор WHEN ... DO видит все изменения, произведённые до оператора вызвавшего ошибку. Однако если вы попытаетесь запротоколировать их в автономной транзакции, то эти изменения будут не доступны, поскольку на момент старта автономной транзакции, транзакция, в которой произошли эти изменения, не подтверждена.

Примеры:

Пример 7.37. Замена стандартной ошибки своей.

```
CREATE EXCEPTION COUNTRY_EXIST '';
SET TERM ^;
CREATE PROCEDURE ADD_COUNTRY (
    ACountryName COUNTRYNAME,
    ACurrency VARCHAR(10) )
AS
BEGIN
  INSERT INTO country (country, currency)
  VALUES (:ACountryName, :ACurrency);

  WHEN SQLCODE -803 DO
    EXCEPTION COUNTRY_EXIST 'Такая страна уже добавлена!';
END^
SET TERM ^;
```

Пример 7.38. Регистрация ошибке в журнале и повторное её возбуждение в блоке WHEN.

```
CREATE PROCEDURE ADD_COUNTRY (
    ACountryName COUNTRYNAME,
    ACurrency VARCHAR(10) )
AS
BEGIN
  INSERT INTO country (country,
                       currency)
  VALUES (:ACountryName,
          :ACurrency);
  WHEN ANY DO
  BEGIN
    -- Записываем ошибку в журнал
    IN AUTONOMOUS TRANSACTION DO
      INSERT INTO ERROR_LOG (PSQL_MODULE,
                             GDS_CODE,
                             SQL_CODE,
                             SQL_STATE)
      VALUES ('ADD_COUNTRY',
              GDSCODE,
              SQLCODE,
              SQLSTATE);
```

```
    -- Повторно возбуждаем ошибку
    EXCEPTION;
  END
END
```

Пример 7.39. Обработка в одном WHEN … DO блоке нескольких ошибок

```
...
WHEN GDSCODE GRANT_OBJ_NOTFOUND,
    GDSCODE GRANT_FLD_NOTFOUND,
    GDSCODE GRANT_NOPRIV,
    GDSCODE GRANT_NOPRIV_ON_BASE
DO
BEGIN
 EXECUTE PROCEDURE LOG_GRANT_ERROR(GDSCODE);
 EXIT;
END
...
```

Пример 7.40. Перехват ошибок по коду SQLSTATE.

```
EXECUTE BLOCK
AS
    DECLARE VARIABLE I INT;
BEGIN
  BEGIN
    I = 1 / 0;
    WHEN SQLSTATE '22003' DO
      EXCEPTION E_CUSTOM_EXCEPTION
        'Numeric value out of range.';
    WHEN SQLSTATE '22012' DO
      EXCEPTION E_CUSTOM_EXCEPTION 'Division by zero.';
    WHEN SQLSTATE '23000' DO
      EXCEPTION E_CUSTOM_EXCEPTION
        'Integrity constraint violation.';
  END
END
```

См. также: EXCEPTION, Коды ошибок SQLSTATE и их описание, Коды ошибок GDSCODE их описание, и SQLCODE, GDSCODE, SQLCODE, SQLSTATE.

Глава 8

Встроенные функции и переменные

Контекстные переменные

CURRENT_CONNECTION

Доступно в: DSQL, PSQL.

Синтаксис:

```
CURRENT_CONNECTION
```

Тип возвращаемого результата: INTEGER

Переменная CURRENT_CONNECTION хранит уникальный идентификатор текущего соединения. Значение переменной хранится в странице заголовка базы и сбрасывается после restore. Переменная увеличивается на единицу при каждом последующем соединении с базой данных (соединения также могут быть внутренними вызванными самим ядром). Следовательно, переменная показывает количество подключений произошедших к базе после её восстановления (или после её создания).

Примеры:

Пример 8.1. Использование переменной CURRENT_CONNECTION

```
SELECT CURRENT_CONNECTION FROM RDB$DATABASE
```

См. также: CURRENT_TRANSACTION.

CURRENT_DATE

Доступно в: DSQL, PSQL, ESQL.

Синтаксис:

```
CURRENT_DATE
```

Тип возвращаемого результата: DATE

Переменная CURRENT_DATE возвращает текущую дату сервера.

Примеры:

Пример 8.2. Использование переменной CURRENT_DATE

```
CREATE DOMAIN DDATE_DNN AS
DATE DEFAULT CURRENT_DATE NOT NULL
```

См. также: 'TODAY', CURRENT_TIMESTAMP, CURRENT_TIME.

CURRENT_ROLE

Доступно в: DSQL, PSQL.

Синтаксис:

```
CURRENT_ROLE
```

Тип возвращаемого результата: VARCHAR(31)

Контекстная переменная CURRENT_ROLE служит для определения роли, с которой произошло подключение к базе данных. В случае если произошло подключение без указания роли, переменная принимает значение NONE.

Примеры:

Пример 8.3. Использование переменной CURRENT_ROLE

```
SELECT CURRENT_ROLE FROM RDB$DATABASE
```

Примечание

Такое же значение можно будет получить и в результате выполнения запроса:

```
SELECT RDB$GET_CONTEXT ('SYSTEM', 'CURRENT_ROLE')
FROM RDB$DATABASE;
```

См. также: RDB$GET_CONTEXT.

CURRENT_TIME

Доступно в: DSQL, PSQL, ESQL.

Синтаксис:

```
CURRENT_TIME [(<precision>)]

<precision> ::= 0 | 1 | 2 | 3
```

Параметры контекстной переменной CURRENT_TIME

precision
Точность. Значение по умолчанию 0. Не поддерживается в ESQL.

Тип возвращаемого результата: TIME

Переменная CURRENT_TIME возвращает текущее время сервера. Точность определяет, сколько учитывать знаков после запятой в долях секунды.

> **Примечание**
>
> В блоке кода PSQL (процедура, триггер, исполняемый блок) значение CURRENT_TIME не меняется по мере выполнения. При вызове вложенного кода, значение также не изменится и будет равно значению в коде самого верхнего уровня. Для определения реального времени используйте другие переменные, например 'NOW' (с полным приведением типа данных).

Примеры:

Пример 8.4. Использование переменной CURRENT_TIME

```
SELECT CURRENT_TIME(2) FROM RDB$DATABASE;
-- результат будет (например) 23:35:33.1200
```

См. также: 'NOW', CURRENT_TIMESTAMP, CURRENT_DATE.

CURRENT_TIMESTAMP

Доступно в: DSQL, PSQL, ESQL.

Синтаксис:

```
CURRENT_TIMESTAMP [(<precision>)]

<precision> ::= 0 | 1 | 2 | 3
```

Параметры контекстной переменной CURRENT_TIMESTAMP

precision
Точность. Значение по умолчанию 0. Не поддерживается в ESQL.

Тип возвращаемого результата: TIMESTAMP

Переменная CURRENT_TIMESTAMP возвращает текущую дату и время сервера. Точность определяет, сколько учитывать знаков после запятой в долях секунды.

Примечание

В блоке кода PSQL (процедура, триггер, исполняемый блок) значение CURRENT_TIMESTAMP не меняется по мере выполнения. При вызове вложенного кода, значение также не изменится и будет равно значению в коде самого верхнего уровня. Для определения реального времени используйте другие переменные, например 'NOW' (с полным приведением типа данных).

Примеры:

Пример 8.5. Использование переменной CURRENT_TIMESTAMP

```
SELECT CURRENT_TIMESTAMP(2) FROM RDB$DATABASE;
-- результат будет (например) 02.03.2014 23:35:33.1200
```

См. также: 'NOW', CURRENT_TIME, CURRENT_DATE.

CURRENT_TRANSACTION

Доступно в: DSQL, PSQL.

Синтаксис:

```
CURRENT_TRANSACTION
```

Тип возвращаемого результата: INTEGER

Переменная CURRENT_TRANSACTION содержит уникальный номер текущей транзакции.

Значение CURRENT_TRANSACTION хранится в странице заголовка базы данных и сбрасывается в 0 после восстановления (или создания базы). Оно увеличивается при старте новой транзакции.

Примеры:

Пример 8.6. Использование переменной CURRENT_TRANSACTION

```
SELECT CURRENT_TRANSACTION FROM RDB$DATABASE;

NEW.TRANS_ID = CURRENT_TRANSACTION;
```

См. также: CURRENT_CONNECTION, RDB$GET_CONTEXT.

CURRENT_USER

Доступно в: DSQL, PSQL.

Синтаксис:

```
CURRENT_USER
```

Тип возвращаемого результата: VARCHAR(31)

Переменная CURRENT_USER содержит имя текущего подключенного пользователя базы данных.

Примеры:

Пример 8.7. Использование переменной CURRENT_USER

```
NEW.ADDED_BY = CURRENT_USER;
```

См. также: USER, CURRENT_ROLE.

DELETING

Доступно в: PSQL.

Синтаксис:

```
DELETING
```

Тип возвращаемого результата: BOOLEAN

Контекстная переменная DELETING доступна только в коде табличных триггеров. Используется в триггерах на несколько типов событий и показывает, что триггер сработал при выполнении операции DELETE.

Примеры:

Пример 8.8. Использование переменной DELETING

```
...
IF (DELETING) THEN
BEGIN
  INSERT INTO REMOVED_CARS (
    ID, MAKE, MODEL, REMOVED)
  VALUES (
    OLD.ID, OLD.MAKE, OLD.MODEL, CURRENT_TIMESTAMP);
END
...
```

См. также: INSERTING, UPDATING.

GDSCODE

Доступно в: PSQL.

Синтаксис:

```
GDSCODE
```

Тип возвращаемого результата: INTEGER

В блоке обработки ошибок "WHEN ... DO" контекстная переменная GDSCODE содержит числовое представление текущего кода ошибки Firebird. До версии Firebird 2.0 GDSCODE можно было получить только с использованием конструкции WHEN GDSCODE. Теперь эту контекстную переменную можно также использовать в блоках WHEN ANY, WHEN SQLCODE и WHEN EXCEPTION при условии, что код ошибки соответствует коду ошибки Firebird. Вне обработчика ошибок GDSCODE всегда равен 0. Вне PSQL GDSCODE не существует вообще.

Примеры:

Пример 8.9. Использование переменной GDSCODE

```
...
WHEN GDSCODE GRANT_OBJ_NOTFOUND,
     GDSCODE GRANT_FLD_NOTFOUND,
     GDSCODE GRANT_NOPRIV,
     GDSCODE GRANT_NOPRIV_ON_BASE
DO
BEGIN
  EXECUTE PROCEDURE LOG_GRANT_ERROR(GDSCODE);
  EXIT;
END
...
```

> **Примечание**
>
> Обратите внимание, пожалуйста: после, WHEN GDSCODE вы должны использовать символьные имена — такие, как *grant_obj_notfound* и т.д. Но контекстная переменная GDSCODE - целое число. Для сравнения его с определённой ошибкой вы должны использовать числовое значение, например, 335544551 для *grant_obj_notfound*.

См. также: SQLCODE, SQLSTATE.

INSERTING

Доступно в: PSQL.

Синтаксис:

```
INSERTING
```

Тип возвращаемого результата: BOOLEAN

Контекстная переменная INSERTING доступна только коде табличных триггеров. Используется в триггерах на несколько типов событий и показывает, что триггер сработал при выполнении операции INSERT.

Примеры:

Пример 8.10. Использование переменной INSERTING

```
...
IF (INSERTING OR UPDATING) THEN
BEGIN
  IF (NEW.SERIAL_NUM IS NULL) THEN
    NEW.SERIAL_NUM = GEN_ID (GEN_SERIALS, 1);
END
...
```

См. также: UPDATING, DELETING.

NEW

Доступно в: PSQL.

Синтаксис:

```
NEW
```

Контекстная переменная NEW доступна только в коде табличных триггеров. Значение NEW содержит новые значения полей данных, которое возникли в базе во время операции обновления или вставки.

В AFTER триггерах переменная доступна только для чтения.

> **Примечание**
>
> Для табличных триггеров, срабатывающих на несколько типов событий, переменная NEW доступна всегда. Однако в случае если триггер сработал на операцию удаления, то для него новая версия данных не имеет смысла. В этой ситуации чтение переменной NEW всегда вернёт NULL.

> **Важно**
>
> Попытка записи в переменную NEW в AFTER триггере вызовет исключение в коде.

Примеры:

Пример 8.11. Использование переменной NEW

```
...
  IF (NEW.SERIAL_NUM IS NULL) THEN
    NEW.SERIAL_NUM = GEN_ID (GEN_SERIALS, 1);
```

```
...
```

См. также: OLD.

'NOW'

Доступно в: DSQL, PSQL, ESQL.

Синтаксис:

```
'NOW'
```

Не является переменной, а является строковым литералом. При использовании преобразования типов данных, например, с помощью функции CAST() в тип даты/времени позволяет получить текущую дату и/или время. Значение "после запятой" у переменной показывают число миллисекунд. Точность составляет 3 знака после запятой (миллисекунды). Написание 'NOW' зависит от регистра, при преобразовании в дату функция игнорирует все пробелы слева и справа от слова.

> **Примечание**
>
> Поскольку 'NOW' всегда возвращает актуальные значения даты и времени при использовании CAST() для приведения типов данных она может использоваться для измерения временных интервалов и скорости выполнения кода в процедурах, триггерах и блоках кода PSQL. Будьте внимательны при использовании 'NOW', т.к. использование сокращённого преобразования типов 'NOW' оценивается во время синтаксического анализа, а затем время её "замораживается", даже при многократном выполнении кода.

Примеры:

Пример 8.12. Использование переменной 'NOW'

```
SELECT CAST('Now' AS DATE) FROM rdb$database;
-- возвратит, например 2014-10-03

SELECT CAST('now' AS TIME) FROM rdb$database;
-- возвратит, например 14:20:19.6170

SELECT CAST('NOW' AS TIMESTAMP) FROM rdb$database;
-- возвратит, например 2014-10-03 14:20:19.6170
```

См. также: CURRENT_TIMESTAMP, CURRENT_DATE, CURRENT_TIME, TODAY, TOMORROW, YESTERDAY.

OLD

Доступно в: PSQL.

Синтаксис:

```
OLD
```

Контекстная переменная OLD доступна только коде триггеров. Значения, содержащееся в OLD, хранит прошлые значения полей, которые были в базе до операции изменения или удаления.

Переменная OLD доступна только для чтения.

> **Примечание**
>
> Для табличных триггеров, срабатывающих на несколько типов событий, значения для переменной OLD всегда возможны. Однако для триггеров, сработавших на вставку записи, значение данной переменной не имеет смысла, поэтому в этой ситуации чтение OLD возвратит NULL, а попытка записи в неё вызовет исключение в коде.

Примеры:

Пример 8.13. Использование переменной OLD

```
...
  IF (NEW.QUANTITY IS DISTINCT FROM OLD.QUANTITY) THEN
    DELTA = NEW.QUANTITY - OLD.QUANTITY;
...
```

См. также: NEW.

ROW_COUNT

Доступно в: PSQL.

Синтаксис:

```
ROW_COUNT
```

Тип возвращаемого результата: INTEGER

Контекстная переменная ROW_COUNT содержит число строк, затронутых последним оператором DML (INSERT, UPDATE, DELETE, SELECT или FETCH) в текущем триггере, хранимой процедуре или исполняемом блоке.

Поведение с SELECT и FETCH:

- После выполнения singleton SELECT запроса (запроса, который может вернуть не более одной строки данных), ROW_COUNT равна 1, если была получена строка данных и 0 в противном случае;
- В цикле FOR SELECT переменная ROW_COUNT увеличивается на каждой итерации (начиная с 0 в качестве первого значения);
- После выборки (FETCH) из курсора, ROW_COUNT равна 1, если была получена строка данных и 0 в противном случае. Выборка нескольких записей из одного курсора не увеличивает ROW_COUNT после 1.

Важно

Переменная ROW_COUNT не может быть использована для определения количества строк, затронутых при выполнении операторов EXECUTE STATEMENT или EXECUTE PROCEDURE. Для оператора MERGE переменная ROW_COUNT будет содержать 0 или 1, даже если было затронуто более записей

Примеры:

Пример 8.14. Использование переменной ROW_COUNT

```
...
UPDATE Figures SET Number = 0 WHERE id = :id;
IF (row_count = 0) THEN
  INSERT INTO Figures (id, Number)
  VALUES (:id, 0);
...
```

SQLCODE

Доступно в: PSQL.

Синтаксис:

```
SQLCODE
```

Тип возвращаемого результата: INTEGER

В блоках обработки ошибок "WHEN ... DO" контекстная переменная SQLCODE содержит текущий код ошибки SQL. До Firebird 2.0 значение SQLCODE можно было получить только в блоках обработки ошибок WHEN SQLCODE и WHEN ANY. Теперь она может быть отлична от нуля в блоках WHEN GDSCODE и WHEN EXCEPTION при условии, что ошибка, вызвавшее срабатывание блока, соответствует коду ошибки SQL. Вне обработчиков ошибок SQLCODE всегда равен 0, а вне PSQL не существует вообще.

Примеры:

Пример 8.15. Использование переменной SQLCODE

```
...
WHEN ANY DO
BEGIN
  IF (SQLCODE <> 0) THEN
    MSG = 'Обнаружена ошибка SQL!';
  ELSE
    MSG = 'Ошибки нет!';
  EXCEPTION EX_CUSTOM MSG;
END
...
```

См. также: GDSCODE, SQLSTATE.

SQLSTATE

Доступно в: PSQL.

Синтаксис:

```
SQLSTATE
```

Тип возвращаемого результата: CHAR(5)

В блоках обработки ошибок "WHEN ... DO" контекстная переменная SQLSTATE переменная содержит 5 символов SQL-2003 - совместимого кода состояния, переданного оператором, вызвавшим ошибку. Вне обработчиков ошибок SQLSTATE всегда равен '00000', а вне PSQL не существует вообще.

> **Примечание**
>
> - SQLSTATE предназначен для замены SQLCODE. Последняя, в настоящее время устарела и буден удалена будущих версиях Firebird;
> - Любой код SQLSTATE состоит из двух символов класса и трёх символов подкласса. Класс 00 (успешное выполнение), 01 (предупреждение) и 02 (нет данных) представляют собой условия завершения. Каждый код статуса вне этих классов является исключением. Поскольку классы 00, 01 и 02 не вызывают ошибку, они никогда не будут обнаруживаться в переменной SQLSTATE.

Пример 8.16. Использование переменной SQLSTATE

```
WHEN ANY DO
BEGIN
  MSG = CASE SQLSTATE
          WHEN '22003' THEN
            'Число вышло за пределы диапазона!'
          WHEN '22012' THEN
            'Деление на ноль!'
          WHEN '23000' THEN
            'Нарушение ограничения целостности!'
          ELSE 'Ошибок нет! SQLSTATE = ' || SQLSTATE;
        END;
  EXCEPTION EX_CUSTOM MSG;
END
```

См. также: GDSCODE, SQLCODE, Коды ошибок SQLSTATE и их описание.

'TODAY'

Доступно в: DSQL, PSQL, ESQL.

Синтаксис:

```
'TODAY'
```

Не является переменной, а является строковым литералом. При использовании преобразования типов данных, например, с помощью функции CAST() в тип даты/времени позволяет получить текущую дату. Написание 'TODAY' не зависит от регистра, при преобразовании в дату функция игнорирует все пробелы слева и справа от слова.

Примеры:

Пример 8.17. Использование переменной 'TODAY'

```
SELECT CAST('Today' AS DATE) FROM rdb$database;
 -- возвратит, например 2014-10-03
SELECT CAST('TODAY' AS TIMESTAMP) FROM rdb$database;
 -- возвратит, например 2014-10-03 00:00:00.0000

-- при использовании короткого, «C-Style» преобразования:
SELECT DATE 'Today' FROM rdb$database;
SELECT TIMESTAMP 'TODAY' FROM rdb$database;
```

См. также: CURRENT_DATE, TOMORROW, YESTERDAY.

'TOMORROW'

Доступно в: DSQL, PSQL, ESQL.

Синтаксис:

```
'TOMORROW'
```

Не является переменной, а является строковым литералом. При использовании преобразования, например, с помощью функции CAST() в тип даты / времени позволяет получить дату, следующую за текущей.

Примеры:

Пример 8.18. Использование переменной 'TOMORROW'

```
SELECT CAST('Tomorrow' AS DATE) FROM rdb$database;
-- возвратит, например 2014-10-04

SELECT CAST('TOMORROW' AS TIMESTAMP) FROM rdb$database;
-- возвратит, например 2014-10-04 00:00:00.0000

-- при использовании короткого, «C-Style» преобразования:
SELECT DATE 'Tomorrow' FROM rdb$database;
```

См. также: CURRENT_DATE, TODAY, YESTERDAY.

UPDATING

Доступно в: PSQL.

Синтаксис:

```
UPDATING
```

Тип возвращаемого результата: BOOLEAN

Контекстная переменная UPDATING доступна только коде табличных триггеров. Используется в триггерах на несколько типов событий и показывает, что триггер сработал при выполнении операции UPDATE.

Примеры:

Пример 8.19. Использование переменной UPDATING

```
...
IF (INSERTING OR UPDATING) THEN
BEGIN
  IF (NEW.SERIAL_NUM IS NULL) THEN
    NEW.SERIAL_NUM = GEN_ID (GEN_SERIALS, 1);
END
...
```

См. также: INSERTING, DELETING.

'YERSTERDAY'

Доступно в: DSQL, PSQL, ESQL.

Синтаксис:

```
'YERSTERDAY'
```

Не является переменной, а является строковым литералом. При использовании преобразования, например, с помощью функции CAST() в тип даты / времени позволяет получить дату, предыдущую перед текущей.

Примеры:

Пример 8.20. Использование переменной 'YERSTERDAY'

```
SELECT CAST('Yesterday' AS DATE) FROM rdb$database;
--возвратит, например 2014-10-04

-- при использовании короткого, «C-Style» преобразования
SELECT DATE 'Yesterday' FROM rdb$database;
SELECT TIMESTAMP 'YESTERDAY' FROM rdb$database;
```

См. также: CURRENT_DATE, TODAY, TOMORROW.

USER

Доступно в: DSQL, PSQL.

Синтаксис:

```
USER
```

Тип возвращаемого результата: VARCHAR(31)

Переменная USER содержит имя текущего подключенного пользователя базы данных.

Примеры:

Пример 8.21. Использование переменной USER

```
NEW.ADDED_BY = USER;
```

См. также: CURRENT_USER, CURRENT_ROLE.

Скалярные функции

> **Примечание**
>
> Если в вашей базе данных имя декларированной внешней функции (DECLARE EXTERNAL FUNCTION) совпадает с именем встроенной, то будет вызываться внешняя функция. Для того чтобы была доступна внутренняя функция, необходимо удалить или изменить имя внешней функции.

Функции для работы с контекстными переменными

RDB$GET_CONTEXT

Доступно в: DSQL, PSQL.

Синтаксис:

```
RDB$GET_CONTEXT('<namespace>', 'varname')

<namespace> ::= SYSTEM | DDL_TRIGGER | USER_SESSION | USER_TRANSACTION
```

Параметры функции RDB$GET_CONTEXT

namespace
Пространство имён.

varname
Имя переменной. Зависит от регистра. Максимальная длина 80 символов.

Тип возвращаемого результата: VARCHAR(255)

Функция RDB$GET_CONTEXT возвращает значение контекстной переменной из одного из пространства имён.

В настоящий момент существуют следующие пространства имён:

- SYSTEM — предоставляет доступ к системным контекстным переменным. Эти переменные доступны только для чтения;
- USER_SESSION — предоставляет доступ к пользовательским контекстным переменным, заданным через функцию RDB$SET_CONTEXT. Переменные существуют в течение подключения;
- USER_TRANSACTION — предоставляет доступ к пользовательским контекстным переменным, заданным через функцию RDB$SET_CONTEXT. Переменные существуют в течение транзакции;
- DDL_TRIGGER — предоставляет доступ к системным контекстным переменным, доступным только во время выполнения DDL триггера. Эти переменные доступны только для чтения.

Пространства имён USER_SESSION и USER_TRANSACTION — изначально пусты и пользователь сам создаёт переменные и наполняет их при помощи функции RDB $SET_CONTEXT.

> **Примечание**
>
> Для предотвращения DoS атак, существует ограничение на 1000 переменных в одном "пространстве имён".

Если запрашиваемая функцией переменная существует в указанном пространстве имён, то будет возвращено её значение в виде строки VARCHAR(255). При обращении к несуществующей переменной в пространстве SYSTEM возникает ошибка, если такое происходит с пространствами имён USER_SESSION или USER_TRANSACTION – функция возвращает NULL.

Использование пространства имён DDL_TRIGGER допустимо, только во время работы DDL триггера. Его использование также допустимо в хранимых процедурах и функциях, вызванных триггерами DDL.

Контекст DDL_TRIGGER работает как стек. Перед возбуждением DDL триггера, значения, относящиеся к выполняемой команде, помещаются в этот стек. После завершения работы триггера значения выталкиваются. Таким образом. В случае каскадных DDL операторов, когда

каждая пользовательская DDL команда возбуждает DDL триггер, и этот триггер запускает другие DDL команды, с помощью EXECUTE STATEMENT, значения переменных в пространстве имён DDL_TRIGGER будут соответствовать команде, которая вызвала последний DDL триггер в стеке вызовов.

Таблица 8.1. Переменные пространства имён SYSTEM

Переменная	Описание
CLIENT_ADDRESS	Для TCPv4 – IP адрес, для XNET – локальный ID процесса. Для остальных случаев NULL.
CLIENT_PID	PID процесса на клиентском компьютере.
CLIENT_PROCESS	Полный путь к клиентскому приложению, подключившемуся к базе данных. Позволяет не использовать системную таблицу MON$ATTACHMENTS (поле MON$REMOTE_PROCESS).
CURRENT_ROLE	Глобальная переменная CURRENT_ROLE.
CURRENT_USER	Глобальная переменная CURRENT_USER.
DB_NAME	Полный путь к базе данных или алиас к базе данных, из строки подключения к базе данных.
ENGINE_VERSION	Версия сервера Firebird.
ISOLATION_LEVEL	Уровень изоляции текущей транзакции — CURRENT_TRANSACTION. Значения: "READ_COMMITED", "SNAPSHOT" или "CONSISTENCY".
LOCK_TIMEOUT	Время ожидания транзакцией высвобождения ресурса при блокировке, в секундах.
NETWORK_PROTOCOL	Протокол, используемый для соединения с базой данных. Возможные значения: "TCPv4", "TCPv6", "WNET", "XNET", NULL.
READ_ONLY	Отображает, является ли транзакция, транзакцией только для чтения. FALSE для R/W транзакций TRUE для ReadOnly.
SESSION_ID	Глобальная переменная CURRENT_CONNECTION.
TRANSACTION_ID	Глобальная переменная CURRENT_TRANSACTION.

Таблица 8.2. Переменные пространства имён DDL_TRIGGER

Переменная	Описание
EVENT_TYPE	Тип события (CREATE, ALTER, DROP).
OBJECT_TYPE	Тип объекта (TABLE, VIEW и др.).
DDL_EVENT	Имя события. DDL_EVENT = EVENT_TYPE || ' ' || OBJECT_TYPE
OBJECT_NAME	Имя объекта метаданных.
OLD_OBJECT_NAME	Имя объекта метаданных до переименования.

Переменная	Описание
NEW_OBJECT_NAME	Имя объекта метаданных после переименования.
SQL_TEXT	Текст SQL запроса.

Примечание

Ещё раз обратите внимание на то, что пространства имён и имена переменных регистрочувствительны, должны быть непустыми строками, и заключены в кавычки!

Примеры:

Пример 8.22. Использование функции RDB$GET_CONTEXT

```
NEW.USER_ADR = RDB$GET_CONTEXT ('SYSTEM', 'CLIENT_ADDRESS');
```

См. также: RDB$SET_CONTEXT.

RDB$SET_CONTEXT

Доступно в: DSQL, PSQL.

Синтаксис:

```
RDB$SET_CONTEXT('<namespace>', 'varname', {<value> | NULL})

<namespace> ::= USER_SESSION | USER_TRANSACTION
```

Параметры функции RDB$SET_CONTEXT

namespace
Пространство имён.

varname
Имя переменной. Зависит от регистра. Максимальная длина 80 символов.

value
Данные любого типа при условии, что их можно привести к типу VARCHAR(255).

Тип возвращаемого результата: INTEGER

Функция RDB$SET_CONTEXT создаёт, устанавливает значение или обнуляет переменную в одном из используемых пользователем пространстве имён: USER_SESSION или USER_TRANSACTION.

Функция возвращает 1, если переменная уже существовала до вызова и 0, если не существовала. Для удаления переменной надо установить её значение в NULL. Если данное пространство имён не существует, то функция вернёт ошибку. Пространство имён и имя переменной зависят от регистра, должны быть не пустыми строками, и заключены в кавычки.

Примечание

- Пространство имён SYSTEM доступно только для чтения;
- Максимальное число переменных в рамках одного соединения (для пространства USER_SESSION) или одной транзакции (для пространства USER_TRANSACTION) равно 1000;
- Все переменные в пространстве имён USER_TRANSACTION сохраняются при ROLLBACK RETAIN или ROLLBACK TO SAVEPOINT, независимо от того, в какой точке во время выполнения транзакции они были установлены.

Примеры:

Пример 8.23. Использование функции RDB$SET_CONTEXT

```
SELECT RDB$SET_CONTEXT ('USER_SESSION', 'DEBUGL', 3)
FROM RDB$DATABASE;

-- в PSQL доступен такой синтаксис
RDB$SET_CONTEXT('USER_SESSION', 'RECORDSFOUND', RECCOUNTER);

SELECT RDB$SET_CONTEXT ('USER_TRANSACTION', 'SAVEPOINTS', 'YES')
FROM RDB$DATABASE;
```

Пример 8.24. Использование функций для работы с контекстными переменными

```
SET TERM ^;
CREATE PROCEDURE set_context(User_ID VARCHAR(40),
                             Trn_ID INT) AS
BEGIN
  RDB$SET_CONTEXT('USER_TRANSACTION', 'Trn_ID', Trn_ID);
  RDB$SET_CONTEXT('USER_TRANSACTION', 'User_ID', User_ID);
END^
SET TERM ;^

CREATE TABLE journal (
   jrn_id INTEGER NOT NULL PRIMARY KEY,
   jrn_lastuser VARCHAR(40),
   jrn_lastaddr VARCHAR(255),
   jrn_lasttran INTEGER
);

SET TERM ^;
CREATE TRIGGER UI_JOURNAL
FOR JOURNAL BEFORE INSERT OR UPDATE
AS
BEGIN
  new.jrn_lastuser = RDB$GET_CONTEXT('USER_TRANSACTION',
                                     'User_ID');
  new.jrn_lastaddr = RDB$GET_CONTEXT('SYSTEM',
                                     'CLIENT_ADDRESS');
  new.jrn_lasttran = RDB$GET_CONTEXT('USER_TRANSACTION',
                                        'Trn_ID');
END^
```

```
SET TERM ;^

EXECUTE PROCEDURE set_context('skidder', 1);

INSERT INTO journal(jrn_id) VALUES(0);

COMMIT;
```

См. также: RDB$GET_CONTEXT.

Математические функции

ABS

Доступно в: DSQL, PSQL.

Синтаксис:

```
ABS (value)
```

Параметры функции ABS

value
Выражение числового типа.

Тип возвращаемого результата: тот же что и входной аргумент.

Функция ABS возвращает абсолютное значение (модуль) аргумента.

ACOS

Доступно в: DSQL, PSQL.

Синтаксис:

```
ACOS (value)
```

Параметры функции ACOS

value
Выражение числового типа в диапазоне [-1; 1].

Тип возвращаемого результата: DOUBLE PRECISION.

Функция ACOS возвращает арккосинус (в радианах) аргумента.

В случае если аргумент функции вне границы диапазона [-1, 1], то функция вернёт неопределённое значения NaN.

См. также: COS.

ACOSH

Доступно в: DSQL, PSQL.

Синтаксис:

```
ACOSH (value)
```

Параметры функции ACOSH

value
Выражение числового типа в диапазоне [1; +∞].

Тип возвращаемого результата: DOUBLE PRECISION.

Функция ACOSH возвращает гиперболический арккосинус (в радианах) аргумента.

См. также: COSH.

ASIN

Доступно в: DSQL, PSQL.

Синтаксис:

```
ASIN (value)
```

Параметры функции ASIN

value
Выражение числового типа в диапазоне [-1; 1].

Тип возвращаемого результата: DOUBLE PRECISION.

Функция ASIN возвращает арксинус (в радианах) аргумента.

В случае если аргумент функции вне границы диапазона [-1, 1], то функция вернёт неопределённое значения NaN.

См. также: SIN.

ASINH

Доступно в: DSQL, PSQL.

Синтаксис:

```
ASINH (value)
```

Параметры функции ASINH

value
Выражение числового типа.

Тип возвращаемого результата: DOUBLE PRECISION.

Функция ASINH возвращает гиперболический арксинус (в радианах) аргумента.

См. также: ASINH.

ATAN

Доступно в: DSQL, PSQL.

Синтаксис:

```
ATAN (value)
```

Параметры функции ATAN

value
Выражение числового типа.

Тип возвращаемого результата: DOUBLE PRECISION.

Функция ATAN возвращает арктангенс аргумента.

Функция возвращает угол в радианах в диапазоне [-π/2; π/2].

См. также: ATAN2, TAN.

ATAN2

Доступно в: DSQL, PSQL.

Синтаксис:

```
ATAN2 (y, x)
```

Параметры функции ATAN2

x
Выражение числового типа.

y
Выражение числового типа.

Тип возвращаемого результата: DOUBLE PRECISION.

Функция ATAN2 возвращает угол как отношение синуса к косинусу, аргументы, у которых задаются этими двумя параметрами, а знаки синуса и косинуса соответствуют знакам параметров. Это позволяет получать результаты по всей окружности, включая углы -π/2 и π/2.

Особенности использования:

- Результат — угол в диапазоне [-π, π] радиан;
- Если *x* отрицательный, то при нулевом значении y результат равен π, а при значении 0 равен -π;
- Если и y и x равны 0, то результат бессмыслен.

> **Примечание**
>
> - Полностью эквивалентное описание этой функции следующее: ATAN2 (y, x) является углом между положительной осью X и линией от начала координат до точки (x, y). Это также делает очевидным, что значение ATAN2 (0, 0) не определено;
> - Если *x* больше, чем 0, ATAN2 (y, x) совпадает с ATAN (y/x);
> - Если известны и синус, и косинус угла, то ATAN2 (SIN, COS) возвращает угол.

См. также: ATAN, SIN, COS.

ATANH

Доступно в: DSQL, PSQL.

Синтаксис:

```
ATANH (value)
```

Параметры функции ATANH

value
Выражение числового типа.

Тип возвращаемого результата: DOUBLE PRECISION.

Функция ATANH возвращает гиперболический арктангенс (в радианах) аргумента.

См. также: TANH.

CEIL, CEILING

Доступно в: DSQL, PSQL.

Синтаксис:

```
CEIL[ING] (number)
```

Параметры функции CEIL[ING]

number
Выражение числового типа.

Тип возвращаемого результата: BIGINT или DOUBLE PRECISION.

Функция CEIL возвращает наименьшее целое число, большее или равное аргументу.

См. также: FLOOR, TRUNC.

COS

Доступно в: DSQL, PSQL.

Синтаксис:

```
COS (angle)
```

Параметры функции COS

angle
Угол, выраженный в радианах.

Тип возвращаемого результата: DOUBLE PRECISION.

Функция COS возвращает косинус угла. Аргумент должен быть задан в радианах.

Любой NOT NULL результат находится в диапазоне [-1, 1].

См. также: ACOS.

COSH

Доступно в: DSQL, PSQL.

Синтаксис:

```
COSH (number)
```

Параметры функции COSH

number
Выражение числового типа.

Тип возвращаемого результата: DOUBLE PRECISION.

Функция COSH возвращает гиперболический косинус аргумента.

Любой NOT NULL результат находится в диапазоне [1, +∞].

См. также: ACOSH.

COT

Доступно в: DSQL, PSQL.

Синтаксис:

```
COT (angle)
```

Параметры функции COT

angle
Угол, выраженный в радианах.

Тип возвращаемого результата: DOUBLE PRECISION.

Функция COT возвращает котангенс угла. Аргумент должен быть задан в радианах.

См. также: TAN.

EXP

Доступно в: DSQL, PSQL.

Синтаксис:

```
EXP (number)
```

Параметры функции EXP

number
Выражение числового типа.

Тип возвращаемого результата: DOUBLE PRECISION.

Функция EXP возвращает значение натуральной экспоненты, e^{number}.

См. также: LN.

FLOOR

Доступно в: DSQL, PSQL.

Синтаксис:

```
FLOOR (number)
```

Параметры функции FLOOR

number
Выражение числового типа.

Тип возвращаемого результата: BIGINT или DOUBLE PRECISION.

Функция FLOOR возвращает целое число, меньшее или равное аргументу.

См. также: CEIL, CEILING, TRUNC.

LN

Доступно в: DSQL, PSQL.

Синтаксис:

```
LN (number)
```

Параметры функции LN

number
Выражение числового типа.

Тип возвращаемого результата: DOUBLE PRECISION.

Функция LN возвращает натуральный логарифм аргумента.

> **Примечание**
>
> В случае если передан отрицательный или нулевой аргумент функция вернёт ошибку.

См. также: EXP.

LOG

Доступно в: DSQL, PSQL.

Синтаксис:

```
LOG (x, y)
```

Параметры функции LOG

x
Основание. Выражение числового типа.

y
Выражение числового типа.

Тип возвращаемого результата: DOUBLE PRECISION.

Функция LOG возвращает логарифм *y* (второй аргумент) по основанию *x* (первый аргумент).

Особенности использования:

- Если один из аргументов меньше или равен 0, то возникает ошибка;

- Если оба аргумента равны 1, то результатом функции будет NaN (Not-a-Number — не число);
- Если $x = 1$ и $y < 1$, то результатом функции будет -INF (-∞);
- Если $x = 1$ и $y > 1$, то результатом функции будет +INF (+∞).

LOG10

Доступно в: DSQL, PSQL.

Синтаксис:

```
LOG10 (number)
```

Параметры функции LOG10

number
Выражение числового типа.

Тип возвращаемого результата: DOUBLE PRECISION.

Функция LOG10 возвращает десятичный логарифм аргумента.

> **Примечание**
>
> Если входной аргумент отрицательный или равен 0, возникает ошибка.

MOD

Доступно в: DSQL, PSQL.

Синтаксис:

```
MOD (a, b)
```

Параметры функции MOD

a
Выражение числового типа.

b
Выражение числового типа.

Тип возвращаемого результата: INTEGER или BIGINT.

Функция MOD возвращает остаток от целочисленного деления.

> **Примечание**
>
> Вещественные числа округляются до выполнения деления. Например, результатом 7.5 mod 2.5 будет 2 (8 mod 3), а не 0.

PI

Доступно в: DSQL, PSQL.

Синтаксис:

```
PI ()
```

Тип возвращаемого результата: DOUBLE PRECISION.

Функция PI возвращает число π.

POWER

Доступно в: DSQL, PSQL.

Синтаксис:

```
POWER (x, y)
```

Параметры функции POWER

x
: Выражение числового типа.

y
: Выражение числового типа.

Тип возвращаемого результата: DOUBLE PRECISION.

Функция POWER возвращает результат возведения числа *x* в степень *y*.

> **Примечание**
>
> Если *x* меньше нуля, возникает ошибка.

RAND

Доступно в: DSQL, PSQL.

Синтаксис:

```
RAND ()
```

Тип возвращаемого результата: DOUBLE PRECISION.

Функция RAND возвращает псевдослучайное число в интервале от 0 до 1.

ROUND

Доступно в: DSQL, PSQL.

Синтаксис:

```
ROUND (number [, scale])
```

Параметры функции ROUND

number
Выражение числового типа.

scale
Масштаб - целое число, определяющее число десятичных разрядов, к которым должен быть проведено округление, т.е.

```
2 для округления к самому близкому кратному 0.01 числу
1 для округления к самому близкому кратному 0.1 числу
0 для округления к самому близкому целому числу
-1 для округления к самому близкому кратному 10 числу
-2 для округления к самому близкому кратному 100 числу
```

По умолчанию 0.

Тип возвращаемого результата: INTEGER, масштабируемый BIGINT, DOUBLE PRECISION.

Функция ROUND округляет число до ближайшего целого числа. Если дробная часть равна 0.5, то округление до ближайшего большего целого числа для положительных чисел и до ближайшего меньшего для отрицательных чисел. С дополнительным опциональным параметром *scale* число может быть округлено до одной из степеней числа 10 (десятки, сотни, десятые части, сотые части и т.д.) вместо просто целого числа.

> **Примечание**
>
> Если используется параметр *scale*, то результат имеет такой же масштаб, как и первый параметр *number*.

Примеры:

Пример 8.25. Использование функции ROUND

```
ROUND(123.654, 1) -- Результат: 123.700 (а не 123.7)
ROUND(8341.7, -3) -- Результат: 8000.0 (а не 8000)
ROUND(45.1212, 0) -- Результат: 45.0000 (а не 45)
ROUND(45.1212)    -- Результат: 45
```

См. также: TRUNC.

SIGN

Доступно в: DSQL, PSQL.

Синтаксис:

```
SIGN (number)
```

Параметры функции SIGN

number
Выражение числового типа.

Тип возвращаемого результата: SMALLINT.

Функция SIGN возвращает знак входного параметра.

Таблица 8.3. Таблица результатов функции SIGN

Результат	Значение
-1	число меньше нуля
0	число равно нулю
1	число больше нуля

SIN

Доступно в: DSQL, PSQL.

Синтаксис:

```
SIN (angle)
```

Параметры функции SIN

angle
Угол, выраженный в радианах.

Тип возвращаемого результата: DOUBLE PRECISION.

Функция SIN возвращает синус угла. Аргумент должен быть задан в радианах.

Любой NOT NULL результат находится в диапазоне [-1, 1].

См. также: ASIN.

SINH

Доступно в: DSQL, PSQL.

Синтаксис:

```
SINH (number)
```

Параметры функции SINH

number
Выражение числового типа.

Тип возвращаемого результата: DOUBLE PRECISION.

Функция SINH возвращает гиперболический синус аргумента.

См. также: ASINH.

SQRT

Доступно в: DSQL, PSQL.

Синтаксис:

```
SQRT (number)
```

Параметры функции SQRT

number
Выражение числового типа.

Тип возвращаемого результата: DOUBLE PRECISION.

Функция SQRT возвращает квадратный корень аргумента.

TAN

Доступно в: DSQL, PSQL.

Синтаксис:

```
TAN (angle)
```

Параметры функции TAN

angle
Угол, выраженный в радианах.

Тип возвращаемого результата: DOUBLE PRECISION.

Функция TAN возвращает тангенс угла. Аргумент должен быть задан в радианах.

См. также: ATAN, ATAN2.

TANH

Доступно в: DSQL, PSQL.

Синтаксис:

```
TANH (number)
```

Параметры функции TANH

number
Выражение числового типа.

Тип возвращаемого результата: DOUBLE PRECISION.

Функция TANH возвращает гиперболический тангенс аргумента.

Любой NOT NULL результат находится в диапазоне [-1, 1].

См. также: ATANH.

TRUNC

Доступно в: DSQL, PSQL.

Синтаксис:

```
TRUNC (number [, scale])
```

Параметры функции TRUNC

number
Выражение числового типа.

scale
Масштаб - целое число, определяющее число десятичных разрядов, к которым должен быть проведено усечение, т.е.

```
2 для усечения к самому близкому кратному 0.01 числу
1 для усечения к самому близкому кратному 0.1 числу
0 для усечения к самому близкому целому числу
-1 для усечения к самому близкому кратному 10 числу
-2 для усечения к самому близкому кратному 100 числу
```

По умолчанию 0.

Тип возвращаемого результата: INTEGER, масштабируемый BIGINT, DOUBLE PRECISION.

Функция TRUNC усекает число до ближайшего целого числа. С дополнительным опциональным параметром *scale* число может быть усечено до одной из степеней числа 10 (десятки, сотни, десятые части, сотые части и т.д.) вместо просто целого числа.

Примечание

Если используется параметр *scale*, то результат имеет такой же масштаб, как и первый параметр *number*.

Важно

Функция всегда увеличивает отрицательные числа, поскольку она обрезает дробную часть.

Примеры:

Пример 8.26. Использование функции TRUNC

```
TRUNC(789.2225, 2)  -- Результат: 789.2200 (а не 789.22)
TRUNC(345.4, -2)  -- Результат: 300.0 (а не 300)
TRUNC(-163.41, 0) -- Результат: -163.00 (а не -163)
TRUNC(-163.41)      -- Результат: -163
```

См. также: ROUND, CEIL, CEILING, FLOOR.

Функции для работы со строками

ASCII_CHAR

Доступно в: DSQL, PSQL.

Синтаксис:

```
ASCII_CHAR (code)
```

Параметры функции ASCII_CHAR

code
Целое число в диапазоне от 0 до 255.

Тип возвращаемого результата: [VAR]CHAR(1) CHARSET NONE.

Функция ASCII_CHAR возвращает ASCII символ соответствующий номеру, переданному в качестве аргумента.

См. также: ASCII_VAL.

ASCII_VAL

Доступно в: DSQL, PSQL.

Синтаксис:

```
ASCII_VAL (ch)
```

Параметры функции ASCII_VAL

ch
Строка типа данных [VAR]CHAR или текстовый BLOB максимального размера 32767 байт.

Тип возвращаемого результата: SMALLINT.

Функция ASCII_VAL возвращает ASCII код символа, переданного в качестве аргумента.

Особенности использования:

- Если строка содержит более одного символа, то возвращается код первого символа строки;
- Если строка пустая, возвращается ноль;
- Если аргумент NULL, то возвращаемое значение также NULL.

См. также: ASCII_CHAR.

BIT_LENGTH

Доступно в: DSQL, PSQL.

Синтаксис:

```
BIT_LENGTH (str)
```

Параметры функции BIT_LENGTH

str
Выражение строкового типа.

Тип возвращаемого результата: BIGINT.

Функция BIT_LENGTH возвращает длину входной строки в битах. Для многобайтных наборов символов результат может быть в 8 раз больше, чем количество символов в "формальном" числе байт на символ, записанном в RDB$CHARACTER_SETS.

С параметрами типа CHAR эта функция берет во внимание всю формальную строковую длину (например, объявленная длина поля или переменной). Если вы хотите получить "логическую" длину в битах, не считая пробелов, то перед передачей аргумента в BIT_LENGTH надо выполнить над ним операцию RIGHT TRIM.

Примеры:

Пример 8.27. Использование функции BIT_LENGTH

```
SELECT BIT_LENGTH ('Hello!') FROM RDB$DATABASE
-- возвращает 48

SELECT BIT_LENGTH (_ISO8859_1 'Grüß Di!')
FROM RDB$DATABASE
```

```
-- возвращает 64: каждый, и ü, и ß занимают один байт в ISO8859_1

SELECT BIT_LENGTH (
CAST (_ISO8859_1 'Grüß di!' AS VARCHAR (24)
CHARACTER SET UTF8))
FROM RDB$DATABASE
-- возвращает 80: каждый, и ü, и ß занимают по два байта в UTF8

SELECT BIT_LENGTH (
CAST (_ISO8859_1 'Grüß di!' AS CHAR (24)
CHARACTER SET UTF8))
FROM RDB$DATABASE
-- возвращает 208: размер всех 24 позиций CHAR и два из них 16-битные
```

См. также: CHAR_LENGTH, OCTET_LENGTH.

CHAR_LENGTH, CHARACTER_LENGTH

Доступно в: DSQL, PSQL.

Синтаксис:

```
CHAR_LENGTH (str)

CHARACTER_LENGTH (str)
```

Параметры функции CHAR_LENGTH

str
Выражение строкового типа.

Тип возвращаемого результата: BIGINT.

Функция CHAR_LENGTH возвращает длину (в символах) строки, переданной в качестве аргумента.

> **Примечание**
>
> С параметрами типа CHAR эта функция берет во внимание всю формальную строковую длину (например, объявленная длина поля или переменной). Если вы хотите получить "логическую" длину без учёта пробелов, то перед передачей аргумента в CHAR[ACTER]_LENGTH надо выполнить над ним операцию RIGHT TRIM.

См. также: BIT_LENGTH, OCTET_LENGTH.

HASH

Доступно в: DSQL, PSQL.

Синтаксис:

```
HASH (str)
```

Параметры функции HASH

str
Выражение строкового типа.

Тип возвращаемого результата: BIGINT.

Функция HASH возвращает хэш-значение входной строки. Эта функция полностью поддерживает текстовые BLOB любой длины и с любым набором символов.

LEFT

Доступно в: DSQL, PSQL.

Синтаксис:

```
LEFT (str, num)
```

Параметры функции LEFT

str
Выражение строкового типа.

num
Целое число. Определяет количество возвращаемых символов.

Тип возвращаемого результата: VARCHAR(N) или BLOB.

Функция LEFT возвращает левую часть строки, количество возвращаемых символов определяется вторым параметром.

Особенности использования:

- Функция поддерживает текстовые блоки любой длины и с любыми наборами символов;
- Если строковый аргумент BLOB, результатом будет BLOB, в противном случае результатом будет VARCHAR(N), при этом N – будет равно длине строкового параметра;
- Если числовой параметр превысит длину текста, результатом будет исходный текст.

> **Предупреждение**
>
> При использовании BLOB в параметрах функции может потребоваться загрузить объект полностью в память. При больших объёмах BLOB могут наблюдаться потери производительности.

Примеры:

Пример 8.28. Использование функции LEFT

```
SELECT LEFT('ABC', 2) FROM rdb$database;
-- результат AB
```

См. также: RIGHT, SUBSTRING.

LOWER

Доступно в: DSQL, PSQL, ESQL.

Синтаксис:

```
LOWER (str)
```

Параметры функции LOWER

str
Выражение строкового типа.

Тип возвращаемого результата: VARCHAR(N) или BLOB.

Функция LOWER возвращает входную строку в нижнем регистре. Точный результат зависит от набора символов входной строки. Например, для наборов символов NONE и ASCII только ASCII символы переводятся в нижний регистр; для OCTETS — вся входная строка возвращается без изменений.

Примеры:

Пример 8.29. Использование функции LOWER

```
/* Результат: 'debacle', в соответствии с французскими
 * правилами приведения в нижний регистр
 */
SELECT LOWER(_ISO8859_1 'Débâcle' COLLATE FR_FR)
FROM RDB$DATABASE
```

См. также: UPPER.

LPAD

Доступно в: DSQL, PSQL.

Синтаксис:

```
LPAD (str, endlen [, padstr])
```

Параметры функции LPAD

str
Выражение строкового типа.

endlen
Длина выходной строки.

padstr
: Строка, которой дополняется исходная строка до указанной длины. По умолчанию является пробелом (' ').

Тип возвращаемого результата: VARCHAR(*endlen*) или BLOB.

Функция LPAD дополняет слева входную строку пробелами или определённой пользователем строкой до заданной длины.

Особенности использования:

- Функция поддерживает текстовые блоки любой длины и с любыми наборами символов;
- Если входная строка имеет тип BLOB, то результат также будет BLOB, в противном случае результат будет VARCHAR(*endlen*).
- Если аргумент *padstr* задан, но равен пустой строке (''), то дополнения строки не происходит! В случае если *endlen* меньше длины входной строки, то в результате происходит её усечение до длины *endlen*, даже если параметр *padstr* равен пустой строке.

> **Предупреждение**
>
> При использовании BLOB в параметрах функции может потребоваться загрузить объект полностью в память. При больших объёмах BLOB могут наблюдаться потери производительности.

Примеры:

Пример 8.30. Использование функции LPAD

```
LPAD ('Hello', 12)                -- возвращает '       Hello'
LPAD ('Hello', 12, '-')           -- возвращает '-------Hello'
LPAD ('Hello', 12, '')            -- возвращает 'Hello'
LPAD ('Hello', 12, 'abc')         -- возвращает 'abcabcaHello'
LPAD ('Hello', 12, 'abcdefghij')  -- возвращает 'abcdefgHello'
LPAD ('Hello', 2)                 -- возвращает 'He'
LPAD ('Hello', 2, '-')            -- возвращает 'He'
LPAD ('Hello', 2, '')             -- возвращает 'He'
```

См. также: RPAD.

OCTET_LENGTH

Доступно в: DSQL, PSQL.

Синтаксис:

```
OCTET_LENGTH (str)
```

Параметры функции OCTET_LENGTH

str
: Выражение строкового типа.

Тип возвращаемого результата: BIGINT.

Функция OCTET_LENGTH возвращает количество байт занимаемое строкой.

При работе с параметрами типа CHAR функция возвращает значение всей формальной строковой длины. Для того чтобы узнать "логическую" длину строки в байтах, то перед передачей аргумента функции следует применить RIGHT TRIM.

> **Примечание**
>
> Следует помнить, что не во всех наборах символов количество байт занимаемых строкой равно количеству символов.

Примеры:

Пример 8.31. Использование функции OCTET_LENGTH

```
SELECT OCTET_LENGTH('Hello!')
FROM rdb$database
-- возвратит 6

SELECT OCTET_LENGTH(_iso8859_1 'Grüß di!')
FROM rdb$database
-- возвратит 8: ü и ß занимают не более 1 байта в ISO8859_1

SELECT
  OCTET_LENGTH(CAST(_iso8859_1 'Grüß di!' AS VARCHAR(24) CHARACTER SET utf8))
FROM rdb$database
-- возвратит 10: ü и ß занимают 2 байта в UTF8

SELECT
  OCTET_LENGTH(CAST(_iso8859_1 'Grüß di!' AS CHAR(24) CHARACTER SET utf8))
FROM rdb$database
-- возвратит 26: всего 24 CHAR позиции, и две из них занимают 2 байта
```

См. также: BIT_LENGTH, CHAR_LENGTH.

OVERLAY

Доступно в: DSQL, PSQL.

Синтаксис:

```
OVERLAY (string PLACING replacement FROM pos [FOR length])
```

Параметры функции OVERLAY

string
Строка, в которой происходит замена.

replacement
Строка, которой заменяется.

pos
Позиция, с которой происходит замена.

length
Количество символов, которые будут удалены из исходной строки.

Тип возвращаемого результата: VARCHAR(N) или BLOB.

Функция OVERLAY предназначена для замены части строки другой строкой.

По умолчанию число удаляемых из строки символов равняется длине заменяемой строки. Дополнительный четвёртый параметр позволяет пользователю задать своё число символов, которые будут удалены.

Особенности использования:

- Функция полностью поддерживает тестовые BLOB с любым набором символов и любой длины;
- Если входная строка имеет тип BLOB, то и результат будет иметь тип BLOB. В противном случае тип результата будет VARCHAR(n), где n является суммой длин параметров *string* и *replacement*;
- Как и во всех строковых функциях SQL параметр *pos* является определяющим;
- Если *pos* больше длины строки, то *replacement* помещается сразу после окончания строки;
- Если число символов от *pos* до конца строки меньше, чем длина *replacement* (или, чем параметр *length*, если он задан), то строка усекается до значения pos и *replacement* помещается после него;
- При нулевом параметре *length* (FOR 0) *replacement* просто вставляется в строку, начиная с позиции *pos*;
- Если любой из параметров имеет значение NULL, то и результат будет NULL;
- Если параметры *pos* и *length* не являются целым числом, то используется банковское округление (до чётного): 0.5 становится 0, 1.5 становится 2, 2.5 становится 2, 3.5 становится 4 и т.д.

> **Предупреждение**
>
> При использовании BLOB функции может потребоваться загрузить весь объект в память. При больших размерах BLOB это может повлиять на производительность.

Примеры:

Пример 8.32. Использование функции OVERLAY

```
OVERLAY ('Goodbye' PLACING 'Hello' FROM 2) -- Результат: 'Ghelloe'
OVERLAY ('Goodbye' PLACING 'Hello' FROM 5) -- Результат: 'GoodHello'
OVERLAY ('Goodbye' PLACING 'Hello' FROM 8) -- Результат: 'GoodbyeHello'
OVERLAY ('Goodbye' PLACING 'Hello' FROM 20) -- Результат: 'GoodbyeHello'
OVERLAY ('Goodbye' PLACING 'Hello' FROM 2 FOR 0) -- Результат: 'GHellooodbye'
OVERLAY ('Goodbye' PLACING 'Hello' FROM 2 FOR 3) -- Результат: 'GHellobye'
OVERLAY ('Goodbye' PLACING 'Hello' FROM 2 FOR 6) -- Результат: 'GHello'
OVERLAY ('Goodbye' PLACING 'Hello' FROM 2 FOR 9) -- Результат: 'Ghello'
OVERLAY ('Goodbye' PLACING '' FROM 4) -- Результат: 'Goodbye'
OVERLAY ('Goodbye' PLACING '' FROM 4 FOR 3) -- Результат: 'Gooe'
OVERLAY ('Goodbye' PLACING '' FROM 4 FOR 20) -- Результат: 'Goo'
OVERLAY ('' PLACING 'Hello' FROM 4) -- Результат: 'Hello'
```

```
OVERLAY ('' PLACING 'Hello' FROM 4 FOR 0) -- Результат: 'Hello'
OVERLAY ('' PLACING 'Hello' FROM 4 FOR 20) -- Результат: 'Hello'
```

См. также: SUBSTRING, REPLACE.

POSITION

Доступно в: DSQL, PSQL.

Синтаксис:

```
POSITION (<args>)

<args> ::=
    substr IN string
  | substr, string [, startpos]
```

Параметры функции POSITION

substr
Подстрока, позиция которой ищется.

string
Строка, в которой ищется позиция.

startpos
Позиция, с которой начинается поиск подстроки.

Тип возвращаемого результата: INTEGER.

Функция POSITION возвращает позицию первого вхождения подстроки в строку. Отсчёт начинается с 1. Третий аргумент (опциональный) задаёт позицию в строке, с которой начинается поиск подстроки, тем самым игнорирую любые вхождения подстроки в строку до этой позиции. Если совпадение не найдено, функция возвращает 0.

Особенности использования:

- Опциональный третий параметр поддерживается только вторым вариантом синтаксиса (синтаксис с запятой);
- Пустую строку, функция считает подстрокой любой строки. Поэтому при входном параметре *substr*, равном '' (пустая строка), и при параметре *string*, отличном от NULL, результатом будет:
  - 1, если параметр *startpos* не задан;
  - 1, если параметр *startpos* не задан;
  - *startpos*, если *startpos* не превышает длину параметра *string*.

Примеры:

Пример 8.33. Использование функции POSITION

```
POSITION ('be' IN 'To be or not to be')    -- Результат: 4
POSITION ('be', 'To be or not to be')      -- Результат: 4
POSITION ('be', 'To be or not to be', 4)   -- Результат: 4
POSITION ('be', 'To be or not to be', 8)   -- Результат: 17
POSITION ('be', 'To be or not to be', 18) -- Результат: 0
POSITION ('be' in 'Alas, poor Yorick!') -- Результат: 0
```

См. также: SUBSTRING.

REPLACE

Доступно в: DSQL, PSQL.

Синтаксис:

```
REPLACE (str, find, repl)
```

Параметры функции REPLACE

str
Строка, в которой делается замена.

find
Строка, которая ищется.

repl
Строка, на которую происходит замена.

Тип возвращаемого результата: VARCHAR(N) или BLOB.

Функция REPLACE заменяет в строке все вхождения одной строки на другую строку.

Особенности использования:

- Функция поддерживает текстовые блоки любой длины и с любыми наборами символов;
- Если один из аргументов имеет тип BLOB, то результат будет иметь тип BLOB. В противном случае результат будет иметь тип VARCHAR(N), где N рассчитывается из длин *str*, *find* и *repl* таким образом, что даже максимальное количество замен не будет вызывать переполнения поля.
- Если параметр *find* является пустой строкой, то возвращается *str* без изменений;
- Если параметр *repl* является пустой строкой, то все вхождения *find* удаляются из строки *str*;
- Если любой из аргументов равен NULL, то результатом всегда будет NULL, даже если не было произведено ни одной замены.

> **Предупреждение**
>
> При использовании BLOB в параметрах функции может потребоваться загрузить объект полностью в память. При больших объёмах BLOB могут наблюдаться потери производительности.

Примеры:

Пример 8.34. Использование функции REPLACE

```
REPLACE ('Billy Wilder', 'il', 'oog')  -- возвращает 'Boogly Woogder'
REPLACE ('Billy Wilder', 'il', '')     -- возвращает 'Bly Wder'
REPLACE ('Billy Wilder', null, 'oog')  -- возвращает NULL
REPLACE ('Billy Wilder', 'il', null)   -- возвращает NULL
REPLACE ('Billy Wilder', 'xyz', null)  -- возвращает NULL (!)
REPLACE ('Billy Wilder', 'xyz', 'abc') -- возвращает 'Billy Wilder'
REPLACE ('Billy Wilder', '', 'abc')    -- возвращает 'Billy Wilder'
```

См. также: OVERLAY.

REVERSE

Доступно в: DSQL, PSQL.

Синтаксис:

```
REVERSE (str)
```

Параметры функции REVERSE

str
Выражение строкового типа.

Тип возвращаемого результата: VARCHAR(N).

Функция REVERSE возвратит строку перевёрнутую "задом наперёд".

Примеры:

Пример 8.35. Использование функции REVERSE

```
REVERSE ('spoonful')              -- возвращает 'lufnoops'
REVERSE ('Was it a cat I saw?')   -- возвращает '?was I tac a ti saW'
```

Подсказка

Данная функция очень удобна, если вам предстоит работать (сортировать или группировать информацию) которая находится в окончаниях строк. Пример такой информации – доменные имена или имена адресов электронной почты.

```
CREATE INDEX ix_people_email ON people
COMPUTED BY (reverse(email));

SELECT * FROM people
WHERE REVERSE(email) STARTING WITH reverse('.br');
```

RIGHT

Доступно в: DSQL, PSQL.

Синтаксис:

```
RIGHT (str, num)
```

Параметры функции RIGHT

str
Выражение строкового типа.

num
Целое число. Определяет количество возвращаемых символов.

Тип возвращаемого результата: VARCHAR(N) или BLOB.

Функция RIGHT возвращает конечную (правую) часть входной строки. Длина возвращаемой подстроки определяется вторым параметром.

Особенности использования:

- Функция поддерживает текстовые блоки любой длины и с любыми наборами символов;
- Если строковый аргумент BLOB, результатом будет BLOB, в противном случае результатом будет VARCHAR(N), при этом N – будет равно длине строкового параметра;
- Если числовой параметр превысит длину текста, результатом будет исходный текст.

> **Предупреждение**
>
> При использовании BLOB в параметрах функции может потребоваться загрузить объект полностью в память. При больших объёмах BLOB могут наблюдаться потери производительности.

Примеры:

Пример 8.36. Использование функции RIGHT

```
SELECT RIGHT('ABC', 1) FROM rdb$database;
-- результат C
```

См. также: LEFT, SUBSTRING.

RPAD

Доступно в: DSQL, PSQL.

Синтаксис:

```
RPAD (str, endlen [, padstr])
```

Параметры функции RPAD

str
Выражение строкового типа.

endlen
Длина выходной строки.

padstr
Строка, которой дополняется исходная строка до указанной длины. По умолчанию является пробелом (' ').

Тип возвращаемого результата: VARCHAR(*endlen*) или BLOB.

Функция RPAD дополняет справа входную строку пробелами или определённой пользователем строкой до заданной длины.

Особенности использования:

- Функция поддерживает текстовые блоки любой длины и с любыми наборами символов;
- Если входная строка имеет тип BLOB, то результат также будет BLOB, в противном случае результат будет VARCHAR(*endlen*).
- Если аргумент *padstr* задан, но равен пустой строке (''), то дополнения строки не происходит! В случае если *endlen* меньше длины входной строки, то в результате происходит её усечение до длины *endlen*, даже если параметр *padstr* равен пустой строке.

> **Предупреждение**
>
> При использовании BLOB в параметрах функции может потребоваться загрузить объект полностью в память. При больших объёмах BLOB могут наблюдаться потери производительности.

Примеры:

Пример 8.37. Использование функции RPAD

```
RPAD ('Hello', 12)                -- возвращает 'Hello       '
RPAD ('Hello', 12, '-')           -- возвращает 'Hello-------'
RPAD ('Hello', 12, '')            -- возвращает 'Hello'
RPAD ('Hello', 12, 'abc')         -- возвращает 'Helloabcabca'
RPAD ('Hello', 12, 'abcdefghij')  -- возвращает 'Helloabcdefg'
RPAD ('Hello', 2)                 -- возвращает 'He'
RPAD ('Hello', 2, '-')            -- возвращает 'He'
RPAD ('Hello', 2, '')             -- возвращает 'He'
```

См. также: LPAD.

SUBSTRING

Доступно в: DSQL, PSQL.

Синтаксис:

```
SUBSTRING (<args>)

<args> ::=
    str FROM startpos [FOR length]
  | str SIMILAR <similar_pattern> ESCAPE <escape>

<similar_pattern> ::=
  <similar pattern: R1>
  <escape>"<similar pattern: R2><escape>"
  <similar pattern: R3>
```

Параметры функции SUBSTRING

str
Выражение строкового типа.

startpos
Позиция, с которой начинается извлечение подстроки. Целочисленное выражение.

length
Длина возвращаемой подстроки. Целочисленное выражение.

similar_pattern
Шаблон регулярного выражения SQL, по которому ищется подстрока.

escape
Символ экранирования.

Тип возвращаемого результата: VARCHAR(N) или BLOB.

Функция SUBSTRING возвращает подстроку строки *str*, начиная с позиции *startpos* (позиция начинается с 1) до конца строки или указанной длины. Без предложения FOR возвращаются все оставшиеся символы в строке. С предложением FOR возвращается *length* символов или остаток строки, в зависимости от того что короче.

Функция полностью поддерживает двоичные и текстовые BLOB любой длины и с любым набором символов. Если параметр *str* имеет тип BLOB, то и результат будет иметь тип BLOB. Для любых других типов результатом будет тип VARCHAR(n). Для входного параметра *str*, не являющегося BLOB, длина результата функции всегда будет равна длине строки *str*, независимо от значений параметров *startpos* и *length*.

Функция SUBSTRING с регулярным выражением возвращает часть строки соответствующей шаблону в предложении SIMILAR. Если соответствия не найдено, то возвращается NULL.

Особенности использования функции SUBSTRING с регулярным выражением:

- Если любая из частей (R1, R2 или R3) регулярного выражения не является пустой строкой и не соответствует формату *<similar_pattern>*, будет возбуждено исключение;
- Возвращаемое значение соответствует части R2 регулярного выражения. Для этого значения истинно выражение

```
str SIMILAR TO R1 || R2 || R3 ESCAPE <escape>
```

Если любой из входных параметров имеет значение NULL, то и результат тоже будет иметь значение NULL.

> **Предупреждение**
>
> При использовании BLOB в параметрах функции может потребоваться загрузить объект в память полностью. При больших объёмах BLOB могут наблюдаться потери производительности.

Примеры:

Пример 8.38. Использование функции SUBSTRING

```
SUBSTRING('Привет!' FROM 4 FOR 3)  -- вернёт подстроку 'вет'
```

Пример 8.39. Использование функции SUBSTRING с регулярными выражениями

```
SUBSTRING('abcabc' SIMILAR 'a#"bcab#"c' ESCAPE '#')  -- bcab
SUBSTRING('abcabc' SIMILAR 'a#"%#"c' ESCAPE '#')     -- bcab
SUBSTRING('abcabc' SIMILAR '_#"%#"_' ESCAPE '#')     -- bcab
SUBSTRING('abcabc' SIMILAR '#"(abc)*#"' ESCAPE '#')  -- abcabc
SUBSTRING('abcabc' SIMILAR '#"abc#"' ESCAPE '#')     -- <null>
```

См. также: POSITION, LEFT, RIGHT, CHAR_LENGTH, SIMILAR TO.

TRIM

Доступно в: DSQL, PSQL.

Синтаксис:

```
TRIM ([<adjust>] str)

<adjust> ::=  {[<where>] [what]} FROM

<where> ::=  BOTH | LEADING | TRAILING
```

Параметры функции TRIM

str
: Выражение строкового типа.

where
: Из какого места необходимо удалить подстроку. По умолчанию BOTH.

what
: Подстрока, которую надо удалить (неоднократно, если таких вхождений несколько) из входной строки *str* в её начале и/или конце. По умолчанию является пробелом (' ').

Тип возвращаемого результата: VARCHAR(N) или BLOB.

Функция TRIM удаляет начальные и /или концевые пробелы (или текст согласно настройкам) из входной строки.

Таблица 8.4. Спецификация опций функции TRIM

Опция	Описание
BOTH	с обеих сторон строки (по умолчанию)
LEADING	с начала строки
TRAILING	с конца строки

Особенности использования:

- Если входной параметр *str* имеет тип BLOB, то и результат будет иметь тип BLOB. В противном случае результат будет иметь тип VARCHAR(n), где n является длиной параметра *str*;
- Подстрока для удаления, если она, конечно, задана, не должна иметь длину больше, чем 32767 байта. Однако при повторениях подстроки в начале и/или конце входного параметра *str* общее число удаляемых байтов может быть гораздо больше.

> **Предупреждение**
>
> При использовании BLOB в параметрах функции может потребоваться загрузить объект в память полностью. При больших объёмах BLOB могут наблюдаться потери производительности.

Примеры:

Пример 8.40. Использование функции TRIM

```
SELECT TRIM (' Waste no space ')
FROM RDB$DATABASE -- Результат: 'Waste no space'

SELECT TRIM (LEADING FROM ' Waste no space ')
FROM RDB$DATABASE -- Результат: 'Waste no space '

SELECT TRIM (LEADING '.' FROM ' Waste no space ')
FROM RDB$DATABASE -- Результат: ' Waste no space '

SELECT TRIM (TRAILING '!' FROM 'Help!!!!')
FROM RDB$DATABASE -- Результат: 'Help'

SELECT TRIM ('la' FROM 'lalala I love you Ella')
FROM RDB$DATABASE -- Результат: ' I love you El'
```

См. также: OVERLAY, REPLACE.

UPPER

Доступно в: DSQL, PSQL, ESQL.

Синтаксис:

```
UPPER (str)
```

Параметры функции UPPER

str
Выражение строкового типа.

Тип возвращаемого результата: VARCHAR(N) или BLOB.

Функция UPPER возвращает входную строку в верхнем регистре. Точный результат зависит от набора символов входной строки. Например, для наборов символов NONE и ASCII только ASCII символы переводятся в верхний регистр; для OCTETS — вся входная строка возвращается без изменений.

Примеры:

Пример 8.41. Использование функции UPPER

```
/* Результат: 'DEBACLE', в соответствии с французскими
 * правилами приведения в верхний регистр
 */
SELECT UPPER(_ISO8859_1 'Débâcle' COLLATE FR_FR)
FROM RDB$DATABASE
```

См. также: LOWER.

Функции для работы с датой и временем

DATEADD

Доступно в: DSQL, PSQL.

Синтаксис:

```
DATEADD (<args>)

<args> ::= amount <unit> TO datetime
         | <unit>, amount, datetime

<unit> ::=
    YEAR | MONTH | WEEK | DAY | WEEKDAY | YEARDAY
  | HOUR | MINUTE | SECOND | MILLISECOND
```

Параметры функции DATEADD

amount
Выражение типа SMALLINT, INTEGER, BIGINT или NUMERIC (отрицательное вычитается).

unit
Составляющая даты/времени.

datetime
Выражение типа DATE, TIME или TIMESTAMP.

Тип возвращаемого результата: определяется третьим аргументом функции.

Функция DATEADD позволяет добавить заданное число лет, месяцев, недель, часов, минут, секунд, миллисекунд к заданному значению даты/времени.

> **Примечание**
>
> - С аргументом типа TIMESTAMP и DATE можно использовать любую составляющую даты/времени (*<unit>*);
> - Для типа данных TIME разрешается использовать только HOUR, MINUTE, SECOND и MILLISECOND.

Примеры:

Пример 8.42. Использование функции DATEADD

```
DATEADD (28 DAY TO CURRENT_DATE)
DATEADD (-6 HOUR TO CURRENT_TIME)
DATEADD (MONTH, 9, DATEOFCONCEPTION)
DATEADD (-38 WEEK TO DATEOFBIRTH)
DATEADD (MINUTE, 90, TIME 'NOW')
DATEADD (? YEAR TO DATE '11-SEP-1973')

SELECT
  CAST(DATEADD(-1 * EXTRACT(MILLISECOND FROM ts) MILLISECOND TO ts) AS VARCHAR(30)) AS t,
  EXTRACT(MILLISECOND FROM ts) AS ms
FROM (
    SELECT TIMESTAMP'2014-06-09 13:50:17.4971' as ts
    FROM RDB$DATABASE
) a
```

```
T                               MS
----------------------------------
2014-06-09 13:50:17.0000 497.1
```

См. также: DATEDIFF, Операции, использующие значения даты и времени.

DATEDIFF

Доступно в: DSQL, PSQL.

Синтаксис:

```
DATEDIFF (<args>)
```

```
<args> ::= <unit> FROM moment_1 TO moment_2
         | <unit>, moment_1, moment_2

<unit> ::=
    YEAR | MONTH | WEEK | DAY | WEEKDAY | YEARDAY
  | HOUR | MINUTE | SECOND | MILLISECOND
```

Параметры функции DATEDIFF

unit
Составляющая даты/времени.

monent_1
Выражение типа DATE, TIME или TIMESTAMP.

monent_2
Выражение типа DATE, TIME или TIMESTAMP.

Тип возвращаемого результата: BIGINT.

Функция DATEDIFF возвращает количество лет, месяцев, недель, дней, часов, минут, секунд или миллисекунд между двумя значениями даты/времени.

Особенности использования:

- Параметры DATE и TIMESTAMP могут использоваться совместно. Совместное использование типа TIME с типами DATE и TIMESTAMP не разрешается;
- С аргументом типа TIMESTAMP и DATE можно использовать любую составляющую даты/времени (*<unit>*);
- Для типа данных TIME разрешается использовать только HOUR, MINUTE, SECOND и MILLISECOND.

Примечание

- Функция DATEDIFF не проверяет разницу в более мелких составляющих даты/времени, чем задана в первом аргументе (*<unit>*). В результате получаем:

```
- DATEDIFF (YEAR, DATE '1-JAN-2009', DATE '31-DEC-2009') = 0, но
- DATEDIFF (YEAR, DATE '31-DEC-2009', DATE '1-JAN-2010') = 1
```

- Однако для более мелких составляющих даты/времени имеем:

```
- DATEDIFF (DAY, DATE '26-JUN-1908', DATE '11-SEP-1973') = 23818
- DATEDIFF (DAY, DATE '30-NOV-1971', DATE '8-JAN-1972') = 39
```

- Отрицательное значение функции говорит о том, что дата/время в *moment_2* меньше, чем в *moment_1*.

Примеры:

Пример 8.43. Использование функции DATEDIFF

```
DATEDIFF (HOUR FROM CURRENT_TIMESTAMP TO TIMESTAMP '12-JUN-2059 06:00')
DATEDIFF (MINUTE FROM TIME '0:00' TO CURRENT_TIME)
DATEDIFF (MONTH, CURRENT_DATE, DATE '1-1-1900')
DATEDIFF (DAY FROM CURRENT_DATE TO CAST (? AS DATE))
```

См. также: DATEADD, Операции, использующие значения даты и времени.

EXTRACT

Доступно в: DSQL, PSQL, ESQL.

Синтаксис:

```
EXTRACT (<part> FROM datetime)

<part> ::=
    YEAR | MONTH | WEEK | DAY | WEEKDAY | YEARDAY
  | HOUR | MINUTE | SECOND | MILLISECOND
```

Параметры функции EXTRACT

part
Составляющая даты/времени.

datetime
Выражение типа DATE, TIME или TIMESTAMP.

Тип возвращаемого результата: SMALLINT или NUNERIC.

Функция EXTRACT извлекает составляющие даты и времени из типов данных DATE, TIME и TIMESTAMP.

Таблица 8.5. Типы и диапазоны результатов функции EXTRACT

Составляющая даты/времени	Тип	Диапазон	Комментарий
YEAR	SMALLINT	1–9999	
MONTH	SMALLINT	1–12	
WEEK	SMALLINT	1–53	
DAY	SMALLINT	1–31	
WEEKDAY	SMALLINT	0–6	0 = Воскресенье
YEARDAY	SMALLINT	0–365	0 = 1 января
HOUR	SMALLINT	0–23	
MINUTE	SMALLINT	0–59	
SECOND	NUMERIC(9,4)	0.0000–59.9999	Включает в себя миллисекунды

Составляющая даты/времени	Тип	Диапазон	Комментарий
MILLISECOND	NUMERIC(9,1)	0.0–999.9	

> **Примечание**
>
> Если составляющая даты/времени не присутствует в аргументе дата/время, например SECOND в аргументе с типом DATE или YEAR в TIME, то функция вызовет ошибку.

Из аргумента с типом данных DATE или TIMESTAMP можно извлекать номер недели. В соответствии со стандартом ISO-8601 неделя начинается с понедельника и всегда включает в себя 7 дней. Первой неделей года является первая неделя, у которой в ней больше дней в новом году (по крайней мере, 4): дни 1-3 могут принадлежать предыдущей неделе (52 или 53) прошлого года. По аналогии дни 1-3 текущего года могут принадлежать 1 неделе следующего года.

Примеры:

Пример 8.44. Использование функции EXTRACT

```
/* получить по дате номер квартала */
SELECT (EXTRACT(MONTH FROM CURRENT_TIMESTAMP)-1)/3+1
FROM RDB$DATABASE
```

См. также: Типы данных для работы с датой и временем.

Функции преобразования типов

CAST

Доступно в: DSQL, PSQL, ESQL.

Синтаксис:

```
CAST(value | NULL AS <type>)

<type> ::=
    <datatype>
  | [TYPE OF] domain
  | TYPE OF COLUMN relname.colname

<datatype> ::=
    {SMALLINT | INTEGER | BIGINT}
  | {FLOAT | DOUBLE PRECISION}
  | {DATE | TIME | TIMESTAMP}
  | BOOLEAN
  | {DECIMAL | NUMERIC} [(precision [, scale])]
  | {CHAR | CHARACTER | CHARACTER VARYING | VARCHAR} [(size)]
    [CHARACTER SET charset]
```

```
  | {NCHAR | NATIONAL CHARACTER | NATIONAL CHAR} [VARYING] [(size)]
  | BLOB [SUB_TYPE {subtype_num | subtype_name}]
    [SEGMENT SIZE seglen] [CHARACTER SET charset]
  | BLOB [(seglen [, subtype_num])]
```

Сокращённый синтаксис:

```
<datatype> 'date/timestring'

<datatype> ::= DATE | TIME | TIMESTAMP
```

Параметры функции CAST

value
SQL выражение.

datatype
Тип данных SQL.

domain
Домен.

relname
Имя таблицы или представления.

colname
Имя столбца таблицы или представления.

precision
Точность. От 1 до 18.

scale
Масштаб. От 0 до 18, должно быть меньше или равно *precision*.

size
Максимальный размер строки в символах.

charset
Набор символов.

subtype_num
Номер подтипа BLOB.

subtype_name
Мнемоника подтипа BLOB.

seqlen
Размер сегмента, не может превышать 65535

Тип возвращаемого результата: *<type>*.

Функция CAST служит для явного преобразования данных из одного типа данных в другой тип данных или домен. Если это невозможно будет выдана ошибка.

Сокращённый синтаксис поддерживается только для литералов даты и времени. Доступен только в 3 диалекте.

> **Примечание**
>
> Сокращенный синтаксис вычисляется сразу во время синтаксического анализа, в результате чего значение сохраняется до тех пор пока для оператора не сделано unprepare. Для литералов даты и времени таких как '12-OCT-2012' это не имеет никакого значения. Но для псевдо-переменных 'NOW', 'YESTERDAY', 'TODAY' и 'TOMORROW' вы можете получить не то, что хотите. Если вам нужно значение, которое будет вычисляться при каждом вызове, используйте CAST().

Таблица 8.6. Допустимые преобразования для функции CAST

Из типа	В тип
Числовые типы	Числовые типы [VAR]CHAR BLOB
[VAR]CHAR BLOB	[VAR]CHAR BLOB BOOLEAN Числовые типы DATE TIME TIMESTAMP
DATE TIME	[VAR]CHAR BLOB TIMESTAMP
TIMESTAMP	[VAR]CHAR BLOB TIME DATE
BOOLEAN	[VAR]CHAR BLOB

Имейте ввиду, что иногда информация может быть потерянна, например, когда вы преобразуете тип TIMESTAMP к DATE. Кроме того, тот факт, что типы совместимы для функции CAST, ещё не гарантирует, что преобразование будет успешным. "CAST (123456789 AS SMALLINT)" безусловно приведёт к ошибке, так же как и "CAST('Judgement Day' as DATE)".

Примеры:

Полный синтаксис:

```
SELECT CAST ('12' || '-June-' || '1959' AS DATE) FROM rdb$database
```

Сокращённый синтаксис для преобразование литералов дат/времени:

```
```

```
UPDATE People SET AgeCat = 'Old'
WHERE BirthDate < date '1-Jan-1943'
```

Заметьте, что вы можете не использовать даже краткий синтаксис преобразования в примере выше, т.к. Firebird поймёт из контекста (сравнение с полем типа DATE) как интерпретировать строку:

```
UPDATE People SET AgeCat = 'Old'
WHERE BirthDate < '1-Jan-1943'
```

Но это не всегда возможно. Преобразование в примере ниже не может быть опущено, т.к. система будет пытаться преобразовать строку к числу. что бы вычесть из неё число:

```
SELECT date 'today' - 7 FROM rdb$database
```

Вы можете применить преобразование типа к параметрам оператора:

```
CAST (? AS INTEGER)
```

Это дает вам контроль над типом полей ввода. Обратите внимание, что с параметрами операторов, вы всегда должны использовать полный синтаксис преобразования, сокращённый синтаксис не поддерживается.

Преобразование к домену или к его базовому типу

При преобразовании к домену должны быть удовлетворены любые ограничения (NOT NULL и/или CHECK) объявленные для домена, иначе преобразование не будет выполнено. Помните, что проверка CHECK проходит, если его вычисление даёт TRUE или UNKNOWN (NULL). Для следующих операторов:

```
CREATE DOMAIN quint AS INT CHECK (VALUE >= 5000)
SELECT CAST (2000 AS quint) FROM rdb$database -- (1)
SELECT CAST (8000 AS quint) FROM rdb$database -- (2)
SELECT CAST (null AS quint) FROM rdb$database -- (3)
```

только (1) завершится с ошибкой.

При использовании модификатора TYPE OF выражение будет преобразовано к базовому типу домена, игнорируя любые ограничения. Для домена quint, объявленного выше, оба преобразования будут эквивалентны и оба будут успешно выполнены:

```
SELECT CAST (2000 AS TYPE OF quint) FROM rdb$database
SELECT CAST (2000 AS INT) FROM rdb$database
```

При использовании TYPE OF с (VAR)CHAR типом, его набор символов и порядок сортировки (collate) сохраняются.

```
CREATE DOMAIN iso20 VARCHAR(20) CHARACTER SET iso8859_1;
CREATE DOMAIN dunl20 VARCHAR(20) CHARACTER SET iso8859_1 COLLATE du_nl;
CREATE TABLE zinnen (zin VARCHAR(20));
COMMIT;
INSERT INTO zinnen VALUES ('Deze');
INSERT INTO zinnen VALUES ('Die');
INSERT INTO zinnen VALUES ('die');
INSERT INTO zinnen VALUES ('deze');
SELECT CAST(zin AS TYPE OF iso20) FROM zinnen ORDER BY 1;
-- returns Deze -> Die -> deze -> die
SELECT CAST(zin AS TYPE OF dunl20) FROM zinnen ORDER BY 1;
-- returns deze -> Deze -> die -> Die
```

Предупреждение

Если определение домена изменяется, то существующие преобразования к домену или его типу могут стать ошибочными. Если такие преобразования происходят в PSQL модулях, то их ошибки могут быть обнаружены. См. Поле RDB$VALID_BLR.

Преобразование к типу столбца

Разрешено преобразовывать выражение к типу столбца существующей таблицы или представления. При этом будет использован только сам тип, для строковых типов будет использован так же набор символов, но не последовательность сортировки. Ограничения и значения по умолчанию исходного столюца не применяются.

```
CREATE TABLE ttt (
  s VARCHAR(40) CHARACTER SET utf8 COLLATE unicode_ci_ai
);
COMMIT;
SELECT CAST ('Jag har många vänner' AS TYPE OF COLUMN ttt.s)
FROM rdb$database;
```

Предупреждение

Если определение столбца изменяется, то существующие преобразования к его типу могут стать ошибочными. Если такие преобразования происходят в PSQL модулях, то их ошибки могут быть обнаружены. См. Поле RDB$VALID_BLR.

См. также: Явное преобразование типов данных.

Функции побитовых операций

BIN_AND

Доступно в: DSQL, PSQL.

Синтаксис:

```
BIN_AND (number [, number ...])
```

Параметры функции BIN_AND

number
Целое число.

Тип возвращаемого результата: INTEGER или BIGINT.

Функция BIN_AND возвращает результат побитовой операции AND (И) аргументов.

См. также: BIN_OR, BIN_XOR.

BIN_OR

Доступно в: DSQL, PSQL.

Синтаксис:

```
BIN_OR (number [, number ...])
```

Параметры функции BIN_OR

number
Целое число.

Тип возвращаемого результата: INTEGER или BIGINT.

Функция BIN_OR возвращает результат побитовой операции OR (ИЛИ) аргументов.

См. также: BIN_AND, BIN_XOR.

BIN_SHL

Доступно в: DSQL, PSQL.

Синтаксис:

```
BIN_SHL (number, shift)
```

Параметры функции BIN_SHL

number
Целое число.

shift
Количество бит, на которое смещается значение *number*.

Тип возвращаемого результата: BIGINT.

Функция BIN_SHL возвращает первый параметр, побитно смещённый влево на значение второго параметра.

См. также: BIN_SHR.

BIN_SHR

Доступно в: DSQL, PSQL.

Синтаксис:

```
BIN_SHR (number, shift)
```

Параметры функции BIN_SHR

number
Целое число.

shift
Количество бит на которое смещается значение number.

Тип возвращаемого результата: BIGINT.

Функция BIN_SHR возвращает первый параметр, побитно смещённый вправо на значение второго параметра.

> **Примечание**
>
> Выполняемая операция является арифметическим сдвигом вправо (SAR), а это означает, что знак первого операнда всегда сохраняется.

См. также: BIN_SHL.

BIN_XOR

Доступно в: DSQL, PSQL.

Синтаксис:

```
BIN_XOR (number [, number ...])
```

Параметры функции BIN_XOR

number
Целое число.

Тип возвращаемого результата: INTEGER или BIGINT.

Функция BIN_XOR возвращает результат побитовой операции XOR аргументов.

См. также: BIN_AND, BIN_OR.

Функции для работы с UUID

CHAR_TO_UUID

Доступно в: DSQL, PSQL.

Синтаксис:

```
CHAR_TO_UUID (ascii_uuid)
```

Параметры функции CHAR_TO_UUID

ascii_uuid
: 36-символьное представление UUID. '-' (дефис) в положениях 9, 14, 19 и 24; допустимые шестнадцатеричные цифры в любых других позициях.

Тип возвращаемого результата: CHAR(16) CHARACTER SET OCTETS.

Функция CHAR_TO_UUID преобразует читабельную 36-ти символьную символику UUID к соответствующему 16-ти байтовому значению UUID.

Примеры:

Пример 8.45. Использование функции CHAR_TO_UUID

```
SELECT CHAR_TO_UUID('A0bF4E45-3029-2a44-D493-4998c9b439A3') FROM rdb$database
-- returns A0BF4E4530292A44D4934998C9B439A3 (16-byte string)

SELECT CHAR_TO_UUID('A0bF4E45-3029-2A44-X493-4998c9b439A3') FROM rdb$database
-- error: -Human readable UUID argument for CHAR_TO_UUID must
-- have hex digit at position 20 instead of "X (ASCII 88)"
```

См. также: GEN_UUID, UUID_TO_CHAR.

GEN_UUID

Доступно в: DSQL, PSQL.

Синтаксис:

```
GEN_UUID()
```

Тип возвращаемого результата: CHAR(16) CHARACTER SET OCTETS.

Функция возвращает универсальный уникальный идентификатор ID в виде 16-байтной строки символов, отвечающий требованиям стандарта RFC-4122. Функция возвращает строку UUID 4-ой версии, где несколько битов зарезервированы, а остальные являются случайными.

Примеры:

Пример 8.46. Использование функции GEN_UUID

```
SELECT GEN_UUID() AS GUID FROM RDB$DATABASE

/* результат будет возвращён в виде
 XXXXXXXX-XXXX-4XXX-YXXX-XXXXXXXXXXXX
 где 4 это номер версии, а Y может принимать значение 8, 9, A или B.
*/
```

```
852C5DD9-3453-430B-B697-D0A46B4D2531
```

См. также: CHAR_TO_UUID, UUID_TO_CHAR.

UUID_TO_CHAR

Доступно в: DSQL, PSQL.

Синтаксис:

```
UUID_TO_CHAR (uuid)
```

Параметры функции UUID_TO_CHAR

uuid
16-байтный UUID.

Тип возвращаемого результата: CHAR(36).

Функция UUID_TO_CHAR конвертирует 16-ти байтный UUID в его 36-ти знаковое ASCII человекочитаемое представление. Тип возвращаемого значения CHAR(36).

Примеры:

Пример 8.47. Использование функции UUID_TO_CHAR

```
SELECT UUID_TO_CHAR(GEN_UUID()) FROM RDB$DATABASE;

SELECT UUID_TO_CHAR(x'876C45F4569B320DBCB4735AC3509E5F') FROM RDB$DATABASE;
-- returns '876C45F4-569B-320D-BCB4-735AC3509E5F'

SELECT UUID_TO_CHAR(GEN_UUID()) FROM RDB$DATABASE;
-- returns e.g. '680D946B-45FF-DB4E-B103-BB5711529B86'

SELECT UUID_TO_CHAR('Firebird swings!') FROM RDB$DATABASE;
-- returns '46697265-6269-7264-2073-77696E677321'
```

См. также: GEN_UUID, CHAR_TO_UUID.

Функции для работы с генераторами (последовательностями)

GEN_ID

Доступно в: DSQL, PSQL, ESQL.

Синтаксис:

```
GEN_ID(gen_name, step)
```

Параметры функции GEN_ID

gen_name
Имя генератора (последовательности).

step
Шаг приращения.

Тип возвращаемого результата: BIGINT.

Функция GEN_ID увеличивает значение генератора или последовательности и возвращает новое значение.

> **Примечание**
>
> Если значение параметра *step* меньше нуля, произойдёт уменьшение значения генератора. Внимание! Следует быть крайне аккуратным при таких манипуляциях в базе данных, они могут привести к потере целостности данных. Если *step* равен 0, функция не будет ничего делать со значением генератора и вернёт его текущее значение.

Начиная с Firebird 2.0 для получения следующего значение последовательности (генератора) стало доступно использование оператора NEXT VALUE FOR.

Примеры:

Пример 8.48. Использование функции GEN_ID

```
NEW.ID = GEN_ID (GEN_TABLE_ID, 1);
```

См. также: NEXT VALUE FOR, SEQUENCE (GENERATOR), ALTER SEQUENCE, SET GENERATOR.

Условные функции

COALESCE

Доступно в: DSQL, PSQL.

Синтаксис:

```
COALESCE(expr1, expr2 [, exprN ...])
```

Параметры функции COALESCE

expr1, *expr2* ... *exprN*
Выражения любого совместимого типа.

Тип возвращаемого результата: тот же что и первый аргумент функции *expr1*.

Функция COALESCE принимает два или более аргумента возвращает значение первого NOT NULL аргумента. Если все аргументы имеют значение NULL, то и результат будет NULL.

Примеры:

Пример 8.49. Использование функции COALESCE

```
SELECT
  COALESCE(PE.NICKNAME, PE.FIRSTNAME, 'Mr./Mrs.') ||
  ' ' || PE.LASTNAME AS FULLNAME
FROM PERSONS PE
```

В данном примере предпринимается попытка использовать все имеющиеся данные для составления полного имени. Выбирается поле NICKNAME из таблицы PERSONS. Если оно имеет значение NULL, то берётся значение из поля FIRSTNAME. Если и оно имеет значение NULL, то используется строка 'Mr./Mrs.'. Затем к значению функции COALESCE фамилия (поле LASTNAME). Обратите внимание, что эта схема нормально работает, только если выбираемые поля имеют значение NULL или не пустое значение: если одно из них является пустой строкой, то именно оно и возвратится в качестве значения функции COALESCE.

Пример 8.50. Использование функции COALESCE с агрегатными функциями

```
-- в случае получения при суммировании NULL, вернёт 0.
SELECT coalesce (sum (q), 0)
FROM bills
WHERE ...
```

См. также: CASE.

DECODE

Доступно в: DSQL, PSQL.

Синтаксис:

```
DECODE(testexpr,
       expr1, result1
       [, expr2, result2 ...]
       [, defaultresult])
```

эквивалентная конструкция CASE

```
CASE testexpr
  WHEN expr1 THEN result1
  [WHEN expr2 THEN result2 ...]
  [ELSE defaultresult]
END
```

Параметры функции DECODE

testexpr
Выражения любого совместимого типа, которое сравнивается с выражениями *expr1*, *expr2* ... *exprN*

expr1, *expr2* ... *exprN*
Выражения любого совместимого типа, с которыми сравнивают с выражением *testexpr*.

result1, *result2* ... *resultN*
Возвращаемые выражения любого типа.

defaultresult
Выражения, возвращаемое если ни одно из условий не было выполнено.

Тип возвращаемого результата: тот же что и первый результат *result1*.

Данная функция эквивалентна конструкции Простой CASE, в которой заданное выражение сравнивается с другими выражениями до нахождения совпадения. Результатом является значение, указанное после выражения, с которым найдено совпадение. Если совпадений не найдено, то возвращается значение по умолчанию (если оно, конечно, задано – в противном случае возвращается NULL).

> **Внимание**
>
> Совпадение эквивалентно оператору "=", т.е. если *testexpr* имеет значение NULL, то он не соответствует ни одному из *expr*, даже тем, которые имеют значение NULL.

Примеры:

Пример 8.51. Использование функции DECODE

```
SELECT
  name,
  age,
  decode (upper(sex),
          'M', 'М',
          'F', 'Ж',
          'не указано') AS sexname,
  UID
FROM people
```

См. также: CASE.

IIF

Доступно в: DSQL, PSQL.

Синтаксис:

```
IIF(<condition>, resultT, resultF)
```

Параметры функции IIF

condition
Выражение логического типа.

resultT
Возвращаемое значение, если *condition* является истинным.

resultF
Возвращаемое значение, если *condition* является ложным.

Тип возвращаемого результата: тот же что и аргумент функции *resultT*.

Функция IIF имеет три аргумента. Если первый аргумент является истиной, то результатом будет второй параметр, в противном случае результатом будет третий параметр.

По сути, функция IIF это короткая запись оператора CASE

```
CASE WHEN <condition> THEN resultT ELSE resultF END
```

Оператор IIF также можно сравнить в тройным оператором "?:" в C-подобных языках.

Примеры:

Пример 8.52. Использование функции IIF

```
SELECT IIF(SEX = 'M', 'Sir', 'Madam') FROM CUSTOMERS
```

См. также: CASE.

MAXVALUE

Доступно в: DSQL, PSQL.

Синтаксис:

```
MAXVALUE(expr1 [, exprN ...])
```

Параметры функции MAXVALUE

expr1 ... *exprN*
Выражения любого совместимого типа.

Тип возвращаемого результата: тот же что и первый аргумент функции *expr1*.

Возвращает максимальное значение из входного списка чисел, строк или параметров с типом DATE/TIME/TIMESTAMP.

> **Примечание**
>
> Если один или более входных параметров имеют значение NULL, то результатом функции MAXVALUE тоже будет NULL в отличие от агрегатной функции MAX.

Примеры:

Пример 8.53. Использование функции MAXVALUE

```
SELECT MAXVALUE(PRICE_1, PRICE_2) AS PRICE
FROM PRICELIST
```

См. также: MINVALUE.

MINVALUE

Доступно в: DSQL, PSQL.

Синтаксис:

```
MINVALUE(expr1 [, exprN ...])
```

Параметры функции MINVALUE

expr1 ... *exprN*
Выражения любого совместимого типа.

Тип возвращаемого результата: тот же что и первый аргумент функции *expr1*.

Возвращает минимальное значение из входного списка чисел, строк или параметров с типом DATE/TIME/TIMESTAMP.

> **Примечание**
>
> Если один или более входных параметров имеют значение NULL, то результатом функции MINVALUE тоже будет NULL в отличие от агрегатной функции MIN.

Примеры:

Пример 8.54. Использование функции MINVALUE

```
SELECT MINVALUE(PRICE_1, PRICE_2) AS PRICE
FROM PRICELIST
```

См. также: MAXVALUE.

NULLIF

Доступно в: DSQL, PSQL.

Синтаксис:

```
NULLIF(expr1, expr2)
```

Параметры функции NULLIF

expr1, *expr2*
Выражения любого совместимого типа.

Тип возвращаемого результата: тот же что и первый аргумент функции *expr1*.

Функция возвращает значение первого аргумента, если он неравен второму. В случае равенства аргументов возвращается NULL.

Примеры:

Пример 8.55. Использование функции NULLIF

```
/* запрос вернёт среднее значение поля weight по таблице,
   за исключением строк, где он не указан (равен -1).
   Если бы не было этой функции простой оператор avg(weight)
   вернул бы некорректное значение */
SELECT AVG(NULLIF(weight, -1)) FROM cargo;
```

См. также: COALESCE, CASE.

Агрегатные функции

Агрегатные функции выполняют вычисление на наборе значений и возвращают одиночное значение. Агрегатные функции, за исключением COUNT, не учитывают значения NULL. Агрегатные функции часто используются совместно с предложением GROUP BY.

Агрегатные функции могут быть использованы в качестве выражений только в следующих случаях:

- Список выбора инструкции SELECT (вложенный или внешний запрос);
- Предложение HAVING.

Любая агрегатная функция может быть использована в качестве оконной. Подробнее см. OVER.

AVG

Доступно в: DSQL.

Синтаксис:

```
AVG([ALL | DISTINCT] <expr>)
  [OVER ([<partition_exp>] [<order_exp>])]
```

Параметры функции AVG

expr
: Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Тип возвращаемого результата: тот же что и аргумент функции *expr*.

Функция AVG возвращает среднее значение для группы. Значения NULL пропускаются.

Параметр ALL применяет агрегатную функцию ко всем значениям. ALL является параметром по умолчанию. Параметр DISTINCT указывает на то, что функция AVG будет выполнена только для одного экземпляра каждого уникального значения, независимо от того, сколько раз встречается это значение.

В случае если выборка записей пустая или содержит только значения NULL, результат будет содержать NULL.

Примеры:

Пример 8.56. Использование функции AVG

```
SELECT
  dept_no,
  AVG(salary)
FROM employee
GROUP BY dept_no
```

См. также: SELECT.

COUNT

Доступно в: DSQL.

Синтаксис:

```
COUNT({[ALL | DISTINCT] <expr> | *})
  [OVER ([<partition_exp>] [<order_exp>])]
```

Параметры функции COUNT

expr
: Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF. Агрегатные функции в качестве выражения не допускаются.

Тип возвращаемого результата: BIGINT.

Функция COUNT возвращает количество значений в группе, которые не являются NULL.

При указании DISTINCT из выборки устраняются дубликаты, ALL является значением по умолчанию для всех выборки значений не NULL.

Если вместо выражения *expr* указана звёздочка (*), то будут подсчитаны все строки. Функция COUNT(*) не принимает параметры и не может использоваться с ключевым словом DISTINCT. Для функции COUNT(*) не нужен параметр *expr*, так как по определению она не использует сведения о каких-либо конкретных столбцах. Функция COUNT(*) возвращает количество строк в указанной таблице, не отбрасывая дублированные строки. Она подсчитывает каждую строку отдельно. При этом учитываются и строки, содержащие значения NULL.

Для пустой выборки данных или если при выборке окажутся одни значения, содержащие NULL, функция возвратит значение равное 0.

Примеры:

Пример 8.57. Использование функции COUNT

```
SELECT
  dept_no,
  COUNT(*) AS cnt,
  COUNT(DISTINCT name) AS cnt_name
FROM employee
GROUP BY dept_no
```

См. также: SELECT.

LIST

Доступно в: DSQL.

Синтаксис:

```
LIST([ALL | DISTINCT] <expr> [, separator])
  [OVER ([<partition_exp>])]
```

Параметры функции LIST

expr
: Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает строковый тип данных или BLOB. Поля типа дата / время и числовые преобразуются к строке. Агрегатные функции в качестве выражения не допускаются.

separator
: Разделитель. Выражение строкового типа. По умолчанию разделителем является запятая.

Тип возвращаемого результата: BLOB.

Функция LIST возвращает строку, содержащую значения элементов выборки, которые не равны NULL. При пустой выборке функция возвратит NULL. Тип возвращаемого значения текстовый BLOB за исключением тех случаев, когда выражением являются BLOB других подтипов.

ALL является опцией по умолчанию. При ней обрабатываются все значения в выборке, не содержащие NULL. При указании DISTINCT из выборки устраняются дубликаты.

Значения выражения *expr* и разделитель *separator* поддерживают тип данных BLOB любого размера и набора символов. Поля типа дата / время и числовые перед проведением операции конкатенации преобразуются в строки.

> **Примечание**
>
> Порядок конкатенации строк определяется порядком чтения записей из источников, который в общем случае не определён. Для придания списку необходимого порядка вы можете предварительно упорядочить источник данных, например с помощью производной таблицы.

Примеры:

Пример 8.58. Использование функции LIST

```
-- Получение списка, порядок не определён
SELECT LIST (display_name, '; ')
FROM GR_WORK;

-- Получение списка в алфавитном порядке
SELECT LIST (display_name, '; ')
FROM (SELECT display_name
      FROM GR_WORK
      ORDER BY display_name);
```

См. также: SELECT.

MAX

Доступно в: DSQL.

Синтаксис:

```
MAX([ALL | DISTINCT] <expr>)
  [OVER ([<partition_exp>] [<order_exp>])]
```

Параметры функции MAX

expr
: Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF. Агрегатные функции в качестве выражения не допускаются.

Тип возвращаемого результата: DOUBLE PRECISION или масштабируемый BIGINT в зависимости от типа аргумента функции *expr*.

Функция MAX возвращает максимальный элемент выборки, которые не равны NULL. При пустой выборке, или при выборке из одних NULL функция возвратит NULL. Если аргумент функции строка, то функция вернёт значение, которое окажется последним в сортировке при применении COLLATE.

> **Примечание**
>
> Параметр DISTINCT не имеет смысла при использовании функцией MAX и доступен только для совместимости со стандартом.

Примеры:

Пример 8.59. Использование функции MAX

```
SELECT
  dept_no,
  MAX(salary)
FROM employee
GROUP BY dept_no
```

См. также: SELECT, MIN.

MIN

Доступно в: DSQL.

Синтаксис:

```
MIN([ALL | DISTINCT] <expr>)
  [OVER ([<partition_exp>] [<order_exp>])]
```

Параметры функции MIN

expr
: Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF. Агрегатные функции в качестве выражения не допускаются.

Тип возвращаемого результата: тот же что и аргумент функции *expr*.

Функция MIN возвращает минимальный элемент выборки, которые не равны NULL. При пустой выборке, или при выборке из одних NULL функция возвратит NULL. Если аргумент функции строка, то функция вернёт значение, которое окажется первым в сортировке при применении COLLATE.

> **Примечание**
>
> Параметр DISTINCT не имеет смысла при использовании функцией MIN и доступен только для совместимости со стандартом.

Примеры:

Пример 8.60. Использование функции MIN

```
SELECT
  dept_no,
  MIN(salary)
FROM employee
GROUP BY dept_no
```

См. также: SELECT, MAX.

SUM

Доступно в: DSQL.

Синтаксис:

```
SUM([ALL | DISTINCT] <expr>)
  [OVER ([<partition_exp>] [<order_exp>])]
```

Параметры функции SUM

expr
: Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Тип возвращаемого результата: тот же что и аргумент функции *expr*.

Функция SUM возвращает сумму элементов выборки, которые не равны NULL. При пустой выборке, или при выборке из одних NULL функция возвратит NULL.

ALL является опцией по умолчанию. При ней обрабатываются все значения из выборки, не содержащие NULL. При указании DISTINCT из выборки устраняются дубликаты, после осуществляется подсчёт.

Примеры:

Пример 8.61. Использование функции SUM

```
SELECT
  dept_no,
  SUM(salary)
FROM employee
GROUP BY dept_no
```

См. также: SELECT.

Статистические функции

Статистические функции являются агрегатными функциями. Эти функции не учитывают значения NULL. К аргументу статистической функции не применимы параметры ALL и DISTINCT.

Статистические функции часто используются совместно с предложением GROUP BY. Любую из статистических функций можно использовать в качестве оконной. Подробнее см. OVER.

CORR

Доступно в: DSQL.

Синтаксис:

```
CORR(<expr1>, <expr2>)
  [OVER ([<partition_exp>] [<order_exp>])]
```

Параметры функции CORR

expr1
: Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

expr2
: Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Тип возвращаемого результата: DOUBLE PRECISION.

Функция CORR возвращает коэффициент корреляции для пары выражений, возвращающих числовые значения.

Функция

```
CORR(<expr1>, <expr2>)
```

эквивалентна

```
COVAR_POP(<expr1>, <expr2>) / (STDDEV_POP(<expr2>) * STDDEV_POP(<expr1>))
```

В статистическом смысле, корреляция — это степень связи между переменными. Связь между переменными означает, что значение одной переменной можно в определённой степени предсказать по значению другой. Коэффициент корреляции представляет степень корреляции в виде числа в диапазоне от -1 (высокая обратная корреляция) до 1 (высокая корреляция). Значение 0 соответствует отсутствию корреляции.

В случае если выборка записей пустая или содержит только значения NULL, результат будет содержать NULL.

Примеры:

Пример 8.62. Использование функции CORR

```
SELECT
    CORR(alength, aheight) AS c_corr
FROM measure
```

См. также: COVAR_POP, STDDEV_POP.

COVAR_POP

Доступно в: DSQL.

Синтаксис:

```
COVAR_POP(<expr1>, <expr2>)
  [OVER ([<partition_exp>] [<order_exp>])]
```

Параметры функции COVAR_POP

expr1
Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

expr2
Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Тип возвращаемого результата: DOUBLE PRECISION.

Функция COVAR_POP возвращает ковариацию совокупности (population covariance) пар выражений с числовыми значениями.

Функция

```
COVAR_POP(<expr1>, <expr2>)
```

эквивалентна

```
(SUM(<expr1> * <expr2>) - SUM(<expr1>) * SUM(<expr2>) / COUNT(*))
  / COUNT(*)
```

В случае если выборка записей пустая или содержит только значения NULL, результат будет содержать NULL.

Примеры:

Пример 8.63. Использование функции COVAR_POP

```
SELECT
    COVAR_POP(alength, aheight) AS c_corr
FROM measure
```

См. также: COVAR_SAMP, SUM, COUNT.

COVAR_SAMP

Доступно в: DSQL.

Синтаксис:

```
COVAR_SAMP(<expr1>, <expr2>)
  [OVER ([<partition_exp>] [<order_exp>])]
```

Параметры функции COVAR_SAMP

expr1
: Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

expr2
: Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Тип возвращаемого результата: DOUBLE PRECISION.

Функция COVAR_SAMP возвращает выборочную ковариацию (sample covariance) пары выражений с числовыми значениями.

Функция

```
COVAR_SAMP(<expr1>, <expr2>)
```

эквивалентна

```
(SUM(<expr1> * <expr2>) - SUM(<expr1>) * SUM(<expr2>) / COUNT(*))
  / (COUNT(*) - 1)
```

В случае если выборка записей пустая, содержит только 1 запись или содержит только значения NULL, результат будет содержать NULL.

Примеры:

Пример 8.64. Использование функции COVAR_SAMP

```
SELECT
    COVAR_SAMP(alength, aheight) AS c_corr
```

```
FROM measure
```

См. также: COVAR_POP, SUM, COUNT.

STDDEV_POP

Доступно в: DSQL.

Синтаксис:

```
STDDEV_POP(<expr>)
  [OVER ([<partition_exp>] [<order_exp>])]
```

Параметры функции STDDEV_POP

expr
: Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Тип возвращаемого результата: DOUBLE PRECISION или NUMERIC в зависимости от типа *expr*.

Функция STDDEV_POP возвращает среднеквадратичное отклонение для группы. Значения NULL пропускаются.

Функция

```
STDDEV_POP(<expr>)
```

эквивалентна

```
SQRT(VAR_POP(<expr>))
```

В случае если выборка записей пустая или содержит только значения NULL, результат будет содержать NULL.

Примеры:

Пример 8.65. Использование функции STDDEV_POP

```
SELECT
  dept_no,
  STDDEV_POP(salary)
FROM employee
GROUP BY dept_no
```

См. также: STDDEV_SAMP, VAR_POP.

STDDEV_SAMP

Доступно в: DSQL.

Синтаксис:

```
STDDEV_SAMP(<expr>)
  [OVER ([<partition_exp>] [<order_exp>])]
```

Параметры функции STDDEV_SAMP

expr
: Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Тип возвращаемого результата: DOUBLE PRECISION или NUMERIC в зависимости от типа *expr*.

Функция STDDEV_SAMP возвращает стандартное отклонение для группы. Значения NULL пропускаются.

Функция

```
STDDEV_SAMP(<expr>)
```

эквивалентна

```
SQRT(VAR_SAMP(<expr>))
```

В случае если выборка записей пустая, содержит только 1 запись или содержит только значения NULL, результат будет содержать NULL.

Примеры:

Пример 8.66. Использование функции STDDEV_SAMP

```
SELECT
  dept_no,
  STDDEV_SAMP(salary)
FROM employee
GROUP BY dept_no
```

См. также: STDDEV_POP, VAR_SAMP.

VAR_POP

Доступно в: DSQL.

Синтаксис:

```
VAR_POP(<expr>)
  [OVER ([<partition_exp>] [<order_exp>])]
```

Параметры функции VAR_POP

expr
Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Тип возвращаемого результата: DOUBLE PRECISION или NUMERIC в зависимости от типа *expr*.

Функция VAR_POP возвращает выборочную дисперсию для группы. Значения NULL пропускаются.

Функция

```
VAR_POP(<expr>)
```

эквивалентна

```
(SUM(<expr> * <expr>) - SUM(<expr>) * SUM(<expr>) / COUNT(<expr>))
  / COUNT(<expr>)
```

В случае если выборка записей пустая или содержит только значения NULL, результат будет содержать NULL.

Примеры:

Пример 8.67. Использование функции VAR_POP

```
SELECT
  dept_no,
  VAR_POP(salary)
FROM employee
GROUP BY dept_no
```

См. также: VAR_SAMP, SUM, COUNT.

VAR_SAMP

Доступно в: DSQL.

Синтаксис:

```
VAR_SAMP(<expr>)
  [OVER ([<partition_exp>] [<order_exp>])]
```

Параметры функции VAR_SAMP

expr
Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Тип возвращаемого результата: DOUBLE PRECISION или NUMERIC в зависимости от типа *expr*.

Функция VAR_SAMP возвращает несмещённую выборочную дисперсию для группы. Значения NULL пропускаются.

Функция

```
VAR_SAMP(<expr>)
```

эквивалентна

```
(SUM(<expr> * <expr>) - SUM(<expr>) * SUM(<expr>) / COUNT(<expr>))
  / (COUNT(<expr>) - 1)
```

В случае если выборка записей пустая, содержит только 1 запись или содержит только значения NULL, результат будет содержать NULL.

Примеры:

Пример 8.68. Использование функции VAR_SAMP

```
SELECT
  dept_no,
  VAR_SAMP(salary)
FROM employee
GROUP BY dept_no
```

См. также: VAR_POP, SUM, COUNT.

Функции линейной регрессии

Функции линейной регрессии полезны для продолжения линии тренда. Линия тренда – это, как правило, закономерность, которой придерживается набор значений. Линия тренда полезна для прогнозирования будущих значений. Этот означает, что тренд будет продолжаться и в будущем. Для продолжения линии тренда необходимо знать угол наклона и точку пересечения с осью Y. Набор линейных функций включает функции для вычисления этих значений.

В синтаксисе функций, *y* интерпретируется в качестве переменной, зависящей от *x*.

Любую функции линейной регрессии из статистических функций можно использовать в качестве оконной. Подробнее см. OVER.

REGR_AVGX

Доступно в: DSQL.

Синтаксис:

```
REGR_AVGX(y, x)
  [OVER ([<partition_exp>] [<order_exp>])]
```

Параметры функции REGR_AVGX

y
Зависимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

x
Независимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Тип возвращаемого результата: DOUBLE PRECISION.

Функция REGR_AVGX вычисляет среднее независимой переменной линии регрессии.

Функция

```
REGR_AVGX(y, x)
```

эквивалентна

```
SUM(<X>) / REGR_COUNT(y, x)

<X> ::= CASE WHEN x IS NOT NULL AND y IS NOT NULL THEN x END
```

См. также: REGR_COUNT, REGR_AVGY.

REGR_AVGY

Доступно в: DSQL.

Синтаксис:

```
REGR_AVGY(y, x)
  [OVER ([<partition_exp>] [<order_exp>])]
```

Параметры функции REGR_AVGY

y
Зависимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

x
Независимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Тип возвращаемого результата: DOUBLE PRECISION.

Функция REGR_AVGY вычисляет среднее зависимой переменной линии регрессии.

Функция

```
REGR_AVGY(y, x)
```

эквивалентна

```
SUM(<Y>) / REGR_COUNT(y, x)

<Y> ::= CASE WHEN x IS NOT NULL AND y IS NOT NULL THEN y END
```

См. также: REGR_COUNT, REGR_AVGX.

REGR_COUNT

Доступно в: DSQL.

Синтаксис:

```
REGR_COUNT(y, x)
  [OVER ([<partition_exp>] [<order_exp>])]
```

Параметры функции REGR_COUNT

y
: Зависимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

x
: Независимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Тип возвращаемого результата: BIGINT.

Функция REGR_COUNT возвращает количество непустых пар, используемых для создания линии регрессии.

Функция

```
REGR_COUNT(y, x)
```

эквивалентна

```
SUM(CASE WHEN x IS NOT NULL AND y IS NOT NULL THEN 1 END)
```

REGR_INTERCEPT

Доступно в: DSQL.

Синтаксис:

```
REGR_INTERCEPT(y, x)
  [OVER ([<partition_exp>] [<order_exp>])]
```

Параметры функции REGR_INTERCEPT

y
Зависимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

x
Независимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Тип возвращаемого результата: DOUBLE PRECISION.

Функция REGR_INTERCEPT вычисляет точку пересечения линии регрессии с осью Y.

Функция

```
REGR_INTERCEPT(y, x)
```

эквивалентна

```
REGR_AVGY(y, x) - REGR_SLOPE(y, x) * REGR_AVGX(y, x)
```

См. также: REGR_AVGY, REGR_AVGX, REGR_SLOPE.

REGR_R2

Доступно в: DSQL.

Синтаксис:

```
REGR_R2(y, x)
  [OVER ([<partition_exp>] [<order_exp>])]
```

Параметры функции REGR_R2

y
Зависимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

x
Независимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Тип возвращаемого результата: DOUBLE PRECISION.

Функция REGR_R2 вычисляет коэффициент детерминации, или R-квадрат, линии регрессии.

Функция

```
REGR_R2(y, x)
```

эквивалентна

```
POWER(CORR(y, x), 2)
```

См. также: CORR.

REGR_SLOPE

Доступно в: DSQL.

Синтаксис:

```
REGR_SLOPE(y, x)
  [OVER ([<partition_exp>] [<order_exp>])]
```

Параметры функции REGR_SLOPE

y
: Зависимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

x
: Независимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Тип возвращаемого результата: DOUBLE PRECISION.

Функция REGR_SLOPE вычисляет угол наклона линии регрессии.

Функция

```
REGR_SLOPE(y, x)
```

эквивалентна

```
COVAR_POP(y, x) / VAR_POP(<X>)

<X> ::= CASE WHEN x IS NOT NULL AND y IS NOT NULL THEN x END
```

См. также: COVAR_POP.

REGR_SXX

Доступно в: DSQL.

Синтаксис:

```
REGR_SXX(y, x)
  [OVER ([<partition_exp>] [<order_exp>])]
```

Параметры функции REGR_SXX

y
Зависимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

x
Независимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Тип возвращаемого результата: DOUBLE PRECISION.

Диагностическая статистика, используемая для анализа регрессии. Вычисляется следующим образом:

```
REGR_COUNT(y, x) * VAR_POP(<X>)

<X> ::= CASE WHEN x IS NOT NULL AND y IS NOT NULL THEN x END
```

См. также: REGR_COUNT, VAR_POP.

REGR_SXY

Доступно в: DSQL.

Синтаксис:

```
REGR_SXY(y, x)
  [OVER ([<partition_exp>] [<order_exp>])]
```

Параметры функции REGR_SXY

y
Зависимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

x
Независимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Тип возвращаемого результата: DOUBLE PRECISION.

Диагностическая статистика, используемая для анализа регрессии. Вычисляется следующим образом:

```
REGR_COUNT(y, x) * COVAR_POP(y, x)
```

См. также: REGR_COUNT, COVAR_POP.

REGR_SYY

Доступно в: DSQL.

Синтаксис:

```
REGR_SYY(y, x)
  [OVER ([<partition_exp>] [<order_exp>])]
```

Параметры функции REGR_SYY

y
Зависимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

x
Независимая переменная линии регрессии. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF, которая возвращает числовой тип данных. Агрегатные функции в качестве выражения не допускаются.

Тип возвращаемого результата: DOUBLE PRECISION.

Диагностическая статистика, используемая для анализа регрессии. Вычисляется следующим образом:

```
REGR_COUNT(y, x) * VAR_POP(<Y>)

<Y> ::= CASE WHEN x IS NOT NULL AND y IS NOT NULL THEN y END
```

См. также: REGR_COUNT, VAR_POP.

Оконные (аналитические) функции

Согласно SQL спецификации оконные функции (также известные как аналитические функции) являются своего рода агрегатными функциями, не уменьшающими степень детализации. При этом агрегированные данные выводятся вместе с неагрегированными.

Синтаксически вызов оконной функции есть указание её имени, за которым всегда следует ключевое слово OVER() с возможными аргументами внутри скобок. В этом и заключается её

синтаксическое отличие от обычной функции или агрегатной функции. Оконные функции могут находиться только в списке SELECT и предложении ORDER BY.

Предложение OVER может содержать указания выполнить действия с разбивкой по группам ("секционирование") и сортировку.

Доступно в: DSQL.

Синтаксис:

```
<window function> ::=
  <window function name>([<expr> [, <expr> ...]])
    OVER ([<partition_exp>] [<order_exp>])

<partition_exp> ::= PARTITION BY <expr> [, <expr> ...]

<order_exp> ::=
  ORDER BY <expr> [<direction>] [<nulls placement>]
         [, <expr> [<direction>] [<nulls placement>] ...]

<direction> ::= {ASC | DESC}

<nulls placement> ::= NULLS {FIRST | LAST}

<window function name> ::=
     <aggregate function>
   | <ranking function>
   | <navigation function>

<ranking function> ::= RANK | DENSE_RANK | ROW_NUMBER

<navigation function> ::= LEAD | LAG | FIRST_VALUE | LAST_VALUE | NTH_VALUE
```

Параметры оконных функций

expr
: Выражение. Может содержать столбец таблицы, константу, переменную, выражение, скалярную или агрегатную функцию. Оконные функции в качестве выражения не допускаются.

partition_exp
: Выражение секционирования.

order_exp
: Выражение сортировки.

direction
: Наравление сортировки.

nulls placement
: Положение псевдозначения NULL в отсортированном наборе.

aggregate function
: Агрегатная функция.

ranking function
: Ранжирующая функция.

navigation function
: Навигационная функция.

Агрегатные функции

Все агрегатные функции могут быть использованы в качестве оконных функций, при добавлении предложения OVER.

Допустим, у нас есть таблица EMPLOYEE со столбцами ID, NAME и SALARY. Нам необходимо показать для каждого сотрудника, соответствующую ему заработную плату и процент от фонда заработной платы.

Простым запросом это решается следующим образом:

```
select
    id,
    department,
    salary,
    salary / (select sum(salary) from employee) percentage
from employee
order by id;
```

Результат:

```
id department salary percentage
-- ---------- ------ ----------
1  R & D       10.00     0.2040
2  SALES       12.00     0.2448
3  SALES        8.00     0.1632
4  R & D        9.00     0.1836
5  R & D       10.00     0.2040
```

Запрос повторяется и может работать довольно долго, особенно если EMPLOYEE является сложным представлением.

Этот запрос может быть переписан в более быстрой и элегантной форме с использованием оконных функций:

```
select
  id,
  department,
  salary,
  salary / sum(salary) OVER () percentage
from employee
order by id;
```

Здесь **sum(salary) OVER ()** вычисляет сумму всех зарплат из запроса (таблицы сотрудников).

Секционирование

Как и для агрегатных функций, которые могут работать отдельно или по отношению к группе, оконные функции тоже могут работать для групп, которые называются "секциями" (partition).

Синтаксис:

```
<window function>(...) OVER (PARTITION BY <expr> [, <expr> ...])
```

Для каждой строки, оконная функция обсчитывает только строки, которые попадают в то же самую секцию, что и текущая строка.

Агрегирование над группой может давать более одной строки, таким образом, к результирующему набору, созданному секционированием, присоединяются результаты из основного запроса, используя тот же список выражений, что и для секции.

Продолжая пример с сотрудниками, вместо того чтобы считать процент зарплаты каждого сотрудника от суммарной зарплаты сотрудников, посчитаем процент от суммарной зарплаты сотрудников того же отдела:

```
select
  id,
  department,
  salary,
  salary / sum(salary) OVER (PARTITION BY department) percentage
from employee
order by id;
```

Результат:

```
id department salary percentage
-- ---------- ------ ----------
1  R & D       10.00     0.3448
2  SALES       12.00     0.6000
3  SALES        8.00     0.4000
4  R & D        9.00     0.3103
5  R & D       10.00     0.3448
```

Сортировка

Предложение ORDER BY может быть использовано с секционированием или без него. Предложение ORDER BY внутри OVER задаёт порядок, в котором оконная функция будет обрабатывать строки. Этот порядок не обязан совпадать с порядком вывода строк. Для стандартных агрегатных функций, предложение ORDER BY заставляет возвращать частичные результаты агрегации по мере обработки записей.

Пример:

```
select
  id,
  salary,
sum(salary) over (order by salary) cumul_salary
from employee
order by salary;
```

Результат:

```
id salary cumul_salary
-- ------ ------------
3    8.00         8.00
4    9.00        17.00
1   10.00        37.00
5   10.00        37.00
2   12.00        49.00
```

В этом случае cumul_salary возвращает частичную/накопительную агрегацию (функции SUM). Может показаться странным, что значение 37,00 повторяется для идентификаторов 1 и 5, но так и должно быть. Сортировка (ORDER BY) ключей группирует их вместе, и агрегат вычисляется единожды (но суммируя сразу два значения 10,00). Чтобы избежать этого, вы можете добавить поле ID в конце предложения ORDER BY.

Вы можете использовать несколько окон с различными сортировками, и дополнять предложение ORDER BY опциями ASC/DESC и NULLS FIRST/LAST.

С секциями предложение ORDER BY работает таким же образом, но на границе каждой секции агрегаты сбрасываются.

Все агрегатные функции могут использовать предложение ORDER BY, за исключением LIST().

Ранжирующие функции

Ранжирующие функции вычисляют порядковый номер ранга внутри секции окна.

Эти функции могут применяться с использованием секционирования и сортировки и без них. Однако их использование без сортировки почти никогда не имеет смысла.

Функции ранжирования могут быть использованы для создания различных типов инкрементных счётчиков. Рассмотрим SUM(1) OVER (ORDER BY SALARY) в качестве примера того, что они могут делать, каждая из них различным образом. Ниже приведён пример запроса, который позволяет сравнить их поведение по сравнению с SUM.

Пример:

```
SELECT
  id,
  salary,
  DENSE_RANK() OVER (ORDER BY salary),
  RANK() OVER (ORDER BY salary),
  ROW_NUMBER() OVER (ORDER BY salary),
  SUM(1) OVER (ORDER BY salary)
```

```
FROM employee
ORDER BY salary;
```

Результат:

```
id salary dense_rank rank row_number sum
-- ------ ---------- ---- ---------- ---
3    8.00          1    1          1   1
4    9.00          2    2          2   2
1   10.00          3    3          3   4
5   10.00          3    3          4   4
2   12.00          4    5          5   5
```

DENSE_RANK

Доступно в: DSQL.

Синтаксис:

```
DENSE_RANK() OVER ([<partition_exp>] [<order_exp>])
```

Тип возвращаемого результата: BIGINT

Возвращает ранг строк в секции результирующего набора без промежутков в ранжировании. Строки с одинаковыми значениями *<order_exp>* получают одинаковый ранг в пределах группы *<partition_exp>*, если она указана. Ранг строки равен количеству различных значений рангов в секции, предшествующих текущей строке, увеличенному на единицу.

Пример:

```
SELECT
  id,
  salary,
  DENSE_RANK() OVER (ORDER BY salary)
FROM employee
ORDER BY salary;
```

Результат:

```
id salary dense_rank
-- ------ ----------
3    8.00          1
4    9.00          2
1   10.00          3
5   10.00          3
2   12.00          4
```

См. также: SELECT, PARTITION BY, ORDER BY, RANK, ROW_NUMBER.

RANK

Доступно в: DSQL.

Синтаксис:

```
RANK() OVER ([<partition_exp>] [<order_exp>])
```

Тип возвращаемого результата: BIGINT

Возвращает ранг каждой строки в секции результирующего набора. Строки с одинаковыми значениями *<order_exp>* получают одинаковый ранг в пределах группы *<partition_exp>*, если она указана. Ранг строки вычисляется как единица плюс количество рангов, находящихся до этой строки.

Пример:

```
SELECT
  id,
  salary,
  RANK() OVER (ORDER BY salary)
FROM employee
ORDER BY salary;
```

Результат:

```
id salary rank
-- ------ ------
3    8.00      1
4    9.00      2
1   10.00      3
5   10.00      3
2   12.00      5
```

См. также: SELECT, PARTITION BY, ORDER BY, DENSE_RANK, ROW_NUMBER.

ROW_NUMBER

Доступно в: DSQL.

Синтаксис:

```
ROW_NUMBER() OVER ([<partition_exp>] [<order_exp>])
```

Тип возвращаемого результата: BIGINT

Возвращает последовательный номер строки в секции результирующего набора, где 1 соответствует первой строке в каждой из секций.

Пример:

```
SELECT
  id,
  salary,
  ROW_NUMBER() OVER (ORDER BY salary)
FROM employee
ORDER BY salary;
```

Результат:

```
id salary row_number
-- ------ ----------
3    8.00          1
4    9.00          2
1   10.00          3
5   10.00          4
2   12.00          5
```

См. также: SELECT, PARTITION BY, ORDER BY, RANK, DENSE_RANK.

Навигационные функции

Навигационные функции получают простые (не агрегированные) значения выражения из другой строки запроса в той же секции.

Важно

Функции FIRST_VALUE, LAST_VALUE и NTH_VALUE оперируют на кадрах окна. В настоящее время в Firebird кадры всегда определены с первой до текущей строки, но не последней, т.е.

```
ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
```

Из-за этого результаты функций NTH_VALUE и в особенности LAST_VALUE могут показаться странными.

Речь идёт о предложении кадрирования (фрейма). Для каждой строки, внутри её разбиения, существует список строк, который называется фрейм окна. Многие (но не все) оконные функции работают только со строками фрейма окна, а не со всем разбиением. По умолчанию, если задано предложение ORDER BY, то фрейм состоит из всех строк, от начала разбиения до текущей строки, плюс любые следующие строки, которые равны текущей строке в соответствии с предложением ORDER BY. Когда ORDER BY опускается, по умолчанию фрейм состоит из всех строк в разбиении.

В настоящее время предложение фрейма не реализовано см. CORE-3647.

Пример 8.69. Навигационные функции

```
SELECT
```

```
  id,
  salary,
  FIRST_VALUE(salary) OVER (ORDER BY salary),
  LAST_VALUE(salary) OVER (ORDER BY salary),
  NTH_VALUE(salary, 2) OVER (ORDER BY salary),
  LAG(salary) OVER (ORDER BY salary),
  LEAD(salary) OVER (ORDER BY salary)
FROM employee
ORDER BY salary;
```

Результат:

```
id salary first_value last_value nth_value    lag   lead
-- ------ ----------- ---------- ---------- ------ ------
3    8.00        8.00       8.00    <null> <null>   9.00
4    9.00        8.00       9.00      9.00   8.00  10.00
1   10.00        8.00      10.00      9.00   9.00  10.00
5   10.00        8.00      10.00      9.00  10.00  12.00
2   12.00        8.00      12.00      9.00  10.00 <null>
```

FIRST_VALUE

Доступно в: DSQL.

Синтаксис:

```
FIRST_VALUE(<expr>) OVER ([<partition_exp>] [<order_exp>])
```

Параметры оператора FIRST_VALUE

expr
: Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF. Агрегатные функции в качестве выражения не допускаются.

Тип возвращаемого результата: тот же что и аргумент функции *expr*.

Возвращает первое значение из упорядоченного набора значений.

См. также: SELECT, PARTITION BY, ORDER BY, LAST_VALUE, NTH_VALUE.

LAG

Доступно в: DSQL.

Синтаксис:

```
LAG(<expr> [, <offset> [, <default>]])
  OVER([<partition_exp>] [<order_exp>])
```

Параметры оператора LAG

expr
Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF. Агрегатные функции в качестве выражения не допускаются.

offset
Количество строк до строки перед текущей строкой, из которой необходимо получить значение. Если значение аргумента не указано, то по умолчанию принимается 1. *offset* может быть столбцом, вложенным запросом или другим выражением, с помощью которого вычисляется целая положительная величина, или другим типом, который может быть неявно преобразован в bigint. *offset* не может быть отрицательным значением или аналитической функцией.

default
Значение по умолчанию, которое возвращается, в случае если смещение (*offset*) указывает за пределы секции. По умолчанию равно NULL

Тип возвращаемого результата: тот же что и аргумент функции *expr*.

Функция LAG обеспечивает доступ к строке с заданным физическим смещением (*offset*) перед началом текущей строки.

Если смещение (*offset*) указывает за пределы секции, то будет возвращено значение *default*, которое по умолчанию равно NULL.

Примеры:

Пример 8.70. Использование функции LAG

Предположим у вас есть таблица rate, которая хранит курс валюты на каждый день. Необходимо проследить динамику изменения курса за последние пять дней.

```
SELECT
  bydate,
  cost,
  cost - LAG(cost) OVER(ORDER BY bydate) AS change,
  100 * (cost - LAG(cost) OVER(ORDER BY bydate)) /
    LAG(cost) OVER(ORDER BY bydate) AS percent_change
FROM rate
WHERE bydate BETWEEN DATEADD(-4 DAY TO current_date)
  AND current_date
ORDER BY bydate
```

Результат:

```
bydate        cost    change   percent_change
----------   -----   -------   --------------
27.10.2014   31.00    <null>           <null>
28.10.2014   31.53      0.53           1.7096
29.10.2014   31.40     -0.13          -0.4123
30.10.2014   31.67      0.27           0.8598
31.10.2014   32.00      0.33           1.0419
```

См. также: SELECT, PARTITION BY, ORDER BY, LEAD.

LAST_VALUE

Доступно в: DSQL.

Синтаксис:

```
LAST_VALUE(<expr>) OVER([<partition_exp>] [<order_exp>])
```

Параметры оператора LAST_VALUE

expr
: Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF. Агрегатные функции в качестве выражения не допускаются.

Тип возвращаемого результата: тот же что и аргумент функции *expr*.

Возвращает последнее значение из упорядоченного набора значений.

См. также: SELECT, PARTITION BY, ORDER BY, FIRST_VALUE, NTH_VALUE.

LEAD

Доступно в: DSQL.

Синтаксис:

```
LEAD(<expr> [, <offset> [, <default>]])
  OVER ([<partition_exp>] [<order_exp>])
```

Параметры оператора LEAD

expr
: Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF. Агрегатные функции в качестве выражения не допускаются.

offset
: Количество строк после текущей строки до строки, из которой необходимо получить значение. Если значение аргумента не указано, то по умолчанию принимается 1. *offset* может быть столбцом, вложенным запросом или другим выражением, с помощью которого вычисляется целая положительная величина, или другим типом, который может быть неявно преобразован в bigint. *offset* не может быть отрицательным значением или аналитической функцией.

default
: Значение по умолчанию, которое возвращается, в случае если смещение (*offset*) указывает за пределы секции. По умолчанию равно NULL.

Тип возвращаемого результата: тот же что и аргумент функции *expr*.

Функция LEAD обеспечивает доступ к строке на заданном физическом смещении (*offset*) после текущей строки.

Если смещение (*offset*) указывает за пределы секции, то будет возвращено значение *default*, которое по умолчанию равно NULL.

См. также: SELECT, PARTITION BY, ORDER BY, LAG.

NTH_VALUE

Доступно в: DSQL.

Синтаксис:

```
NTH_VALUE(<expr> [, <offset>]) [FROM FIRST | FROM LAST]
  OVER ([<partition_exp>] [<order_exp>])
```

Параметры оператора NTH_VALUE

expr
: Выражение. Может содержать столбец таблицы, константу, переменную, выражение, неагрегатную функцию или UDF. Агрегатные функции в качестве выражения не допускаются.

offset
: Номер записи, начиная с первой (опция FROM FIRST) или последней (опция FROM LAST) записи.

Тип возвращаемого результата: тот же что и аргумент функции *expr*.

Функция NTH_VALUE возвращает N-ое значение, начиная с первой (опция FROM FIRST) или последней (опция FROM LAST) записи. По умолчанию используется опция FROM FIRST. Смещение 1 от первой записи будет эквивалентно функции FIRST_VALUE, смещение 1 от последней записи будет эквивалентно функции LAST_VALUE.

См. также: SELECT, PARTITION BY, ORDER BY, FIRST_VALUE, LAST_VALUE.

Агрегатные функции внутри оконных

В качестве аргументов оконных функций, а также в предложении OVER разрешено использование агрегатных функций (но не оконных). В этом случае сначала вычисляются агрегатные функции, а только затем на них накладываются окна оконных функций.

> **Примечание**
>
> При использовании агрегатных функции в качестве аргументов оконных функций, все столбцы, не используемые в агрегатных функциях должны быть указаны в предложении GROUP BY.

Пример 8.71. Использование агрегатной функции в качестве аргумента оконной

```
SELECT
    code_employee_group,
    AVG(salary) AS avg_salary,
    RANK() OVER(ORDER BY AVG(salary)) AS salary_rank
FROM employee
GROUP BY code_employee_group
```

Глава 9

Управление транзакциями

Всё в Firebird выполняется в рамках транзакций. Транзакция — логическая единица изолированной работы группы последовательных операций над базой данных. Изменения над данными остаются обратимыми до тех пор, пока клиентское приложение не выдаст серверу инструкцию COMMIT.

Операторы управления транзакциями

Firebird имеет небольшое количество SQL операторов, которые могут использоваться клиентскими приложениями для старта, управления, подтверждения или отмены транзакций, но достаточное для всех задач над базой данных:

- SET TRANSACTION — задание параметров транзакции и её старт;
- COMMIT — завершение транзакции и сохранение изменений;
- ROLLBACK — отмена изменений произошедший в рамках транзакции;
- SAVEPOINT — установка точки сохранения для частичного отката изменений, если это необходимо;
- RELEASE SAVEPOINT — удаление точки сохранения.

SET TRANSACTION

Назначение: Задаёт параметры транзакции и стартует её.

Доступно в: DSQL, ESQL.

Синтаксис:

```
SET TRANSACTION
  [NAME tr_name]
  [READ WRITE | READ ONLY]
  [[ISOLATION LEVEL] {
       SNAPSHOT [TABLE STABILITY]
     | READ COMMITTED [[NO] RECORD_VERSION] }]
  [NO WAIT | WAIT [LOCK TIMEOUT seconds]]
  [NO AUTO UNDO]
  [IGNORE LIMBO]
  [RESERVING <tables> | USING <dbhandles>]

<tables> ::= <table_spec> [, <table_spec> ...]

<table_spec> ::= tablename [, tablename ...]
  [FOR [SHARED | PROTECTED] {READ | WRITE}]
```

```
<dbhandles> ::= dbhandle [, dbhandle ...]
```

Параметры оператора SET TRANSACTION

tr_name
Имя транзакции. Доступно только в ESQL.

seconds
Время ожидания оператора (statement) в секундах при возникновении конфликта.

tables
Список таблиц для резервирования.

dbhandles
Список баз данных, к которым база данных может получить доступ. Доступно только в ESQL.

table_spec
Спецификация резервирования таблицы.

tablename
Имя таблицы для резервирования.

dbhandle
Хендл базы данных, к которой транзакция может получить доступ. Доступно только в ESQL.

Оператор SET TRANSACTION задаёт параметры транзакции и стартует её. Старт транзакции осуществляется только клиентскими приложениями, но не сервером (за исключением автономных транзакций и некоторых фоновых системных потоков/процессов, например, таких как sweep).

Каждое клиентское приложение может запускать произвольное количество одновременно выполняющихся транзакций. Фактически есть ограничение на общее количество выполняемых транзакций во всех клиентских приложениях, работающих с одной конкретной базой данных с момента последнего восстановления базы данных с резервной копии или с момента первоначального создания базы данных. Это количество равняется числу $2^{32} – 1$, то есть 4 294 967 295.

Все предложения в операторе SET TRANSACTION являются необязательными. Если в операторе запуска транзакции на выполнение не задано никакого предложения, то предполагается старт транзакции со значениями всех характеристик по умолчанию (режим доступа, режим разрешения блокировок и уровень изолированности).

По умолчанию транзакция стартует со следующими характеристиками.

```
SET TRANSACTION
READ WRITE
WAIT ISOLATION LEVEL SNAPSHOT;
```

При старте со стороны клиента любой транзакции (заданной явно или по умолчанию) сервер передаёт клиенту дескриптор транзакции (целое число). На стороне сервера транзакциям последовательно присваиваются номера. Этот номер средствами SQL можно получить, используя контекстную переменную CURRENT_TRANSACTION.

Параметры транзакции

Основными характеристиками транзакции являются:

- режим доступа к данным (READ WRITE, READ ONLY);
- режим разрешения блокировок (WAIT, NO WAIT) с возможным дополнительным уточнением LOCK TIMEOUT;
- уровень изоляции (READ COMMITTED, SNAPSHOT, TABLE STABILITY);
- средства резервирования или освобождения таблиц (предложение RESERVING).

Имя транзакции

Необязательное предложение NAME задаёт имя транзакции. Предложение NAME доступно только в Embedded SQL. Если предложение NAME не указано, то оператор SET TRANSACTION применяется к транзакции по умолчанию. За счёт именованных транзакций позволяется одновременный запуск нескольких активных транзакций в одном приложении. При этом должна быть объявлена и инициализирована одноименная переменная базового языка. В DSQL, это ограничение предотвращает динамическую спецификацию имён транзакций.

Режим доступа

Для транзакций существует два режима доступа к данным базы данных: READ WRITE и READ ONLY.

- При режиме доступа READ WRITE операции в контексте данной транзакции могут быть как операциями чтения, так и операциями изменения данных. Это режим по умолчанию.
- В режиме READ ONLY в контексте данной транзакции могут выполняться только операции выборки данных SELECT. Любая попытка изменения данных в контексте такой транзакции приведёт к исключениям базы данных. Однако это не относиться к глобальным временным таблицам (GTT), которые разрешено модифицировать в READ ONLY транзакциях.

Режим разрешения блокировок

При работе с одной и той же базой данных нескольких клиентских приложений могут возникать блокировки. Блокировки могут возникать, когда одна транзакция вносит неподтверждённые изменения в строку таблицы или удаляет строку, а другая транзакция пытается изменять или удалять эту же строку. Такие блокировки называются конфликтом обновления.

Блокировки также могут возникнуть и в других ситуациях при использовании некоторых уровней изоляции транзакций.

Существуют два режима разрешения блокировок: WAIT и NO WAIT.

Режим WAIT

В режиме WAIT (режим по умолчанию) при появлении конфликта с параллельными транзакциями, выполняющими конкурирующие обновления данных в той же базе данных, такая транзакция будет ожидать завершения конкурирующей транзакции путём её подтверждения (COMMIT) или отката (ROLLBACK). Иными словами, клиентское приложение будет переведено в режим ожидания до момента разрешения конфликта.

Если для режима WAIT задать предложение LOCK TIMEOUT, то ожидание будет продолжаться только указанное в этом предложении количество секунд. По истечении этого срока будет выдано сообщение об ошибке: "Lock time-out on wait transaction" (Истечение времени ожидания блокировки для транзакции WAIT).

Этот режим даёт несколько отличные формы поведения в зависимости от уровня изоляции транзакций.

Режим NO WAIT

Если установлен режим разрешения блокировок NO WAIT, то при появлении конфликта блокировки данная транзакция немедленно вызовет исключение базы данных.

ISOLATION LEVEL

Уровень изолированности транзакций — значение, определяющее уровень, при котором в транзакции допускаются несогласованные данные, то есть степень изолированности одной транзакции от другой. Изменения, внесённые некоторым оператором, будут видны всем последующим операторам, запущенным в рамках этой же транзакции, независимо от её уровня изолированности. Изменения произведённые в рамках другой транзакции остаются невидимыми для текущей транзакции до тех пор пока они не подтверждены. Уровень изолированности, а иногда, другие атрибуты, определяет, как транзакции будут взаимодействовать с другой транзакцией, которая хочет подтвердить изменения.

Необязательное предложение ISOLATION LEVEL задаёт уровень изолированности запускаемой транзакции. Это самая важная характеристика транзакции, которая определяет её поведение по отношению к другим одновременно выполняющимся транзакциям.

Существует три уровня изолированности транзакции:

- SNAPSHOT
- SNAPSHOT TABLE STABILITY
- READ COMMITTED с двумя уточнениями (NO RECORD_VERSION и RECORD_VERSION)

Уровень изолированности SNAPSHOT

Уровень изолированности SNAPSHOT (уровень изолированности по умолчанию) означает, что этой транзакции видны лишь те изменения, фиксация которых произошла не позднее момента старта этой транзакции. Любые подтверждённые изменения, сделанные другими конкурирующими транзакциями, не будут видны в такой транзакции в процессе ее активности без её перезапуска. Чтобы увидеть эти изменения, нужно завершить транзакцию (подтвердить её или выполнить полный откат, но не откат на точку сохранения) и запустить транзакцию заново.

> **Примечание**
>
> Изменения, вносимые автономными транзакциями, также не будут видны в контексте той ("внешней") транзакции, которая запустила эти автономные транзакции, если она работает в режиме SNAPSHOT.

Уровень изолированности SNAPSHOT TABLE STABILITY

Уровень изоляции транзакции SNAPSHOT TABLE STABILITY позволяет, как и в случае SNAPSHOT, также видеть только те изменения, фиксация которых произошла не позднее

момента старта этой транзакции. При этом после старта такой транзакции в других клиентских транзакциях невозможно выполнение изменений ни в каких таблицах этой базы данных, уже каким-либо образом измененных первой транзакцией. Все такие попытки в параллельных транзакциях приведут к исключениям базы данных. Просматривать любые данные другие транзакции могут совершенно свободно.

При помощи предложения резервирования (RESERVING) можно разрешить другим транзакциям изменять данные в некоторых таблицах.

Если на момент старта клиентом транзакции с уровнем изоляции SNAPSHOT TABLE STABILITY какая-нибудь другая транзакция выполнила неподтверждённое изменение данных любой таблицы базы данных, то запуск транзакции с таким уровнем изоляции приведёт к ошибке базы данных.

Уровень изолированности READ COMMITTED

Уровень изолированности READ COMMITTED позволяет в транзакции без её перезапуска видеть все подтверждённые изменения данных базы данных, выполненные в других параллельных транзакциях. Неподтверждённые изменения не видны в транзакции и этого уровня изоляции.

Для получения обновлённого списка строк интересующей таблицы необходимо лишь повторное выполнение оператора SELECT в рамках активной транзакции READ COMMITTED без её перезапуска.

RECORD_VERSION

Для этого уровня изолированности можно указать один из двух значений дополнительной характеристики в зависимости от желаемого способа разрешения конфликтов: RECORD_VERSION и NO RECORD_VERSION. Как видно из их имён они являются взаимоисключающими.

- NO RECORD_VERSION (значение по умолчанию) является в некотором роде механизмом двухфазной блокировки. В этом случае транзакция не может прочитать любую запись, которая была изменена параллельной активной (неподтвержденной) транзакцией.

  Если указана стратегия разрешения блокировок NO WAIT, то будет немедленно выдано соответствующее исключение.

  Если указана стратегия разрешения блокировок WAIT, то это приведёт к ожиданию завершения или откату конкурирующей транзакции. Если конкурирующая транзакция откатывается, или, если она завершается и её идентификатор старее (меньше), чем идентификатор текущей транзакции, то изменения в текущей транзакции допускаются. Если конкурирующая транзакция завершается и её идентификатор новее (больше), чем идентификатор текущей транзакции, то будет выдана ошибка конфликта блокировок.

- При задании RECORD_VERSION транзакция всегда читает последнюю подтверждённую версию записей таблиц, независимо от того, существуют ли изменённые и ещё не подтверждённые версии этих записей. В этом случае режим разрешения блокировок (WAIT или NO WAIT) никак не влияет на поведение транзакции при её старте.

NO AUTO UNDO

При использовании опции NO AUTO UNDO оператор ROLLBACK только помечает транзакцию как отменённую без удаления созданных в этой транзакции версий, которые будут удалены

позднее в соответствии с выбранной политикой сборки мусора (см. параметр *GCPolicy* в `firebird.conf`).

Эта опция может быть полезна при выполнении транзакции, в рамках которой производится много отдельных операторов, изменяющих данные, и при этом есть уверенность, что эта транзакция будет чаще всего завершаться успешно, а не откатываться.

Для транзакций, в рамках которых не выполняется никаких изменений, опция NO AUTO UNDO игнорируется.

IGNORE LIMBO

При указании опции IGNORE LIMBO игнорируются записи, создаваемые "потерянными" (т.е. не завершёнными) транзакциями (limbo transaction). Транзакции считается "потерянной", если не завершён второй этап двухфазного подтверждения (two-phase commit).

RESERVING

Предложение RESERVING в операторе SET TRANSACTION резервирует указанные в списке таблицы. Резервирование запрещает другим транзакциям вносить в эти таблицы изменения или (при определённых установках характеристик предложения резервирования) даже читать данные из этих таблиц, в то время как выполняется данная транзакция. Либо, наоборот, в этом предложении можно указать список таблиц, в которые параллельные транзакции могут вносить изменения, даже если запускается транзакция с уровнем изоляции SNAPSHOT TABLE STABILITY.

В одном предложении резервирования можно указать произвольное количество резервируемых таблиц используемой базы данных.

Если опущено одно из ключевых слов SHARED или PROTECTED, то предполагается SHARED. Если опущено все предложение FOR, то предполагается FOR SHARED READ. Варианты осуществления резервирования таблиц по их названиям не являются очевидными.

Таблица 9.1. Совместимости различных блокировок

	SHARED READ	SHARED WRITE	PROTECTED READ	PROTECTED WRITE
SHARED READ	да	да	да	да
SHARED WRITE	да	да	нет	нет
PROTECTED READ	да	нет	да	нет
PROTECTED WRITE	да	нет	нет	нет

Для транзакции запущенной в режиме изолированности SNAPSHOT для таблиц, указанных в предложении RESERVING, в параллельных транзакциях в зависимости от их уровня изоляции допустимы при различных способах их резервирования следующие варианты поведения:

- SHARED READ — не оказывает никакого влияния на выполнение параллельных транзакций;
- SHARED WRITE — на поведение параллельных транзакций с уровнями изолированности SNAPSHOT и READ COMMITTED не оказывает никакого влияния, для транзакций с уровнем изолированности SNAPSHOT TABLE STABILITY запрещает не только запись, но также и чтение данных из указанных таблиц;

- PROTECTED READ — допускает только чтение данных из резервируемых таблиц для параллельных транзакций с любым уровнем изолированности, попытка внесения изменений приводит к исключению базы данных;
- PROTECTED WRITE — для параллельных транзакций с уровнями изолированности SNAPSHOT и READ COMMITTED запрещает запись в указанные таблицы, для транзакций с уровнем изолированности SNAPSHOT TABLE STABILITY запрещает также и чтение данных из резервируемых таблиц.

Для транзакции запущенной в режиме изолированности SNAPSHOT TABLE STABILITY для таблиц, указанных в предложении RESERVING, в параллельных транзакциях в зависимости от их уровня изолированности допустимы при различных способах их резервирования следующие варианты поведения:

- SHARED READ — позволяет всем параллельным транзакциям независимо от их уровня изолированности не только читать, но и выполнять любые изменения в резервируемых таблицах (если параллельная транзакция имеет режим доступа READ WRITE);
- SHARED WRITE — для всех параллельных транзакций с уровнем доступа READ WRITE и с уровнями изолированности SNAPSHOT и READ COMMITTED позволяет читать данные из таблиц и писать данные в указанные таблицы, для транзакций с уровнем изолированности SNAPSHOT TABLE STABILITY запрещает не только запись, но также и чтение данных из указанных таблиц;
- PROTECTED READ — допускает только лишь чтение данных из резервируемых таблиц для параллельных транзакций с любым уровнем изолированности;
- PROTECTED WRITE — для параллельных транзакций с уровнями изолированности SNAPSHOT и READ COMMITTED запрещает запись в указанные таблицы, для транзакций с уровнем изолированности SNAPSHOT TABLE STABILITY запрещает также и чтение данных из резервируемых таблиц.

Для транзакции запущенной в режиме изолированности READ COMMITTED для таблиц, указанных в предложении RESERVING, в параллельных транзакциях в зависимости от их уровня изоляции допустимы при различных способах их резервирования следующие варианты поведения:

- SHARED READ — позволяет всем параллельным транзакциям независимо от их уровня изолированности не только читать, но и выполнять любые изменения в резервируемых таблицах (при уровне доступа READ WRITE);
- SHARED WRITE — для всех транзакций с уровнем доступа READ WRITE и с уровнями изолированности SNAPSHOT и READ COMMITTED позволяет читать и писать данные в указанные таблицы, для транзакций с уровнем изолированности SNAPSHOT TABLE STABILITY запрещает не только запись, но также и чтение данных из указанных таблиц;
- PROTECTED READ — допускает только чтение данных из резервируемых таблиц для параллельных транзакций с любым уровнем изолированности;
- PROTECTED WRITE — для параллельных транзакций с уровнями изолированности SNAPSHOT и READ COMMITTED разрешает только чтение данных и запрещает запись в указанные в данном списке таблицы, для транзакций с уровнем изолированности SNAPSHOT TABLE STABILITY запрещает не только изменение данных, но и чтение данных из резервируемых таблиц.

> **Подсказка**
>
> Предложение USING может быть использовано для сохранения системных ресурсов за счёт ограничения количества баз данных, к которым имеет доступ транзакция. Доступно только в Embedded SQL.

См. также: COMMIT, ROLLBACK.

COMMIT

Назначение: Подтверждение транзакции.

Доступно в: DSQL, ESQL.

Синтаксис:

```
COMMIT [WORK] [TRANSACTION tr_name]
  [RELEASE] [RETAIN [SNAPSHOT]];
```

Параметры оператора COMMIT

tr_name
Имя транзакции. Доступно только в ESQL.

Оператор COMMIT подтверждает все изменения в данных, выполненные в контексте данной транзакции (добавления, изменения, удаления). Новые версии записей становятся доступными для других транзакций, и если предложение RETAIN не используется освобождаются все ресурсы сервера, связанные с выполнением данной транзакции.

Если в процессе подтверждения транзакции возникли ошибки в базе данных, то транзакция не подтверждается. Пользовательская программа должна обработать ошибочную ситуацию и заново подтвердить транзакцию или выполнить ее откат.

Необязательное предложение TRANSACTION задаёт имя транзакции. Предложение TRANSACTION доступно только в Embedded SQL. Если предложение TRANSACTION не указано, то оператор COMMIT применяется к транзакции по умолчанию.

> **Примечание**
>
> За счёт именованных транзакций позволяется одновременный запуск нескольких активных транзакций в одном приложении. При этом должна быть объявлена и инициализирована одноименная переменная базового языка. В DSQL, это ограничение предотвращает динамическую спецификацию имён транзакций.

Необязательное ключевое слово WORK может быть использовано лишь для совместимости с другими системами управления реляционными базами данных.

Ключевое слово RELEASE доступно только в Embedded SQL. Оно позволяет отключиться ото всех баз данных после завершения текущей транзакции. RELEASE поддерживается только для обратной совместимости со старыми версиями INTERBASE. В настоящее время вместо него используется оператор ESQL DISCONNECT.

Если используется предложение RETAIN [SNAPSHOT], то выполняется так называемое мягкое (soft) подтверждение. Выполненные действия в контексте данной транзакции фиксируются в базе данных, а сама транзакция продолжает оставаться активной, сохраняя свой идентификатор, а также состояние курсоров, которое было до мягкой фиксации транзакции. В этом случае нет необходимости опять стартовать транзакцию и заново выполнять оператор SELECT для получения данных.

Если уровень изоляции такой транзакции SNAPSHOT или SNAPSHOT TABLE STABILITY, то после мягкого подтверждения транзакция продолжает видеть то состояние базы данных, которое было при первоначальном запуске транзакции, то есть клиентская программа не видит новых подтверждённых результатов изменения данных других транзакций. Кроме того, мягкое подтверждение не освобождает ресурсов сервера (открытые курсоры не закрываются).

> **Подсказка**
>
> Для транзакций, которые выполняют только чтение данных из базы данных, рекомендуется также использовать оператор COMMIT, а не ROLLBACK, поскольку этот вариант требует меньшего количества ресурсов сервера и улучшает производительность всех последующих транзакций.

См. также: SET TRANSACTION, ROLLBACK.

ROLLBACK

Назначение: Откат транзакции.

Доступно в: DSQL, ESQL.

Синтаксис:

```
ROLLBACK [WORK] [TRANSACTION tr_name]
  [RETAIN [SNAPSHOT] | TO SAVEPOINT sp_name] [RELEASE];
```

Параметры оператора ROLLBACK

tr_name
: Имя транзакции. Доступно только в ESQL.

sp_name
: Имя точки сохранения. Доступно только в DSQL.

Оператор ROLLBACK отменяет все изменения данных базы данных (добавление, изменение, удаление), выполненные в контексте этой транзакции. Оператор ROLLBACK никогда не вызывает ошибок. Если не указано предложение RETAIN, то при его выполнении освобождаются все ресурсы сервера, связанные с выполнением данной транзакции.

Необязательное предложение TRANSACTION задаёт имя транзакции. Предложение TRANSACTION доступно только в Embedded SQL. Если предложение TRANSACTION не указано, то оператор ROLLBACK применяется к транзакции по умолчанию.

> **Примечание**
>
> За счёт именованных транзакций позволяется одновременный запуск нескольких активных транзакций в одном приложении. При этом должна быть объявлена и инициализирована одноименная переменная базового языка. В DSQL, это ограничение предотвращает динамическую спецификацию имён транзакций.

Необязательное ключевое слово WORK может быть использовано лишь для совместимости с другими системами управления реляционными базами данных.

Ключевое слово RETAIN указывает, что все действия по изменению данных в контексте этой транзакции, отменяются, а сама транзакция продолжает оставаться активной, сохраняя свой идентификатор, а также состояние курсоров, которое было до мягкой фиксации транзакции. Таким образом, выделенные ресурсы для транзакции не освобождаются.

Для уровней изоляции SNAPSHOT и SNAPSHOT TABLE STABILITY состояние базы данных остаётся в том виде, которое база данных имела при первоначальном старте такой транзакции, однако в случае уровня изоляции READ COMMITTED база данных будет иметь вид, соответствующий новому состоянию на момент выполнения оператора ROLLBACK RETAIN. В случае отмены транзакции с сохранением её контекста нет необходимости заново выполнять оператор SELECT для получения данных из таблицы.

См. также: SET TRANSACTION, COMMIT.

ROLLBACK TO SAVEPOINT

Необязательное предложение TO SAVEPOINT в операторе ROLLBACK задаёт имя точки сохранения, на которую происходит откат. В этом случае отменяются все изменения, произошедшие в рамках транзакции, начиная с созданной точки сохранения (SAVEPOINT).

Оператор ROLLBACK TO SAVEPOINT выполняет следующие операции:

- Все изменения в базе данных, выполненные в рамках транзакции начиная с созданной точки сохранения, отменяются. Пользовательские переменные, заданные с помощью функции RDB$SET_CONTEXT() остаются неизменными;
- Все точки сохранения, создаваемые после названной, уничтожаются. Все более ранние точки сохранения, как сама точка сохранения, остаются. Это означает, что можно откатываться к той же точке сохранения несколько раз;
- Все явные и неявные блокированные записи, начиная с точки сохранения, освобождаются. Другие транзакции, запросившие ранее доступ к строкам, заблокированным после точки сохранения, должны продолжать ожидать, пока транзакция не фиксируется или откатывается. Другие транзакции, которые ещё не запрашивали доступ к этим строкам, могут запросить и сразу же получить доступ к разблокированным строкам.

См. также: SAVEPOINT.

SAVEPOINT

Назначение: Создание точки сохранения.

Доступно в: DSQL.

Синтаксис:

```
SAVEPOINT sp_name
```

Параметры оператора SAVEPOINT

sp_name
Имя точки сохранения. Должно быть уникальным в рамках транзакции.

Оператор SAVEPOINT создаёт SQL 99 совместимую точку сохранения, к которой можно позже откатывать работу с базой данных, не отменяя все действия, выполненные с момента

старта транзакции. Механизмы точки сохранения также известны под термином "вложенные транзакции" ("nested transactions").

Если имя точки сохранения уже существует в рамках транзакции, то существующая точка сохранения будет удалена, и создаётся новая с тем же именем.

Для отката изменений к точке сохранения используется оператор ROLLBACK TO SAVEPOINT.

> **Примечание**
>
> Внутренний механизм точек сохранения может использовать большие объёмы памяти, особенно если вы обновляете одни и те же записи многократно в одной транзакции. Если точка сохранения уже не нужна, но вы ещё не готовы закончить транзакцию, то можно ее удалить оператором RELEASE SAVEPOINT, тем самым освобождая ресурсы.

Примеры:

Пример 9.1. DSQL сессия с использованием точек сохранения

```
CREATE TABLE TEST (ID INTEGER);
COMMIT;
INSERT INTO TEST VALUES (1);
COMMIT;
INSERT INTO TEST VALUES (2);
SAVEPOINT Y;
DELETE FROM TEST;
SELECT * FROM TEST; -- возвращает пустую строку
ROLLBACK TO Y;
SELECT * FROM TEST; -- возвращает две строки
ROLLBACK;
SELECT * FROM TEST; -- возвращает одну строку
```

См. также: ROLLBACK TO SAVEPOINT, RELEASE SAVEPOINT.

RELEASE SAVEPOINT

Назначение: Удаление точки сохранения.

Доступно в: DSQL.

Синтаксис:

```
RELEASE SAVEPOINT sp_name [ONLY]
```

Параметры оператора RELEASE SAVEPOINT

sp_name
Имя точки сохранения.

Оператор RELEASE SAVEPOINT удаляет именованную точку сохранения, освобождая все связанные с ней ресурсы. По умолчанию удаляются также все точки сохранения, создаваемые

после указанной. Если указано предложение ONLY, то удаляется только точка сохранения с заданным именем.

См. также: SAVEPOINT.

Внутренние точки сохранения

По умолчанию сервер использует автоматическую системную точку сохранения уровня транзакции для выполнения её отката. При выполнении оператора ROLLBACK, все изменения, выполненные в транзакции, откатываются до системной точки сохранения и после этого транзакция подтверждается.

Когда объем изменений, выполняемых под системной точкой сохранения уровня транзакции, становится большим (затрагивается порядка 50000 записей) сервер освобождает системную точку сохранения и, при необходимости отката транзакции, использует механизм TIP.

> **Подсказка**
>
> Если вы ожидаете, что объем изменений в транзакции будет большим, то можно задать опцию NO AUTO UNDO в операторе SET TRANSACTION, или – если используется API – установить флаг TPB isc_tpb_no_auto_undo. В обеих вариантах предотвращается создание системной точки сохранения уровня транзакции.

Точки сохранения и PSQL

Использование операторов управления транзакциями в PSQL не разрешается, так как это нарушит атомарность оператора, вызывающего процедуру. Но Firebird поддерживает вызов и обработку исключений в PSQL, так, чтобы действия, выполняемые в хранимых процедурах и триггерах, могли быть выборочно отменены без полного отката всех действий в них. Внутренне автоматические точки сохранения используется для:

- отмены всех действий внутри блока BEGIN ... END, где происходит исключение;
- отмены всех действий, выполняемых в хранимой процедуре/триггере (или, в случае селективной хранимой процедуры, всех действий, выполненных с момента последнего оператора SUSPEND), если они завершаются преждевременно из-за непредусмотренной ошибки или исключения.

Каждый блок обработки исключений PSQL также ограничен автоматическими точками сохранения сервера.

> **Примечание**
>
> Сами по себе блок BEGIN..END не создаёт автоматическую точку сохранения. Она создаётся только в блоках, которых присутствует блок WHEN для обработки исключений или ошибок.

Глава 10

Безопасность

Базы данных, как и данные, хранимые в файлах базы данных, должны быть защищены. Firebird обеспечивает двухуровневую защиту данных — аутентификация пользователя на уровне сервера и привилегии на уровне базы данных. В данной главе рассказывается, каким образом управлять безопасностью вашей базы данных на каждом из уровней.

Аутентификация пользователя

Безопасность всей базы данных зависит от проверки подлинности идентификатора пользователя. Подлинность пользователя может выполняться несколькими способами в зависимости от установок параметра *AuthServer* в файле конфигурации `firebird.conf`. Этот параметр содержит список доступных плагинов проверки подлинности. Если проверить подлинность с помощью первого плагина не удалось, то сервер переходит к следующему плагину и т.д. Если ни один плагин не подтвердил подлинность, то пользователь получает сообщение об ошибке.

Информация о пользователях, зарегистрированных для конкретного сервера Firebird, хранится в особой базе данных безопасности (security database) — `security3.fdb`. Для каждой базы данных база данных безопасности может переопределена в файле `databases.conf` (параметр *SecurityDatabase*). Любая база данных может быть базой данных безопасности для самой себя.

Имя пользователя может состоять максимум из 31 символа. Максимальная длина пароля зависит от плагина проверки подлинности и плагина управления пользователями (параметр *UserManager*), регистр — учитывается. По умолчанию будет выбран первый плагин из списка плагинов управления пользователями. Этот плагин можно изменить в SQL командах управления пользователями. Для плагина SRP эффективная длина пароля ограничена 20 байтами *. Для плагина Legacy_UserManager максимальная длина пароля равна 8 байт.

> ***Почему эффективная длина пароля ограничена 20 символами?**
>
> На длину пароля нет ограничения в 20 байт и он может быть использован. Хэши различных паролей, длина которых более 20 байт, тоже различны. Предел эффективности наступает из-за ограниченной длины хэша в SHA1 равном 20 байт или 160 бит. Рано или поздно найдётся более короткий пароль с тем же хэшем с помощью атаки Brute Force. Именно поэтому часто говорят, что эффективная длина пароля для алгоритма SHA1 составляет 20 байт.

Встроенная версия сервера (embedded), не использует аутентификацию. Тем не менее, имя пользователя, и если необходимо роль, должны быть указаны в параметрах подключения, поскольку они используются для контроля доступа к объектам базы данных.

Пользователь SYSDBA или пользователь вошедший с ролью RDB$ADMIN, получают неограниченный доступ к базе данных. Если пользователь является владельцем базы данных, то без указания роли RDB$ADMIN он получает неограниченный доступ ко всем объектам принадлежащим этой базе данных.

Специальные учётные записи

В Firebird существует специальная учётная запись SYSDBA, которая существует вне всех ограничений безопасности и имеет полный доступ ко всем базам данных сервера.

Особенности POSIX

В POSIX системах, включая MacOSX, Firebird будет трактовать пользователя POSIX точно так же как и пользователя, хранящегося в собственной базе данных безопасности Firebird, до тех пор, пока сервер видит клиента в качестве доверенного хоста. Учетные записи пользователя должны существовать как на клиенте, так и на сервере. Для того чтобы установить доверительные отношения с хостом клиента, необходимо на сервере занести соответствующую запись в файл `/etc/hosts.equiv` или `/etc/gds_hosts.equiv`. В файле `hosts.equiv` прописываются доверительные отношения на уровне операционных систем, которые, соответственно, распространяются на все сервисы (например, rlogin, rsh, rcp). В файле `gds_hosts.equiv` устанавливаются доверительные отношения между хостами, только для Firebird. Формат записи идентичен для обоих файлов, и выглядит следующим образом:

```
hostname [username]
```

В POSIX системах пользователь root может выступать в роли SYSDBA. Firebird в этом случае будет трактовать имя пользователя root как SYSDBA, и вы будете иметь доступ ко всем базам данных сервера.

Особенности Windows

В операционных системах семейства Windows NT вы также можете пользоваться учётными записями ОС. Для этого необходимо, чтобы в файле конфигурации `firebird.conf` (параметр *`AuthServer`*) в списке плагинов присутствовал провайдер Win_Sspi. Кроме того, этот плагин должен присутствовать и в списке плагинов клиентской стороны (параметр *`AuthClient`*).

Администраторы операционной системы Windows автоматически не получают права SYSDBA при подключении к базе данных (если, конечно, разрешена доверенная авторизация). Имеют ли администраторы автоматические права SYSDBA, зависит от установки значения флага AUTO ADMIN MAPPING.

> **Примечание**
>
> До Firebird 3.0 при включенной доверительной аутентификации, пользователи прошедшие проверку по умолчанию автоматически отображались в CURRENT_USER. В Firebird 3 и выше отображение должно быть сделано явно для систем с несколькими базами данных безопасности и включенной доверительной аутентификацией. См. CREATE MAPPING.

Операторы управления пользователями

В данном разделе описываются операторы создания, модификации и удаления учётных записей пользователей Firebird средствами операторов SQL. Такая возможность предоставлена следующим пользователям:

- SYSDBA;
- Любому пользователю, имеющему права на роль RDB$ADMIN в базе данных пользователей и права на ту же роль для базы данных в активном подключении (пользователь должен подключаться к базе данных с ролью RDB$ADMIN);
- При включенном флаге AUTO ADMIN MAPPING в базе данных пользователей (`security3.fdb` или той, что установлена для вашей базы данных в файле `databases.conf`) — любой администратор операционной системы Windows (при условии использования сервером доверенной авторизации — trusted authentication) без указания роли. При этом не важно, включен или выключен флаг AUTO ADMIN MAPPING в самой базе данных.

Непривилегированные пользователи могут использовать только оператор ALTER USER для изменения собственной учётной записи.

CREATE USER

Назначение: Создание учётной записи пользователя Firebird.

Доступно в: DSQL.

Синтаксис:

```
CREATE USER username PASSWORD 'password'
  [FIRSTNAME 'firstname']
  [MIDDLENAME 'middlename']
  [LASTNAME 'lastname']
  [ACTIVE | INACTIVE]
  [USING PLUGIN 'pluginname']
  [TAGS (<tag> [, <tag> [, <tag> ...]] )]
  [GRANT ADMIN ROLE];

<tag> ::= tagname = 'string value'
```

Параметры оператора CREATE USER

username
: Имя пользователя. Максимальная длина 31 символ.

password
: Пароль пользователя. Может включать в себя до 32 символов. Чувствительно к регистру.

firstname
: Вспомогательная информация: имя пользователя. Максимальная длина 32 символа.

middlename
: Вспомогательная информация: "второе имя" (отчество, "имя отца") пользователя. Максимальная длина 32 символа.

lastname
: Вспомогательная информация: фамилия пользователя. Максимальная длина 32 символа.

pluginname
: Имя плагина управления пользователями, в котором необходимо создать нового пользователя.

tagname
: Имя пользовательского атрибута. Максимальная длина 31 символ. Имя атрибута должно подчиняться правилам наименования SQL идентификаторов.

string value
: Значение пользовательского атрибута. Максимальная длина 255 символов.

Оператор CREATE USER создаёт учётную запись пользователя Firebird. Пользователь должен отсутствовать в текущей базе данных безопасности Firebird иначе будет выдано соответствующее сообщение об ошибке.

Важно

Начиная с Firebird 3.0 имена пользователей подчиняются общему правилу наименования идентификаторов объектов метаданных. Таким образом, пользователь с именем "Alex" и с именем "ALEX" будут разными пользователями.

```
CREATE USER ALEX PASSWORD 'bz23ds';

-- этот пользователь такой же как и первый
CREATE USER Alex PASSWORD 'bz23ds';

-- этот пользователь такой же как и первый
CREATE USER "ALEX" PASSWORD 'bz23ds';

-- а это уже другой пользователь
CREATE USER "Alex" PASSWORD 'bz23ds';
```

Предложение PASSWORD задаёт пароль пользователя. Максимальная длина пароля зависит от того какой менеджер пользователей задействован (параметр *UserManager*). Для менеджер пользователей SRP эффективная длина пароля ограничена 20 байтами *. Для менеджер пользователей Legacy_UserManager максимальная длина пароля равна 8 байт.

***Почему эффективная длина пароля ограничена 20 символами?**

На длину пароля нет ограничения в 20 байт и он может быть использован. Хэши различных паролей, длина которых более 20 байт, тоже различны. Предел эффективности наступает из-за ограниченной длины хэша в SHA1 равном 20 байт или 160 бит. Рано или поздно найдётся более короткий пароль с тем же хэшем с помощью атаки Brute Force. Именно поэтому часто говорят, что эффективная длина пароля для алгоритма SHA1 составляет 20 байт.

Необязательные предложения FIRSTNAME, MIDDLENAME и LASTNAME задают дополнительные атрибуты пользователя, такие как имя пользователя (имя человека), отчество и фамилия соответственно.

Кроме того вы можете задать неограниченное количество пользовательских атрибутов с помощью необязательного предложения TAGS.

Если при создании учётной записи будет указан атрибут INACTIVE, то пользователь будет создан в "неактивном состоянии", т.е. подключиться с его учётной записью будет невозможно.

При указании атрибута ACTIVE пользователь будет создан в активном состоянии. По умолчанию пользователь создаётся активным.

Если указана опция GRANT ADMIN ROLE, то новая учётная запись пользователя создаётся с правами роли RDB$ADMIN в текущей базе данных безопасности. Это позволяет вновь созданному пользователю управлять учётными записями пользователей, но не даёт ему специальных полномочий в обычных базах данных.

Необязательное предложение USING PLUGIN позволяет явно указывать какой плагин управления пользователями будет использован. По умолчанию используется тот плагин, который был указан первым в списке параметра *UserManager* в файле конфигурации firebird.conf. Допустимыми являются только значения, перечисленные в параметре *UserManager*.

> **Важно:**
>
> Учтите что одноименные пользователи, созданные с помощью разных плагинов управления пользователями — это разные пользователи. Поэтому пользователя созданного с помощью одного плагина управления пользователями можно удалить или изменить, указав только тот же самый плагин.

Для создания учётной записи пользователя текущий пользователь должен обладать административными привилегиями.

Примеры:

Пример 10.1. Создание пользователя.

```
CREATE USER bigshot PASSWORD 'buckshot';
```

Пример 10.2. Создание пользователя с помощью плагина управления пользователями Legacy_UserManager.

```
CREATE USER godzilla PASSWORD 'robot'
USING PLUGIN Legacy_UserManager;
```

Пример 10.3. Создание пользователя с пользовательскими атрибутами.

```
CREATE USER john PASSWORD 'fYe_3Ksw'
FIRSTNAME 'John'
LASTNAME 'Doe'
TAGS (BIRTHYEAR = '1970', CITY = 'New York');
```

Пример 10.4. Создание пользователя в неактивном состоянии.

```
CREATE USER john PASSWORD 'fYe_3Ksw'
FIRSTNAME 'John'
```

```
LASTNAME 'Doe'
INACTIVE;
```

Пример 10.5. Создание пользователя с возможностью управления пользователями.

```
CREATE USER superuser PASSWORD 'kMn8Kjh'
GRANT ADMIN ROLE;
```

См. также: ALTER USER, CREATE OR ALTER USER, DROP USER.

ALTER USER

Назначение: Изменение учётной записи пользователя Firebird.

Доступно в: DSQL.

Синтаксис:

```
ALTER {USER username | CURRENT USER}
{
  [SET]
  [PASSWORD 'password']
  [FIRSTNAME 'firstname']
  [MIDDLENAME 'middlename']
  [LASTNAME 'lastname']
  [ACTIVE | INACTIVE]
  [TAGS (<tag> | DROP tagname [, <tag> | DROP tagname ...] )]
}
[USING PLUGIN 'pluginname']
[{GRANT | REVOKE} ADMIN ROLE];

<tag> ::= tagname = 'string value'
```

Параметры оператора ALTER USER

username
Имя пользователя.

password
Пароль пользователя. Может включать в себя до 32 символов. Чувствительно к регистру.

firstname
Вспомогательная информация: имя пользователя. Максимальная длина 32 символа.

middlename
Вспомогательная информация: "второе имя" (отчество, "имя отца") пользователя. Максимальная длина 32 символа.

lastname
Вспомогательная информация: фамилия пользователя. Максимальная длина 32 символа.

pluginname
Имя плагина управления пользователями, в котором был создан данный пользователь.

tagname
Имя пользовательского атрибута. Максимальная длина 31 символ. Имя атрибута должно подчиняться правилам наименования SQL идентификаторов.

string value
Значение пользовательского атрибута. Максимальная длина 255 символов.

Оператор ALTER USER изменяет данные учётной записи пользователя. В операторе ALTER USER должен присутствовать хотя бы одно из необязательных предложений.

Необязательное предложение PASSWORD задаёт новый пароль пользователя. Необязательные предложения FIRSTNAME, MIDDLENAME и LASTNAME позволяют изменить дополнительные атрибуты пользователя, такие как имя пользователя (имя человека), отчество и фамилия соответственно.

Атрибут INACTIVE позволяет сделать учётную запись неактивной. Это удобно когда необходимо временно отключить учётную запись без её удаления. Атрибут ACTIVE позволяет вернуть неактивную учётную запись в активное состояние.

Необязательное предложение TAGS позволяет задать, изменить или удалить пользовательские атрибуты. Если в списке атрибутов, атрибута с заданным именем не было, то он будет добавлен, иначе его значение будет изменено. Атрибуты не указанные в списке не будут изменены. Для удаления пользовательского атрибута перед его именем в списке атрибутов необходимо указать ключевое слово DROP.

Предложение GRANT ADMIN ROLE предоставляет указанному пользователю привилегии роли RDB$ADMIN в текущей базе данных безопасности. Это позволяет указанному пользователю управлять учётными записями пользователей, но не даёт ему специальных полномочий в обычных базах данных.

Предложение REVOKE ADMIN ROLE отбирает у указанного пользователя привилегии роли RDB$ADMIN в текущей базе данных безопасности. Это запрещает указанному пользователю управлять учётными записями пользователей.

Необязательное предложение USING PLUGIN позволяет явно указывать какой плагин управления пользователями будет использован. По умолчанию используется тот плагин, который был указан первым в списке параметра *UserManager* в файле конфигурации `firebird.conf`. Допустимыми являются только значения, перечисленные в параметре *UserManager*.

> **Важно:**
>
> Учтите что одноименные пользователи, созданные с помощью разных плагинов управления пользователями — это разные пользователи. Поэтому пользователя созданного с помощью одного плагина управления пользователями можно удалить или изменить, указав только тот же самый плагин.

Если требуется изменить свою учётную запись, то вместо указания имени текущего пользователя можно использовать предложение CURRENT USER.

Для модификации чужой учётной записи пользователя текущий пользователь должен обладать административными привилегиями. Свои собственные учётные записи могут изменять любые

пользователи, однако это не относится к опциям GRANT/REVOKE ADMIN ROLE и атрибуту ACTIVE/INACTIVE для изменения которых необходимы административные привилегии.

Примеры:

Пример 10.6. Изменение пользователя и выдача ему привилегии управления пользователями.

```
ALTER USER bobby PASSWORD '67-UiT_G8'
GRANT ADMIN ROLE;
```

Пример 10.7. Изменение пароля пользователя, созданного с помощью плагина управления пользователями Legacy_UserManager.

```
ALTER USER godzilla PASSWORD 'robot12'
USING PLUGIN Legacy_UserManager;
```

Пример 10.8. Изменение дополнительных атрибутов своей учётной записи.

```
ALTER CURRENT USER
FIRSTNAME 'No_Jack'
LASTNAME 'Kennedy';
```

Пример 10.9. Отключение пользователя.

```
ALTER USER dan INACTIVE;
```

Пример 10.10. Отбор привилегии управления пользователями у пользователя.

```
ALTER USER dumbbell
REVOKE ADMIN ROLE;
```

Пример 10.11. Изменение пользовательских атрибутов своей учётной записи.

```
ALTER CURRENT USER
TAGS (BIRTHYEAR = '1971', DROP CITY);
```

Атрибуту BIRTHDAY будет установлено новое значение, а атрибут CITY будет удалён.

См. также: CREATE USER, CREATE OR ALTER USER, DROP USER.

CREATE OR ALTER USER

Назначение: Создание или изменение учётной записи пользователя Firebird.

Доступно в: DSQL.

Синтаксис:

```
CREATE OR ALTER USER username
{
  [SET]
  [PASSWORD 'password']
  [FIRSTNAME 'firstname']
  [MIDDLENAME 'middlename']
  [LASTNAME 'lastname']
  [ACTIVE | INACTIVE]
  [TAGS (<tag> | DROP tagname [, <tag> | DROP tagname ...] )]
}
[USING PLUGIN 'pluginname']
[{GRANT | REVOKE} ADMIN ROLE];

<tag> ::= tagname = 'string value'
```

Параметры оператора CREATE OR ALTER USER

username
Имя пользователя. Максимальная длина 31 символ.

password
Пароль пользователя. Может включать в себя до 32 символов. Чувствительно к регистру.

firstname
Вспомогательная информация: имя пользователя. Максимальная длина 32 символа.

middlename
Вспомогательная информация: "второе имя" (отчество, "имя отца") пользователя. Максимальная длина 32 символа.

lastname
Вспомогательная информация: фамилия пользователя. Максимальная длина 32 символа.

pluginname
Имя плагина управления пользователями, в котором необходимо создать нового пользователя или в котором он был создан ранее.

tagname
Имя пользовательского атрибута. Максимальная длина 31 символ. Имя атрибута должно подчиняться правилам наименования SQL идентификаторов.

string value
Значение пользовательского атрибута. Максимальная длина 255 символов.

Оператор CREATE OR ALTER USER создаёт новую или изменяет учётную запись. Если пользователя не существует, то он будет создан с использованием предложения CREATE USER. Если он уже существует, то он будет изменён, при этом существующие привилегии сохраняются.

Примеры:

Пример 10.12. Создание или изменение пользователя.

```
CREATE OR ALTER USER john PASSWORD 'fYe_3Ksw'
FIRSTNAME 'John'
LASTNAME 'Doe'
INACTIVE;
```

См. также: CREATE USER, ALTER USER.

DROP USER

Назначение: Удаление учётной записи пользователя Firebird.

Доступно в: DSQL.

Синтаксис:

```
DROP USER username
[USING PLUGIN 'pluginname'];
```

Параметры оператора DROP USER

username
Имя пользователя.

pluginname
Имя плагина управления пользователями, в котором был создан данный пользователь.

Оператор DROP USER удаляет учётную запись пользователя Firebird.

Необязательное предложение USING PLUGIN позволяет явно указывать какой плагин управления пользователями будет использован. По умолчанию используется тот плагин, который был указан первым в списке параметра *UserManager* в файле конфигурации firebird.conf. Допустимыми являются только значения, перечисленные в параметре *UserManager*.

> **Важно**
>
> Учтите что одноименные пользователи, созданные с помощью разных плагинов управления пользователями — это разные пользователи. Поэтому пользователя созданного с помощью одного плагина управления пользователями можно удалить или изменить, указав только тот же самый плагин.

Для удаления учётной записи пользователя текущий пользователь должен обладать административными привилегиями.

Примеры:

Пример 10.13. Удаление пользователя.

```
DROP USER bobby;
```

Пример 10.14. Удаление пользователя, созданного с помощью плагина управления пользователями Legacy_UserManager.

```
DROP USER Godzilla USING PLUGIN Legacy_UserManager;
```

См. также: CREATE USER, ALTER USER.

Операторы управления ролями

Роль (role) — объект базы данных, представляющий набор привилегий. Роли реализуют концепцию управления безопасностью на групповом уровне.

В данном разделе рассматриваются вопросы создания и удаления ролей, а также разрешения или запрещения автоматического предоставления роли RDB$ADMIN администраторам Windows (Trusted Authentication).

CREATE ROLE

Назначение: Создание новой роли.

Доступно в: DSQL, ESQL.

Синтаксис:

```
CREATE ROLE rolename;
```

Параметры оператора CREATE ROLE

rolename
Имя роли. Максимальная длина 31 символ.

Оператор CREATE ROLE создаёт новую роль. Имя роли должно быть уникальным среди имён ролей.

> **Предупреждение:**
>
> Желательно также чтобы имя роли было уникальным не только среди имён ролей, но и среди имён пользователей. Если вы создадите роль с тем же именем существующего пользователя, то такой пользователь не сможет подключиться к базе данных.

Создать новую роль могут:

- SYSDBA;
- Владелец базы данных;

- Любой пользователь, которому выдана привилегия на создание ролей (GRANT CREATE ROLE);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Пользователь, создавший роль, становится её владельцем.

Примеры:

Пример 10.15. Создание роли.

```
CREATE ROLE SELLERS;
```

См. также: DROP ROLE, GRANT, REVOKE.

ALTER ROLE

Назначение: Разрешает или запрещает автоматическое предоставление роли RDB $ADMIN администраторам Windows, если используется доверительная авторизация (Trusted Authentication).

Доступно в: DSQL.

Синтаксис:

```
ALTER ROLE RDB$ADMIN {SET | DROP} AUTO ADMIN MAPPING;
```

Оператор ALTER ROLE RDB$ADMIN разрешает или запрещает автоматическое предоставление роли RDB$ADMIN администраторам Windows в текущей базе данных, если используется доверительная авторизация (Trusted Authentication). По умолчанию автоматическое предоставление роли RDB$ADMIN отключено.

Примечание

На самом деле данный оператор является упрощённым видом оператора CREATE MAPPING. Эквивалентным оператором оператору

```
ALTER ROLE RDB$ADMIN SET AUTO ADMIN MAPPING
```

выглядит следующим образом:

```
CREATE MAPPING WIN_ADMINS
USING PLUGIN WIN_SSPI
FROM Predefined_Group
DOMAIN_ANY_RID_ADMINS
TO ROLE RDB$ADMIN;
```

Примечание

Оператор ALTER ROLE не может включить или выключить флаг AUTO ADMIN MAPPING в базе данных пользователей. Для этого можно использовать только утилиту командной строки gsec:

```
gsec -mapping set
gsec -mapping drop
```

Другим способом включения флага AUTO ADMIN MAPPING в базе данных пользователей является использование оператора:

```
CREATE GLOBAL MAPPING WIN_ADMINS
USING PLUGIN WIN_SSPI
FROM Predefined_Group
DOMAIN_ANY_RID_ADMINS
TO ROLE RDB$ADMIN;
```

Этот оператор может быть выполнен пользователями с достаточными правами, а именно:

- SYSDBA;
- Владелец базы данных;
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры:

Пример 10.16. Разрешение администраторам Windows автоматически предоставлять роль RDB$ADMIN.

```
ALTER ROLE RDB$ADMIN SET AUTO ADMIN MAPPING;
```

Пример 10.17. Запретить администратор Windows автоматически предоставлять роль RDB$ADMIN.

```
ALTER ROLE RDB$ADMIN DROP AUTO ADMIN MAPPING;
```

См. также: RDB$ADMIN, CREATE MAPPING.

DROP ROLE

Назначение: Удаление существующей роли.

Доступно в: DSQL, ESQL.

Синтаксис:

```
DROP ROLE rolename;
```

Параметры оператора DROP ROLE

rolename
Имя роли.

Оператор DROP ROLE удаляет существующую роль. При удалении роли все привилегии, предоставленные этой роли, отменяются.

Удалить роль могут:

- SYSDBA;
- Владелец базы данных;
- Владелец роли;
- Любой пользователь, которому выдана привилегия на удаление любой роли (GRANT DROP ANY ROLE);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

Примеры:

Пример 10.18. Удаление роли.

```
DROP ROLE SELLERS;
```

См. также: CREATE ROLE, GRANT, REVOKE.

SQL привилегии

Как уже отмечалось, в Firebird реализована двухуровневая модель безопасности. На первом уровне осуществляется аутентификация пользователя в момент подключения к базе данных, при этом используется база данных безопасности. Второй уровень реализуется уже на уровне самой базы данных. Все привилегии по доступу к объектам базы данных хранятся в самой базе. Авторизованный пользователь не имеет никаких привилегий до тех пор, пока какие либо права не будут предоставлены ему явно. Только создатель объекта и SYSDBA имеет привилегии на него и может назначать привилегии другим пользователям, ролям или объектам.

Владелец объекта базы данных

Пользователь, создавший объект базы данных, становится его владельцем. Создать объект базы данных могут SYSDBA, владелец базы данных, пользователи, вошедшие с ролью RDB $ADMIN, и пользователи которым назначены привилегии на создание объектов базы данных данного типа. Только владелец объекта базы данных, пользователи с административными

привилегиями (SYSDBA, и вошедшие с ролью RDB$ADMIN) и пользователи, получившие привилегии на изменение/удаление объектов базы данных данного типа могут изменять или удалять объект базы данных.

SYSDBA или владелец объекта могут выдавать привилегии другим пользователям, в том числе и привилегии на право выдачи привилегий другим пользователям. Собственно сам процесс раздачи привилегий на уровне SQL реализуется двумя операторами: GRANT, REVOKE.

GRANT

Назначение: Предоставление привилегий или назначение ролей.

Доступно в: DSQL.

Синтаксис:

```
<grant_stmt> ::=
    { GRANT <privileges>
      TO {<object_list> | PUBLIC | <user_list> [WITH GRANT OPTION] }
      [{GRANTED BY | AS} [USER] grantor] }
  | { GRANT <role_granted>
      TO {PUBLIC | <role_grantee_list> [WITH ADMIN OPTION]}
      [{GRANTED BY | AS} [USER] grantor] }

<privileges> ::=
    <table_privileges>
  | <execute_privileges>
  | <usage_privileges>
  | <ddl_privileges>

<table_privileges> ::=
  { ALL [PRIVILEGES] | <table_privileges_list> }
  ON [TABLE] {table_name | view_name}

<table_privilege_list> ::= {
    SELECT
  | DELETE
  | INSERT
  | UPDATE [(col [,col ...])]
  | REFERENCES [(col [,col ...])]
} [, <table_privilege_list> ...]

<execute_privileges> ::= EXECUTE ON {
    PROCEDURE proc_name
  | FUNCTION func_name
  | PACKAGE package_name
}

<usage_privileges> ::= USAGE ON {
    DOMAIN domain_name
  | EXCEPTION exception_name
  | {GENERATOR | SEQUENCE} generator_name
  | CHARACTER SET charset_name
  | COLLATION collation_name
}
```

```
<ddl_privileges> ::=
    {CREATE | ALTER ANY | DROP ANY} <object_type>
  | {CREATE | ALTER | DROP} DATABASE

<object_type> ::= {
    CHARACTER SET
  | COLLATION
  | DOMAIN
  | EXCEPTION
  | FILTER
  | FUNCTION
  | GENERATOR
  | PACKAGE
  | PROCEDURE
  | ROLE
  | SEQUENCE
  | TABLE
  | VIEW
}

<object_list> ::= {
    PROCEDURE proc_name
  | FUNCTION func_name
  | PACKAGE package_name
  | TRIGGER trig_name
  | VIEW view_name } [, <object_list> ...]

<user_list> ::= {
    [USER] username
  | [ROLE] rolename
  | GROUP Unix_group
  | Unix_user } [, <user_list> ...]

<role_granted> ::= rolename [, rolename ...]

<role_grantee_list> ::= [USER] username [, [USER] username ...]
```

Параметры оператора GRANT

table_name
Имя таблицы, к которой должно быть применена привилегия.

view_name
Имя представления, к которому должно быть применена привилегия или которому будут выданы привилегии.

col
Столбец таблицы, к которому должна быть применена привилегия.

proc_name
Имя хранимой процедуры, для которой должна быть выдана привилегия EXECUTE или которой будут даны привилегии.

func_name
Имя хранимой функции (или UDF), для которой должна быть выдана привилегия EXECUTE или которой будут даны привилегии.

package_name
Имя пакета, для которого должна быть выдана привилегия EXECUTE или которому будут даны привилегии.

domain_name
Имя домена, для которого должна быть выдана привилегия USAGE.

exception_name
Имя исключения, для которого должна быть выдана привилегия USAGE.

generator_name
Имя генератора (последовательности), для которого должна быть выдана привилегия USAGE.

charset_name
Имя набора символов, для которого должна быть выдана привилегия USAGE.

collation_name
Имя сортировки, для которой должна быть выдана привилегия USAGE.

object_type
Тип объекта метаданных.

object_list
Список объектов метаданных, которым будут даны привилегии.

trig_name
Имя триггера, которому будут даны привилегии.

user_list
Список пользователей/ролей, которым будут выданы привилегии.

username
Имя пользователя, для которого выдаются привилегии или которому назначается роль.

rolename
Имя роли.

Unix_group
Имя группы пользователей в операционных системах семейства UNIX.

Unix_user
Имя пользователя в операционной системе семейства UNIX.

role_granted
Список ролей, которые будут назначены.

role_grantee_list
Список пользователей, которым будут назначены роли.

grantor
Пользователь от имени, которого предоставляются привилегии.

Оператор GRANT предоставляет одну или несколько привилегий для объектов базы данных пользователям, ролям, хранимым процедурам, функциям, пакетам, триггерам и представлениям.

Авторизованный пользователь не имеет никаких привилегий до тех пор, пока какие либо права не будут предоставлены ему явно. При создании объекта только его создатель и SYSDBA имеет привилегии на него и может назначать привилегии другим пользователям, ролям или объектам.

Для различных типов объектов метаданных существует различный набор привилегий. Эти привилегии будут описаны далее отдельно для каждого из типов объектов метаданных.

Предложение TO

В предложении TO указывается список пользователей, ролей и объектов базы данных (процедур, функций, пакетов, триггеров и представлений) для которых будут выданы перечисленные привилегии. Необязательные предложения USER и ROLE позволяют уточнить, кому именно выдаётся привилегия. Если ключевое слово USER или ROLE не указано, то сервер проверяет, существует ли роль с данным именем, если таковой не существует, то привилегии назначаются пользователю. Существование пользователя, которому выдаются права, не проверяются при выполнении оператора GRANT. Если привилегия выдаётся объекту базы данных, то необходимо обязательно указывать тип объекта.

> **Рекомендация**
>
> Несмотря на то, что ключевые слова USER и ROLE не обязательные, желательно использовать их, чтобы избежать путаницы.

Пользователь PUBLIC

В SQL существует специальный пользователь PUBLIC, представляющий всех пользователей. Если какая-то операция разрешена пользователю PUBLIC, значит, любой аутентифицированный пользователь может выполнить эту операцию над указанным объектом.

> **Важно**
>
> Если привилегии назначены пользователю PUBLIC, то и отозваны они должны быть у пользователя PUBLIC.

WITH GRANT OPTION

Необязательное предложение WITH GRANT OPTION позволяет пользователям, указанным в списке пользователей, передавать другим пользователям привилегии указанные в списке привилегий.

GRANTED BY

При предоставлении прав в базе данных в качестве лица, предоставившего эти права, обычно записывается текущий пользователь. Используя предложение GRANTED BY можно предоставлять права от имени другого пользователя. При использовании оператора REVOKE после GRANTED BY права будут удалены только в том случае, если они были зарегистрированы от удаляющего пользователя. Для облегчения миграции из некоторых других реляционных СУБД нестандартное предложение AS поддерживается как синоним оператора GRANTED BY.

Предложение GRANTED BY может использовать:

- Владелец базы данных;

- SYSDBA;
- Любой пользователь, имеющий права на роль RDB$ADMIN и указавший её при соединении с базой данных;
- При использовании флага AUTO ADMIN MAPPING — любой администратор операционной системы Windows (при условии использования сервером доверенной авторизации — trusted authentication), даже без указания роли.

Даже владелец роли не может использовать GRANTED BY, если он не находится в вышеупомянутом списке.

Табличные привилегии

Для таблиц и представлений в отличие от других объектов метаданных возможно использовании сразу нескольких привилегий.

Список привилегий для таблиц

SELECT
: Разрешает выборку данных из таблицы или представления.

INSERT
: Разрешает добавлять записи в таблицу или представление.

UPDATE
: Разрешает изменять записи в таблице или представлении. Можно указать ограничения, чтобы можно было изменять только указанные столбцы.

DELETE
: Разрешает удалять записи из таблицы или представления.

REFERENCES
: Разрешает ссылаться на указанные столбцы внешним ключом. Необходимо указать для столбцов, на которых построен первичный ключ таблицы, если на неё есть ссылка внешним ключом другой таблицы.

ALL
: Объединяет привилегии SELECT, INSERT, UPDATE, DELETE и REFERENCES.

Примеры:

Пример 10.19. Назначение привилегий для таблиц

```
-- Привилегии SELECT, INSERT пользователю ALEX
GRANT SELECT, INSERT ON TABLE SALES
TO USER ALEX;

-- Привилегия SELECT ролям MANAGER, ENGINEER и пользователю IVAN
GRANT SELECT ON TABLE CUSTOMER
TO ROLE MANAGER, ROLE ENGINEER, USER IVAN;

-- Все привилегии для роли ADMINISTRATOR
-- с возможностью передачи своих полномочий
GRANT ALL ON TABLE CUSTOMER
TO ROLE ADMINISTRATOR WITH GRANT OPTION;
```

```
-- Привилегии SELECT и REFRENCE для столбца NAME для всех пользователей
GRANT SELECT, REFERENCES (NAME) ON TABLE COUNTRY
TO PUBLIC;

-- Выдача привилегии SELECT для пользователя IVAN от имени пользователя ALEX
GRANT SELECT ON TABLE EMPLOYEE
TO USER IVAN GRANTED BY ALEX;

-- Привилегия UPDATE для столбцов FIRST_NAME, LAST_NAME
GRANT UPDATE (FIRST_NAME, LAST_NAME) ON TABLE EMPLOYEE
TO USER IVAN;

-- Привилегия INSERT для хранимой процедуры ADD_EMP_PROJ
GRANT INSERT ON EMPLOYEE_PROJECT
TO PROCEDURE ADD_EMP_PROJ;
```

Привилегия EXECUTE

Привилегия EXECUTE (выполнение) применима к хранимым процедурам, хранимым функциям, пакетам и унаследованным внешним функциям (UDF), определяемых как DECLARE EXTERNAL FUNCTION.

Для хранимых процедур привилегия EXECUTE позволяет не только выполнять хранимые процедуры, но и делать выборку данных из процедур выбора (с помощью оператора SELECT).

> **Замечание:**
>
> Привилегия может быть назначена только для всего пакета, а не для отдельных его подпрограмм.

Примеры:

Пример 10.20. Назначение привилегии EXECUTE

```
-- Привилегия EXECUTE для хранимой процедуры
GRANT EXECUTE ON PROCEDURE ADD_EMP_PROJ
TO ROLE MANAGER;

-- Привилегия EXECUTE для хранимой функции
GRANT EXECUTE ON FUNCTION GET_BEGIN_DATE TO ROLE MANAGER;

-- Привилегия EXECUTE для пакета
GRANT EXECUTE ON PACKAGE APP_VAR TO PUBLIC;

-- Привилегия EXECUTE для функции выданная пакету
GRANT EXECUTE ON FUNCTION GET_BEGIN_DATE
TO PACKAGE APP_VAR;
```

Привилегия USAGE

Для использования объектов метаданных, отличных от таблиц, представлений, хранимых процедур и функций, триггеров и пакетов, в пользовательских запросах необходимо

предоставить пользователю привилегию USAGE для этих объектов. Поскольку в Firebird хранимые процедуры и функции, триггеры и подпрограммы пакетов выполняются с привилегиями вызывающего пользователя, то при использовании таких объектов метаданных в них, может потребоваться назначить привилегию USAGE и для них.

> **Замечание:**
>
> В Firebird 3 привилегия USAGE проверяется только для исключений (exception) и генераторов/последовательностей (в gen_id(*gen_name*, 1) или next value for *gen_name*). Привилегии для других объектов метаданных могут быть включены в следующих релизах, если покажется целесообразным.

> **Замечание:**
>
> Привилегия USAGE даёт права только на приращения генераторов (последовательностей) с помощью функции GEN_ID или конструкции NEXT VALUE FOR. Оператор SET GENERATOR является аналогом оператора ALTER SEQUENCE ... RESTART WITH, которые относятся к DDL операторам. По умолчанию права на такие операции имеет только владелец генератора (последовательности). Права на установку начального значения любого генератора (последовательности) можно предоставить с помощью GRANT ALTER ANY SEQUENCE, что не рекомендуется для обычных пользователей.

Примеры:

Пример 10.21. Назначение привилегии USAGE

```
-- Привилегия USAGE для последовательности выданная роли
GRANT USAGE ON SEQUENCE GEN_AGE TO ROLE MANAGER;

-- Привилегия USAGE для последовательности выданная триггеру
GRANT USAGE ON SEQUENCE GEN_AGE TO TRIGGER TR_AGE_BI;

-- Привилегия USAGE для исключения выданная пакету
GRANT USAGE ON EXCEPTION E_ACCESS_DENIED
TO PACKAGE PKG_BILL;
```

DDL привилегии

Создать новый объект метаданных могут:

- SYSDBA;
- Владелец базы данных;
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Любой пользователь, получивший привилегии на создание объекта метаданных данного типа.

Изменить/удалить объект метаданных могут:

- SYSDBA;
- Владелец базы данных;
- Владелец объекта метаданных;
- Любой пользователь, вошедший с ролью RDB$ADMIN;

- Любой пользователь, получивший привилегии на изменение/удаление объектов метаданных данного типа.

Список DDL привилегий

CREATE
Разрешает создание объекта указанного типа метаданных.

ALTER ANY
Разрешает изменение любого объекта указанного типа метаданных.

DROP ANY
Разрешает удаление любого объекта указанного типа метаданных.

> **Примечание:**
>
> Метаданные триггеров и индексов наследуют привилегии таблиц, которые владеют ими.

Примеры:

Пример 10.22. Назначение привилегий на изменение метаданных

```
-- Разрешение пользователю Joe создавать таблицы
GRANT CREATE TABLE TO Joe;

-- Разрешение пользователю Joe изменять любые процедуры
GRANT ALTER ANY PROCEDURE TO Joe;
```

Назначение ролей

Оператор GRANT может быть использован для назначения ролей для группы перечисленных пользователей. В этом случае после предложения GRANT следует список ролей, которые будут назначены списку пользователей, указанному после предложения TO.

Необязательное предложение WITH ADMIN OPTION позволяет пользователям, указанным в списке пользователей, назначать роли из списка ролей, указанных в списке ролей, другим пользователям.

Примеры:

Пример 10.23. Назначение ролей для пользователей

```
-- Назначение ролей DIRECTOR и MANAGER пользователю IVAN
GRANT DIRECTOR, MANAGER TO USER IVAN;

-- Назначение роли ADMIN пользователю ALEX
-- с возможностью назначить эту другим пользователям
GRANT MANAGER TO USER ALEX WITH ADMIN OPTION;
```

См. также: REVOKE.

REVOKE

Назначение: Отмена привилегий или отбор ролей.

Доступно в: DSQL.

Синтаксис:

```
<revoke_stmt> ::=
    { REVOKE [GRANT OPTION FOR] <privileges>
      FROM {<object_list> | PUBLIC | <user_list> }
      [{GRANTED BY | AS} [USER] grantor] }
  | { REVOKE [ADMIN OPTION FOR] <role_granted>
      FROM {PUBLIC | <role_grantee_list> }
      [{GRANTED BY | AS} [USER] grantor] }
  | { REVOKE ALL ON ALL FROM <user_list> }

<privileges> ::=
    <table_privileges>
  | <execute_privileges>
  | <usage_privileges>
  | <ddl_privileges>

<table_privileges> ::=
  { ALL [PRIVILEGES] | <table_privileges_list> }
  ON [TABLE] {table_name | view_name}

<table_privilege_list> ::= {
    SELECT
  | DELETE
  | INSERT
  | UPDATE [(col [,col ...])]
  | REFERENCES [(col [,col ...])]
} [, <table_privilege_list> ...]

<execute_privileges> ::= EXECUTE ON {
    PROCEDURE proc_name
  | FUNCTION func_name
  | PACKAGE package_name
}

<usage_privileges> ::= USAGE ON {
    DOMAIN domain_name
  | EXCEPTION exception_name
  | {GENERATOR | SEQUENCE} generator_name
  | CHARACTER SET charset_name
  | COLLATION collation_name
}

<ddl_privileges> ::=
    {CREATE | ALTER ANY | DROP ANY} <object_type>
  | {CREATE | ALTER | DROP} DATABASE

<object_type> ::= {
    CHARACTER SET
  | COLLATION
  | DOMAIN
```

```
  | EXCEPTION
  | FILTER
  | FUNCTION
  | GENERATOR
  | PACKAGE
  | PROCEDURE
  | ROLE
  | SEQUENCE
  | TABLE
  | VIEW
}

<object_list> ::= {
    PROCEDURE proc_name
  | FUNCTION func_name
  | PACKAGE package_name
  | TRIGGER trig_name
  | VIEW view_name } [, <object_list> ...]

<user_list> ::= {
    [USER] username
  | [ROLE] rolename
  | GROUP Unix_group
  | Unix_user } [, <user_list> ...]

<role_granted> ::= rolename [, rolename ...]

<role_grantee_list> ::= [USER] username [, [USER] username ...]
```

Параметры оператора REVOKE

table_name
Имя таблицы, у которой должна быть отозвана привилегия.

view_name
Имя представления, к которому должно быть применена привилегия или которого будет отозваны привилегии.

col
Столбец таблицы, у которого должна быть отозвана привилегия.

proc_name
Имя хранимой процедуры, для которой должна быть отозвана привилегия EXECUTE или у которой должны быть отозваны привилегии.

func_name
Имя хранимой функции (или UDF), для которой должна быть отозвана привилегия EXECUTE или у которой должны быть отозваны привилегии.

package_name
Имя пакета, для которого должна быть отозвана привилегия EXECUTE или у которого должны быть отозваны привилегии.

domain_name
Имя домена, для которого должна быть отозвана привилегия USAGE.

exception_name
Имя исключения, для которого должна быть отозвана привилегия USAGE.

generator_name
Имя генератора (последовательности), для которого должна быть отозвана привилегия USAGE.

charset_name
Имя набора символов, для которого должна быть отозвана привилегия USAGE.

collation_name
Имя сортировки, для которой должна быть отозвана привилегия USAGE.

object_type
Тип объекта метаданных.

object_list
Список объектов метаданных, у которых будут отозваны привилегии.

trig_name
Имя триггера, у которого будут отозваны привилегии.

user_list
Список пользователей/ролей, у которых будут отозваны привилегии.

username
Имя пользователя, для которого отзываются привилегии или у которого отбирается роль.

rolename
Имя роли.

Unix_group
Имя группы пользователей в операционных системах семейства UNIX.

Unix_user
Имя пользователя в операционной системе семейства UNIX.

role_granted
Список ролей, которые будут отобраны.

role_grantee_list
Список пользователей, у которых будут отобраны роли.

grantor
Пользователь от имени, которого отзываются привилегии.

Оператор REVOKE отменяет привилегии для пользователей, ролей, хранимых процедур, хранимых функций, пакетов, триггеров и представлений выданные оператором GRANT. Подробное описание различных типов привилегий см. в GRANT.

Предложение FROM

В предложении FROM указывается список пользователей, ролей и объектов базы данных (процедур, функций, пакетов, триггеров и представлений) у которых будут отняты перечисленные привилегии. Необязательные предложения USER и ROLE позволяют уточнить,

у кого именно выдаётся привилегия. Если ключевое слово USER или ROLE не указано, то сервер проверяет, существует ли роль с данным именем, если таковой не существует, то привилегии отбираются у пользователя.

> **Рекомендация**
>
> Несмотря на то, что ключевые слова USER и ROLE не обязательные, желательно использовать их, чтобы избежать путаницы.

Существование пользователя, у которого отбираются права, не проверяются при выполнении оператора REVOKE. Если привилегия отбирается у объекта базы данных, то необходимо обязательно указывать тип объекта.

> **Важно**
>
> Если привилегии были назначены специальному пользователю PUBLIC, то отменять привилегии необходимо для пользователя PUBLIC. Специальный пользователь PUBLIC используется, когда необходимо предоставить привилегии сразу всем пользователям. Однако не следует рассматривать PUBLIC как группу пользователей.

GRANT OPTION FOR

Необязательное предложение GRANT OPTION FOR отменяет для соответствующего пользователя или роли право предоставления другим пользователям или ролям привилегии к таблицам, представлениям, триггерам, хранимым процедурам.

Отмена назначенных ролей

Другое назначение оператора REVOKE в отборе назначенных группе пользователей ролей оператором GRANT. В этом случае после предложения REVOKE следует список ролей, которые будут отозваны у списка пользователей, указанных после предложения FROM.

Необязательное предложение ADMIN OPTION FOR отменяет ранее предоставленную административную опцию (право на передачу предоставленной пользователю роли другим) из грантополучателей, не отменяя прав на роль. В одном операторе могут быть обработаны несколько ролей и/или грантополучателей.

GRANTED BY

При предоставлении прав в базе данных в качестве лица, предоставившего эти права, обычно записывается текущий пользователь. Используя предложение GRANTED BY можно предоставлять права от имени другого пользователя. При использовании оператора REVOKE после GRANTED BY права будут удалены только в том случае, если они были зарегистрированы от удаляющего пользователя. Для облегчения миграции из некоторых других реляционных СУБД нестандартное предложение AS поддерживается как синоним оператора GRANTED BY.

Предложение GRANTED BY может использовать:

- Владелец базы данных;
- SYSDBA;
- Любой пользователь, имеющий права на роль RDB$ADMIN и указавший её при соединении с базой данных;

- При использовании флага AUTO ADMIN MAPPING — любой администратор операционной системы Windows (при условии использования сервером доверенной авторизации — trusted authentication), даже без указания роли.

Даже владелец роли не может использовать GRANTED BY, если он не находится в вышеупомянутом списке.

REVOKE ALL ON ALL

Если после ключевого слова REVOKE указано предложение ALL ON ALL, то это позволяет отменить все привилегии (включая роли) на всех объектах от одного или более пользователей и/или ролей. Это быстрый способ "очистить" (отобрать) права, когда пользователю должен быть заблокирован доступ к базе данных.

Примечания:

- Когда оператор REVOKE ALL ON ALL вызывается привилегированным пользователем (владельцем базы данных, SYSDBA или любым пользователем, у которого CURRENT_ROLE — RDB$ADMIN), удаляются все права независимо от того, кто их предоставил. В противном случае удаляются только права, предоставленные текущим пользователем;
- Не поддерживается предложение GRANTED BY;
- Этот оператор не удаляет флаг пользователя, давшего права на хранимые процедуры, триггеры или представлений (права на такие объекты конечно удаляются).

Примеры:

Пример 10.24. Отзыв привилегий на таблицу

```
-- отзыв привилегий SELECT, INSERT у таблицы
REVOKE SELECT, INSERT ON TABLE SALES FROM USER ALEX

-- отзыв привилегии SELECT у ролей MANAGER и ENGINEER и пользователя IVAN
REVOKE SELECT ON TABLE CUSTOMER
FROM ROLE MANAGER, ROLE ENGINEER, USER IVAN;

-- отмена возможности передавать любую из привилегии на таблицу
-- другим пользователям или ролям у роли ADMINISTRATOR
REVOKE GRANT OPTION FOR ALL ON TABLE CUSTOMER
FROM ROLE ADMINISTRATOR;

-- отзыв привилегий SELECT и REFERENCES у пользователя PUBLIC
REVOKE SELECT, REFERENCES (NAME) ON TABLE COUNTRY
FROM PUBLIC;

-- отзыв привилегии SELECT у пользователя IVAN,
-- которая была выдана пользователем ALEX
REVOKE SELECT ON TABLE EMPLOYEE
FROM USER IVAN GRANTED BY ALEX;

-- отзыв привилегии UPDATE для столбцов FIRST_NAME, LAST_NAME
REVOKE UPDATE (FIRST_NAME, LAST_NAME) ON TABLE EMPLOYEE
FROM USER IVAN;

-- отзыв привилегии INSERT у хранимой процедуры ADD_EMP_PROJ
```

```
REVOKE INSERT ON EMPLOYEE_PROJECT
FROM PROCEDURE ADD_EMP_PROJ;
```

Пример 10.25. Отзыв привилегии EXECUTE

```
-- отзыв привилегии EXECUTE для процедуры
REVOKE EXECUTE ON PROCEDURE ADD_EMP_PROJ
FROM USER IVAN;

-- отзыв привилегии EXECUTE для функции
-- и отбор права передавать эту привилегию
-- другим пользователям и ролям
REVOKE GRANT OPTION FOR
EXECUTE ON FUNCTION GET_BEGIN_DATE
FROM ROLE MANAGER;

-- отзыв привилегии EXECUTE для пакета
REVOKE EXECUTE ON PACKAGE DATE_UTILS
FROM USER ALEX;
```

Пример 10.26. Отзыв привилегии USAGE

```
-- Отзыв привилегии USAGE для последовательности выданной роли
REVOKE USAGE ON SEQUENCE GEN_AGE FROM ROLE MANAGER;

-- Отзыв привилегии USAGE для последовательности выданной триггеру
REVOKE USAGE ON SEQUENCE GEN_AGE FROM TRIGGER TR_AGE_BI;

-- Отзыв привилегии USAGE для исключения выданной пакету
REVOKE USAGE ON EXCEPTION E_ACCESS_DENIED
FROM PACKAGE PKG_BILL;
```

Пример 10.27. Отзыв привилегий на изменение метаданных

```
-- Отзыв у пользователя Joe привилегии на создание таблиц
REVOKE CREATE TABLE FROM Joe;

-- Отзыв у пользователя Joe привилегии на изменение любой процедуры
REVOKE ALTER ANY PROCEDURE FROM Joe;
```

Пример 10.28. Отзыв ролей

```
-- Отзыв ролей DIRECTOR, MANAGER у пользователя IVAN
REVOKE DIRECTOR, MANAGER FROM USER IVAN;

-- Отзыв роли MANAGER и права назначать её другим пользователям
REVOKE ADMIN OPTION FOR MANAGER FROM USER ALEX;
```

Пример 10.29. Отзыв всех привилегий и ролей у пользователя

```
REVOKE ALL ON ALL FROM IVAN;
```

После выполнения этой команды у пользователя IVAN нет вообще никаких прав.

См. также: GRANT.

Изменение текущей роли

SET ROLE

Назначение: Изменение текущей роли.

Доступно в: DSQL.

Синтаксис:

```
SET ROLE rolename
```

Параметры оператора SET ROLE

rolename
Имя устанавливаемой роли.

Согласно стандарту SQL-2008 оператор SET ROLE позволяет установить контекстной переменной CURRENT_ROLE одну из назначенных ролей для пользователя CURRENT_USER или роль, полученную в результате доверительной аутентификации (в этом случае оператор принимает вид SET TRUSTED ROLE).

Примеры:

Пример 10.30. Изменение текущей роли

```
SET ROLE manager;
select current_role from rdb$database;
```

```
ROLE
=======================
MANAGER
```

SET TRUSTED ROLE

Назначение: Установка доверенной роли.

Доступно в: DSQL.

Синтаксис:

```
SET TRUSTED ROLE
```

Оператор SET TRUSTED ROLE включает доступ доверенной роли, при условии, что CURRENT_USER получен с помощью доверительной аутентификации и роль доступна.

Идея отдельной команды SET TRUSTED ROLE состоит в том, чтобы при подключении доверенного пользователя не указывать никакой дополнительной информации о роли, SET TRUSTED ROLE делает доверенную роль (если таковая существует) текущей ролью без дополнительной деятельности, связанной с установкой параметров DBP.

Доверенная роль это не специальный тип роли, ей может быть любая роль, созданная с помощью оператора CREATE ROLE или предопределённая системная роль RDB$ADMIN. Она становится доверенной ролью для подключения, когда подсистема отображения объектов безопасности (security objects mapping subsystem) находит соответствие между результатом аутентификации, полученным от плагина и локальным или глобальным отображением (mapping) для текущей базы данных. Роль даже может быть той, которая не предоставлена явно этому доверенному пользователю.

> **Примечание**
>
> Доверенная роль не назначается при подключении по умолчанию. Можно изменить это поведение, используя соответствующий плагин аутентификации и команды CREATE/ALTER MAPPING.

Примером использования доверенной роли является назначение системной роли RDB$ADMIN для администраторов Windows, когда используется доверительная аутентификация Windows.

Системные роли

RDB$ADMIN

Системная роль RDB$ADMIN, присутствует в каждой базе данных. Предоставление пользователю роли RDB$ADMIN в базе данных даёт ему права SYSDBA, но только в этой базе данных. В обычной базе данных это означает полный контроль над всеми объектами. В базе данных пользователей это означает возможность создавать, изменять и удалять учётные записи пользователей. В обоих случаях пользователь с правами RDB$ADMIN роли может всегда передавать эту роль другим. Другими словами, "WITH ADMIN OPTION" уже встроен в эту роль и эту опцию можно не указывать.

Предоставление роли RDB$ADMIN в обычной базе данных

Для предоставления и удаления роли RDB$ADMIN в обычной базе данных используются операторы GRANT и REVOKE, как и для назначения и отмены остальных ролей.

Синтаксис:

```
GRANT RDB$ADMIN TO username

REVOKE RDB$ADMIN FROM username
```

Параметры операторов установки и отмены роли RDB$ADMIN

username
Имя пользователя, которому назначается роль RDB$ADMIN или у которого она отбирается.

Права на роль RDB$ADMIN могут давать:

- SYSDBA;
- Владелец базы данных;
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

См. также: GRANT, REVOKE.

Использование роли RDB$ADMIN в обычной базе данных

Для использования прав роли RDB$ADMIN пользователь просто указывает её при соединении с базой данных. Он также может указать её позднее с помощью оператора SET ROLE.

Предоставление роли RDB$ADMIN в базе данных пользователей

Так как никто не может соединиться с базой данных пользователей, то операторы GRANT и REVOKE здесь не могут использоваться. Вместо этого роль RDB$ADMIN предоставляют и удаляют SQL командами управления пользователями: CREATE USER и ALTER USER, в которых указываются специальные опции GRANT ADMIN ROLE и REVOKE ADMIN ROLE.

Синтаксис (неполный):

```
CREATE USER newuser
PASSWORD 'password'
GRANT ADMIN ROLE

ALTER USER existinguser
GRANT ADMIN ROLE

ALTER USER existinguser
REVOKE ADMIN ROLE
```

Параметры операторов установки и отмены роли RDB$ADMIN

newuser
Имя вновь создаваемого пользователя. Максимальная длина 31 символ.

existinguser
Имя существующего пользователя.

password
Пароль пользователя. Может включать в себя до 32 символов. Чувствительно к регистру.

> **Важно**
>
> Пожалуйста, помните, что GRANT ADMIN ROLE и REVOKE ADMIN ROLE это не операторы GRANT и REVOKE. Это параметры для CREATE USER и ALTER USER.

Права на роль RDB$ADMIN могут давать:

- SYSDBA;
- Любой пользователь, имеющий права на роль RDB$ADMIN в базе данных пользователей и указавший её при соединении с базой данных (или во время работы с утилитой gsec);
- Пользователь root операционной системы Linux;
- Администраторы Windows, если используется доверительная авторизация (trusted authentication) и включено автоматическое предоставление роли RDB$ADMIN администраторам Windows.

См. также: GRANT, REVOKE.

Использование роли RDB$ADMIN в базе данных пользователей

Для управления учётными записями пользователей через SQL пользователь, имеющий права на роль RDB$ADMIN, должен подключиться к базе данных с этой ролью. Так как к базе данных пользователей не имеет права соединяться никто, то пользователь должен подключиться к обычной базе данных, где он также имеет права на роль RDB$ADMIN. Он определяет роль при соединении с обычной базой данных и может в ней выполнить любой SQL запрос. Это не самое элегантное решение, но это единственный способ управлять пользователями через SQL запросы. Если нет обычной базы данных, где у пользователя есть права на роль RDB$ADMIN, то управление учётными записями посредством SQL запросов недоступно.

Управление отображением объектов безопасности

С введением поддержки множества баз данных безопасности в Firebird появились новые проблемы, которые не могли произойти с единой глобальной базой данных безопасности. Кластеры баз данных, использующие одну и ту же базу данных безопасности, были эффективно разделены. Отображения предоставляют средства для достижения той же эффективности, когда множество баз данных используют каждая свою базу данных безопасности. В некоторых случаях требуется управление для ограничения взаимодействия между такими кластерами. Например:

- когда EXECUTE STATEMENT ON EXTERNAL DATA SOURCE требует обмена данными между кластерами;
- когда обще серверный SYSDBA доступ к базам данных необходим от других кластеров, использующих службы;
- аналогичные проблемы существовали в Firebird 2.1 и 2.5 под Windows, из-за поддержки доверительной аутентификации: два отдельных списка пользователей – один в базе данных безопасности, а другой в Windows, и необходимо связать их.

Единое решение для всех этих случаев является отображение информации о пользователе, входящего в систему, на внутренние объекты безопасности – CURRENT_USER и CURRENT_ROLE.

> **Примечание**
>
> В Firebird имеется одно встроенное глобальное правило, действующее по умолчанию: пользователи прошедшие проверку в базе данных безопасности всегда отображается в любую базу данных один к одному. Это безопасное правило: для базы данных безопасности не имеет смысла не доверять себе.

CREATE MAPPING

Назначение: Создание отображения объекта безопасности.

Доступно в: DSQL.

Синтаксис:

```
CREATE [GLOBAL] MAPPING name
USING {
    PLUGIN plugin_name [IN database]
  | ANY PLUGIN [IN database | SERVERWIDE]
  | MAPPING [IN database]
  | '*' [IN database] }
FROM { ANY type | type from_name }
TO { USER | ROLE } [to_name]
```

Параметры оператора CREATE MAPPING

name
: Имя отображения. Может содержать до 31 символа.

plugin_name
: Имя плагина аутентификации.

database
: Имя базы данных, в которой прошла аутентификация.

type
: Тип объекта, который будет отображён.

from_name
: Имя объекта, который будет отображён.

to_name
: Имя объекта (пользователи или роли) на которое будет произведено отображение.

Оператор CREATE MAPPING создаёт отображение объектов безопасности (пользователей, групп, ролей) одного или нескольких плагинов аутентификации на внутренние объекты безопасности – CURRENT_USER и CURRENT_ROLE. Имя отображения должно быть уникальным среди имён отображений.

Если присутствует опция GLOBAL, то отображение будет применено не только для текущей базы данных, но и для всех баз данных находящимся в том же кластере, в том числе и базы данных безопасности.

> **Важно**
>
> Если существуют одноименные глобальные и локальные отображение, то вам следует знать, что это разные объекты.

> **Примечание**
>
> Глобальное отображение работает, если в качестве базы данных безопасности используется база данных Firebird 3 или более высокой версии. Если вы планируете использовать другую базу данных, например, для целей использования собственного поставщика, то вам необходимо создать таблицу в ней и назвать её RDB$MAP с той же структурой, что и RDB $MAP в базе данных Firebird 3 и дать доступ на запись только для SYSDBA.

Предложение USING описывает источник отображения. Оно имеет весьма сложный набор опций:

- явное указание имени плагина (опция PLUGIN) означает, что оно будет работать только с этим плагином;
- оно может использовать любой доступный плагин (опция ANY PLUGIN), даже если источник является продуктом предыдущего отображения;
- оно может быть сделано так, чтобы работать только с обще серверными плагинами (опция SERVERWIDE);
- оно может быть сделано так, чтобы работать только с результатами предыдущего отображения (опция MAPPING);
- вы можете опустить использование любого из методов, используя звёздочку (*) в качестве аргумента;
- оно может содержать имя базы данных (опция IN), из которой происходит отображение объекта FROM.

> **Примечание**
>
> Этот аргумент не является допустимым для отображения с обще серверной аутентификацией.

Предложение FROM описывает отображаемый объект. Оно принимает обязательный аргумент – тип объекта. Особенности:

- при отображении имён из плагинов, тип определяется плагином;
- при отображении продукта предыдущего отображения, типом может быть только USER и ROLE;
- если имя объекта будет указано явно, то оно будет учитываться при отображении;
- при использовании ключевого слова ANY будут отображены объекты с любыми именами данного типа.

В предложении TO указывается пользователь или роль, на которого будет произведено отображение. NAME является не обязательным аргументом. Если он не указан, то в качестве имени объекта будет использовано оригинальное имя из отображаемого объекта.

Создать отображение могут:

- SYSDBA;
- Владелец базы данных (если отображение локальное);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux.

Примеры:

Пример 10.31. Включение использования доверительной аутентификации Windows во всех базах данных, которые используют текущую базу данных безопасности.

```
CREATE GLOBAL MAPPING TRUSTED_AUTH
USING PLUGIN WIN_SSPI
FROM ANY USER
TO USER;
```

Пример 10.32. Включение SYSDBA подобного доступа для администраторов Windows в текущей базе данных.

```
CREATE MAPPING WIN_ADMINS
USING PLUGIN WIN_SSPI
FROM Predefined_Group
DOMAIN_ANY_RID_ADMINS
TO ROLE RDB$ADMIN;
```

> **Примечание**
>
> Группа DOMAIN_ANY_RID_ADMINS не существует в Windows, но такое имя будет добавлено плагином win_sspi для обеспечения точной обратной совместимости.

Пример 10.33. Включение доступа определённому пользователю из другой базы данных к текущей базе данных под другим именем.

```
CREATE MAPPING FROM_RT
USING PLUGIN SRP IN "rt"
FROM USER U1 TO USER U2;
```

> **Важно**
>
> Имена баз данных должны быть заключены в двойные кавычки на операционных системах, которые имею регистр чувствительные имена файлов.

Пример 10.34. Включение SYSDBA сервера (от основной базы данных безопасности) для доступа к текущей базе данных.

Предположим, что база данных использует базу данных безопасности не по умолчанию.

```
CREATE MAPPING DEF_SYSDBA
USING PLUGIN SRP IN "security.db"
FROM USER SYSDBA
TO USER;
```

Пример 10.35. Обеспечение гарантирование, что у пользователей, которые подключаются унаследованным плагином аутентификации не слишком много прав.

```
CREATE MAPPING LEGACY_2_GUEST
USING PLUGIN legacy_auth
FROM ANY USER
TO USER GUEST;
```

См. также: ALTER MAPPING, CREATE OR ALTER MAPPING, DROP MAPPING.

ALTER MAPPING

Назначение: Изменение отображения объекта безопасности.

Доступно в: DSQL.

Синтаксис:

```
ALTER [GLOBAL] MAPPING name
USING {
    PLUGIN plugin_name [IN database]
  | ANY PLUGIN [IN database | SERVERWIDE]
  | MAPPING [IN database]
  | '*' [IN database] }
FROM { ANY type | type from_name }
TO { USER | ROLE } [to_name]
```

Параметры оператора ALTER MAPPING

name
Имя отображения.

plugin_name
Имя плагина аутентификации.

database
Имя базы данных, в которой прошла аутентификация.

type
Тип объекта, который будет отображён.

from_name
Имя объекта, который будет отображён.

to_name
Имя объекта (пользователи или роли) на которое будет произведено отображение.

Оператор ALTER MAPPING позволяет изменять любые опции существующего отображения.

> **Важно**
>
> Одноименные глобальные и локальные отображение — это разные объекты.

Изменить отображение могут:

- SYSDBA;
- Владелец базы данных (если отображение локальное);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux.

Примеры:

Пример 10.36. Изменение отображения.

```
ALTER MAPPING FROM_RT
USING PLUGIN SRP IN "rt"
FROM USER U1 TO USER U3;
```

См. также: CREATE MAPPING, CREATE OR ALTER MAPPING, DROP MAPPING.

CREATE OR ALTER MAPPING

Назначение: Создание или изменение отображения объекта безопасности.

Доступно в: DSQL.

Синтаксис:

```
CREATE OR ALTER [GLOBAL] MAPPING name
USING {
    PLUGIN plugin_name [IN database]
  | ANY PLUGIN [IN database | SERVERWIDE]
  | MAPPING [IN database]
  | '*' [IN database] }
FROM { ANY type | type from_name }
TO { USER | ROLE } [to_name]
```

Параметры оператора CREATE OR ALTER MAPPING

name
Имя отображения. Может содержать до 31 символа.

plugin_name
Имя плагина аутентификации.

database
Имя базы данных, в которой прошла аутентификация.

type
Тип объекта, который будет отображён.

from_name
Имя объекта, который будет отображён.

to_name
Имя объекта (пользователи или роли) на которое будет произведено отображение.

Оператор CREATE OR ALTER MAPPING создаёт новое или изменяет существующее отображение. Если отображение не существует, то оно будет создано с использованием предложения CREATE MAPPING.

> **Важно**
>
> Одноименные глобальные и локальные отображение — это разные объекты.

Примеры:

Пример 10.37. Создание нового или изменение существующего отображения.

```
CREATE OR ALTER MAPPING FROM_RT
USING PLUGIN SRP IN "rt"
FROM USER U1 TO USER U4;
```

См. также: CREATE MAPPING, ALTER MAPPING, DROP MAPPING.

DROP MAPPING

Назначение: Удаление отображения объекта безопасности.

Доступно в: DSQL.

Синтаксис:

```
DROP [GLOBAL] MAPPING name
```

Параметры оператора DROP MAPPING

name
Имя отображения.

Оператор DROP MAPPING удаляет существующее отображение. Если указана опция GLOBAL, то будет удалено глобальное отображение.

> **Важно**
>
> Одноименные глобальные и локальные отображение — это разные объекты.

Изменить отображение могут:

- SYSDBA;
- Владелец базы данных (если отображение локальное);
- Любой пользователь, вошедший с ролью RDB$ADMIN;
- Пользователь root операционной системы Linux.

Примеры:

Пример 10.38. Удаление отображения.

```
DROP MAPPING FROM_RT;
```

См. также: CREATE MAPPING.

Шифрование базы данных

В Firebird существует возможность зашифровать данные хранимые в базе данных. Не весь файл базы данных шифруется: только страницы данных, индексов и blob.

Для того чтобы сделать шифрование базы данных возможным необходимо получить или написать плагин шифрования базы данных.

> **Примечание**
>
> Пример плагина шифрования в examples/dbcrypt не производит реального шифрования, это просто пример того, как можно написать этот плагин.

Основная проблема с шифрованием базы данных состоит в том, как хранить секретный ключ. Firebird предоставляет помощника для передачи этого ключа от клиента, но это вовсе не означает, что хранение ключей на клиенте является лучшим способом: это не более чем одна возможных альтернатив. Хранение ключей на том же диске что и база данных является очень плохим вариантом.

Для эффективного разделения шифрования и доступа к ключу, плагин шифрования базы данных разделён на две части: само шифрование и держатель секретного ключа. Это может быть эффективным подходом, когда вы хотите использовать некоторый хороший алгоритм шифрования, но у вас есть собственный секретный способ хранение ключей.

После того как вы определитесь с плагином и ключом, вы можете включить процесс шифрования.

Синтаксис:

```
ALTER DATABASE ENCRYPT WITH plugin_name
```

Параметры оператора ALTER DATABASE ENCRYPT

plugin_name
Имя плагина шифрования.

Шифрование начинается сразу после этого оператора и будет выполняться в фоновом режиме. Нормальная работа с базами данных не нарушается во время шифрования.

> **Подсказка**
>
> Процесс шифрования может быть проконтролирован с помощью поля MON$CRYPT_PAGE в псевдо-таблице MON$DATABASE или смотреть страницу заголовка базы данных с помощью *gstat -e*.
>
> *gstat –h* также будет предоставлять ограниченную информацию о состоянии шифрования.

Для дешифрования базы данных выполните:

```
ALTER DATABASE DECRYPT
```

Для Linux пример плагина с именем *libDbCrypt_example.so* можно найти в поддиректории */plugins/*.

Приложение A: Дополнительные статьи

Поле RDB$VALID_BLR

В системных таблицах RDB$PROCEDURES, RDB$FUNCTIONS и RDB$TRIGGERS присутствует поле RDB$VALID_BLR. Оно предназначено для отображения возможной невалидности модуля PSQL (процедуры, функции или триггера) при изменении доменов или столбцов таблиц, от которых он зависит. При возникновении описанной выше ситуации флаг поле RDB$VALID_BLR устанавливается в 0 для процедур, функций или триггеров, код которых возможно является не валидным.

Нижеприведённый запрос находит процедуры и триггеры, зависящие от определённого домена (в примере это домен 'MYDOMAIN'), и выводит информацию о состоянии поля RDB $VALID_BLR:

```
WITH VALID_PSQL (
    PSQL_TYPE,
    ROUTE_NAME,
    VALID)
AS (SELECT
        'Procedure',
        RDB$PROCEDURE_NAME,
        RDB$VALID_BLR
    FROM
        RDB$PROCEDURES
    WHERE
          RDB$PROCEDURES.RDB$PACKAGE_NAME IS NULL
    UNION ALL
    SELECT
        'Function',
        RDB$FUNCTION_NAME,
        RDB$VALID_BLR
    FROM
        RDB$FUNCTIONS
    WHERE
          RDB$FUNCTIONS.RDB$PACKAGE_NAME IS NULL
    UNION ALL
    SELECT
        'Package',
        RDB$PACKAGE_NAME,
        RDB$VALID_BODY_FLAG
    FROM
        RDB$PACKAGES
    UNION ALL
    SELECT
        'Trigger',
        RDB$TRIGGER_NAME,
```

```
        RDB$VALID_BLR
    FROM
        RDB$TRIGGERS
    WHERE
          RDB$TRIGGERS.RDB$SYSTEM_FLAG = 0)
SELECT
    PSQL_TYPE,
    ROUTE_NAME,
    VALID
FROM
    VALID_PSQL
WHERE
      EXISTS(SELECT
                 *
             FROM
                 RDB$DEPENDENCIES
             WHERE
                   RDB$DEPENDENT_NAME = VALID_PSQL.ROUTE_NAME
               AND RDB$DEPENDED_ON_NAME = 'MYDOMAIN');

/*
  Замените MYDOMAIN фактическим именем проверяемого
  домена. Используйте заглавные буквы, если
  домен создавался нечувствительным к регистру — в
  противном случае используйте точное написание
  имени домена с учётом регистра
*/
```

Следующий запрос находит процедуры и триггеры, зависящие от определённого столбца таблицы (в примере это столбец 'MYCOLUMN' таблицы 'MYTABLE'), и выводит информацию о состоянии поля RDB$VALID_BLR:

```
WITH VALID_PSQL (
    PSQL_TYPE,
    ROUTE_NAME,
    VALID)
AS (SELECT
        'Procedure',
        RDB$PROCEDURE_NAME,
        RDB$VALID_BLR
    FROM
        RDB$PROCEDURES
    WHERE
          RDB$PROCEDURES.RDB$PACKAGE_NAME IS NULL
    UNION ALL
    SELECT
        'Function',
        RDB$FUNCTION_NAME,
        RDB$VALID_BLR
    FROM
        RDB$FUNCTIONS
    WHERE
          RDB$FUNCTIONS.RDB$PACKAGE_NAME IS NULL
    UNION ALL
    SELECT
```

```
        'Package',
        RDB$PACKAGE_NAME,
        RDB$VALID_BODY_FLAG
    FROM
        RDB$PACKAGES
    UNION ALL
    SELECT
        'Trigger',
        RDB$TRIGGER_NAME,
        RDB$VALID_BLR
    FROM
        RDB$TRIGGERS
    WHERE
          RDB$TRIGGERS.RDB$SYSTEM_FLAG = 0)
SELECT
    PSQL_TYPE,
    ROUTE_NAME,
    VALID
FROM
    VALID_PSQL
WHERE
      EXISTS(SELECT
                 *
             FROM
                 RDB$DEPENDENCIES D
             WHERE
                   D.RDB$DEPENDENT_NAME = VALID_PSQL.ROUTE_NAME
               AND D.RDB$DEPENDED_ON_NAME = 'MYTABLE'
               AND D.RDB$FIELD_NAME = 'MYCOLUMN');

/*
  Замените MYTABLE и MYCOLUMN фактическими именами
  проверяемой таблицы и её столбца.
  Используйте заглавные буквы, если таблица и её
  столбец создавались нечувствительными к регистру —
  в противном случае используйте точное написание
  имени таблицы и её столбца с учётом регистра
*/
```

К сожалению, не все случаи невалидности кода PSQL будут отражены в поле RDB $VALID_BLR. Поэтому после изменения домена или столбца таблицы желательно тщательно проанализировать все процедуры, функции и триггеры, о которых сообщают вышеупомянутые запросы — даже те, которые имеют 1 в столбце "RDB$VALID_BLR".

Обратите внимание на то, что для модулей PSQL, наследованных от более ранних версий Firebird (включая многие системные триггеры, даже если база данных создавалась под версией Firebird 2.1 или выше), поле RDB$VALID_BLR имеет значение NULL. Это не означает, что их BLR является недействительным.

Команды утилиты командной строки isql SHOW PROCEDURES, SHOW FUNCTIONS и SHOW TRIGGERS при выводе информации отмечают звёздочкой модули, у которых поле RDB$VALID_BLR равно 0. Команды SHOW PROCEDURE PROCNAME, SHOW FUNCTION FUNCNAME и SHOW TRIGGER TRIGNAME, выводящие на экран код PSQL модуля, не сигнализируют пользователя о недопустимом BLR.

Замечание о равенстве

Оператор "=", который явно используется во многих условиях соединении и неявно в соединениях именованными столбцами и естественных соединениях, сравнивает только значения со значениями. В соответствии со стандартом SQL, NULL не является значением и, следовательно, два значения NULL не равны и ни неравны друг с другом. Если необходимо, чтобы значения NULL соответствовали друг другу при объединении, используйте оператор IS NOT DISTINCT FROM. Этот оператор возвращает истину, если операнды имеют то же значение, или, если оба они равны NULL.

```
SELECT *
FROM A
JOIN B ON A.id IS NOT DISTINCT FROM B.code
```

Точно так же (крайне редко), в случае соединения по неравенству, используйте оператор IS DISTINCT FROM вместо оператора "<>", если вы хотите чтобы значения NULL отличались от любого значения и два значения NULL считались равными.

```
SELECT *
FROM A
JOIN B ON A.id IS DISTINCT FROM B.code
```

Это замечание об операторах равенства и неравенства применяется повсюду в СУБД Firebird, не только в условия соединения.

Приложение B: Обработка ошибок, коды и сообщения

В данном приложении будут рассмотрены вопросы, касающиеся модулей PSQL, а конкретно, что происходит, если в результате работы происходит исключение, вопросы перехвата исключения и их обработки в выполняемом коде.

Типы исключений

При работе кода PSQL возможны следующие типы исключений:

- *Ошибки SQL* – сообщения SQL, при которых в системе взводится признак ошибки;
- *Внутренние ошибки Firebird*, которые имеют отношение к конкурирующему взаимодействию, данным, метаданным и условиям окружения;
- *Пользовательские исключения*, которые декларируются в базе как постоянные объекты и вызываются в коде PSQL для того, чтобы код, например, соответствовал определённым бизнес правилам.

В коде PSQL исключения обрабатываются при помощи оператора WHEN. Если исключение будет обработано в вашем коде, то вы обеспечите исправление или обход ошибки и позволите продолжить выполнение, — то клиенту не возвращается никакого сообщения об исключении.

Исключение приводит к прекращению выполнения в блоке. Вместо того чтобы передать выполнение на конечный оператор END, теперь процедура отыскивает уровни во вложенных блоках, начиная с блока где была вызвана ошибка, и переходит на внешние блоки, чтобы найти код обработчика, который "знает" о таком исключении. Она отыскивает первый оператор WHEN, который может обработать эту ошибку.

Пользовательские исключения не должны дублировать работу внутренне определённых исключений. Нужно определять в коде пользовательские исключения для использования там, где мы хотим выявлять ошибочные ситуации, которые нарушают *наши* бизнес – правила.

Смотрите также контекстные переменные GDSCODE, SQLCODE, SQLSTATE, таблицы со значениями контекстных переменных для различных видов системных исключений: Коды ошибок SQLSTATE и их описание, Коды ошибок GDSCODE их описание, и SQLCODE; WHEN ... DO, EXCEPTION, CREATE EXCEPTION.

Системные исключения

Исключение представляет собой сообщение, которое генерируется, когда возникает ошибка.

Все обрабатываемые СУБД исключения имеют заранее определённые числовые значение для контекстных переменных и связанные с ними тексты сообщений. Сообщения об ошибке

написаны по умолчанию на английском языке. Существуют и локализованные сборки СУБД, в которых сообщения об ошибках переведены на другие языки.

> **Примечание**
>
> В селективных хранимых процедурах выходные строки, которые уже были получены клиентом в предыдущих циклах FOR SELECT ... DO ... SUSPEND, остаются доступными для клиента, в случае если по мере выборки потом возникло исключение!

См. также: Коды ошибок SQLSTATE и их описание, Коды ошибок GDSCODE их описание, и SQLCODE.

Пользовательские исключения. Создание

В Firebird существует простой синтаксис DDL CREATE EXCEPTION, для создания пользовательских исключений, с текстами сообщений до 1021 символов. Вы можете динамически модифицировать текст сообщения в зависимости от контекста. Имя исключения должно быть уникальным в базе данных среди исключений, а в диалекте 3 — может быть заключено в кавычки; в этом случае имя исключения будет чувствительно к регистру.

Синтаксис:

```
CREATE EXCEPTION exception_name '<message>';

<message> ::= {txt | @n} <message>
```

> **Предупреждение**
>
> Текстовое сообщение хранится в наборе символов NONE.

Примеры:

Пример B.1. Создание пользовательского исключения

```
CREATE EXCEPTION ERROR_REFIN_RATE
  'Пересечение диапазона ставок рефинансирования';
```

Пример B.2. Вызов исключения с динамически формируемым текстом

```
...
EXCEPTION EX_BAD_TYPE 'Неверный тип записи с id ' || new.id;
...
```

Пользовательские исключения. Изменение и удаление

Изменением и/или удалением пользовательских исключений в Firebird может заниматься их владелец или пользователь SYSDBA.

Если исключение используется только в хранимых процедурах, вы можете изменять или удалять его в любое время. Если исключение используется в триггере, вы можете только его изменять и изменять только его текст сообщения.

Исходя из описанного выше, возьмите себе за правило удалять исключение из кода хранимых процедур при удалении исключения. Иначе при выполнении таких процедур возникнет ошибка выполнения по причине отсутствия пользовательского исключения в базе.

Примеры:

Пример B.3. Изменение/удаление исключения

```
DROP EXCEPTION ERROR_REFIN_RATE;

ALTER EXCEPTION ERROR_REFIN_RATE 'Пересечение диапазона';
```

Подсказка

В скриптах на обновление системы группируйте операторы CREATE EXCEPTION вместе для удобства работы и документирования. Рекомендуется использовать в наименовании исключений систему префиксов в соответствии с категориями пользовательских исключений.

Коды ошибок SQLSTATE и их описание

В данной главе приведены коды ошибок для контекстной переменной SQLSTATE и их описания. Коды ошибок SQLSTATE построены следующим образом: пяти символьный код ошибки состоит и SQL класса ошибки (2 символа) и SQL подкласса (3 символа).

В большинстве случаев, коды ошибок SQLCODE не соотносятся с кодами SQLSTATE "один в один". Коды ошибок SQLSTATE соответствуют SQL стандарту. В то же время SQLCODE использовались много лет и в настоящий момент считаются устаревшими. В следующих вервиях поддержка SQLCODE может полностью прекратиться.

Таблица B.1. Коды ошибок SQLSTATE

Код SQLSTATE	Связанное сообщение	Примечание
SQLCLASS 00 (Success)		
0	Success	Успех
SQLCLASS 01 (Warning)		Класс 01 (предупреждения)
1000	General Warning	Общее предупреждение
1001	Cursor operation conflict	Конфликт при операции с курсором
1002	Disconnect error	Ошибка, связанная с разъединением
1003	NULL value eliminated in set function	Значение NULL устраняется в определении функции

Код SQLSTATE	Связанное сообщение	Примечание
1004	String data, right-truncated	Строковые данные, обрезание справа
1005	Insufficient item descriptor areas	Недостаточно элементов в области дескрипторов
1006	Privilege not revoked	Привилегии не отозваны
1007	Privilege not granted	Привилегии не выданы
1008	Implicit zero-bit padding	Неявное обрезание нулевого бита
1100	Statement reset to unprepared	Оператор сброшен в состояние unprepared
1101	Ongoing transaction has been committed	Текущая транзакция завершена COMMIT
1102	Ongoing transaction has been rolled back	Текущая транзакция завершена ROLLED BACK
SQLCLASS 02 (No Data)		Класс ошибок 02 (Нет данных)
2000	No data found or no rows affected	Данные не обнаружены или не затронуты строки
SQLCLASS 07 (Dynamic SQL error)		Класс ошибок 07 (Ошибки DSQL)
7000	Dynamic SQL error	Ошибка DSQL
7001	Wrong number of input parameters	Неверное число входных параметров
7002	Wrong number of output parameters	Неверное число выходных параметров
7003	Cursor specification cannot be executed	Определение курсора не может быть выполнено
7004	USING clause required for dynamic parameters	Для динамического параметра требуется предложение USING
7005	Prepared statement not a cursor-specification	Подготовленный оператор не является курсор - специфичным
7006	Restricted data type attribute violation	Исключение по причине запрещенного типа данных для атрибута
7007	USING clause required for result fields	Для возвращаемого поля требуется предложение USING
7008	Invalid descriptor count	Неверный счетчик дескрипторов
7009	Invalid descriptor index	Неверный индекс дескриптора
SQLCLASS 08 (Connection Exception)		Класс ошибок 08 (Исключения коннекта)

Код SQLSTATE	Связанное сообщение	Примечание
8001	Client unable to establish connection	Клиент не может установить соединение
8002	Connection name in use	Имя соединения уже используется
8003	Connection does not exist	Соединение не существует
8004	Server rejected the connection	Сервер отверг подключение
8006	Connection failure	Ошибка при подключении
8007	Transaction resolution unknown	Неизвестно разрешение транзакции
SQLCLASS 0A (Feature Not Supported)		Класс ошибок 0A (Возможность не поддерживается)
0A000	Feature Not Supported	Возможность (конструкция) не поддерживается
SQLCLASS 0B (Invalid Transaction Initiation)		Класс ошибок 0B (неверная инициализация транзакции)
0B000	Invalid transaction initiation	Неверная инициализация транзакции
SQLCLASS 0L (Invalid Grantor)		Неверный грантодатель
0L000	Invalid grantor	Неверный грантодатель
SQLCLASS 0P (Invalid Role Specification)		Класс ошибок 0P (неверная спецификация роли)
0P000	Invalid role specification	Неверная спецификация роли
SQLCLASS 0U (Attempt to Assign to Non-Updatable Column)		Класс ошибок 0U (попытка присвоения не обновляемому столбцу)
0U000	Attempt to assign to non-updatable column	Попытка присвоения не обновляемому столбцу
SQLCLASS 0V (Attempt to Assign to Ordering Column)		Класс ошибок 0V (попытка присвоения сортируемому столбцу)
0V000	Attempt to assign to Ordering column	Попытка присвоения сортируемому столбцу
SQLCLASS 20 (Case Not Found For Case Statement)		Класс 20 (не обнаружено вариантов для предложения CASE)
20000	Case not found for case statement	Не обнаружено вариантов для предложения CASE
SQLCLASS 21 (Cardinality Violation)		Класс 21 (Нарушения определения)
21000	Cardinality violation	Нарушение определения

Код SQLSTATE	Связанное сообщение	Примечание
21S01	Insert value list does not match column list	Список вставляемых значений не соответствует списку столбцов
21S02	Degree of derived table does not match column list	Состояние производной таблицы не соответствует списку столбцов
SQLCLASS 22 (Data Exception)		Класс ошибок 22 (исключения, вызванные данными)
22000	Data exception	Исключения данных
22001	String data, right truncation	Строковые данные, усечены справа
22002	Null value, no indicator parameter	Значение NULL, параметр не обозначен
22003	Numeric value out of range	Числовое значение вышло за предел допустимого
22004	Null value not allowed	Значение NULL не допустимо
22005	Error in assignment	Ошибка присваивания
22006	Null value in field reference	Значение NULL в поле ссылки
22007	Invalid datetime format	Неверный формат даты/времени
22008	Datetime field overflow	Переполнение в поле даты/времени
22009	Invalid time zone displacement value	Неверная временная зона, неверное значение
2200A	Null value in reference target	Значение NULL в целевой ссылке
2200B	Escape character conflict	Конфликт символа управления
2200C	Invalid use of escape character	Неверное использование управляющего символа
2200D	Invalid escape octet	Неверный октет для управляющего символа
2200E	Null value in array target	Значение NULL в массиве назначения
2200F	Zero-length character string	Нулевая длина строки символов
2200G	Most specific type mismatch	Наиболее определенное несоответствие типов
22010	Invalid indicator parameter value	Неверный индикатор значения параметра
22011	Substring error	Ошибка подстроки
22012	Division by zero	Деление на ноль

Код SQLSTATE	Связанное сообщение	Примечание
22014	Invalid update value	Неверное значение в операции update
22015	Interval field overflow	Переполнение интервала в поле
22018	Invalid character value for cast	Неверный символ для преобразования типов
22019	Invalid escape character	Неверный символ управления
2201B	Invalid regular expression	Неверное регулярное выражение
2201C	Null row not permitted in table	Запись содержащая NULL не допустима для таблицы
22020	Invalid limit value	Неверное значение лимита
22021	Character not in repertoire	Символ вне диапазона
22022	Indicator overflow	Переполнение индикатора
22023	Invalid parameter value	Неверное значение параметра
22024	Character string not properly terminated	Символьная строка имеет некорректный замыкающий символ
22025	Invalid escape sequence	Неверная управляющая последовательность
22026	String data, length mismatch	Строковые данные, длина неверная
22027	Trim error	Ошибка операции TRIM
22028	Row already exists	Строка уже существует
2202D	Null instance used in mutator function	NULL экземпляр используется для мутирующей функции
2202E	Array element error	Ошибка элемента массива
2202F	Array data, right truncation	Данные массива, обрезание справа
SQLCLASS 23 (Integrity Constraint Violation)		Класс ошибок 23 (Нарушение ограничения целостности)
23000	Integrity constraint violation	Нарушение ограничения целостности
SQLCLASS 24 (Invalid Cursor State)		Класс ошибок 24 (неверное состояние курсора)
24000	Invalid cursor state	Неверное состояние курсора
24504	The cursor identified in the UPDATE, DELETE, SET, or GET statement is not positioned on a row	Курсор определенный для UPDATE, DELETE, SET или GET операции не позиционирован по строке

Код SQLSTATE	Связанное сообщение	Примечание
SQLCLASS 25 (Invalid Transaction State)		Класс ошибок 25 (неверное состояние транзакции)
25000	Invalid transaction state	Неверное состояние транзакции
25S01	Transaction state	Неверное состояние транзакции
25S02	Transaction is still active	Транзакция до сих пор активная
25S03	Transaction is rolled back	Транзакция откачена
SQLCLASS 26 (Invalid SQL Statement Name)		Класс ошибок 26 (неверное имя SQL предложения)
26000	Invalid SQL statement name	Неверное имя SQL предложения
SQLCLASS 27 (Triggered Data Change Violation)		Класс ошибок 27 (ошибки изменения данных триггером)
27000	Triggered data change violation	Ошибки изменения данных триггером
SQLCLASS 28 (Invalid Authorization Specification)		Класс ошибок 28 (неверная спецификация авторизации)
28000	Invalid authorization specification	Неверная спецификация авторизации
SQLCLASS 2B (Dependent Privilege Descriptors Still Exist)		Класс ошибок 2B (зависимые описания привилегий еще существуют)
2B000	Dependent privilege descriptors still exist	Зависимые описания привилегий еще существуют
SQLCLASS 2C (Invalid Character Set Name)		Класс ошибок 2C (неверное имя набора символов)
2C000	Invalid character set name	Неверное имя набора символов
SQLCLASS 2D (Invalid Transaction Termination)		Класс ошибок 2D (неверное завершение транзакции)
2D000	Invalid transaction termination	Неверное завершение транзакции
SQLCLASS 2E (Invalid Connection Name)		Класс ошибок 2E (неверное имя соединения)
2E000	Invalid connection name	Неверное имя соединения
SQLCLASS 2F (SQL Routine Exception)		Класс ошибок 2F (процедурные исключения SQL)
2F000	SQL routine exception	Процедурное исключение SQL
2F002	Modifying SQL-data not permitted	На модификацию SQL данных нет доступа

Код SQLSTATE	**Связанное сообщение**	**Примечание**
2F003	Prohibited SQL-statement attempted	Встретилось запрещенное SQL предложение
2F004	Reading SQL-data not permitted	Нет доступа на чтение SQL данных
2F005	Function executed no return statement	Исполняемая функция не имеет возвращаемого выражения
SQLCLASS 33 (Invalid SQL Descriptor Name)		Класс ошибок 33 (неверное имя SQL описания)
33000	Invalid SQL descriptor name	Неверное имя SQL описания
SQLCLASS 34 (Invalid Cursor Name)		Класс ошибок 34 (неверное имя курсора)
34000	Invalid cursor name	Неверное имя курсора
SQLCLASS 35 (Invalid Condition Number)		Класс ошибок 35 (неверный номер условия)
35000	Invalid condition number	Неверный номер условия
SQLCLASS 36 (Cursor Sensitivity Exception)		Класс ошибок 36 (ошибка восприятия курсора)
36001	Request rejected	Запрос отвергнут
36002	Request failed	Запрос ошибочный
SQLCLASS 37 (Invalid Identifier)		Класс ошибок 37 (неверный идентификатор)
37000	Invalid identifier	Неверный идентификатор
37001	Identifier too long	Идентификатор слишком длинный
SQLCLASS 38 (External Routine Exception)		Класс ошибок 38 (ошибки внешних процедур)
38000	External routine exception	Ошибка внешней процедуры
SQLCLASS 39 (External Routine Invocation Exception)		Класс ошибок 39 (ошибка вызова внешней процедуры)
39000	External routine invocation exception	Ошибка вызова внешней процедуры
SQLCLASS 3B (Invalid Save Point)		Класс ошибок 3B (неверная точка сохранения)
3B000	Invalid save point	Неверная точка сохранения
SQLCLASS 3C (Ambiguous Cursor Name)		Класс ошибок 3C (имя курсора неоднозначное)
3C000	Ambiguous cursor name	Имя курсора неоднозначное

Код SQLSTATE	Связанное сообщение	Примечание
SQLCLASS 3D (Invalid Catalog Name)		Класс ошибок 3D (неверное имя каталога)
3D000	Invalid catalog name	Неверное имя каталога
3D001	Catalog name not found	Каталог с таким именем не обнаружен
SQLCLASS 3F (Invalid Schema Name)		Класс ошибок 3F (неверное имя схемы)
3F000	Invalid schema name	Неверное имя схемы
SQLCLASS 40 (Transaction Rollback)		Класс ошибок 40 (откат транзакции)
40000	Ongoing transaction has been rolled back	Текущая транзакция была откачена
40001	Serialization failure	Отказ сериализации
40002	Transaction integrity constraint violation	Нарушение условия целостности транзакции
40003	Statement completion unknown	Неизвестно состояние завершения транзакции
SQLCLASS 42 (Syntax Error or Access Violation)		Класс ошибок 42 (синтаксическая ошибка или ошибка доступа)
42000	Syntax error or access violation	Синтаксическая ошибка или ошибка доступа
42702	Ambiguous column reference	Неоднозначная ссылка на столбец
42725	Ambiguous function reference	Неоднозначная ссылка на функцию
42818	The operands of an operator or function are not compatible	Операнды оператора или функции являются не совместимыми
42S01	Base table or view already exists	Таблица в базе или view уже существует
42S02	Base table or view not found	Таблица в базе или view не найдена
42S11	Index already exists	Индекс уже существует
42S12	Index not found	Индекс не найден
42S21	Column already exists	Столбец уже существует
42S22	Column not found	Столбец не найден
SQLCLASS 44 (With Check Option Violation)		Класс ошибок 44 (нарушение опции WITH CHECK)
44000	WITH CHECK OPTION Violation	Нарушение опции WITH CHECK

Код SQLSTATE	Связанное сообщение	Примечание
SQLCLASS 45 (Unhandled User-defined Exception)		Класс ошибок 45 (необработанное исключение определенное пользователем)
45000	Unhandled user-defined exception	Необработанное исключение, определенное пользователем
SQLCLASS 54 (Program Limit Exceeded)		Класс ошибок 54 (превышены ограничения программы)
54000	Program limit exceeded	Превышены ограничения программы
54001	Statement too complex	Выражение слишком сложное
54011	Too many columns	Слишком много столбцов
54023	Too many arguments	Слишком много аргументов
SQLCLASS HY (CLI-specific Condition)		Класс ошибок HY (условия CLI-specific)
HY000	CLI-specific condition	Условия CLI-specific
HY001	Memory allocation error	Ошибка выделения памяти
HY003	Invalid data type in application descriptor	Неверный тип данных в дескрипторе приложения
HY004	Invalid data type	Неверный тип данных
HY007	Associated statement is not prepared	Связанный оператор не подготовлен
HY008	Operation canceled	Операция отменена
HY009	Invalid use of null pointer	Неправильное использование нулевого указателя
HY010	Function sequence error	Ошибка последовательности функций
HY011	Attribute cannot be set now	Атрибут не может быть установлен сейчас
HY012	Invalid transaction operation code	Неверный код транзакции операции
HY013	Memory management error	Ошибка управления памятью
HY014	Limit on the number of handles exceeded	Достигнут лимит числа указателей
HY015	No cursor name available	Недоступен курсор без имени
HY016	Cannot modify an implementation row descriptor	Невозможно изменить реализацию дескриптора строки

Код SQLSTATE	Связанное сообщение	Примечание
HY017	Invalid use of an automatically allocated descriptor handle	Неверное использование автоматически выделяемого дескриптора указателей
HY018	Server declined the cancellation request	Сервер отклонил запрос на отмену
HY019	Non-string data cannot be sent in pieces	Не строковые данные не могут быть отправлены по кускам
HY020	Attempt to concatenate a null value	Попытка конкатенации значения NULL
HY021	Inconsistent descriptor information	Противоречивая информация о дескрипторе
HY024	Invalid attribute value	Неверное значение атрибута
HY055	Non-string data cannot be used with string routine	Не строковые данные не могут быть использованы со строковой процедурой
HY090	Invalid string length or buffer length	Неверная длина строки или длина буфера
HY091	Invalid descriptor field identifier	Неверный дескриптор идентификатора поля
HY092	Invalid attribute identifier	Неверный идентификатор атрибута
HY095	Invalid FunctionId specified	Неверное указание ID функции
HY096	Invalid information type	Неверный тип информации
HY097	Column type out of range	Тип столбца вне диапазона
HY098	Scope out of range	Определение вне диапазона
HY099	Nullable type out of range	Типы с допустимыми NULL вне диапазона
HY100	Uniqueness option type out of range	Тип опции "уникальность" вне диапазона
HY101	Accuracy option type out of range	Тип опции "точность" вне диапазона
HY103	Invalid retrieval code	Неверный код поиска
HY104	Invalid LengthPrecision value	Неверное значение длина/точность
HY105	Invalid parameter type	Неверный тип параметра
HY106	Invalid fetch orientation	Неверное направление для fetch
HY107	Row value out of range	Значение строки вне диапазона
HY109	Invalid cursor position	Неверная позиция курсора

Код SQLSTATE	Связанное сообщение	Примечание
HY110	Invalid driver completion	Неверный код завершения драйвера
HY111	Invalid bookmark value	Неверное значение метки bookmark
HYC00	Optional feature not implemented	Опциональная функция не реализована
HYT00	Timeout expired	Достигнут тайм-аут
HYT01	Connection timeout expired	Достигнут тайм-аут соединения
SQLCLASS XX (Internal Error)		SQLCLASS XX (внутренние ошибки)
XX000	Internal error	Внутренняя ошибка
XX001	Data corrupted	Данные разрушены
XX002	Index corrupted	Индекс разрушен

Коды ошибок GDSCODE их описание, и SQLCODE

Таблица ошибок содержит числовое и символьное значения GDSCODE, текст сообщения об ошибке и описание ошибки. Также приводится SQLCODE ошибки.

В настоящее время SQLCODE считаются устаревшим. В следующих версиях поддержка SQLCODE может полностью прекратиться.

Таблица B.2. Коды ошибок GDSCODE, SQLCODE и их описание

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
101	335544366	Segment	Segment buffer length shorter than expected. Длина буфера сегмента меньше, чем ожидается.
100	335544338	from_no_match	No match for first value expression. Нет соответствия для первого значения выражения.
100	335544354	no_record	Invalid database key. Неверный ключ базы данных.
100	335544367	segstr_eof	Attempted retrieval of more segments than exist.

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			Попытка обращения к сегменту большему чем их существует.
100	335544374	stream_eof	Attempt to fetch past the last record in a record stream. Попытка получения в потоке записей записи, следующей за последней.
0	335741039	gfix_opt_SQL_dialect	-sql_dialect set database dialect n.
0	335544875	bad_debug_format	Bad debug info format. Неверный формат отладочной информации.
-84	335544554	nonsql_security_rel	Table/procedure has non-SQL security class defined. Для таблицы/процедуры определен НЕ-SQL класс безопасности.
-84	335544555	nonsql_security_fld	Column has non-SQL security class defined. Для столбца определен НЕ-SQL класс безопасности.
-84	335544668	dsql_procedure_use_err	Procedure @1 does not return any values. Процедура @1 не возвращает никакого значения.
-85	335544747	usrname_too_long	The username entered is too long. Maximum length is 31 bytes. Введенноей имя пользователя очень длинное. Максимальная длина 31 байт.
-85	335544748	password_too_long	The password specified is too long. Maximum length is @1 bytes. Введенный пароль очень длинный. Максимальная длина 8 байт.

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
-85	335544749	usrname_required	A username is required for this operation. Для этой операции требуется имя пользователя.
-85	335544750	password_required	A password is required for this operation. Для этой операции требуется пароль.
-85	335544751	bad_protocol	The network protocol specified is invalid. Указан неверный сетевой протокол.
-85	335544752	dup_usrname_found	A duplicate user name was found in the security database. В базе данных безопасности обнаружено дублирование имен пользователей.
-85	335544753	usrname_not_found	The user name specified was not found in the security database. Указанное имя пользователя не найдено в базе данных безопасности.
-85	335544754	error_adding_sec_record	An error occurred while attempting to add the user. Ошибка произошла при попытке добавления пользователя.
-85	335544755	error_modifying_sec_record	An error occurred while attempting to modify the user record. Ошибка произошла при попытке редактирования записи о пользователе.
-85	335544756	error_deleting_sec_record	An error occurred while attempting to delete the user record. Ошибка произошла при попытке удаления записи о пользователе.

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
-85	335544757	error_updating_sec_db	An error occurred while updating the security database. Ошибка произошла при изменении базы данных безопасности.
-103	335544571	dsql_constant_err	Data type for constant unknown. Неизвестный тип данных для константы.
-104	336003075	dsql_transitional_numeric	Precision 10 to 18 changed from DOUBLE PRECISION in SQL dialect 1 to 64-bit scaled integer in SQL dialect 3. Точность от 10 до 18 в SQL диалекте 1 изменена для DOUBLE PRECISION до 64 битного масштабируемого целого в диалекте 3.
-104	336003077	sql_db_dialect_dtype_unsupport	Database SQL dialect @1 does not support reference to @2 datatype. База данных SQL с диалектом @1 не поддерживает ссылку на @2 тип данных.
-104	336003087	dsql_invalid_label	Label @1 @2 in the current scope. Метка @1 @2 находится в текущей зоне видимости.
-104	336003088	dsql_datatypes_not_comparable	Datatypes @1are not comparable in expression @2. Тип данных @1 не сравним в выражении @2.
-104	335544343	invalid_blr	Invalid request BLR at offset @1. Неверный запрос BLR со смещением @1.
-104	335544390	syntaxerr	BLR syntax error: expected @1 at offset @2, encountered @3.

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
			Ошибка синтаксиса BLR: ожидается @1 по смещению @2, встречено @3.
-104	335544425	ctxinuse	Context already in use (BLR error). Контекст находится в использовании (ошибка BLR).
-104	335544426	ctxnotdef	Context not defined (BLR error). Контекст не определен (ошибка BLR).
-104	335544429	badparnum	Bad parameter number. Неверный номер параметра.
-104	335544440	bad_msg_vec	-
-104	335544456	invalid_sdl	Invalid slice description language at offset @1. Неверный фрагмент языка описания по смещению @1.
-104	335544570	dsql_command_err	Invalid command. Неверная команда.
-104	335544579	dsql_internal_err	Internal error. Внутренняя ошибка.
-104	335544590	dsql_dup_option	Option specified more than once. Режим указан более одного раза.
-104	335544591	dsql_tran_err	Unknown transaction option. Неизвестный режим транзакции.
-104	335544592	dsql_invalid_array	Invalid array reference. Неверная ссылка на массив.
-104	335544608	command_end_err	Unexpected end of command. Неожиданное завершение команды.
-104	335544612	token_err	Token unknown.

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
			Неизвестный синтаксический элемент.
-104	335544634	dsql_token_unk_err	Token unknown - line @1, column @2. Неизвестный синтаксический элемент – строка @1, символ @2.
-104	335544709	dsql_agg_ref_err	Invalid aggregate reference. Неверная ссылка на агрегат.
-104	335544714	invalid_array_id	Invalid blob id. Неверный идентификатор BLOB.
-104	335544730	cse_not_supported	Client/Server Express not supported in this release. Client/Server Express не поддерживается в этом релизе.
-104	335544743	token_too_long	Token size exceeds limit. Размер синтаксического элемента превышает предел.
-104	335544763	invalid_string_constant	A string constant is delimited by double quotes. Строковая константа определена в кавычках.
-104	335544764	transitional_date	DATE must be changed to TIMESTAMP. DATE должно измениться TIMESTAMP.
-104	335544796	sql_dialect_datatype_unsupport	Client SQL dialect @1 does not support reference to @2 datatype. SQL диалект @1 клиента не поддерживает ссылку на тип данных @2.
-104	335544798	depend_on_uncommitted_rel	You created an indirect dependency on uncommitted metadata. You must roll back the current transaction.

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
			Вы создали непрямую зависимость на неподтвержденные метаданные. Вы должны отменить текущую транзакцию.
-104	335544821	dsql_column_pos_err	Invalid column position used in the @1 clause. В предложении @1 используется неверная позиция столбца.
-104	335544822	dsql_agg_where_err	Cannot use an aggregate function in a WHERE clause, use HAVING instead. Невозможно использовать агрегатную функцию в предложении WHERE, заместо этого используйте HAVING.
-104	335544823	dsql_agg_group_err	Cannot use an aggregate function in a GROUP BY clause. Невозможно использовать агрегатную функцию в кляузе GROUP BY.
-104	335544824	dsql_agg_column_err	Invalid expression in the @1 (not contained in either an aggregate function or the GROUP BY clause). Неверное выражение в @1 (ни содержится ни в агрегатной функции ни в кляузе GROUP BY).
-104	335544825	dsql_agg_having_err	Invalid expression in the @1 (neither an aggregate function nor a part of the GROUP BY clause). Неверное выражение в @2 (не агрегатная функция, ни часть кляузы GROUP BY).
-104	335544826	dsql_agg_nested_err	Nested aggregate functions are not allowed. Вложенные агрегатные функции не допустимы.
-104	335544849	malformed_string	Malformed string.

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
			Искаженная (некорректная) строка.
-104	335544851	command_end_err2	Unexpected end of command- line @1, column @2. Неожиданный конец команд – линия @1, колонка @2.
-104	336397215	dsql_max_sort_items	Cannot sort on more than 255 items. Не могу сортировать условие более чем из 255 элементов.
-104	336397216	dsql_max_group_items	Cannot group on more than 255 items. Не могу группировать более чем 255 элементов.
-104	336397217	dsql_conflicting_sort_field	Cannot include the same field (@1.@2) twice in the ORDER BY clause with conflicting sorting options. Не могу включить такое же поле (@1.@2) дважды в кляузу ORDER BY с противоречивыми параметрами сортировки.
-104	336397218	dsql_derived_table_more_columns	Column list from derived table @1 has more columns than the number of items in its SELECT statement. Список столбцов в производной таблице @1 содержит больше столбцов, чем количество элементов в SELECT.
-104	336397219	dsql_derived_table_less_columns	Column list from derived table @1 has less columns than the number of items in its SELECT statement. Список столбцов из производной таблицы @1 имеет меньше столбцов, чем количество элементов в SELECT.

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
-104	336397220	dsql_derived_field_unnamed	No column name specified for column number @1 in derived table @2. Нет имени столбца указанного для столбца под номером @1 в производной таблице @2.
-104	336397221	dsql_derived_field_dup_name	Column @1 was specified multiple times for derived table @2. Столбец @1 был указан несколько раз для производной таблицы @2.
-104	336397222	dsql_derived_alias_select	Internal dsql error: alias type expected by pass1_expand_select_node. Внутренняя ошибка DSQL: тип псевдоним ожидался pass1_expand_select_node.
-104	336397223	dsql_derived_alias_field	Internal dsql error: alias type expected by pass1_field. Внутренняя ошибка DSQL: тип псевдоним ожидался pass1_field.
-104	336397224	dsql_auto_field_bad_pos	Internal dsql error: column position out of range in pass1_union_auto_cast. Внутренняя ошибка DSQL: позиция столбца вышла из диапазона в pass1_union_auto_cast.
-104	336397225	dsql_cte_wrong_reference	Recursive CTE member (@1) can refer itself only in FROM clause. Рекурсивная часть CTE (@1) может ссылаться сама на себя только в предложении FROM.
-104	336397226	dsql_cte_cycle	CTE '@1' has cyclic dependencies. CTE '@1' имеет циклические зависимости.

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
-104	336397227	dsql_cte_outer_join	Recursive member of CTE can't be member of an outer join. Рекурсивная часть CTE не может являться членом outer join.
-104	336397228	dsql_cte_mult_references	Recursive member of CTE can't reference itself more than once. Рекурсивный член CTE не может ссылаться на себя более одного раза.
-104	336397229	dsql_cte_not_a_union	Recursive CTE (@1) must be an UNION. Рекурсивный CTE (@1) должен содержать UNION.
-104	336397230	dsql_cte_nonrecurs_after_recurs	CTE '@1' defined non-recursive member after recursive. В CTE '@1' определена не рекурсивная часть после рекурсии.
-104	336397231	dsql_cte_wrong_clause	Recursive member of CTE '@1' has @2 clause. Рекурсивная часть CTE '@1' содержит предложение @2.
-104	336397232	dsql_cte_union_all	Recursive members of CTE (@1) must be linked with another members via UNION ALL. Рекурсивная часть CTE (@1) должна быть связана с остальными частями через UNION ALL.
-104	336397233	dsql_cte_miss_nonrecursive	Non-recursive member is missing in CTE '@1'. Не рекурсивная часть отсутствует в CTE '@1'.
-104	336397234	dsql_cte_nested_with	WITH clause can't be nested. Предложение WITH не может быть вложенным.

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
-104	336397235	dsql_col_more_than_once_using	Column @1 appears more than once in USING clause. Столбец @1 используется более одного раза в предложении USING.
-104	336397237	dsql_cte_not_used	CTE "@1" is not used in query. CTE "@1" не используется в запросе.
-105	335544702	like_escape_invalid	Invalid ESCAPE sequence. Неверная ESCAPE последовательность.
-105	335544789	extract_input_mismatch	Specified EXTRACT part does not exist in input datatype. Указанная часть-параметр для EXTRACT не существует для входного типа данных.
-150	335544360	read_only_rel	Attempted update of read-only table. Попытка обновления таблицы только для чтения.
-150	335544362	read_only_view	Cannot update read-only view @1. Не могу обновить представление @1 только для чтения.
-150	335544446	non_updatable	Not updatable. Не обновляемое.
-150	335544546	constaint_on_view	Cannot define constraints on views. Не могу определять констрейны на view.
-151	335544359	read_only_field	Attempted update of read - only column. Попытка обновить доступный только для чтения столбец.
-155	335544658	dsql_base_table	@1 is not a valid base table of the specified view.

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			@1 не является верной базовой таблицей для указанного view.
-157	335544598	specify_field_err	Must specify column name for view select expression. Вы обязаны указать имя столбца для выражения выборки view.
-158	335544599	num_field_err	Number of columns does not match select list. Число столбцов не соответствует списку выборки.
-162	335544685	no_dbkey	Dbkey not available for multi - table views. Значение Dbkey не доступно для мультитабличных view.
-170	335544512	prcmismat	Input parameter mismatch for procedure @1. Входные параметры не соответствуют для процедуры @1.
-170	335544619	extern_func_err	External functions cannot have morethan 10 parametrs. UDF не может иметь более чем 10 параметров.
-170	335544850	prc_out_param_mismatch	Output parameter mismatch for procedure @1. Несоответствующие входные параметры для процедуры @1.
-171	335544439	funmismat	Function @1 could not be matched. Функция @1 не может быть согласована.
-171	335544458	invalid_dimension	Column not array or invalid dimensions (expected @1, encountered @2). Столбец не является массивом или неверная размерность

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
			(ожидается @1, встретилась @2).
-171	335544618	return_mode_err	Return mode by value not allowed for this data type. Режим возврата по значению для допускается для этого типа данных.
-171	335544873	array_max_dimensions	Array data type can use up to @1 dimensions. Тип данных массив не может использовать свыше @1 размерностей.
-172	335544438	funnotdef	Function @1 is not defined. Функция @1 не определена.
-203	335544708	dyn_fld_ambiguous	Ambiguous column reference. Неоднозначная ссылка на столбец.
-204	336003085	dsql_ambiguous_field_name	Ambiguous field name between @1 and @2. Неоднозначное имя поля между @1 и @2.
-204	335544463	gennotdef	Generator @1 is not defined. Генератор @1 не определен.
-204	335544502	stream_not_defined	Reference to invalid stream number. Ссылка на неверный номер потока.
-204	335544509	charset_not_found	CHARACTER SET @1 is not defined. Набор символов @1 не определен.
-204	335544511	prcnotdef	Procedure @1 is not defined. Процедура @1 не определена.
-204	335544515	codnotdef	Status code @1 unknown.

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
			Код статуса @1 неизвестен.
-204	335544516	xcpnotdef	Exception @1 not defined. Пользовательское исключение @1 не определено.
-204	335544532	ref_cnstrnt_notfound	Name of Referential Constraint not defined in constraints table. Имя ссылочного ограничения не определено в таблице ограничений.
-204	335544551	grant_obj_notfound	Could not find table/procedure for GRANT. Для операции GRANT не могу найти таблицу / процедуру.
-204	335544568	text_subtype	Implementation of text subtype @1 not located. Реализация текстового подтипа @1 не обнаружена.
-204	335544573	dsql_datatype_err	Data type unknown. Неизвестный тип данных.
-204	335544580	dsql_relation_err	Table unknown. Неизвестная таблица.
-204	335544581	dsql_procedure_err	Procedure unknown. Неизвестная процедура.
-204	335544588	collation_not_found	COLLATION @1 for CHARACTER SET @2 is not defined. Тип сортировки @1 для набора символов @2 не определен.
-204	335544589	collation_not_for_charset	COLLATION @1 is not valid for specified CHARACTER SET. Тип сортировки @1 не верная для указанного набора символов.
-204	335544595	dsql_trigger_err	Trigger unknown.

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
			Триггер неизвестен.
-204	335544620	alias_conflict_err	Alias @1 conflicts with an alias in the same statement. Алиас @1 конфликтует с алиасом в том же выражении.
-204	335544621	procedure_conflict_error	Alias @1 conflicts with a procedure in the same statement. Алиас @1 конфликтует в процедурой в том же выражении.
-204	335544622	relation_conflict_err	Alias @1 conflicts with a table in the same statement. Алиас @2 конфликтует с таблицей в том же выражении.
-204	335544635	dsql_no_relation_alias	There is no alias or table named @1 at this scope level. Там нет алиаса или так именуется таблица на этом уровне области.
-204	335544636	indexname	There is no index @1 for table @2. Там нет индекса @1 для таблицы @2.
-204	335544640	collation_requires_text	Invalid use of CHARACTER SET or COLLATE. Неверное использование набора символов или типа сортировки.
-204	335544662	dsql_blob_type_unknown	BLOB SUB_TYPE @1 is not defined. Подтип BLOB @1 не определен.
-204	335544759	bad_default_value	Can not define a not null column with NULL as default value. Не могу описать not null столбец при значении по умолчанию NULL.
-204	335544760	invalid_clause	Invalid clause - '@1'.

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
			Неверная кляуза - '@1'.
-204	335544800	too_many_contexts	Too many Contexts of Relation/ Procedure/Views. Maximum allowed is 255. В контексте слишком большое количество таблиц/процедур/ view. Максимально допустимое количество 255.
-204	335544817	bad_limit_param	Invalid parameter to FIRST.Only integers >= 0 are allowed. Неверный параметр для FIRST. Возможны только целые числа >=0.
-204	335544818	bad_skip_param	Invalid parameter to SKIP. Only integers >= 0 are allowed. Неверный параметр для SKIP. Возможны только целые числа >=0.
-204	335544837	bad_substring_offset	Invalid offset parameter @1 to SUBSTRING. Only positive integers are allowed. Неверный параметр смещения @1 для SUBSTRING. Возможны только положительные целые числа.
-204	335544853	bad_substring_length	Invalid length parameter @1 to SUBSTRING. Negative integers are not allowed. Неверный параметр длины @1 для SUBSTRING. Отрицательные целые числа не доступны.
-204	335544854	charset_not_installed	CHARACTER SET @1 is not installed. Набор символов @1 не установлен.
-204	335544855	collation_not_installed	COLLATION @1 for CHARACTER SET @2 is not installed.

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			Сортировка @1 для набора символов @2 не установлена.
-204	335544867	subtype_for_internal_use	Blob sub _ types bigger than 1 (text) are for internal use only. Подтип BLOB со значением более чем 1 (TEXT) предназначены только для внутреннего использования.
-205	335544396	fldnotdef	Column @1 is not defined in table @2. Столбец @1 не определен в таблице @2.
-205	335544552	grant_fld_notfound	Could not find column for GRANT. Не могу найти строку для операции GRANT.
-205	335544883	fldnotdef2	Column @1 is not defined in procedure @2. Столбец @1 не определен в процедуре @2.
-206	335544578	dsql_field_err	Column unknown. Столбец неизвестен.
-206	335544587	dsql_blob_err	Column is not a BLOB. Столбец не является BLOB.
-206	335544596	dsql_subselect_err	Subselect illegal in this context. В данном контексте подзапрос запрещен.
-206	336397208	dsql_line_col_error	At line @1, column @2. В строке @1, колонка @2.
-206	336397209	dsql_unknown_pos	At unknown line and column. В неизвестных строке и столбце.
-206	336397210	dsql_no_dup_name	Column @1 cannot be repeated in @2 statement.

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
			Столбец @1 не может быть повторен в выражении @2.
-208	335544617	order_by_err	Invalid ORDER BY clause. Неверное предложение ORDER BY.
-219	335544395	relnotdef	Table @1 is not defined. Таблица @1 не определена.
-219	335544872	domnotdef	Domain @1 is not defined. Домен @1 не определен.
-230	335544487	walw_err	WAL Writer error. Ошибка записи WAL.
-231	335544488	logh_small	Log file header of @1 too small. Заголовок файла лога @1 слишком мал.
-232	335544489	logh_inv_version	Invalid version of log file @1. Неверная версия файла лога @1.
-233	335544490	logh_open_flag	Log file @1 not latest in the chain but open flag still set. Лог файл @1 не последний в цепочке, но флаг «открыт» еще установлен.
-234	335544491	logh_open_flag2	Log file @1 not closed properly; database recovery may be required. Лог файл @1 не верно закрыт; может понадобится операция восстановления базы данных.
-235	335544492	logh_diff_dbname	Database name in the log file @1 is different. Имя базы данный в файле лога @1 отличается.
-236	335544493	logf_unexpected_eof	Unexpected end of log file @1 at offset @2.

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
			Неожиданное окончание фала лога @1 по смещению @2.
-237	335544494	logr_incomplete	Incomplete log record at offset @1 in log file @2. Неполная запись лога по смещению @1 в файл лога @2.
-238	335544495	logr_header_small	Log record header too small at offset @1 in log file @2. Заголовок записи лога слишком мал по смещению @1 в файле лога @2.
-239	335544496	logb_small	Log block too small at offset @1 in log file @2. Блок лога слишком мал по смещению @1 в файле лога @2.
-239	335544691	cache_too_small	Insufficient memory to allocate page buffer cache. Недостаточно памяти для размещения страниц буфера кэш.
-239	335544693	log_too_small	Log size too small. Размер лога слишком мал.
-239	335544694	partition_too_small	Log partition size too small. Размер раздела лога слишком мал.
-243	335544500	no_wal	Database does not use Write-ahead Log. База данных не использует запись с упреждением лога.
-257	335544566	start_cm_for_wal	WAL defined; Cache Manager must be started first. Обнаружено WAL; Менеджер КЭШа должен стартовать первым.
-260	335544690	cache_redef	Cache redefined.

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			Кэш переопределен.
-260	335544692	log_redef	Log redefined. Лог переопределен.
-261	335544695	partition_not_supp	Partitions not supported in series of log file specification. Разделы не поддерживаются в серии спецификации файла журнала.
-261	335544696	log_length_spec	Total length of a partitioned log must be specified. Общая длина раздельного лога должна быть определена.
-281	335544637	no_stream_plan	Table @1 is not referenced in plan. Таблица @1 не упоминается в плане.
-282	335544638	stream_twice	Table @1 is referenced more than once in plan; use aliases to distinguish. Таблица @1 упомянута более раза в плане; используйте алиасы чтобы различить.
-282	335544643	dsql_self_join	The table @1 is referenced twice; use aliases to differentiate. Таблица @1 упомянута дважды; используйте алиасы чтобы различить.
-282	335544659	duplicate_base_table	Table @1 is referenced twice in view; use an alias to distinguish. Таблица @1 упомянута дважды во view; используйте алиасы чтобы различить.
-282	335544660	view_alias	View @1 has more than one base table; use aliases to distinguish. View @1 использует более одного раза базовую таблицу;

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			используйте алиасы чтобы различить их.
-282	335544710	complex_view	Navigational stream @1 references a view with more than one base table. Навигационный поток @1 ссылается на view с более чем одной базовой таблицей.
-283	335544639	stream_not_found	Table @1 is referenced in the plan but not the from list. В плане ссылаются на таблицу @1, но она не в списке.
-284	335544642	index_unused	Index @1 cannot be used in the specified plan. Индекс @1 не может быть использован в указанном плане.
-291	335544531	primary_key_notnull	Column used in a PRIMARY constraint must be NOT NULL. Столбец, использованный в первичном ключе должен быть NOT NULL.
-292	335544534	ref_cnstrnt_update	Cannot update constraints (RDB $REF_CONSTRAINTS). Не могу обновить ограничения (RDB$REF_CONSTRAINTS).
-293	335544535	check_cnstrnt_update	Cannot update constraints (RDB $CHECK_CONSTRAINTS). Не могу обновить ограничения (RDB$CHECK_CONSTRAINTS).
-294	335544536	check_cnstrnt_del	Cannot delete CHECK constraint entry (RDB $CHECK_CONSTRAINTS). Не удается удалить ограничение CHECK (RDB $CHECK_CONSTRAINTS).
-295	335544545	rel_cnstrnt_update	Cannot update constraints (RDB $RELATION_CONSTRAINTS).

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			Не могу обновить ограничения (RDB $RELATION_CONSTRAINTS).
-296	335544547	invld_cnstrnt_type	Internal gds software consistency check (invalid RDB $CONSTRAINT_TYPE). Внутренняя ошибка программного обеспечения на согласованность (неверная таблица RDB $CONSTRAINT_TYPE).
-297	335544558	check_constraint	Operation violates check constraint @1 on view or table @2. Операция нарушает проверочное ограничение @1 на представление или таблицу @2.
-313	336003099	upd_ins_doesnt_match_pk	UPDATE OR INSERT field list does not match primary key of table @1. В UPDATE OR INSERT список полей не соответствует первичному ключу таблицы @1.
-313	336003100	upd_ins_doesnt_ match _matching	UPDATE OR INSERT field list does not match MATCHING clause. В UPDATE OR INSERT список полей не соответствует кляузе MATCHING.
-313	335544669	dsql_count_mismatch	Count of column list and variable list do not match. Количество столбцов и переменных в списке не соответствует.
-314	335544565	transliteration_failed	Cannot transliterate character between character sets. Не могу подвергнуть транслитерации символ между наборами символов.
-315	336068815	dyn_dtype_invalid	Cannot change datatype for column @1.Changing datatype is not

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			supported for BLOB or ARRAY columns. Не могу изменить тип данных для столбца @1. Изменение типа данных не поддерживается для BLOB полей и полей с массивами.
-383	336068814	dyn_dependency_exists	Column @1 from table @2 is referenced in @3. Столбец @1 из таблицы @2 упоминается в @3.
-401	335544647	invalid_operator	Invalid comparison operator for find operation. Неверный оператор сравнения для операции поиска.
-402	335544368	segstr_no_op	Attempted invalid operation on a BLOB. Попытка неверной операции с BLOB.
-402	335544414	blobnotsup	BLOB and array data types are not supported for @1 operation. Типы данных BLOB и массив не поддерживаются для операции @1.
-402	335544427	datnotsup	Data operation not supported. Операция данных не поддерживается.
-406	335544457	out_of_bounds	Subscript out of bounds. Индекс вне границ.
-407	335544435	nullsegkey	Null segment of UNIQUE KEY. Null сегмент для уникального ключа.
-413	335544334	convert_error	Conversion error from string "@1". Ошибка конвертации для строки "@1".

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
-413	335544454	nofilter	Filter not found to convert type @1 to type @2. Фильтр не обнаружен для конвертации типа @1 в тип @2.
-413	335544860	blob_convert_error	Unsupported conversion to target type BLOB (subtype @1). Не поддерживается преобразование в целевой тип BLOB (подтип @1).
-413	335544861	array_convert_error	Unsupported conversion to target type ARRAY. Не поддерживается преобразование в целевой тип массив.
-501	335544577	dsql_cursor_close_err	Attempt to reclose a closed cursor. Попытка перезакрыть закрытый курсор.
-502	336003090	dsql_cursor_redefined	Statement already has a cursor @1 assigned. Для предложения уже имеется назначенный @1 курсор.
-502	336003091	dsql_cursor_not_found	Cursor @1 is not found in the current context. Курсор @1 не обнаружен для текущего контекста.
-502	336003092	dsql_cursor_exists	Cursor @1 already exists in the current context. Курсор @1 уже существует для указанного контекста.
-502	336003093	dsql_cursor_rel_ambiguous	Relation @1 is ambiguous in cursor @2. Соотношение @1 неоднозначно в курсоре @2.
-502	336003094	dsql_cursor_rel_not_found	Relation @1 is not found in cursor @2.

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			Отношение @1 не найдено в курсоре @2.
-502	336003095	dsql_cursor_not_open	Cursor is not open. Курсор не открыт.
-502	335544574	dsql_decl_err	Invalid cursor declaration. Неверная декларация курсора.
-502	335544576	dsql_cursor_open_err	Attempt to reopen an open cursor. Попытка переоткрыть открытый курсор.
-504	336003089	dsql_cursor_invalid	Empty cursor name is not allowed. Для курсора не доступно пустое имя.
-504	335544572	dsql_cursor_err	Invalid cursor reference. Недопустимая ссылка на курсор.
-508	335544348	no_cur_rec	No current record for fetch operation. Нет текущей записи для операции FETCH.
-510	335544575	dsql_cursor_update_err	Cursor @1 is not updatable. Курсор @1 является не обновляемым.
-518	335544582	dsql_request_err	Request unknown. Запрос неизвестен.
-519	335544688	dsql_open_cursor_request	The prepare statement identifies a prepare statement with an open cursor. Подготовка выражения определила подготовку выражения с открытием курсора.
-530	335544466	foreign_key	Violation of FOREIGN KEY constraint "@1" on table "@2".

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			Нарушение ограничения внешнего ключа @1 для таблицы @2.
-530	335544838	foreign_key_target_doesnt_exist	Foreign key reference target does not exist. Ссылка на целевое значение внешнего ключа не существует.
-530	335544839	foreign_key_references_present	Foreign key references are present for the record. Ссылки внешнего ключа присутствуют для записи.
-531	335544597	dsql_crdb_prepare_err	Cannot prepare a CREATE DATABASE/SCHEMA statement. Не могу подготовить к выполнению оператор CREATE DATABASE/SCHEMA.
-532	335544469	trans_invalid	Transaction marked invalid by I/O error. Транзакция помечена как недействительная из-за ошибки ввода-вывода.
-551	335544352	no_priv	No permission for @1 access to @2 @3. Нет разрешения @1 для доступа к @2 @3.
-551	335544790	insufficient_svc_privileges	Service @1 requires SYSDBA permissions. Reattach to the Service Manager using the SYSDBA account. Сервис @1 требует привилегии SYSDBA. Переприсоединитесь к менеджеру сервисов используя учетную запись SYSDBA.
-552	335544550	not_rel_owner	Only the owner of a table may reassign ownership. Только владелец таблицы может переназначить права владения.

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
-552	335544553	grant_nopriv	User does not have GRANT privileges for operation. Пользователь не имеет привилегий для операции GRANT.
-552	335544707	grant_nopriv_on_base	User does not have GRANT privileges on base table/view for operation. Пользователь не имеет GRANT привилегии на таблицу / представление для этой операции.
-553	335544529	existing_priv_mod	Cannot modify an existing user privilege. Не могу изменить существующие пользовательские привилегии.
-595	335544645	stream_crack	The current position is on a crack. Текущая позиция «в трещине».
-596	335544644	stream_bof	Illegal operation when at beginning of stream. Неверная операция при начале потока.
-597	335544632	dsql_file_length_err	Preceding file did not specify length, so @1 must include starting page number. Предыдущий файл не указал длины, так что @1 должен включать число стартовых страниц.
-598	335544633	dsql_shadow_number_err	Shadow number must be a positive integer. Номер тени должен быть целым положительным числом.
-599	335544607	node_err	Gen.c: node not supported. Gen.c: ноды (узлы) не поддерживаются.

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
-599	335544625	node_name_err	A node name is not permitted in a secondary, shadow, cache or log file name. Имя узла не допускается в именах вторичного, теневого, КЭШа или в имени файла лога.
-600	335544680	crrp_data_err	Sort error: corruption in data structure. Ошибка сортировки: разрушения в структуре данных.
-601	335544646	db_or_file_exists	Database or file exists. База данных или файл не существует.
-604	335544593	dsql_max_arr_dim_exceeded	Array declared with too many dimensions. Массив задекларирован со слишком многими размерностями.
-604	335544594	dsql_arr_range_error	Illegal array dimension range. Неверный диапазон размерности массива.
-605	335544682	dsql_field_ref	Inappropriate self-reference of column. Жалоба на ссылку «сам на себя» столбца.
-607	336003074	dsql_dbkey_from_non_table	Cannot SELECT RDB$DB_KEY from a stored procedure. Невозможно выполнить SELECT RDB$DB_KEY из хранимой процедуры.
-607	336003086	dsql_udf_return_pos_err	External function should have return position between 1 and @1. Внешняя функция должна иметь позицию возврата между 1 и @1.

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
-607	336003096	dsql_type_not_supp_ext_tab	Data type @1 is not supported for EXTERNAL TABLES. Relation '@2', field '@3'. Тип данных @1 не поддерживается для внешних таблиц. Таблица '@2', поле '@3'.
-607	335544351	no_meta_update	Unsuccessful metadata update. Неудачное обновление метаданных.
-607	335544549	systrig_update	Cannot modify or erase a system trigger. Не возможно изменить или стереть системный триггер.
-607	335544657	dsql_no_blob_array	Array/BLOB/DATE data types not allowed in arithmetic. Типы данных Массив/BLOB/ даты не возможны для арифметических операций.
-607	335544746	reftable_requires_pk	"REFERENCES table" without "(column)" requires PRIMARY KEY on referenced table. "Таблицы ссылок" без "(столбца)" требуют первичного ключа связанной таблице.
-607	335544815	generator_name	GENERATOR @1.
-607	335544816	udf_name	UDF @1.
-607	335544858	must_have_phys_field	Can't have relation with only computed fields or constraints. Невозможна таблица состоящая только из одних вычисляемых полей или ограничений.
-607	336397206	dsql_table_not_found	Table @1 does not exist. Таблица @1 не существует.
-607	336397207	dsql_view_not_found	View @1 does not exist.

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			Представление @1 не существует.
-607	336397212	dsql_no_array_computed	Array and BLOB data types not allowed in computed field. Типы данных массив и BLOB не подходят для вычисляемых полей.
-607	336397214	dsql_only_can_subscript_array	Scalar operator used on field @1 which is not an array. Скалярный оператор используется по полю @1, которое не является массивом.
-612	336068812	dyn_domain_name_exists	Cannot rename domain @1 to @2. A domain with that name already exists. Не возможно переименовать домен из @1 в @2. Домен с таким именем уже существует.
-612	336068813	dyn_field_name_exists	Cannot rename column @1 to @2.A column with that name already exists in table @3. Не возможно переименовать столбец @1 в @2. Столбец с таким именем уже существует в таблице @3.
-615	335544475	relation_lock	Lock on table @1 conflicts with existing lock. Блокировка в таблице @1 конфликтует с существующей блокировкой.
-615	335544476	record_lock	Requested record lock conflicts with existing lock. Требуемая блокировка записи конфликтует с существующей блокировкой.
-615	335544507	range_in_use	Refresh range number @1 already in use.

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
			Обновленный диапазон номеров @1 уже используется.
-616	335544530	primary_key_ref	Cannot delete PRIMARY KEY being used in FOREIGN KEY definition. Не могу удалить первичный ключ используемый в определении внешнего ключа.
-616	335544539	integ_index_del	Cannot delete index used by an Integrity Constraint. Не возможно удалить индекс используемый в ограничении.
-616	335544540	integ_index_mod	Cannot modify index used by an Integrity Constraint. Не могу изменить индекс используемый в ограничении.
-616	335544541	check_trig_del	Cannot delete trigger used by a CHECK Constraint. Не могу удалить триггер используемый в ограничении типа CHECK.
-616	335544543	cnstrnt_fld_del	Cannot delete column being used in an Integrity Constraint. Не могу удалить столбец используемый в ограничении целостности.
-616	335544630	dependency	There are @1 dependencies. Есть @1 зависимостей.
-616	335544674	del_last_field	Last column in a table cannot be deleted. Последний столбец таблицы не может быть удален.
-616	335544728	integ_index_deactivate	Cannot deactivate index used by an integrity constraint.

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
			Не могу деактивировать индекс используемый в ограничении целостности.
-616	335544729	integ_deactivate_primary	Cannot deactivate index used by a PRIMARY/UNIQUE constraint. Не могу деактивировать индекс используемый в ограничении первичного ключа/уникальности.
-617	335544542	check_trig_update	Cannot update trigger used by a CHECK Constraint. Не могу обновить триггер используемый в ограничении типа CHECK.
-617	335544544	cnstrnt_fld_rename	Cannot rename column being used in an Integrity Constraint. Не могу переименовать столбец, используемый в ограничении целостности.
-618	335544537	integ_index_seg_del	Cannot delete index segment used by an Integrity Constraint. Не могу удалить сегмент индекса, используемый в ограничении целостности.
-618	335544538	integ_index_seg_mod	Cannot update index segment used by an Integrity Constraint. Не могу обновить сегмент индекса, используемый в ограничении целостности.
-625	335544347	not_valid	Validation error for column @1, value "@2". Ошибка проверки данных для столбца @1, значение "@2".
-625	335544879	not_valid_for_var	Validation error for variable @1, value "@2". Ошибка проверки данных для переменной @1, значение "@2".
-625	335544880	not_valid_for	Validation error for @1, value "@2".

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			Ошибка проверки данных для @1, значение "@2".
-637	335544664	dsql_duplicate_spec	Duplicate specification of @1- not supported. Дубликат спецификации для @1 - не поддерживается.
-637	336397213	dsql_implicit_domain_name	Implicit domain name @1 not allowed in user created domain. Неявное доменное имя @1 не допускается для доменов, создаваемых пользователями.
-660	336003098	primary_key_required	Primary key required on table @1. Для таблицы @1 требуется первичный ключ.
-660	335544533	foreign_key_notfound	Non-existent PRIMARY or UNIQUE KEY specified for FOREIGN KEY. Не существующий первичный или ключ уникальности указан для внешнего ключа.
-660	335544628	idx_create_err	Cannot create index @1. Не могу создать индекс @1.
-663	335544624	idx_seg_err	Segment count of 0 defined for index @1. Количество сегментов равное 0 определено для индекса @1.
-663	335544631	idx_key_err	Too many keys defined for index @1. Слишком много ключей определено для индекса @1.
-663	335544672	key_field_err	Too few key columns found for index @1 (incorrect column name?). Слишком много столбцов ключей обнаружено для индекса @1 (неверные имена столбцов?).

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
-664	335544434	keytoobig	Key size exceeds implementation restriction for index "@1". Размер ключа превысил ограничения реализации для индекса "@1".
-677	335544445	ext_err	@1 extension error. @1 ошибка расширения.
-685	335544465	bad_segstr_type	Invalid BLOB type for operation. Неверный тип BLOB для операции.
-685	335544670	blob_idx_err	Attempt to index BLOB column in index @1. Попытка индексации BLOB столбца в индексе @1.
-685	335544671	array_idx_err	Attempt to index array column in index @1. Попытка индексации столбца с типом массив в индексе @1.
-689	335544403	badpagtyp	Page @1 is of wrong type (expected @2, found @3). Страница @1 имеет неверный тип (ожидается @2, обнаружена @3).
-689	335544650	page_type_err	Wrong page type. Неверный тип страницы.
-690	335544679	no_segments_err	Segments not allowed in expression index @1. Сегменты не допускаются в выражении индекса @1.
-691	335544681	rec_size_err	New record size of @1 bytes is too big. Новый размер записи в @1 байт является слишком большим.

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
-692	335544477	max_idx	Maximum indexes per table (@1) exceeded. Максимум количества индексов на одну таблицу (@1) превышен.
-693	335544663	req_max_clones_exceeded	Too many concurrent executions of the same request. Слишком много конкурентных выполнений одного и того же запроса.
-694	335544684	no_field_access	Cannot access column @1 in view @2. Не могу получить доступ к столбцу @1 представления @2.
-802	335544321	arith_except	Arithmetic exception, numeric overflow, or string truncation. Арифметическое исключение, числовое переполнение или строковое обрезание.
-802	335544836	concat_overflow	Concatenation overflow. Resulting string cannot exceed 32K in length. Переполнение при конкатенации. Результирующая строка не может превышать 32 Кб.
-803	335544349	no_dup	Attempt to store duplicate value (visible to active transactions) in unique index "@1". Попытка сохранить дубликат значения (видимые при активных транзакциях) в уникальном индексе "@1".
-803	335544665	unique_key_violation	Violation of PRIMARY or UNIQUE KEY constraint "@1" on table "@2". Нарушение ограничения первичного или уникального ключа "@1" в таблице "@2".

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
-804	336003097	dsql_feature_not_supported_ods	Feature not supported on ODS version older than @1.@2. Опция не поддерживается в ODS старше чем @1.@2.
-804	335544380	wronumarg	Wrong number of arguments on call. Неверное число аргументов в вызове.
-804	335544583	dsql_sqlda_err	SQLDA missing or incorrect version, or incorrect number/type of variables. SQLDA отсутствует или неверной версии, либо некорректный номер/тип переменной.
-804	335544584	dsql_var_count_err	Count of read - write columns does not equal count of values. Количество столбцов для чтения/записи не равно количеству значений.
-804	335544586	dsql_function_err	Function unknown. Функция неизвестна.
-804	335544713	dsql_sqlda_value_err	Incorrect values within SQLDA structure. Некорректные значения в SQLDA структуре.
-804	336397205	dsql_too_old_ods	ODS versions before ODS@1 are not supported. ODS версии до версии ODS@1 не поддерживаются.
-806	335544600	col_name_err	Only simple column names permitted for VIEW WITH CHECK OPTION. Только простые имена столбцов допустимы для опции VIEW WITH CHECK.

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
-807	335544601	where_err	No WHERE clause for VIEW WITH CHECK OPTION. Нет предложения WHERE для опции VIEW WITH CHECK.
-808	335544602	table_view_err	Only one table allowed for VIEW WITH CHECK OPTION. Только одна таблица позволена для опции VIEW WITH CHECK.
-809	335544603	distinct_err	DISTINCT, GROUP or HAVING not permitted for VIEW WITH CHECK OPTION. Для представления с опцией WITH CHECK OPTION не допустимы DISTINCT, GROUP или HAVING.
-810	335544605	subquery_err	No subqueries permitted for VIEW WITH CHECK OPTION. Нет подзапросов разрешенных для for VIEW WITH CHECK OPTION.
-811	335544652	sing_select_err	Multiple rows in singleton select. Несколько строк для единичной выпорки.
-816	335544651	ext_readonly_err	Cannot insert because the file is readonly or is on a read only medium. Не могу вставить по причине файла только для чтения или среды носителя только для чтения.
-816	335544715	extfile_uns_op	Operation not supported for EXTERNAL FILE table @1. Операция не поддерживается для внешней таблицы @1.
-817	336003079	isc_sql_dialect_conflict_num	DB dialect @1 and client dialect @2 conflict with respect to numeric precision @3.

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
			Диалект БД @1 и диалект клиентской программы @2 конфликтует с соблюдением числовой точности @3.
-817	336003101	upd_ins_with_complex_view	UPDATE OR INSERT without MATCHING could not be used with views based on more than one table. UPDATE OR INSERT без MATCHING не могут быть использованы с View базирующимися на более чем одной таблице.
-817	336003102	dsql_incompatible_trigger_type	Incompatible trigger type. Несовместимый тип триггера.
-817	336003103	dsql_db_trigger_type_cant_change	Database trigger type can't be changed. Триггер типа базы данных не может быть изменен.
-817	335544361	read_only_trans	Attempted update during read - only transaction. Попытка выполнить изменения во время выполнения транзакции только для чтения.
-817	335544371	segstr_no_write	Attempted write to read-only BLOB. Попытка записи в BLOB только для чтения.
-817	335544444	read_only	Operation not supported. Операция не поддерживается.
-817	335544765	read_only_database	Attempted update on read - only database. Попытка записи в базу данных находящуюся в режиме только для чтения.
-817	335544766	must_be_dialect_2_and_up	SQL dialect @1 is not supported in this database.

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			SQL диалект @1 не поддерживается в этой базе данных.
-817	335544793	ddl_not_allowed_by_db_sql_dial	Metadata update statement is not allowed by the current database SQL dialect @1. Обновление метаданных этой строкой не поддерживается из-за текущего диалекта базы данных @1.
-820	335544356	obsolete_metadata	Metadata is obsolete. Метаданных являются устаревшими.
-820	335544379	wrong_ods	Unsupported on - disk structure for file @1; found @2.@3, support @4.@5. Неподдерживаемая ODS для файла @1, обнаружена @2.@3, поддерживается @4.@5.
-820	335544437	wrodynver	Wrong DYN version. Неверная версия DYN.
-820	335544467	high_minor	Minor version too high found @1 expected @2. Минорная версия слишком высокая, обнаружена @1, ожидалась @2.
-820	335544881	need_difference	Difference file name should be set explicitly for database on raw device. Файл разницы должен быть явно задан для базы данных на «сыром устройстве».
-823	335544473	invalid_bookmark	Invalid bookmark handle. Неверный дескриптор закладки.
-824	335544474	bad_lock_level	Invalid lock level @1.

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
			Неверный уровень блокировки @1.
-825	335544519	bad_lock_handle	Invalid lock handle. Неверный дескриптор блокировки.
-826	335544585	dsql_stmt_handle	Invalid statement handle. Неверный дескриптор выражения.
-827	335544655	invalid_direction	Invalid direction for find operation. Неверное выражение для операции «поиск».
-827	335544718	invalid_key	Invalid key for find operation. Неверный ключ для операции поиска.
-828	335544678	inval_key_posn	Invalid key position. Неверный положение ключа.
-829	336068816	dyn_char_fld_too_small	New size specified for column @1 must be at least @2 characters. Новый размер указанный для столбца @1 должен быть по крайней мере @2 символов.
-829	336068817	dyn_invalid_dtype_conversion	Cannot change datatype for @1.Conversion from base type @2 to @3 is not supported. Не могу изменить тип данных для @1. Преобразование из базового типа @2 в @3 не поддерживается.
-829	336068818	dyn_dtype_conv_invalid	Cannot change datatype for column @1 from a character type to a non-character type. Не могу помять типа данных для столбца @1 из символьного типа в не символьный.

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
-829	336068829	max_coll_per_charset	Maximum number of collations per character set exceeded. В наборе символов превышено максимальное число наборов сортировок.
-829	336068830	invalid_coll_attr	Invalid collation attributes. Неверный атрибуты параметров сортировки.
-829	336068852	dyn_scale_too_big	New scale specified for column @1 must be at most @2. Новый масштаб, указанный для столбца @1 должен быть не более @2.
-829	336068853	dyn_precision_too_small	New precision specified for column @1 must be at least @2. Новая точность указанная для столбца @1 должна быть по крайней мере @2.
-829	335544616	field_ref_err	Invalid column reference. Неверная ссылка столбца.
-830	335544615	field_aggregate_err	Column used with aggregate. Столбец используется в агрегатах.
-831	335544548	primary_key_exists	Attempt to define a second PRIMARY KEY for the same table. Попытка определить второй первичный ключ для таблицы.
-832	335544604	key_field_count_err	FOREIGN KEY column count does not match PRIMARY KEY. Количество столбцов внешнего ключа не совпадает с первичным ключом.
-833	335544606	expression_eval_err	Expression evaluation not supported.

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			Вычисляемые выражения не поддерживаются.
-833	335544810	date_range_exceeded	Value exceeds the range for valid dates. Значение превышают пределы установленные для действительных дат.
-834	335544508	range_not_found	Refresh range number @1 not found. Обновленный диапазон номеров @1 не найден.
-835	335544649	bad_checksum	Bad checksum. Неверная контрольная сумма.
-836	335544517	except	Exception @1. Исключение @1.
-836	335544848	except2	Exception @1. Исключение @1.
-837	335544518	cache_restart	Restart shared cache manager. Рестарт общего менеджера КЭШа.
-838	335544560	shutwarn	Database @1 shutdown in @2 seconds. База данных @1 уйдет в состояние шатдаун через @2 секунд.
-841	335544677	version_err	Too many versions. Слишком много версий.
-842	335544697	precision_err	Precision must be from 1 to 18. Точность должна быть в пределах от 1 до 18.
-842	335544698	scale_nogt	Scale must be between zero and precision.

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			Масштаб должен быть между нулем и значением точности.
-842	335544699	expec_short	Short integer expected. Ожидается короткое целое.
-842	335544700	expec_long	Long integer expected. Ожидается длинное целое.
-842	335544701	expec_ushort	Unsigned short integer expected. Ожидается беззнаковое короткое целое.
-842	335544712	expec_positive	Positive value expected. Ожидается положительное значение.
-901	335740929	gfix_db_name	Database file name (@1) already given. Имя файла базы данных (@1) уже отдано.
-901	336330753	gbak_unknown_switch	Found unknown switch. Обнаружена неизвестная опция командной строки.
-901	336920577	gstat_unknown_switch	Found unknown switch. Обнаружена неизвестная опция командной строки.
-901	336986113	fbsvcmgr_bad_am	Wrong value for access mode. Неверное значение для режима доступа.
-901	335740930	gfix_invalid_sw	Invalid switch @1. Неверный параметр командной строки @1.
-901	335544322	bad_dbkey	Invalid database key. Неверный ключ базы данных.
-901	336986114	fbsvcmgr_bad_wm	Wrong value for write mode.

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			Неверное значение для режима записи.
-901	336330754	gbak_page_size_missing	Page size parameter missing. Параметра размера страницы отсутствует.
-901	336920578	gstat_retry	Please retry, giving a database name. Пожалуйста повторите, давая имя базы данных.
-901	336986115	fbsvcmgr_bad_rs	Wrong value for reserve space. Неверное значение для зарезервированного пространства.
-901	336920579	gstat_wrong_ods	Wrong ODS version, expected @1, encountered @2. Неверная версия ODS, ожидается @1, встретилась @2.
-901	336330755	gbak_page_size_toobig	Page size specified (@1) greater than limit (16384 bytes). Указанный размер страницы (@1) больше ограничения (16384 bytes).
-901	335740932	gfix_incmp_sw	Incompatible switch combination. Несовместимая комбинация ключей командной строки.
-901	336920580	gstat_unexpected_eof	Unexpected end of database file. Неожиданный конец файла базы данных.
-901	336330756	gbak_redir_ouput_missing	Redirect location for output is not specified. Перенаправление места для вывода не указано.
-901	336986116	fbsvcmgr_info_err	Unknown tag (@1) in info_svr_db_info block after isc_svc_query().

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			Неизвестный тег (@1) в блоке info_svr_db_info после вызова isc_svc_query().
-901	335740933	gfix_replay_req	Replay log pathname required. Воспроизведение лога требует пути до него.
-901	336330757	gbak_switches_conflict	Conflicting switches for backup/ restore. Конфликтующие операции командной строки для бекапа/ рестора.
-901	336986117	fbsvcmgr_query_err	Unknown tag (@1) in isc_svc_query() results. Неизвестный тег в результатах isc_svc_query().
-901	335544326	bad_dpb_form	Unrecognized database parameter block. Блок параметров базы данных не распознан.
-901	335740934	gfix_pgbuf_req	Number of page buffers for cache required. Требуется указать количество страниц для буфера кэша.
-901	336986118	fbsvcmgr_switch_unknown	Unknown switch "@1". Неизвестный параметр командной строки "@1".
-901	336330758	gbak_unknown_device	Device type @1 not known. Тип устройства @1 не известен.
-901	335544327	bad_req_handle	Invalid request handle. Неверный указатель запроса.
-901	335740935	gfix_val_req	Numeric value required. Требуется числовое значение.
-901	336330759	gbak_no_protection	Protection is not there yet.

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
			Опция «защита» еще не есть.
-901	335544328	bad_segstr_handle	Invalid BLOB handle. Неверный указатель на BLOB.
-901	335740936	gfix_pval_req	Positive numeric value required. Требуется положительное числовое значение.
-901	336330760	gbak_page_size_not_allowed	Page size is allowed only on restore or create. Параметр «размер страницы» доступен только при восстановлении или создании.
-901	335544329	bad_segstr_id	Invalid BLOB ID. Неверный ID BLOB.
-901	335740937	gfix_trn_req	Number of transactions per sweep required. Требуется количество транзакция для операции sweep.
-901	336330761	gbak_multi_source_dest	Multiple sources or destinations specified. Несколько источников или направлений указаны.
-901	335544330	bad_tpb_content	Invalid parameter in transaction parameter block. Неверный параметр в блоке параметров транзакции.
-901	336330762	gbak_filename_missing	Requires both input and output filenames. Требуются оба имени файла для ввода и для вывода.
-901	335544331	bad_tpb_form	Invalid format for transaction parameter block. Неверный формат для блока параметров для транзакции.

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
-901	336330763	gbak_dup_inout_names	Input and output have the same name. Disallowed. Ввод и вывод имеют одинаковое имя. Не разрешается.
-901	335740940	gfix_full_req	"full" or "reserve" required. Параметры "full" или "reserve" требуются.
-901	335544332	bad_trans_handle	Invalid transaction handle (expecting explicit transaction start). Неверная ссылка на транзакцию (ожидаю явного старта транзакций).
-901	336330764	gbak_inv_page_size	Expected page size, encountered "@1". Ожидалось размер страницы, встретилось "@1".
-901	335740941	gfix_usrname_req	User name required. Требуется имя пользователя.
-901	336330765	gbak_db_specified	REPLACE specified, but the first file @1 is a database. Указан параметр ЗАМЕНА, но первый файл @1 является базой данных.
-901	335740942	gfix_pass_req	Password required. Требуется пароль.
-901	336330766	gbak_db_exists	Database @1 already exists.To replace it, use the -REP switch. База данных @1 уже существует. Чтобы заменить ее, используйте переключатель командной строки -REP.
-901	335740943	gfix_subs_name	Subsystem name. Имя подсистемы.

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
-901	336723983	gsec_cant_open_db	Unable to open database. Не могу открыть базу данных.
-901	336330767	gbak_unk_device	Device type not specified. Тип устройства не указан.
-901	336723984	gsec_switches_error	Error in switch specifications. Ошибка в указании опций командной строки.
-901	335740945	gfix_sec_req	Number of seconds required. Число секунд требуется.
-901	335544337	excess_trans	Attempt to start more than @1 transactions. Попытка начать более чем @1 транзакций.
-901	336723985	gsec_no_op_spec	No operation specified. Операция не определена.
-901	335740946	gfix_nval_req	Numeric value between 0 and 32767 inclusive required. Цифровое значение между 0 и 32767 включительно требуется.
-901	336723986	gsec_no_usr_name	No user name specified. Не определено имя пользователя.
-901	335740947	gfix_type_shut	Must specify type of shutdown. Требуется определить тип шатдауна.
-901	335544339	infinap	Information type inappropriate for object specified. Тип данных подходит для указанного объекта.
-901	336723987	gsec_err_add	Add record error. Ошибка добавления записи.

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
-901	335544340	infona	No information of this type available for object specified. Нет информации этого типа доступной для указанного объекта.
-901	336723988	gsec_err_modify	Modify record error. Ошибка изменения записи.
-901	336330772	gbak_blob_info_failed	Gds_$blob_info failed. Операция Gds_$blob_info не удалась.
-901	335740948	gfix_retry	Please retry, specifying an option. Пожалуйста повторите, уточните опцию.
-901	335544341	infunk	Unknown information item. Незнакомый элемент информации.
-901	336723989	gsec_err_find_mod	Find / modify record error. Ошибка поиска/изменения записи.
-901	336330773	gbak_unk_blob_item	Do not understand BLOB INFO item @1. Не понятный элемент BLOB INFO @1.
-901	335544342	integ_fail	Action cancelled by trigger (@1) to preserve data integrity. Действие отменено триггером (@1) чтобы сохранить целостность данных.
-901	336330774	gbak_get_seg_failed	Gds_$get_segment failed. Операция Gds_$get_segment не удалась.
-901	336723990	gsec_err_rec_not_found	Record not found for user: @1.

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			Запись не найдена для пользователя: @1.
-901	336723991	gsec_err_delete	Delete record error. Ошибка удаления записи.
-901	336330775	gbak_close_blob_failed	Gds_$close_blob failed. Операция Gds_$close_blob не удалась.
-901	335740951	gfix_retry_db	Please retry, giving a database name. Пожалуйста повторите, давая имя базы данных.
-901	336330776	gbak_open_blob_failed	Gds_$open_blob failed. Операция Gds_$open_blob не удалась.
-901	336723992	gsec_err_find_del	Find / delete record error. Ошибка поиска/удаления записи.
-901	335544345	lock_conflict	Lock conflict on no wait transaction. Ошибка блокировки при не ждущей транзакции.
-901	336330777	gbak_put_blr_gen_id_failed	Failed in put_blr_gen_id. Операция put_blr_gen_id не удалась.
-901	336330778	gbak_unk_type	Data type @1 not understood. Тип данных @1 не понят.
-901	336330779	gbak_comp_req_failed	Gds_$compile_request failed. Операция Gds_$compile_request не удалась.
-901	336330780	gbak_start_req_failed	Gds_$start_request failed. Операция Gds_$start_request не удалась.
-901	336723996	gsec_err_find_disp	Find / display record error.

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			Ошибка в Найти/показать запись.
-901	336330781	gbak_rec_failed	gds_$receive failed. Операция gds_$receive не удалась.
-901	336920605	gstat_open_err	Can't open database file @1. Не могу открыть файл базы данных <строка>.
-901	336723997	gsec_inv_param	Invalid parameter, no switch defined. Неверный параметр, нет ключа командной строки.
-901	335544350	no_finish	Program attempted to exit without finishing database. Программа попыталась завершиться без отсоединения от базы данных.
-901	336920606	gstat_read_err	Can't read a database page. Не могу прочитать страницу базы данных.
-901	336330782	gbak_rel_req_failed	Gds_$release_request failed. Операция Gds_$release_request не удалась.
-901	336723998	gsec_op_specified	Operation already specified. Операция уже определена.
-901	336920607	gstat_sysmemex	System memory exhausted. Системная память исчерпана.
-901	336330783	gbak_db_info_failed	gds_$database_info failed. Операция gds_$database_info не удалась.
-901	336723999	gsec_pw_specified	Password already specified. Пароль уже определен.
-901	336724000	gsec_uid_specified	Uid already specified.

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
			UID уже определен.
-901	336330784	gbak_no_db_desc	Expected database description record. Ожидалась описание записи базы данных.
-901	335544353	no_recon	Transaction is not in limbo. Транзакция не является limbo (не находится в подвешенном состоянии).
-901	336724001	gsec_gid_specified	Gid already specified. GID уже указан.
-901	336330785	gbak_db_create_failed	Failed to create database @1. Ошибка создания базы данных @1.
-901	336724002	gsec_proj_specified	Project already specified. Параметр «проект» уже указан.
-901	336330786	gbak_decomp_len_error	RESTORE: decompression length error. Восстановление: ошибка длины декомпрессии.
-901	335544355	no_segstr_close	BLOB was not closed. BLOB был не закрытым.
-901	336330787	gbak_tbl_missing	Cannot find table @1. Не могу найти таблицу @1.
-901	336724003	gsec_org_specified	Organization already specified. Организация уже указана.
-901	336330788	gbak_blob_col_missing	Cannot find column for BLOB. Не могу найти столбец для BLOB.
-901	336724004	gsec_fname_specified	First name already specified. Параметр «имя» уже указано.

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
-901	335544357	open_trans	Cannot disconnect database with open transactions (@1 active). Не могу отключить базу данных от открытых коннектов (@1 активных).
-901	336330789	gbak_create_blob_failed	Gds_$create_blob failed. Операция Gds_$create_blob не удалась.
-901	336724005	gsec_mname_specified	Middle name already specified. Второе имя (отчество) уже указано.
-901	335544358	port_len	Message length error (encountered @1, expected @2). Ошибка в длине сообщения (встретилась @1, ожидалось @2).
-901	336330790	gbak_put_seg_failed	Gds_$put_segment failed. Операция Gds_$put_segment не удалась.
-901	336724006	gsec_lname_specified	Last name already specified. Фамилия уже указана.
-901	336330791	gbak_rec_len_exp	Expected record length. Ошибка в ожидаемой длине записи.
-901	336724008	gsec_inv_switch	Invalid switch specified. Неверный параметр командной строки указан.
-901	336330792	gbak_inv_rec_len	Wrong length record, expected @1 encountered @2. Неверная длина записи, ожидалось @1, встретилась @2.
-901	336330793	gbak_exp_data_type	Expected data attribute. Ожидаемый атрибут данных.

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
-901	336724009	gsec_amb_switch	Ambiguous switch specified. Неоднозначный переключатель командной строки указан.
-901	336330794	gbak_gen_id_failed	Failed in store_blr_gen_id. Ошибка в store_blr_gen_id.
-901	336724010	gsec_no_op_specified	No operation specified for parameters. Нет операции указанной для параметров.
-901	335544363	req_no_trans	No transaction for request. Нет транзакции для запроса.
-901	336330795	gbak_unk_rec_type	Do not recognize record type @1. Не распознан тип записи @1.
-901	336724011	gsec_params_not_allowed	No parameters allowed for this operation. Нет параметров доступных для этой операции.
-901	335544364	req_sync	Request synchronization error. Ошибка синхронизации запроса.
-901	336724012	gsec_incompat_switch	Incompatible switches specified. Указанные несовместимые переключатели командной строки.
-901	336330796	gbak_inv_bkup_ver	Expected backup version 1..8. Found @1. Ожидалась версия бекапа в пределах 1..8. Обнаружена @1.
-901	335544365	req _ wrong _ db	Request referenced an unavailable database. Запрос ссылается на недоступный базу данных.
-901	336330 797	gbak_missing_bkup_desc	Expected backup description record.

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
			Ожидается описание бекапа.
-901	336330798	gbak_string_trunc	String truncated. Усечение строки.
-901	336330799	gbak_cant_rest_record	warning -- record could not be restored. Предупреждение – запись не может быть восстановлена.
-901	336330800	gbak_send_failed	Gds_$send failed. Ошибка Gds_$send.
-901	335544369	segstr_no_read	Attempted read of a new, open BLOB. Попытка чтения нового, а открыт BLOB.
-901	336330801	gbak_no_tbl_name	No table name for data. Не указано имя таблицы для данных.
-901	335544370	segstr_no_trans	Attempted action on blob outside transaction. Попытка действия на BLOB за пределами транзакции.
-901	336330802	gbak_unexp_eof	Unexpected end of file on backup file. Неожиданный признак конца файла в файле бекапа.
-901	336330803	gbak_db_format_too_old	Database format @1 is too old to restore to. Формат базы данных @1 слишком старый чтобы восстановить базу.
-901	335544372	segstr_wrong_db	Attempted reference to BLOB in unavailable database. Попытка сослаться на BLOB в недоступной базе данных.

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
-901	336330804	gbak_inv_array_dim	Array dimension for column @1 is invalid. Размерность массива для столбца @1 неверная.
-901	336330807	gbak_xdr_len_expected	Expected XDR record length. Ожидаемая длина записи XDR.
-901	335544376	unres_rel	Table @1 was omitted from the transaction reserving list. Таблица @1 была исключена из списка резервирования транзакций.
-901	335544377	uns_ext	Request includes a DSRI extension not supported in this implementation. Запрос включает в себя расширение DSRI не поддерживается в этой реализации.
-901	335544378	wish_list	Feature is not supported. Опция не поддерживается.
-901	335544382	random	@1
-901	335544383	fatal_conflict	Unrecoverable conflict with limbo transaction @1. Неустранимый конфликт с лимбо транзакцией @1.
-901	335740991	gfix_exceed_max	Internal block exceeds maximum size. Внутренний блок превышает максимальный размер.
-901	335740992	gfix_corrupt_pool	Corrupt pool. Разрушение пула.
-901	335740993	gfix_mem_exhausted	Virtual memory exhausted. Виртуальная память исчерпана.
-901	336330817	gbak_open_bkup_error	Cannot open backup file @1.

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			Не могу открыть файл бекапа @1.
-901	335740994	gfix_bad_pool	Bad pool id. Неверный ID пула.
-901	336330818	gbak_open_error	Cannot open status and error output file @1. Не могу получить статус и произошла ошибка выходного файла <строка>.
-901	335740995	gfix_trn_not_valid	Transaction state @1 not in valid range. Состояние транзакции <число> вне пределах верного диапазона.
-901	335544392	bdbincon	Internal error. Внутренняя ошибка.
-901	336724044	gsec_inv_username	Invalid user name (maximum 31 bytes allowed). Неверное имя пользователя. (максимум 31 байт доступен).
-901	336724045	gsec_inv_pw_length	Warning - maximum 8 significant bytes of password used. Внимание – максимум 8 значащих байт для пароля используется.
-901	336724046	gsec_db_specified	Database already specified. База данных уже указана.
-901	336724047	gsec_db_admin_specified	Database administrator name already specified. Имя администратора базы данных уже указано.
-901	336724048	gsec_db_admin_pw_specified	Database administrator password already specified.

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			Пароль администратора базы данных уже указан.
-901	336724049	gsec_sql_role_specified	SQL role name already specified Имя SQL роли уже указаны
-901	335741012	gfix_unexp_eoi	Unexpected end of input Неожиданных конец ввода
-901	335544407	dbbnotzer	Database handle not zero Указатель базы данных не ноль
-901	335544408	tranotzer	Transaction handle not zero Указатель транзакции не ноль
-901	335741018	gfix_recon_fail	Failed to reconnect to a transaction in database @1 Сбой реконнекта транзакции в базе данных < строка >
-901	335544418	trainlim	Transaction in limbo Транзакция в лимбо (зависла)
-901	335544419	notinlim	Transaction not in limbo Транзакция не в лимбо (зависла)
-901	335544420	traoutsta	Transaction outstanding Транзакция выдалась за пределы
-901	335544428	badmsgnum	Undefined message number Неопределенный номер сообщения
-901	335741036	gfix_trn_unknown	Transaction description item unknown Элемент описания транзакции неизвестен
-901	335741038	gfix_mode_req	"read_only" or "read_write" required

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
			Требуются параметры "read_only" или "read_write"
-901	335544431	blocking_signal	Blocking signal has been received Был получен блокирующий сигнал
-901	335741042	gfix_pzval_req	Positive or zero numeric value required Требуется положительное или нулевое значение
-901	335544442	noargacc_read	Database system cannot read argument @1 СУБД не может прочитать аргумент <строка>
-901	335544443	noargacc_write	Database system cannot write argument @1 СУБД не может записать аргумент < строка >
-901	335544450	misc_interpreted	@1
-901	335544468	tra_state	Transaction @1 is @2 Транзакция <строка> является <строка>
-901	335544485	bad_stmt_handle	Invalid statement handle Неверный указатель выражения
-901	336330934	gbak_missing_block_fac	Blocking factor parameter missing Параметр blocking factor отсутствует
-901	336330935	gbak_inv_block_fac	Expected blocking factor, encountered "@1" Ожидался Blocking factor, встретилось < строка >
-901	336330936	gbak_block_fac_specified	A blocking factor may not be used in conjunction with device CT

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			Blocking factor не могут быть использованы в сочетании с устройством CT
-901	336068796	dyn_role_does_not_exist	SQL role @1 does not exist SQL роль <строка> не существует
-901	336330940	gbak_missing_username	User name parameter missing Параметр имя пользователя отсутствует
-901	336330941	gbak_missing_password	Password parameter missing Параметр пароль отсутствует
-901	336068797	dyn_no_grant_admin_opt	User @1 has no grant admin option on SQL role @2 Пользователь < строка > не имеет администраторской опции GRANT в SQL роли < строка >
-901	335544510	lock_timeout	Lock time - out on wait transaction В ждущей транзакции произошел тайм-аут блокировки
-901	336068798	dyn_user_not_role_member	User @1 is not a member of SQL role @2 Пользователь < строка > не является членом SQL роли
-901	336068799	dyn_delete_role_failed	@1 is not the owner of SQL role @2 < строка > не принадлежит SQL роли < строка >
-901	336068800	dyn_grant_role_to_user	@1 is a SQL role and not a user <строка> является SQL ролью, а не пользователем
-901	336068801	dyn_inv_sql_role_name	User name @1 could not be used for SQL role Имя пользователя не может использоваться для названия SQL роли

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
-901	336068802	dyn_dup_sql_role	SQL role @1 already exists SQL роль <строка> уже существует
-901	336068803	dyn_kywd_spec_for_role	Keyword @1 can not be used as a SQL role name Ключевое слово <строка> не может быть использовано в качестве имени роли
-901	336068804	dyn_roles_not_supported	SQL roles are not supported in on older versions of the database.A backup and restore of the database is required. SQL роль не поддерживается в более старых версиях СУБД. Требуется выполнить операцию бекап – рестор для базы данных.
-901	336330952	gbak_missing_skipped_bytes	missing parameter for the number of bytes to be skipped Отсутствующий параметр для количества байт был обойден
-901	336330953	gbak_inv_skipped_bytes	Expected number of bytes to be skipped, encountered "@1" Ожидалось число байтов для пропуска, встретилось "@1"
-901	336068820	dyn_zero_len_id	Zero length identifiers are not allowed Идентификаторы с нулевой длиной не допускаются
-901	336330965	gbak_err_restore_charset	Character set Ошибка при восстановлении: Character set
-901	336330967	gbak_err_restore_collation	Collation Ошибка при восстановлении: Collation
-901	336330972	gbak_read_error	Unexpected I/O error while reading from backup file

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			Неожиданная ошибка операции ввода – вывода при чтении файла бекапа
-901	336330973	gbak_write_error	Unexpected I/O error while writing to backup file Неожиданная ошибка операции ввода – вывода при записи файла бекапа
-901	336068840	dyn_wrong_gtt_scope	@1 cannot reference @2 @1 не может ссылаться на @2
-901	336330985	gbak_db_in_use	Could not drop database @1 (database might be in use) Не могу удалить базу данных @1 (операция DROP) (база данных может использоваться)
-901	336330990	gbak_sysmemex	System memory exhausted Системная память исчерпана
-901	335544559	bad_svc_handle	Invalid service handle Неверный дескриптор службы
-901	335544561	wrospbver	Wrong version of service parameter block Неверная версия блока параметров сервиса
-901	335544562	bad_spb_form	Unrecognized service parameter block Не распознан блок параметров сервиса
-901	335544563	svcnotdef	Service @1 is not defined Сервис @1 не определен
-901	336068856	dyn_ods_not_supp_feature	Feature '@1' is not supported in ODS @2.@3 Возможность '@1' не поддерживается в ODS @2.@3
-901	336331002	gbak_restore_role_failed	SQL role

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			Ошибка восстановления SQL роли
-901	336331005	gbak_role_op_missing	SQL role parameter missing Параметр SQL роль отсутствует
-901	336331010	gbak_page_buffers_missing	Page buffers parameter missing Параметр Page buffers отсутствует
-901	336331011	gbak_page_buffers_wrong_param	Expected page buffers, encountered "@1" Ожидался параметр page buffers, встретилось @1
-901	336331012	gbak_page_buffers_restore	Page buffers is allowed only on restore or create Параметр page buffers позволяет только восстановить или создать
-901	336331014	gbak_inv_size	Size specification either missing or incorrect for file @1 Размер спецификации или отсутствует или некорректный для файла @1
-901	336331015	gbak_file_outof_sequence	File @1 out of sequence Файл @1 выпал из последовательности
-901	336331016	gbak_join_file_missing	Can't join - one of the files missing Не могу соединить – один из файлов отсутствует
-901	336331017	gbak_stdin_not_supptd	standard input is not supported when using join operation Поток stdin не поддерживается при использование операции «соединить» (join)
-901	336331018	gbak_stdout_not_supptd	Standard output is not supported when using split operation

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			Поток stdout не поддерживается при использование операции "разделить" (split)
-901	336331019	gbak_bkup_corrupt	Backup file @1 might be corrupt Файл бэкапа <строка> может быть поврежден
-901	336331020	gbak_unk_db_file_spec	Database file specification missing Файл спецификации базы данных отсутствует
-901	336331021	gbak_hdr_write_failed	Can't write a header record to file @1 Не могут записать заголовок в файл @1
-901	336331022	gbak_disk_space_ex	Free disk space exhausted Свободное место на диске исчерпано
-901	336331023	gbak_size_lt_min	File size given (@1) is less than minimum allowed (@2) Размер файла дан (@1) – является меньшим чем минимально допущенный (@2)
-901	336331025	gbak_svc_name_missing	Service name parameter missing Параметр имя службы отсутствует
-901	336331026	gbak_not_ownr	Cannot restore over current database, must be SYSDBA or owner of the existing database. Не могу восстановить поверх текущей базы данных, должен быть SYSDBA или владелец существующей базы данных.
-901	336331031	gbak_mode_req	"read_only" or "read_write" required Требуются режимы "read_only" или "read_write"

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
-901	336331033	gbak_just_data	Just data ignore all constraints etc. Только данные, игнорируя все условия контроля целостности данных
-901	336331034	gbak_data_only	Restoring data only ignoring foreign key, unique, not null & other constraints Режим восстановления «только данные» игнорирующий все ограничения по внешним ключам, условиям уникальности данных, NOT NULL и прочим условиям проверки целостности данных
-901	335544609	index_name	INDEX @1
-901	335544610	exception_name	EXCEPTION @1
-901	335544611	field_name	COLUMN @1
-901	335544613	union_err	Union not supported Операция UNION не поддерживается
-901	335544614	dsql_construct_err	Unsupported DSQL construct Не поддерживаемая конструкция DSQL
-901	335544623	dsql_domain_err	Illegal use of keyword VALUE Неверное использование слова VALUE
-901	335544626	table_name	TABLE @1
-901	335544627	proc_name	PROCEDURE @1
-901	335544641	dsql_domain_not_found	Specified domain or source column @1 does not exist Указанный домен или столбец – источник @1 не существует
-901	335544656	dsql_var_conflict	Variable @1 conflicts with parameter in same procedure

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			Переменная @1 конфликтует с одноименным параметром в процедуре
-901	335544666	srvr_version_too_old	Server version too old to support all CREATE DATABASE options Слишком старая версия сервера чтобы поддерживать все опции команды CREATE DATABASE
-901	335544673	no_delete	Cannot delete Не могу удалить
-901	335544675	sort_err	Sort error Ошибка сортировки
-901	335544703	svcnoexe	Service @1 does not have an associated executable Сервис @1 не имеет связанного исполнителя
-901	335544704	net_lookup_err	Failed to locate host machine. Не удалось найти хост машины
-901	335544705	service_unknown	Undefined service @1/@2. Неопределенная служба @1/@2.
-901	335544706	host_unknown	The specified name was not found in the hosts file or Domain Name Services. Указанное имя не найдено в файле hosts или в DNS.
-901	335544711	unprepared_stmt	Attempt to execute an unprepared dynamic SQL statement. Попытка выполнить неподготовленный DSQL запрос.
-901	335544716	svc_in_use	Service is currently busy: @1 Сервис в настоящий момент занят: @1
-901	335544731	tra_must_sweep	

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
-901	335544740	udf_exception	A fatal exception occurred during the execution of a user defined function. Фатальное исключение произошло во время выполнения UDF.
-901	335544741	lost_db_connection	Connection lost to database Соединение с базой данных потеряно
-901	335544742	no_write_user_priv	User cannot write to RDB $USER_PRIVILEGES Пользователь не может писать в таблицу RDB $USER_PRIVILEGES
-901	335544767	blob_filter_exception	A fatal exception occurred during the execution of a blob filter. Фатальное исключение произошло во время исполнения BLOB фильтра.
-901	335544768	exception_access_violation	Access violation.The code attempted to access a virtual address without privilege to do so. Ошибка доступа. Код попытался получить доступ к виртуальному адресу без соответствующих привилегий на это.
-901	335544769	exception_datatype_missalignment	Datatype misalignment.The attempted to read or write a value that was not stored on a memory boundary. Тип данных не выровнен. Попытка прочитать или записать значение, которое не хранится в границах области памяти.
-901	335544770	exception_array_bounds_exceeded	Array bounds exceeded. The code attempted to access an array element that is out of bounds.

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
			Превышение границ массива. Код пытался получить доступ к элементам массива за его пределами.
-901	335544771	exception_float_denormal_ operand	Float denormal operand.One of the floating-point operands is too small to represent a standard float value. Аномальное число с плавающей точкой. Один из операндов с плавающей точкой слишком мал, чтобы представить стандартным значением с плавающей точкой.
-901	335544772	exception_float_divide_by_zero	Floating-point divide by zero.The code attempted to divide a floating-point value by zero. Числа с плавающей точкой; деление на ноль. Код попытался выполнить деление числа с плавающей точкой на ноль.
-901	335544773	exception_float_inexact_result	Floating-point inexact result.The result of a floating-point operation cannot be represented as a decimal fraction. Числа с плавающей точкой; неточный результат. Результат операции с плавающей точкой не может быть представлен в виде десятичной дроби.
-901	335544774	exception _float_invalid_operand	Floating-point invalid operand.An indeterminant error occurred during a floating-point operation. Числа с плавающей точкой; неверная операция. Неопределяемая ошибка произошла во время операций с плавающей точкой.
-901	335544775	exception_float_overflow	Floating-point overflow.The exponent of a floating-point operation is greater than the magnitude allowed.

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
			Числа с плавающей точкой; переполнение. Показатель операции с плавающей точкой больше, чем допустимая величина.
-901	335544776	exception_float_stack_check	Floating-point stack check.The stack overflowed or underflowed as the result of a floating-point operation. Числа с плавающей точкой; проверка стека. Стек переполнен или показатель операции с плавающей точкой меньше величины допустимого, в результате операции с плавающей точкой.
-901	335544777	exception_float_underflow	Floating-point underflow.The exponent of a floating-point operation is less than the magnitude allowed. Числа с плавающей точкой. Показатель операции с плавающей точкой меньше величины допустимого.
-901	335544778	exception_integer_divide_by_zero	Integer divide by zero.The code attempted to divide an integer value by an integer divisor of zero. Целые числа; деление на ноль. Код попытался выполнить операцию целочисленного деления целого числа на ноль.
-901	335544779	exception_integer_overflow	Integer overflow.The result of an integer operation caused the most significant bit of the result to carry. Переполнение целого числа. В результате операций с целыми числами был выставлен самый старший бит, отвечающий за перенос.

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
-901	335544780	exception_unknown	An exception occurred that does not have a description.Exception number @1. Произошло исключение, но оно не имеет описания. Номер исключения @1.
-901	335544781	exception_stack_overflow	Stack overflow.The resource requirements of the runtime stack have exceeded the memory available to it. Переполнение стека. Требования ресурсов к операциях к стеку превысили память отведенную под него.
-901	335544782	exception_sigsegv	Segmentation Fault. The code attempted to access memory without priviledges. Ошибка сегментации. Код попытался получить доступ к области памяти без соответствующих привилегий.
-901	335544783	exception_sigill	Illegal Instruction. The Code attempted to perfrom an illegal operation. Неверная инструкция. Код попытался выполнить нелегальную операцию.
-901	335544784	exception_sigbus	Bus Error. The Code caused a system bus error. Ошибка шины. Код вызвал системную ошибку шины.
-901	335544785	exception_sigfpe	Floating Point Error. The Code caused an Arithmetic Exception or a floating point exception. Ошибка операции с плавающей точкой. Код вызвал арифметическое исключение или исключение операций с плавающей точкой.

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
-901	335544786	ext_file_delete	Cannot delete rows from external files. Невозможно удалить строки из внешних таблиц/файлов
-901	335544787	ext_file_modify	Cannot update rows in external files. Не могут обновлять строки во внешних файлах.
-901	335544788	adm_task_denied	Unable to perform operation.You must be either SYSDBA or owner of the database Не могу выполнить операцию. Вы должны быть SYSDBA или владельцем базы данных.
-901	335544794	cancelled	Operation was cancelled Операция была отменена
-901	335544797	svcnouser	User name and password are required while attaching to the services manager Для доступа к менеджеру сервисов требуется имя пользователя и пароль
-901	335544801	datype_notsup	Data type not supported for arithmetic Тип данных не поддерживается для арифметических операций
-901	335544803	dialect_not_changed	Database dialect not changed. Диалект базы данных не изменен.
-9 01	335544804	database_create_failed	Unable to create database @1 Не могу создать базу данных @1
-901	335544805	inv_dialect_specified	Database dialect @1 is not a valid dialect. Диалект базы данных @1 не является верным диалектом

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
-901	335544806	valid_db_dialects	Valid database dialects are @1. Верным диалектом базы данных является @1.
-901	335544811	inv_client_dialect_specified	Passed client dialect @1 is not a valid dialect. Переданный клиентской программой диалект @1 не является верным диалектом
-901	335544812	valid_client_dialects	Valid client dialects are @1. Верным клиентским диалектом является @1.
-901	335544814	service_not_supported	Services functionality will be supported in a later versionof the product Функциональность сервисов будет поддерживаться в дальнейших версиях продукта
-901	335544820	invalid_savepoint	Unable to find savepoint with name @1 in transaction context Не могу найти точку сохранения транзакции с именем @1 в контексте транзакции
-901	335544835	bad_shutdown_mode	Target shutdown mode is invalid for database "@1" Указание режима «шатдаун» неверное для базы данных "@1"
-901	335544840	no_update	Cannot update Не могу обновить
-901	335544842	stack_trace	@1
-901	335544843	ctx_var_not_found	Context variable @1 is not found in namespace @2 Контекстная переменная @1 не найдена в пространстве имен @2
-901	335544844	ctx_namespace_invalid	Invalid namespace name @1 passed to @2

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			Неверное имя пространства имен @1 передается в @2
-901	335544845	ctx_too_big	Too many context variables Слишком много контекстных переменных
-901	335544846	ctx_bad_argument	Invalid argument passed to @1 Неверный аргумент передан в @1
-901	335544847	identifier_too_long	BLR syntax error. Identifier @1... is too long Ошибка синтаксиса BLR. Идентификатор @1 является слишком большим
-901	335544859	invalid_time_precision	Time precision exceeds allowed range (0-@1) Уровень точности времени (тип данных TIME) превышает допустимый диапазон (0 - @1)
-901	335544866	met_wrong_gtt_scope	@1 cannot depend on @2 <строка> не может зависеть от <строка>
-901	335544868	illegal_prc_type	Procedure @1 is not selectable (it does not contain a SUSPEND statement) Процедура @1 не является процедурой выборки (она не содержит оператор SUSPEND)
-901	335544869	invalid_sort_datatype	Datatype @1 is not supported for sorting operation Тип данных @1 не поддерживается для операции сортировка
-901	335544870	collation_name	COLLATION @1 Порядок сортировки @1
-901	335544871	domain_name	DOMAIN @1

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			ДОМЕН @1
-901	335544874	max_db_per_trans_allowed	A multi database transaction cannot span more than @1 databases Мультибазовая транзакция не может занимать более чем @1 баз данных
-901	335544876	bad_proc_BLR	Error while parsing procedure @1' s BLR Ошибка во время разбора BLR процедуры @1
-901	335544877	key_too_big	Index key too big Ключ индекса слишком велик
-901	336397211	dsql_too_many_values	Too many values (more than @1) in member list to match against Слишком много значений (более чем @1) в списке членов, чтобы соответствовать против
-901	336397236	dsql_unsupp_feature_dialect	Feature is not supported in dialect @1 Особенность не поддерживается в диалекте @1
-902	335544333	bug_check	Internal gds software consistency check (@1) Внутренняя проверка целостности программного обеспечения (@1)
-902	335544335	db_corrupt	Database file appears corrupt (@1) Файл базы данных является поврежденным (@1)
-902	335544344	io_error	I/O error for file "@2" Ошибка ввода – вывода для файла @1
-902	335544346	metadata_corrupt	Corrupt system table

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
			Повреждение системной таблицы
-902	335544373	sys_request	Operating system directive @1 failed Директива операционной системы @1 не удалась
-902	335544384	badblk	Internal error Внутренняя ошибка
-902	335544385	invpoolcl	Internal error Внутренняя ошибка
-902	335544387	relbadblk	Internal error Внутренняя ошибка
-902	335544388	blktoobig	Block size exceeds implementation restriction Размер блока превышает ограничение реализации
-902	335544394	badodsver	Incompatible version of on-disk structure Несовместимая версия ODS
-902	335544397	dirtypage	Internal error Внутренняя ошибка
-902	335544398	waifortra	Internal error Внутренняя ошибка
-902	335544399	doubleloc	Internal error Внутренняя ошибка
-902	335544400	nodnotfnd	Internal error Внутренняя ошибка
-902	335544401	dupnodfnd	Internal error Внутренняя ошибка
-902	335544402	locnotmar	Internal error

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
			Внутренняя ошибка
-902	335544404	corrupt	Database corrupted База данных повреждена
-902	335544405	badpage	Checksum error on database page @1 Ошибка контрольной суммы на странице базы данных @1
-902	335544406	badindex	Index is broken Индекс поврежден
-902	335544409	trareqmis	Transaction -- request mismatch (synchronization error) Транзакция – несоответствие запроса (ошибка синхронизации)
-902	335544410	badhndcnt	Bad handle count Неверный счетчик указателей
-902	335544411	wrotpbver	Wrong version of transaction parameter block Неверная версия блока параметров транзакции
-902	335544412	wroblrver	Unsupported BLR version (expected @1, encountered @2) Не поддерживаемая версия BLR (ожидается @1 встретилась @2)
-902	335544413	wrodpbver	Wrong version of database parameter block Неверная версия блока параметров базы данных
-902	335544415	badrelation	Database corrupted База данных разрушена
-902	335544416	nodetach	Internal error Внутренняя ошибка
-902	335544417	notremote	Internal error

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
			Внутренняя ошибка
-902	335544422	dbfile	Internal error Внутренняя ошибка
-902	335544423	orphan	Internal error Внутренняя ошибка
-902	335544432	lockmanerr	Lock manager error Ошибка менеджера блокировок
-902	335544436	sqlerr	SQL error code = @1 SQL ошибка код = @1
-902	335544448	bad_sec_info	
-902	335544449	invalid_sec_info	
-902	335544470	buf_invalid	Cache buffer for page @1 invalid Буфер КЭШа для страницы @1 неверный
-902	335544471	indexnotdefined	There is no index in table @1 with id @2 Не существует индекса в таблице @1 с идентификатором @2
-902	335544472	login	Your user name and password are not defined. Ask your database administrator to set up a Firebird login. Ваши имя и пароль не определены. Чтобы установить соединение с Firebird обратитесь к администратору базы данных.
-902	335544506	shutinprog	Database @1 shutdown in progress Выполняется останов (shutdown) базы данных @1
-902	335544528	shutdown	Database @1 shutdown

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			База данных @1 в режиме «шатдаун»
-902	335544557	shutfail	Database shutdown unsuccessful Не успешный шатдаун базы данных
-902	335544569	dsql_error	Dynamic SQL Error Ошибка динамического SQL
-902	335544653	psw_attach	Cannot attach to password database Невозможно соединиться с базой данных с этим паролем
-902	335544654	psw_start_trans	Cannot start transaction for password database Невозможно стартовать транзакцию для базы данных пароля
-902	335544717	err_stack_limit	Stack size insufficent to execute current request Размер стека недостаточен для выполнения текущего запроса
-902	335544721	network_error	Unable to complete network request to host "@1". Невозможно завершить сетевой запрос на хост "@1"
-902	335544722	net_connect_err	Failed to establish a connection. Ошибка при установлении соединения
-902	335544723	net_connect_listen_err	Error while listening for an incoming connection. Ошибка при прослушивании входного соединения
-902	335544724	net_event_connect_err	Failed to establish a secondary connection for event processing.

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			Ошибка при установлении вторичного соединения для обработки события
-902	335544725	net_event_listen_err	Error while listening for an incoming event connection request. Ошибка при прослушивании входного запроса для события соединения
-902	335544726	net_read_err	Error reading data from the connection. Ошибка чтения данных из соединения
-902	335544727	net_write_err	Error writing data to the connection. Ошибка записи данных в соединение
-902	335544732	unsupported_network_drive	Access to databases on file servers is not supported. Доступ к базам данных в файловых серверах не поддерживается
-902	335544733	io_create_err	Error while trying to create file Ошибка ввода-вывода при попытке создания файла
-902	335544734	io_open_err	Error while trying to open file Ошибка ввода-вывода при попытке открытия файла
-902	335544735	io_close_err	Error while trying to close file Ошибка ввода-вывода при попытке закрытия файла
-902	335544736	io_read_err	Error while trying to read from file Ошибка ввода-вывода при попытке чтения из файла
-902	335544737	io_write_err	Error while trying to write to file

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			Ошибка ввода-вывода при попытке записи в файл
-902	335544738	io_delete_err	Error while trying to delete file Ошибка ввода-вывода при попытке удаления файла
-902	335544739	io_access_err	Error while trying to access file Ошибка ввода-вывода при попытке доступа к файлу
-902	335544745	login_same_as_role_name	Your login @1 is same as one of the SQL role name. Ask your database administrator to set up a valid Firebird login. Ваше регистрационное имя @1 то же, что и имя роли SQL. Уточните у вашего администратора базы данных допустимое регистрационное имя Firebird.
-902	335544791	file_in_use	The file @1 is currently in use by another process.Try again later. Файл @1 в настоящее время используется другим процессом. Попытайтесь позже.
-902	335544795	unexp_spb_form	Unexpected item in service parameter block, expected @1 Неопределенный элемент в блоке параметров сервиса, ожидается @1
-902	335544809	extern_func_dir_error	Function @1 is in @2, which is not in a permitted directory for external functions. Функция @1 находится в @2, что не является доступным каталогом для внешних функций.
-902	335544819	io_32bit_exceeded_err	File exceeded maximum size of 2GB. Add another database file or use a 64 bit I/O version of Firebird.

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
			Файл превысил максимальный размер 2 Гбайт. Добавьте другой файл базы данных или используйте 64битовую версию Firebird.
-902	335544831	conf_access_denied	Access to @1 "@2" is denied by server administrator Доступ к @1 "@2" отвергнут администратором сервера
-902	335544834	cursor_not_open	Cursor is not open Курсор не открыт
-902	335544841	cursor_already_open	Cursor is already open Курсор уже открыт
-902	335544856	att_shutdown	Connection shutdown Соединение остановлено
-902	335544882	long_login	Login name too long (@1 characters, maximum allowed @2) Наименование логина слишком велико (@1 символов, максимально возможно @2)
-904	335544324	bad_db_handle	Invalid database handle (no active connection) Указатель базы данных неверный (нет активного соединения)
-904	335544375	unavailable	Unavailable database Недоступная база данных
-904	335544381	imp_exc	Implementation limit exceeded Исчерпан лимит выполнения
-904	335544386	nopoolids	Too many requests Слишком много запросов
-904	335544389	bufexh	Buffer exhausted Исчерпан буфер

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
-904	335544391	bufinuse	Buffer in use Буфер используется
-904	335544393	reqinuse	Request in use Запрос используется
-904	335544424	no_lock_mgr	No lock manager available Нет доступного менеджера блокировок
-904	335544430	virmemexh	Unable to allocate memory from operating system Невозможно выделить память в операционной системе
-904	335544451	update_conflict	Update conflicts with concurrent update Изменение конфликтует с конкурирующим обновлением
-904	335544453	obj_in_use	Object @1 is in use Объект @1 используется
-904	335544455	shadow_accessed	Cannot attach active shadow file Невозможно соединиться с активным файлом теневой копии
-904	335544460	shadow_missing	A file in manual shadow @1 is unavailable Файл в ручной теневой копии @1 недоступен
-904	335544661	index_root_page_full	Cannot add index, index root page is full. Невозможно добавить индекс, корневая страница индексов заполнена
-904	335544676	sort_mem_err	Sort error: not enough memory Ошибка сортировки: недостаточно памяти

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
-904	335544683	req_depth_exceeded	Request depth exceeded. (Recursive definition?) Превышена глубина запроса (рекурсивное определение?)
-904	335544758	sort_rec_size_err	Sort record size of @1 bytes is too big байт Размер записи сортировки в @1 байт слишком велик
-904	335544761	too_many_handles	Too many open handles to database Слишком много открыто дескрипторов базы данных
-904	335544792	service_att_err	Cannot attach to services manager Не могу подключиться к менеджеру сервисов
-904	335544799	svc_name_missing	The service name was not specified. Не указано имя сервиса
-904	335544813	optimizer_between_err	Unsupported field type specified in BETWEEN predicate. Указан не поддерживаемый тип поля в предикате BETWEEN
-904	335544827	exec_sql_invalid_arg	Invalid argument in EXECUTE STATEMENT-cannot convert to string Неверный аргумент в EXECUTE STATEMENT – невозможно конвертировать в строку
-904	335544828	exec_sql_invalid_req	Wrong request type in EXECUTE STATEMENT '@1' Неверный тип запроса в EXECUTE STATEMENT '@1'
-904	335544829	exec_sql_invalid_var	Variable type (position @1) in EXECUTE STATEMENT '@2' INTO does not match returned column type

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			Тип переменной (позиция @1) в EXECUTE STATEMENT '@2' INTO не соответствует возвращаемому типу столбца
-904	335544830	exec_sql_max_call_exceeded	Too many recursion levels of EXECUTE STATEMENT Слишком много уровней рекурсии в EXECUTE STATEMENT
-904	335544832	wrong_backup_state	Cannot change difference file name while database is in backup mode Не могу изменить файл – разницу пока база данных находится в режиме бекапа
-904	335544852	partner_idx_incompat_type	Partner index segment no @1 has incompatible data type Сегмент @1 индекса – партнера содержит не совместимый тип данных
-904	335544857	blobtoobig	Maximum BLOB size exceeded Достигнут максимальный размер BLOB
-904	335544862	record_lock_not_supp	Stream does not support record locking Поток не поддерживает блокировку записей.
-904	335544863	partner_idx_not_found	Cannot create foreign key constraint @1. Partner index does not exist or is inactive. Не могу создать конструкцию внешнего ключа. Индекс-партнер не существует или является неактивным.
-904	335544864	tra_num_exc	Transactions count exceeded. Perform backup and restore to make database operable again Превышено число допустимых транзакций. Выполнение бекапа

SQLCODE	GDSCODE	SYMBOL	TEXT **Описание и перевод сообщения.**
			и рестор сделает вновь базу данных работоспособной.
-904	335544865	field_disappeared	Column has been unexpectedly deleted Столбец был неожиданно удален
-904	335544878	concurrent_transaction	Concurrent transaction number is @1 Число конкурирующих транзакций @1
-906	335544744	max_att_exceeded	Maximum user count exceeded.Contact your database administrator. Превышен максимум счетчика пользователей. Свяжитесь с вашим администратором базы данных.
-909	335544667	drdb _ completed _ with _ errs	Drop database completed with errors Удаление базы данных завершилось с ошибками
-911	335544459	rec_in_limbo	Record from transaction @1 is stuck in limbo Запись транзакции @1 становится зависшей
-913	335544336	Deadlock	Deadlock Взаимная блокировка
-922	335544323	bad_db_format	File @1 is not a valid database Файл @1 не является допустимой базой данных
-923	335544421	connect_reject	Connection rejected by remote interface Соединение отменено удаленным интерфейсом
-923	335544461	cant_validate	Secondary server attachments cannot validate databases

SQLCODE	GDSCODE	SYMBOL	TEXT Описание и перевод сообщения.
			Вторичные подключения к серверу не могут проверять базу данных
-923	335544464	cant_start_logging	Secondary server attachments cannot start logging Вторичные подключения к серверу не могут начинать логгирование
-924	335544325	bad_dpb_content	Bad parameters on attach or create database Неверные параметры при подключении или создании базы данных
-924	335544441	bad_detach	Database detach completed with errors Отключение от базы данных завершилось с ошибками
-924	335544648	conn_lost	Connection lost to pipe server Потеря соединения с каналом сервера
-926	335544447	no_rollback	No rollback performed Не выполнен откат транзакции
-999	335544689	ib_error	Firebird error Ошибка Firebird

Приложение C: Зарезервированные и ключевые слова

Зарезервированные слова являются частью языка SQL Firebird. Они не могут быть использованы в качестве идентификаторов (например, имён таблиц или процедур), за исключением случаев когда они заключены в двойные кавычки (при использовании 3 диалекта). Вы должны стараться избегать их использования, если на то нет серьёзных причин.

Ключевые слова также являются частью языка. Они имеют особое значение при использовании в определённом контексте, но не являются зарезервированными для монопольного использования самим сервером Firebird. Вы можете использовать их в качестве идентификаторов без заключения в двойные кавычки.

Зарезервированные слова

ADD
ADMIN
ALL
ALTER
AND
ANY
AS
AT
AVG
BEGIN
BETWEEN
BIGINT
BIT_LENGTH
BLOB
BOOLEAN
BOTH
BY
CASE
CAST
CHAR
CHAR_LENGTH
CHARACTER
CHARACTER_LENGTH
CHECK
CLOSE
COLLATE
COLUMN
COMMIT
FETCH
FILTER
FLOAT
FOR
FOREIGN
FROM
FULL
FUNCTION
GDSCODE
GLOBAL
GRANT
GROUP
HAVING
HOUR
IN
INDEX
INNER
INSENSITIVE
INSERT
INSERTING
INT
INTEGER
INTO
IS
JOIN
LEADING
LEFT
LIKE
REFERENCES
REGR_AVGX
REGR_AVGY
REGR_COUNT
REGR_INTERCEPT
REGR_R2
REGR_SLOPE
REGR_SXX
REGR_SXY
REGR_SYY
RELEASE
RETURN
RETURNING_VALUES
RETURNS
REVOKE
RIGHT
ROLLBACK
ROW
ROW_COUNT
ROWS
SAVEPOINT
SCROLL
SECOND
SELECT
SENSITIVE
SET
SIMILAR
SMALLINT

CONNECT
CONSTRAINT
CORR
COUNT
COVAR_POP
COVAR_SAMP
CREATE
CROSS
CURRENT
CURRENT_CONNECTION
CURRENT_DATE
CURRENT_ROLE
CURRENT_TIME
CURRENT_TIMESTAMP
CURRENT_TRANSACTION
CURRENT_USER
CURSOR
DATE
DAY
DEC
DECIMAL
DECLARE
DEFAULT
DELETE
DELETING
DETERMINISTIC
DISCONNECT
DISTINCT
DOUBLE
DROP
ELSE
END
ESCAPE
EXECUTE
EXISTS
EXTERNAL
EXTRACT
FALSE
LONG
LOWER
MAX
MAXIMUM_SEGMENT
MERGE
MIN
MINUTE
MONTH
NATIONAL
NATURAL
NCHAR
NO
NOT
NULL
NUMERIC
OCTET_LENGTH
OF
OFFSET
ON
ONLY
OPEN
OR
ORDER
OUTER
OVER
PARAMETER
PLAN
POSITION
POST_EVENT
PRECISION
PRIMARY
PROCEDURE
RDB$DB_KEY
RDB$RECORD_VERSION
REAL
RECORD_VERSION
RECREATE
RECURSIVE
SOME
SQLCODE
SQLSTATE
START
STDDEV_POP
STDDEV_SAMP
SUM
TABLE
THEN
TIME
TIMESTAMP
TO
TRAILING
TRIGGER
TRIM
TRUE
UNION
UNIQUE
UNKNOWN
UPDATE
UPDATING
UPPER
USER
USING
VALUE
VALUES
VAR_POP
VAR_SAMP
VARCHAR
VARIABLE
VARYING
VIEW
WHEN
WHERE
WHILE
WITH
YEAR

Ключевые слова

Следующие термины имеют особое значение в DSQL Firebird. Некоторые из них также являются и зарезервированными словами.

!<
^<
^=
^>
,
:=
!=
ELSE
ENCRYPT
ENGINE
END
ENTRY_POINT
ESCAPE
EXCEPTION
PRIMARY
PRIOR
PRIVILEGES
PROCEDURE
PROTECTED
RAND
RANK

!>
(
)
<
<=
<>
=
>
>=
||
~<
~=
~>
ABS
ABSOLUTE
ACCENT
ACOS
ACOSH
ACTION
ACTIVE
ADD
ADMIN
AFTER
ALL
ALTER
ALWAYS
AND
ANY
AS
ASC
ASCENDING
ASCII_CHAR
ASCII_VAL
ASIN
ASINH
AT
ATAN
ATAN2
ATANH
AUTO
AUTONOMOUS
AVG
BACKUP
BEFORE
BEGIN
BETWEEN
BIGINT
BIN_AND
BIN_NOT
BIN_OR
BIN_SHL
BIN_SHR
EXECUTE
EXISTS
EXIT
EXP
EXTERNAL
EXTRACT
FALSE
FETCH
FILE
FILTER
FIRST
FIRST_VALUE
FIRSTNAME
FLOAT
FLOOR
FOR
FOREIGN
FREE_IT
FROM
FULL
FUNCTION
GDSCODE
GEN_ID
GEN_UUID
GENERATED
GENERATOR
GLOBAL
GRANT
GRANTED
GROUP
HASH
HAVING
HOUR
IF
IGNORE
IIF
IN
INACTIVE
INCREMENT
INDEX
INNER
INPUT_TYPE
INSENSITIVE
INSERT
INSERTING
INT
INTEGER
INTO
IS
ISOLATION
JOIN
KEY
RDB$GET_CONTEXT
RDB$DB_KEY
RDB$RECORD_VERSION
RDB$SET_CONTEXT
READ
REAL
RECORD_VERSION
RECREATE
RECURSIVE
REFERENCES
REGR_AVGX
REGR_AVGY
REGR_COUNT
REGR_INTERCEPT
REGR_R2
REGR_SLOPE
REGR_SXX
REGR_SXY
REGR_SYY
RELATIVE
RELEASE
REPLACE
REQUESTS
RESERV
RESERVING
RESTART
RESTRICT
RETAIN
RETURN
RETURNING
RETURNING_VALUES
RETURNS
REVERSE
REVOKE
RIGHT
ROLE
ROLLBACK
ROUND
ROW
ROW_COUNT
ROW_NUMBER
ROWS
RPAD
SAVEPOINT
SCALAR_ARRAY
SCHEMA
SCROLL
SECOND
SEGMENT
SELECT
SENSITIVE
SEQUENCE

BIN_XOR
BIT_LENGTH
BLOB
BLOCK
BODY
BOOLEAN
BOTH
BREAK
BY
CALLER
CASCADE
CASE
CAST
CEIL
CEILING
CHAR
CHAR_LENGTH
CHAR_TO_UUID
CHARACTER
CHARACTER_LENGTH
CHECK
CLOSE
COALESCE
COLLATE
COLLATION
COLUMN
COMMENT
COMMIT
COMMITTED
COMMON
COMPUTED
CONDITIONAL
CONNECT
CONSTRAINT
CONTAINING
CONTINUE
CORR
COS
COSH
COT
COUNT
COVAR_POP
COVAR_SAMP
CREATE
CROSS
CSTRING
CURRENT
CURRENT_CONNECTION
CURRENT_DATE
CURRENT_ROLE
CURRENT_TIME
CURRENT_TIMESTAMP
LAG
LAST
LAST_VALUE
LASTNAME
LEAD
LEADING
LEAVE
LEFT
LENGTH
LEVEL
LIKE
LIMBO
LINGER
LIST
LN
LOCK
LOG
LOG10
LONG
LOWER
LPAD
MANUAL
MAPPING
MATCHED
MATCHING
MAX
MAXIMUM_SEGMENT
MAXVALUE
MERGE
MIDDLENAME
MILLISECOND
MIN
MINUTE
MINVALUE
MOD
MODULE_NAME
MONTH
NAME
NAMES
NATIONAL
NATURAL
NCHAR
NEXT
NO
NOT
NTH_VALUE
NULL
NULLIF
NULLS
NUMERIC
OCTET_LENGTH
OF
SERVERWIDE
SET
SHADOW
SHARED
SIGN
SIMILAR
SIN
SINGULAR
SINH
SIZE
SKIP
SMALLINT
SNAPSHOT
SOME
SORT
SOURCE
SPACE
SQLCODE
SQLSTATE
SQRT
STABILITY
START
STARTING
STARTS
STATEMENT
STATISTICS
STDDEV_POP
STDDEV_SAMP
SUB_TYPE
SUBSTRING
SUM
SUSPEND
TABLE
TAGS
TAN
TANH
TEMPORARY
THEN
TIME
TIMEOUT
TIMESTAMP
TO
TRAILING
TRANSACTION
TRIGGER
TRIM
TRUE
TRUNC
TRUSTED
TWO_PHASE
TYPE
UNCOMMITTED

CURRENT_TRANSACTION
CURRENT_USER
CURSOR
DATA
DATABASE
DATE
DATEADD
DATEDIFF
DAY
DDL
DEC
DECIMAL
DECLARE
DECODE
DECRYPT
DEFAULT
DELETE
DELETING
DENSE_RANK
DESC
DESCENDING
DESCRIPTOR
DETERMINISTIC
DIFFERENCE
DISCONNECT
DISTINCT
DO
DOMAIN
DOUBLE
DROP
OFFSET
ON
ONLY
OPEN
OPTION
OR
ORDER
OS_NAME
OUTER
OUTPUT_TYPE
OVER
OVERFLOW
OVERLAY
PACKAGE
PAD
PAGE
PAGE_SIZE
PAGES
PARAMETER
PARTITION
PASSWORD
PI
PLACING
PLAN
PLUGIN
POSITION
POST_EVENT
POWER
PRECISION
PRESERVE
UNDO
UNION
UNIQUE
UNKNOWN
UPDATE
UPDATING
UPPER
USAGE
USER
USING
UUID_TO_CHAR
VALUE
VALUES
VAR_POP
VAR_SAMP
VARCHAR
VARIABLE
VARYING
VIEW
WAIT
WEEK
WEEKDAY
WHEN
WHERE
WHILE
WITH
WORK
WRITE
YEAR
YEARDAY

Приложение D: Описания системных таблиц

При первоначальном создании базы данных система управления базами данных создаёт множество системных таблиц. В системных таблицах хранятся метаданные — описания всех объектов базы данных. Системные таблицы содержат префикс RDB$ в имени.

Таблица D.1. Системные таблицы

Таблица	Содержание
RDB$AUTH_MAPPING	Сведения об отображении объектов безопасности.
RDB$BACKUP_HISTORY	Хранит историю копирования базы данных.
RDB$CHARACTER_SETS	Доступные в базе данных наборы символов.
RDB$CHECK_CONSTRAINTS	Соответствие имён триггеров именам ограничений, связанных с характеристиками NOT NULL, ограничениями CHECK и предложениями ON UPDATE и ON DELETE в ограничениях внешнего ключа.
RDB$COLLATIONS	Порядки сортировки для всех наборов символов.
RDB$DATABASE	Основные данные о базе данных.
RDB$DB_CREATORS	Содержит сведения о пользователях имеющих права на создание базы данных. Используется только в том случае, если текущая база данных назначена как база данных безопасности.
RDB$DEPENDENCIES	Сведения о зависимостях между объектами базы данных.
RDB$EXCEPTIONS	Пользовательские исключения базы данных.
RDB$FIELD_DIMENSIONS	Размерности столбцов, являющихся массивами.
RDB$FIELDS	Характеристики столбцов и доменов, как системных, так и созданных пользователем.
RDB$FILES	Сведения о вторичных файлах и файлах теневых копий.
RDB$FILTERS	Данные о BLOB-фильтрах.
RDB$FORMATS	Данные об изменениях таблиц.
RDB$FUNCTION_ARGUMENTS	Параметры хранимых или внешних функций.
RDB$FUNCTIONS	Описание хранимых или внешних функций.
RDB$GENERATORS	Сведения о генераторах (последовательностях).

Таблица	Содержание
RDB$INDEX_SEGMENTS	Сегменты и позиции индексов.
RDB$INDICES	Определение индексов базы данных (созданных пользователем или системой).
RDB$LOG_FILES	В настоящей версии не используется.
RDB$PACKAGES	Сведения о PSQL пакетах.
RDB$PAGES	Сведения о страницах базы данных.
RDB$PROCEDURE_PARAMETERS	Параметры хранимых процедур.
RDB$PROCEDURES	Описания хранимых процедур.
RDB$REF_CONSTRAINTS	Описания именованных ограничений базы данных (внешних ключей).
RDB$RELATION_CONSTRAINTS	Описание всех ограничений на уровне таблиц.
RDB$RELATION_FIELDS	Характеристики столбцов таблиц.
RDB$RELATIONS	Заголовки таблиц и представлений.
RDB$ROLES	Определение ролей.
RDB$SECURITY_CLASSES	Списки управления доступом.
RDB$TRANSACTIONS	Состояние транзакций при обращении к нескольким базам данных.
RDB$TRIGGER_MESSAGES	Сообщения триггеров.
RDB$TRIGGERS	Описания триггеров.
RDB$TYPES	Описание перечислимых типов данных.
RDB$USER_PRIVILEGES	Полномочия пользователей системы.
RDB$VIEW_RELATIONS	Описывает представления. Не используется в настоящей версии.

RDB$AUTH_MAPPING

Сведения о локальных отображениях объектов безопасности.

Таблица D.2. Описание столбцов таблицы RDB$AUTH_MAPPING

Наименование столбца	Тип данных	Описание
RDB$MAP_NAME	CHAR(31)	Имя отображения.
RDB$MAP_USING	CHAR(1)	Является ли аутентификация общесерверной (S) или обычной (P).

Наименование столбца	Тип данных	Описание
RDB$MAP_PLUGIN	CHAR(31)	Имя плагина аутентификации, из которого происходит отображение.
RDB$MAP_DB	CHAR(31)	Имя базы данных, в которой прошла аутентификация. Из неё происходит отображение.
RDB$MAP_FROM_TYPE	CHAR(31)	Тип объекта, который будет отображён.
RDB$MAP_FROM	CHAR(255)	Имя объекта, из которого будет произведено отображение.
RDB$MAP_TO_TYPE	SMALLINT	Тип объекта, в который будет произведено отображение: • 0 — USER; • 1 — ROLE.
RDB$MAP_TO	CHAR(31)	Наименование объекта, в который будет произведено отображение (имя пользователя или роли).
RDB$SYSTEM_FLAG	SMALLINT	Признак: определён пользователем — значение 0; определён в системе — значение 1.
RDB$DESCRIPTION	BLOB TEXT	Произвольное текстовое описание порядка сортировки.

RDB$BACKUP_HISTORY

Таблица хранит историю копирования базы данных при помощи утилиты nbackup.

Таблица D.3. Описание столбцов таблицы RDB$BACKUP_HISTORY

Наименование столбца	Тип данных	Описание
RDB$BACKUP_ID	INTEGER	Присваиваемый ядром идентификатор.
RDB$TIMESTAMP	DATE	Дата и время выполнения копирования.
RDB$BACKUP_LEVEL	INTEGER	Уровень копирования.
RDB$GUID	CHAR(38)	Уникальный идентификатор.
RDB$SCN	INTEGER	Системный номер.
RDB$FILE_NAME	VARCHAR(255)	Полный путь и имя файла копии.

RDB$CHARACTER_SETS

Содержит наборы символов, доступные в базе данных.

Таблица D.4. Описание столбцов таблицы RDB$CHARACTER_SETS

Наименование столбца	Тип данных	Описание
RDB $CHARACTER_SET_NAME	CHAR(31)	Имя набора символов.
RDB$FORM_OF_USE	CHAR(31)	Не используется.
RDB $NUMBER_OF_CHARACTERS	INTEGER	Количество символов в наборе. Для существующих наборов символов не используется.
RDB $DEFAULT_COLLATE_NAME	CHAR(31)	Имя порядка сортировки по умолчанию для набора символов.
RDB$CHARACTER_SET_ID	SMALLINT	Уникальный идентификатор набора символов.
RDB$SYSTEM_FLAG	SMALLINT	Системный флаг: имеет значение 1, если набор символов был определён в системе при создании базы данных; значение 0 для набора символов, определённого пользователем.
RDB$DESCRIPTION	BLOB TEXT	Произвольное текстовое описание набора символов.
RDB$FUNCTION_NAME	CHAR(31)	Имя внешней функции для наборов символов, определённых пользователем, доступ к которым осуществляется через внешнюю функцию.
RDB $BYTES_PER_CHARACTER	SMALLINT	Количество байтов для представления одного символа.
RDB$SECURITY_CLASS	CHAR(31)	Может ссылаться на класс безопасности, определённый в таблице RDB$SECURITY_CLASSES для применения ограничений управления доступом для всех пользователей этого набора символов.
RDB$OWNER_NAME	CHAR(31)	Имя пользователя — владельца (создателя) набора символов.

RDB$CHECK_CONSTRAINTS

Описывает соответствие имён триггеров именам ограничений, связанных с характеристиками NOT NULL, ограничениями CHECK и предложениями ON UPDATE, ON DELETE в ограничениях внешнего ключа.

Таблица D.5. Описание столбцов таблицы RDB$CHECK_CONSTRAINTS

Наименование столбца	Тип данных	Описание
RDB$CONSTRAINT_NAME	CHAR(31)	Имя ограничения. Задаётся пользователем или автоматически генерируется системой.
RDB$TRIGGER_NAME	CHAR(31)	Для ограничения CHECK — это имя триггера, который поддерживает данное ограничение. Для ограничения NOT NULL — это имя столбца, к которому применяется ограничение. Для ограничения внешнего ключа – это имя триггера, который поддерживает предложения ON UPDATE, ON DELETE.

RDB$COLLATIONS

Порядки сортировки для наборов символов.

Таблица D.6. Описание столбцов таблицы RDB$COLLATIONS

Наименование столбца	Тип данных	Описание
RDB$COLLATION_NAME	CHAR(31)	Имя порядка сортировки.
RDB$COLLATION_ID	SMALLINT	Идентификатор порядка сортировки. Вместе с идентификатором набора символов является уникальным идентификатором порядка сортировки.
RDB$CHARACTER_SET_ID	SMALLINT	Идентификатор набора символов. Вместе с идентификатором порядка сортировки является уникальным идентификатором.
RDB $COLLATION_ATTRIBUTES	SMALLINT	Атрибуты сортировки. Представляет собой битовую маску, где 1-й бит показывает учитывать ли конечные пробелы при сравнении (0 — NO PAD; 1 — PAD SPACE); 2-й бит показывает является ли сравнение чувствительным к регистру символов (0 — CASE SENSITIVE, 1 — CASE INSENSITIVE); 3-й бит показывает будет ли сравнение

Наименование столбца	Тип данных	Описание
		чувствительным к акцентам (0 — ACCENT SENSITIVE, 1 — ACCENT SENSITIVE). Таким образом, значение 5 означает, что сравнение не является чувствительным к конечным пробелам и к акцентированным буквам.
RDB$SYSTEM_FLAG	SMALLINT	Признак: определён пользователем — значение 0; определён в системе — значение 1.
RDB$DESCRIPTION	BLOB TEXT	Произвольное текстовое описание порядка сортировки.
RDB$FUNCTION_NAME	CHAR(31)	В настоящий момент не используется.
RDB $BASE_COLLATION_NAME	CHAR(31)	Имя базового порядка сортировки для данного порядка сортировки.
RDB $SPECIFIC_ATTRIBUTES	BLOB TEXT	Описание особых атрибутов.
RDB$SECURITY_CLASS	CHAR(31)	Может ссылаться на класс безопасности, определённый в таблице RDB$SECURITY_CLASSES для применения ограничений управления доступом для всех пользователей этой сортировки.
RDB$OWNER_NAME	CHAR(31)	Имя пользователя — владельца (создателя) сортировки.

RDB$DATABASE

Основные данные о базе данных. Содержит только одну запись.

Таблица D.7. Описание столбцов таблицы RDB$DATABASE

Наименование столбца	Тип данных	Описание
RDB$DESCRIPTION	BLOB TEXT	Текст примечания для базы данных.
RDB$RELATION_ID	SMALLINT	Количество таблиц и представлений в базе данных.
RDB$SECURITY_CLASS	CHAR(31)	Класс безопасности, определённый в RDB$SECURITY_CLASSES, для обращения к общим для базы данных ограничениям доступа.

Наименование столбца	Тип данных	Описание
RDB $CHARACTER_SET_NAME	CHAR(31)	Имя набора символов по умолчанию для базы данных, установленного в предложении DEFAULT CHARACTER SET при создании базы данных. NULL — набор символов NONE.
RDB$LINGER	INTEGER	Количество секунд "задержки" (установленной оператором alter database set linger) до закрытия последнего соединения базы данных (в SuperServer). Если задержка не установлена, то содержит NULL.

RDB$DB_CREATORS

Содержит сведения о пользователях имеющих права на создание базы данных. Используется только в том случае, если текущая база данных назначена как база данных безопасности.

Таблица D.8. Описание столбцов таблицы RDB$DB_CREATORS

Наименование столбца	Тип данных	Описание
RDB$USER	CHAR(31)	Имя пользователя или роли, которому даны полномочия на создание базы данных.
RDB$USER_TYPE	SMALLINT	Тип пользователя: • 8 — пользователь; • 13 — роль.

RDB$DEPENDENCIES

Сведения о зависимостях между объектами базы данных.

Таблица D.9. Описание столбцов таблицы RDB$DEPENDENCIES

Наименование столбца	Тип данных	Описание
RDB$DEPENDENT_NAME	CHAR(31)	Имя представления, процедуры, триггера, ограничения CHECK или вычисляемого столбца, для которого описывается зависимость.
RDB$DEPENDED_ON_NAME	CHAR(31)	Объект, зависящий от описываемого объекта — таблица, на которую ссылается представление,

Наименование столбца	Тип данных	Описание
		процедура, триггер, ограничение CHECK или вычисляемый столбец.
RDB$FIELD_NAME	CHAR(31)	Имя столбца в зависимой таблице, на который ссылается представление, процедура, триггер, ограничение CHECK или вычисляемый столбец.
RDB$DEPENDENT_TYPE	SMALLINT	Идентифицирует тип объекта, для которого описывается зависимость: • 0 — таблица; • 1 — представление; • 2 — триггер; • 3 — вычисляемый столбец; • 4 — ограничение CHECK; • 5 — процедура; • 6 — выражение для индекса; • 9 — столбец; • 15 — хранимая функция; • 18 — заголовок пакета; • 19 — тело пакета.
RDB$DEPENDED_ON_TYPE	SMALLINT	Идентифицирует тип зависимого объекта: • 0 — таблица (или её столбец); • 1 — представление; • 2 — триггер; • 3 — вычисляемый столбец; • 4 — ограничение CHECK; • 5 — процедура; • 6 — выражение для индекса; • 7 — исключение; • 8 — пользователь; • 9 — столбец; • 10 — индекс; • 14 — генератор (последовательность); • 15 — UDF или хранимая функция; • 17 — сортировка; • 18 — заголовок пакета; • 19 — тело пакета.
RDB$PACKAGE_NAME	CHAR(31)	Пакет процедуры или функции, для которой описывается зависимость.

RDB$EXCEPTIONS

Пользовательские исключения базы данных.

Таблица D.10. Описание столбцов таблицы RDB$EXCEPTIONS

Наименование столбца	Тип данных	Описание
RDB$EXCEPTION_NAME	CHAR(31)	Имя пользовательского исключения.
RDB$EXCEPTION_NUMBER	INTEGER	Назначенный системой уникальный номер исключения.
RDB$MESSAGE	CHAR(1023)	Текст сообщения в исключении.
RDB$DESCRIPTION	BLOB TEXT	Произвольное текстовое описание исключения.
RDB$SYSTEM_FLAG	SMALLINT	Признак: определено пользователем = 0; определено системой = 1 или выше.
RDB$SECURITY_CLASS	CHAR(31)	Может ссылаться на класс безопасности, определённый в таблице RDB$SECURITY_CLASSES для применения ограничений управления доступом для всех пользователей этого исключения.
RDB$OWNER_NAME	CHAR(31)	Имя пользователя — владельца (создателя) исключения.

RDB$FIELD_DIMENSIONS

Размерности столбцов, являющихся массивами.

Таблица D.11. Описание столбцов таблицы RDB$FIELD_DIMENSIONS

Наименование столбца	Тип данных	Описание
RDB$FIELD_NAME	CHAR(31)	Имя столбца, являющегося массивом. Должно содержаться в поле RDB$FIELD_NAME таблицы RDB$FIELDS.
RDB$DIMENSION	SMALLINT	Определяет одну размерность столбца массива. Нумерация размерностей начинается с 0.
RDB$LOWER_BOUND	INTEGER	Нижняя граница этой размерности.
RDB$UPPER_BOUND	INTEGER	Верхняя граница описываемой размерности.

RDB$FIELDS

Характеристики столбцов и доменов, как системных, так и созданных пользователем.

Таблица D.12. Описание столбцов таблицы RDB$FIELDS

Наименование столбца	Тип данных	Описание
RDB$FIELD_NAME	CHAR(31)	Уникальное имя домена, созданного пользователем, или домена, автоматически построенного системой для столбца таблицы. Во втором случае имя будет начинаться с символов 'RDB$'.
RDB$QUERY_NAME	CHAR(31)	Не используется.
RDB$VALIDATION_BLR	BLOB BLR	Двоичное представление (BLR) выражения SQL, задающее проверку значения CHECK у домена.
RDB$VALIDATION_SOURCE	BLOB TEXT	Оригинальный исходный текст на языке SQL, задающий проверку значения CHECK.
RDB$COMPUTED_BLR	BLOB BLR	Двоичное представление (BLR) выражения SQL, которое используется сервером базы данных для вычисления при обращении к столбцу COMPUTED BY.
RDB$COMPUTED_SOURCE	BLOB TEXT	Оригинальный исходный текст выражения, которое определяет столбец COMPUTED BY.
RDB$DEFAULT_VALUE	BLOB BLR	Значение по умолчанию в двоичном виде BLR.
RDB$DEFAULT_SOURCE	BLOB TEXT	Значение по умолчанию в исходном виде на языке SQL.
RDB$FIELD_LENGTH	SMALLINT	Размер столбца в байтах. FLOAT, DATE, TIME, INTEGER занимают 4 байта. DOUBLE PRECISION, BIGINT, TIMESTAMP и идентификатор BLOB — 8 байтов. Для типов данных CHAR и VARCHAR столбец задаёт максимальное количество байтов, указанное при объявлении строкового домена (столбца).
RDB$FIELD_SCALE	SMALLINT	Отрицательное число задаёт масштаб для столбцов DECIMAL и NUMERIC — количество дробных знаков после десятичной точки.
RDB$FIELD_TYPE	SMALLINT	Код типа данных для столбца: • 7 – SMALLINT; • 8 – INTEGER; • 10 – FLOAT;

Наименование столбца	Тип данных	Описание
		• 12 – DATE; • 13 – TIME; • 14 – CHAR; • 16 – BIGINT; • 23 – BOOLEAN; • 27 – DOUBLE PRECISION; • 35 – TIMESTAMP; • 37 – VARCHAR; • 261 – BLOB. Коды для DECIMAL и NUMERIC имеют тот же размер, что и целые типы, используемые для их хранения.
RDB$FIELD_SUB_TYPE	SMALLINT	Для типа данных BLOB задаёт подтип: • 0 – не определён; • 1 – текст; • 2 – BLR; • 3 – список управления доступом; • 4 – резервируется для дальнейшего использования; • 5 – кодированное описание метаданных таблицы; • 6 – описание транзакции к нескольким базам данных, которая не завершилась нормально. Для типа данных CHAR задаёт: • 0 – неопределённые данные; • 1 – фиксированные двоичные данные. Для целочисленных типов данных (SMALLINT, INTEGER, BIGINT) и чисел с фиксированной точкой (NUMERIC, DECIMAL) задаёт конкретный тип данных: • 0 или NULL – тип данных соответствует значению в поле RDB$FIELD_TYPE; • 1 – NUMERIC; • 2 – DECIMAL.
RDB$MISSING_VALUE	BLOB BLR	Не используется.
RDB$MISSING_SOURCE	BLOB TEXT	Не используется.
RDB$DESCRIPTION	BLOB TEXT	Произвольный текст комментария для домена (столбца таблицы).

Наименование столбца	Тип данных	Описание
RDB$SYSTEM_FLAG	SMALLINT	Признак: значение 1 – домен, автоматически созданный системой, значение 0 – домен определён пользователем.
RDB$QUERY_HEADER	BLOB TEXT	Не используется.
RDB$SEGMENT_LENGTH	SMALLINT	Для столбцов BLOB задаёт длину буфера BLOB в байтах. Для остальных типов данных содержит NULL.
RDB$EDIT_STRING	VARCHAR(127)	Не используется.
RDB$EXTERNAL_LENGTH	SMALLINT	Длина столбца в байтах, если он входит в состав внешней таблицы. Всегда NULL для обычных таблиц.
RDB$EXTERNAL_SCALE	SMALLINT	Показатель степени для столбца целого типа данных во внешней таблице; задаётся степенью 10, на которую умножается целое.
RDB$EXTERNAL_TYPE	SMALLINT	Тип данных поля, как он представляется во внешней таблице. • 7 – SMALLINT; • 8 – INTEGER; • 10 – FLOAT; • 12 – DATE; • 13 – TIME; • 14 – CHAR; • 16 – BIGINT; • 23 – BOOLEAN; • 27 – DOUBLE PRECISION; • 35 – TIMESTAMP; • 37 – VARCHAR. Коды для DECIMAL и NUMERIC имеют тот же размер, что и целые типы, используемые для их хранения.
RDB$DIMENSIONS	SMALLINT	Задаёт количество размерностей массива, если столбец был определён как массив. Для столбцов, не являющихся массивами, всегда NULL.
RDB$NULL_FLAG	SMALLINT	Указывает, может ли столбец принимать пустое значение (в поле будет значение NULL) или не может (в поле будет содержаться значение 1).

Наименование столбца	Тип данных	Описание
RDB$CHARACTER_LENGTH	SMALLINT	Длина столбцов CHAR или VARCHAR в символах (не в байтах).
RDB$COLLATION_ID	SMALLINT	Идентификатор порядка сортировки для символьного столбца или домена. Если не задан, значением поля будет 0.
RDB$CHARACTER_SET_ID	SMALLINT	Идентификатора набора символов для символьного столбца, столбца BLOB или домена.
RDB$FIELD_PRECISION	SMALLINT	Указывает общее количество цифр для числового типа данных с фиксированной точкой (DECIMAL и NUMERIC). Для целочисленных типов данных значением является 0, для всех остальных типов данных – NULL.
RDB$SECURITY_CLASS	CHAR(31)	Может ссылаться на класс безопасности, определённый в таблице RDB$SECURITY_CLASSES для применения ограничений управления доступом для всех пользователей этого домена.
RDB$OWNER_NAME	CHAR(31)	Имя пользователя – владельца (создателя) домена.

RDB$FILES

Сведения о вторичных файлах и файлах оперативных копий.

Таблица D.13. Описание столбцов таблицы RDB$FILES

Наименование столбца	Тип данных	Описание
RDB$FILE_NAME	VARCHAR(255)	Полный путь к файлу и имя вторичного файла базы данных в многофайловой базе данных или файла оперативной копии.
RDB$FILE_SEQUENCE	SMALLINT	Порядковый номер вторичного файла в последовательности или номер файла копии в наборе оперативных копий.
RDB$FILE_START	INTEGER	Начальный номер страницы вторичного файла или файла оперативной копии.

Наименование столбца	Тип данных	Описание
RDB$FILE_LENGTH	INTEGER	Длина файла в страницах базы данных.
RDB$FILE_FLAGS	SMALLINT	Для внутреннего использования.
RDB$SHADOW_NUMBER	SMALLINT	Номер набора оперативных копий. Если строка описывает вторичный файл базы данных, то значением поля будет NULL или 0.

RDB$FILTERS

Содержит данные о BLOB-фильтрах.

Таблица D.14. Описание столбцов таблицы RDB$FILTERS

Наименование столбца	Тип данных	Описание
RDB$FUNCTION_NAME	CHAR(31)	Уникальное имя фильтра BLOB.
RDB$DESCRIPTION	BLOB TEXT	Написанная пользователем документация о фильтре BLOB и используемых двух подтипах.
RDB$MODULE_NAME	VARCHAR(255)	Имя динамической библиотеки / совместно используемого объекта, где расположен код фильтра BLOB.
RDB$ENTRYPOINT	CHAR(255)	Точка входа в библиотеке фильтров для этого фильтра BLOB.
RDB$INPUT_SUB_TYPE	SMALLINT	Подтип BLOB для преобразуемых данных.
RDB$OUTPUT_SUB_TYPE	SMALLINT	Подтип BLOB, в который преобразуются входные данные.
RDB$SYSTEM_FLAG	SMALLINT	Признак: внешне определённый фильтр (т.е. определённый пользователем = значение 0, внутренне определённый = значение 1 или более)
RDB$SECURITY_CLASS	CHAR(31)	Может ссылаться на класс безопасности, определённый в таблице RDB$SECURITY_CLASSES для применения ограничений управления доступом для всех пользователей этого BLOB фильтра.
RDB$OWNER_NAME	CHAR(31)	Имя пользователя — владельца (создателя) BLOB фильтра.

RDB$FORMATS

Данные об изменениях таблиц. Каждый раз, когда таблица изменяется, таблица получает новый номер формата. Когда номер формата любой таблицы достигает 255, вся база данных становится недоступной для работы с ней. Тогда нужно выполнить резервное копирование, восстановить эту копию и продолжить работу с заново созданной базой данных.

Таблица D.15. Описание столбцов таблицы RDB$FORMATS

Наименование столбца	Тип данных	Описание
RDB$RELATION_ID	SMALLINT	Идентификатор таблицы или представления.
RDB$FORMAT	SMALLINT	Идентификатор формата таблицы. Форматов может быть до 255.
RDB$DESCRIPTOR	BLOB FORMAT	Отображение в виде BLOB столбцов и характеристик данных на момент, когда была создана запись формата.

RDB$FUNCTION_ARGUMENTS

Параметры хранимых или внешних функций.

Таблица D.16. Описание столбцов таблицы RDB$FUNCTION_ARGUMENTS

Наименование столбца	Тип данных	Описание
RDB$FUNCTION_NAME	CHAR(31)	Имя функции.
RDB$ARGUMENT_POSITION	SMALLINT	Позиция аргумента в списке аргументов.
RDB$MECHANISM	SMALLINT	Механизм передачи параметра для Legacy функций: • 0 — по значению; • 1 — по ссылке; • 2 — через дескриптор; • 3 — через дескриптор BLOB.
RDB$FIELD_TYPE	SMALLINT	Код типа данных аргумента: • 7 — SMALLINT; • 8 — INTEGER; • 12 — DATE; • 13 — TIME; • 14 — CHAR; • 16 — BIGINT; • 23 — BOOLEAN; • 27 — DOUBLE PRECISION; • 35 — TIMESTAMP; • 37 — VARCHAR;

Наименование столбца	Тип данных	Описание
		• 40 — CSTRING (завершаемый нулём текст); • 45 — blob id; • 261 — BLOB.
RDB$FIELD_SCALE	SMALLINT	Масштаб для целого числа или аргумента с фиксированной точкой. Это показатель числа 10.
RDB$FIELD_LENGTH	SMALLINT	Длина аргумента в байтах: • 1 — для BOOLEAN; • 2 — для SMALLINT; • 4 — для INTEGER; • 4 — для DATE; • 4 — для TIME; • 8 — для BIGINT; • 8 — для DOUBLE PRECISION; • 8 — для TIMESTAMP; • 8 — для blob id.
RDB$FIELD_SUB_TYPE	SMALLINT	Для аргумента типа данных BLOB задаёт подтип BLOB.
RDB$CHARACTER_SET_ID	SMALLINT	Идентификатор набора символов для символьного аргумента.
RDB$FIELD_PRECISION	SMALLINT	Количество цифр точности, допустимой для типа данных аргумента.
RDB$CHARACTER_LENGTH	SMALLINT	Длина аргумента CHAR или VARCHAR в символах (но не в байтах).
RDB$PACKAGE_NAME	CHAR(31)	Имя пакета функции (если функция упакованная), в которой используется параметр.
RDB$ARGUMENT_NAME	CHAR(31)	Имя параметра.
RDB$FIELD_SOURCE	CHAR(31)	Имя домена, созданного пользователем (при использовании ссылки на домен вместо типа), или домена, автоматически построенного системой для параметра функции. Во втором случае имя будет начинаться с символов 'RDB$'.
RDB$DEFAULT_VALUE	BLOB BLR	Значение по умолчанию на языке BLR.
RDB$DEFAULT_SOURCE	BLOB TEXT	Значение по умолчанию в исходном виде на языке SQL.

Наименование столбца	Тип данных	Описание
RDB$COLLATION_ID	SMALLINT	Идентификатор используемого порядка сортировки для символьного параметра.
RDB$NULL_FLAG	SMALLINT	Признак допустимости пустого значения NULL.
RDB $ARGUMENT_MECHANISM	SMALLINT	Механизм передачи параметра для не Legacy функций: • 0 — по значению; • 1 — по ссылке; • 2 — через дескриптор; • 3 — через дескриптор BLOB.
RDB$FIELD_NAME	CHAR(31)	Имя столбца, на которое ссылается параметр с помощью предложения TYPE OF COLUMN.
RDB$RELATION_NAME	CHAR(31)	Имя таблицы, на которую ссылается параметр с помощью предложения TYPE OF COLUMN.
RDB$SYSTEM_FLAG	SMALLINT	Указывает, является ли параметр определённым системой (значение 1 и выше) или пользователем (значение 0).
RDB$DESCRIPTION	BLOB TEXT	Текст произвольного примечания к параметру.

RDB$FUNCTIONS

Описание хранимых или внешних функций.

Таблица D.17. Описание столбцов таблицы RDB$FUNCTIONS

Наименование столбца	Тип данных	Описание
RDB$FUNCTION_NAME	CHAR(31)	Имя функции.
RDB$FUNCTION_TYPE	SMALLINT	В настоящий момент не используется.
RDB$QUERY_NAME	CHAR(31)	В настоящий момент не используется.
RDB$DESCRIPTION	BLOB TEXT	Произвольный текст комментария к функции.
RDB$MODULE_NAME	VARCHAR(255)	Имя внешнего модуля (динамической библиотеки), где расположен код функции.

Наименование столбца	Тип данных	Описание
RDB$ENTRYPOINT	CHAR(255)	Имя точки входа в библиотеке, где находится эта функция.
RDB$RETURN_ARGUMENT	SMALLINT	Номер позиции возвращаемого аргумента в списке параметров, соответствующем входным аргументам.
RDB$SYSTEM_FLAG	SMALLINT	Признак определения функции: 0 — определённая системой, 1 — определённая пользователем.
RDB$ENGINE_NAME	CHAR(31)	Имя движка для использования внешних функций. Обычно UDR.
RDB$PACKAGE_NAME	CHAR(31)	Имя пакета, если функция является упакованной.
RDB$PRIVATE_FLAG	SMALLINT	Для неупакованных хранимых функций всегда NULL, для упакованных 0 — если функция описана в заголовке пакета и 1 — если функция описана или реализована только в теле пакета (не описана в заголовке).
RDB$FUNCTION_SOURCE	BLOB TEXT	Исходный код функции на языке SQL.
RDB$FUNCTION_ID	SMALLINT	Уникальный идентификатор функции.
RDB$FUNCTION_BLR	BLOB BLR	Двоичное представление (BLR) кода функции.
RDB$VALID_BLR	SMALLINT	Указывает, остаётся ли текст хранимой функции корректным после последнего изменения функции при помощи оператора ALTER FUNCTION.
RDB$DEBUG_INFO	BLOB	Содержит отладочную информацию о переменных, используемых в хранимой функции.
RDB$SECURITY_CLASS	CHAR(31)	Может указывать на класс безопасности, определённый в системной таблице RDB $SECURITY_CLASSES, для применения ограничений управления доступом.
RDB$OWNER_NAME	CHAR(31)	Имя пользователя — владельца (создателя) функции.

Наименование столбца	Тип данных	Описание
RDB$LEGACY_FLAG	SMALLINT	Признак legacy стиля функции. 1 — если функция описана в legacy стиле (DECLARE EXTERNAL FUNCTION), в противном случае 0 (CREATE FUNCTION).
RDB$DETERMINISTIC_FLAG	SMALLINT	Флаг детерминистической функции. 1 — если функция детерминистическая (DETERMINISTIC), в противном случае — 0.

RDB$GENERATORS

Сведения о генераторах (последовательностях).

Таблица D.18. Описание столбцов таблицы RDB$GENERATORS

Наименование столбца	Тип данных	Описание
RDB$GENERATOR_NAME	CHAR(31)	Уникальное имя генератора.
RDB$GENERATOR_ID	SMALLINT	Назначаемый системой уникальный идентификатор для генератора.
RDB$SYSTEM_FLAG	SMALLINT	Признак: значение 0 — генератор определён пользователем, значение 1 или выше — определён системой. 6 — внутренний генератор для identity столбца.
RDB$DESCRIPTION	BLOB TEXT	Произвольный текст примечания к генератору.
RDB$SECURITY_CLASS	CHAR(31)	Может указывать на класс безопасности, определённый в системной таблице RDB $SECURITY_CLASSES, для применения ограничений управления доступом.
RDB$OWNER_NAME	CHAR(31)	Имя пользователя — владельца (создателя) генератора.
RDB$INITIAL_VALUE	BIGINT	Хранит начальное значение генератора или значение генератора, установленное при предыдущем рестарте (WITH RESTART).
RDB $GENERATOR_INCREMENT	INTEGER	Шаг приращения генератора при использовании оператора NEXT VALUE FOR.

RDB$INDEX_SEGMENTS

Сегменты и позиции индексов. Таблица описывает все столбцы таблицы, входящие в состав конкретного индекса. Для каждого столбца индекса создаётся отдельная строка в данной таблице.

Таблица D.19. Описание столбцов таблицы RDB$INDEX_SEGMENTS

Наименование столбца	Тип данных	Описание
RDB$INDEX_NAME	CHAR(31)	Имя индекса, к которому относится данный сегмент. Должно соответствовать главной записи в системной таблице RDB$INDICES.
RDB$FIELD_NAME	CHAR(31)	Имя одного из столбцов, входящего в состав индекса. Должно соответствовать значению в столбце RDB$FIELD_NAME в таблице RDB $RELATION_FIELDS.
RDB$FIELD_POSITION	SMALLINT	Позиция столбца в индексе. Нумерация начинается с нуля.
RDB$STATISTICS	DOUBLE PRECISION	Селективность индекса по данному столбцу.

RDB$INDICES

Определение индексов базы данных (созданных пользователем или системой). Описывает каждый индекс, созданный пользователем или системой. Для каждого столбца таблицы, входящего в состав индекса, присутствует строка системной таблицы RDB $INDEX_SEGMENTS, где описываются характеристики столбца индекса.

Таблица D.20. Описание столбцов таблицы RDB$INDICES

Наименование столбца	Тип данных	Описание
RDB$INDEX_NAME	CHAR(31)	Уникальное имя индекса, заданное пользователем или автоматически сгенерированное системой.
RDB$RELATION_NAME	CHAR(31)	Имя таблицы, к которой применяется индекс. Соответствует RDB $RELATION_NAME в строке таблицы RDB$RELATIONS.
RDB$INDEX_ID	SMALLINT	Внутренний (системный) идентификатор индекса.
RDB$UNIQUE_FLAG	SMALLINT	Указывает, является ли индекс уникальным:

Наименование столбца	Тип данных	Описание
		• 0 — не уникальный; • 1 — уникальный.
RDB$DESCRIPTION	BLOB TEXT	Произвольный текст комментария к индексу.
RDB$SEGMENT_COUNT	SMALLINT	Количество сегментов (столбцов) в индексе.
RDB$INDEX_INACTIVE	SMALLINT	Указывает, является ли в настоящий момент индекс активным: • 0 — активный; • 1 — неактивный.
RDB$INDEX_TYPE	SMALLINT	В настоящий момент не используется.
RDB$FOREIGN_KEY	CHAR(31)	Имя ассоциированного ограничения внешнего ключа, если существует.
RDB$SYSTEM_FLAG	SMALLINT	Указывает, является ли индекс определённым системой (значение 1 или выше) или пользователем (значение 0).
RDB$EXPRESSION_BLR	BLOB BLR	Выражение, записанное на языке двоичного представления (BLR). Будет использовано для вычисления во время выполнения, когда будут реализованы индексы выражений.
RDB$EXPRESSION_SOURCE	BLOB TEXT	Исходный текст выражения. Будет использовано, когда будут реализованы индексы выражений.
RDB$STATISTICS	DOUBLE PRECISION	Хранит самую последнюю селективность индекса, вычисленную при помощи оператора SET STATISTICS.

RDB$LOG_FILES

В настоящей версии не используется.

Таблица D.21. Описание столбцов таблицы RDB$LOG_FILES

Наименование столбца	Тип данных	Описание
RDB$FILE_NAME	VARCHAR(255)	Не используется.
RDB$FILE_SEQUENCE	SMALLINT	Не используется.

Наименование столбца	Тип данных	Описание
RDB$FILE_LENGTH	INTEGER	Не используется.
RDB$FILE_PARTITIONS	SMALLINT	Не используется.
RDB$FILE_P_OFFSET	INTEGER	Не используется.
RDB$FILE_FLAGS	SMALLINT	Не используется.

RDB$PACKAGES

Сведения о PSQL пакетах.

Таблица D.22. Описание столбцов таблицы RDB$PACKAGES

Наименование столбца	Тип данных	Описание
RDB$PACKAGE_NAME	CHAR(31)	Уникальное имя пакета.
RDB $PACKAGE_HEADER_SOURCE	BLOB TEXT	Исходный код заголовка пакета на языке SQL.
RDB $PACKAGE_BODY_SOURCE	BLOB TEXT	Исходный код тела пакета на языке SQL.
RDB$VALID_BODY_FLAG	SMALLINT	Указывает, остаётся ли текст тела пакета корректным после последнего изменения заголовка пакета или его пересоздания.
RDB$SECURITY_CLASS	CHAR(31)	Может указывать на класс безопасности, определённый в системной таблице RDB $SECURITY_CLASSES, для применения ограничений управления доступом.
RDB$OWNER_NAME	CHAR(31)	Имя пользователя – владельца (создателя) пакета.
RDB$SYSTEM_FLAG	SMALLINT	Указывает, что пакет определён пользователем (значение 0) или системой (значение 1 или выше).
RDB$DESCRIPTION	BLOB TEXT	Произвольный текст примечания к пакету.

RDB$PAGES

Сведения о страницах базы данных.

Таблица D.23. Описание столбцов таблицы RDB$PAGES

Наименование столбца	Тип данных	Описание
RDB$PAGE_NUMBER	INTEGER	Уникальный номер физически созданной страницы базы данных.
RDB$RELATION_ID	SMALLINT	Идентификатор таблицы, для которой выделена эта страница.
RDB$PAGE_SEQUENCE	INTEGER	Последовательный номер страницы по отношению к другим страницам, выделенным для данной таблицы.
RDB$PAGE_TYPE	SMALLINT	Описывает тип страницы. Для системного использования.

RDB$PROCEDURE_PARAMETERS

Описывает параметры хранимых процедур.

Таблица D.24. Описание столбцов таблицы RDB$PROCEDURE_PARAMETERS

Наименование столбца	Тип данных	Описание
RDB$PARAMETER_NAME	CHAR(31)	Имя параметра.
RDB$PROCEDURE_NAME	CHAR(31)	Имя процедуры, в которой используется параметр.
RDB$PARAMETER_NUMBER	SMALLINT	Последовательный номер параметра.
RDB$PARAMETER_TYPE	SMALLINT	Указывает, является ли параметр входным (значение 0) или выходным (значение 1).
RDB$FIELD_SOURCE	CHAR(31)	Имя домена, созданного пользователем (при использовании ссылки на домен вместо типа), или домена, автоматически построенного системой для параметра процедуры. Во втором случае имя будет начинаться с символов 'RDB$'.
RDB$DESCRIPTION	BLOB TEXT	Текст произвольного примечания к параметру.
RDB$SYSTEM_FLAG	SMALLINT	Указывает, является ли параметр определённым системой (значение 1 и выше) или пользователем (значение 0).
RDB$DEFAULT_VALUE	BLOB BLR	Значение по умолчанию на языке BLR.

Наименование столбца	Тип данных	Описание
RDB$DEFAULT_SOURCE	BLOB TEXT	Значение по умолчанию в исходном виде на языке SQL.
RDB$COLLATION_ID	SMALLINT	Идентификатор используемого порядка сортировки для символьного параметра.
RDB$NULL_FLAG	SMALLINT	Признак допустимости пустого значения NULL.
RDB $PARAMETER_MECHANISM	SMALLINT	Механизм передачи параметра: • 0 — по значению; • 1 — по ссылке; • 2 — через дескриптор; • 3 — через дескриптор BLOB.
RDB$FIELD_NAME	CHAR(31)	Имя столбца, на которое ссылается параметр с помощью предложения TYPE OF COLUMN.
RDB$RELATION_NAME	CHAR(31)	Имя таблицы, на которую ссылается параметр с помощью предложения TYPE OF COLUMN.
RDB$PACKAGE_NAME	CHAR(31)	Имя пакета процедуры (если процедура упакованная), в которой используется параметр.

RDB$PROCEDURES

Описывает хранимые процедуры.

Таблица D.25. Описание столбцов таблицы RDB$PROCEDURES

Наименование столбца	Тип данных	Описание
RDB$PROCEDURE_NAME	CHAR(31)	Имя хранимой процедуры.
RDB$PROCEDURE_ID	SMALLINT	Уникальный идентификатор процедуры.
RDB$PROCEDURE_INPUTS	SMALLINT	Указывает количество входных параметров или их отсутствие (значение NULL).
RDB $PROCEDURE_OUTPUTS	SMALLINT	Указывает количество выходных параметров или их отсутствие (значение NULL).
RDB$DESCRIPTION	BLOB TEXT	Произвольный текст примечания к процедуре.

Наименование столбца	Тип данных	Описание
RDB$PROCEDURE_SOURCE	BLOB TEXT	Исходный код процедуры на языке SQL.
RDB$PROCEDURE_BLR	BLOB BLR	Двоичное представление (BLR) кода процедуры.
RDB$SECURITY_CLASS	CHAR(31)	Может указывать на класс безопасности, определённый в системной таблице RDB $SECURITY_CLASSES, для применения ограничений управления доступом.
RDB$OWNER_NAME	CHAR(31)	Имя пользователя - владельца (создателя) процедуры.
RDB$RUNTIME	BLOB	Описание метаданных процедуры. Внутреннее использование для оптимизации.
RDB$SYSTEM_FLAG	SMALLINT	Указывает, что процедура определена пользователем (значение 0) или системой (значение 1 или выше).
RDB$PROCEDURE_TYPE	SMALLINT	Тип процедуры: • 1 — хранимая процедура выбора (содержит в своём составе оператор SUSPEND); • 2 — выполняемая хранимая процедура.
RDB$VALID_BLR	SMALLINT	Указывает, остаётся ли текст хранимой процедуры корректным после последнего изменения процедуры при помощи оператора ALTER PROCEDURE.
RDB$DEBUG_INFO	BLOB	Содержит отладочную информацию о переменных, используемых в хранимой процедуре.
RDB$ENGINE_NAME	CHAR(31)	Имя движка для использования внешних процедур. Обычно UDR.
RDB$ENTRYPOINT	CHAR(255)	Имя точки входа в библиотеке, где находится эта процедура.
RDB$PACKAGE_NAME	CHAR(31)	Имя пакета, если процедура является упакованной.
RDB$PRIVATE_FLAG	SMALLINT	Для неупакованных хранимых процедур всегда NULL, для

Наименование столбца	Тип данных	Описание
		упакованных 0 — если процедура описана в заголовке пакета и 1 — если процедура описана или реализована только в теле пакета (не описана в заголовке).

RDB$REF_CONSTRAINTS

Описания именованных ограничений базы данных (внешних ключей).

Таблица D.26. Описание столбцов таблицы RDB$REF_CONSTRAINTS

Наименование столбца	Тип данных	Описание
RDB$CONSTRAINT_NAME	CHAR(31)	Имя ограничения внешнего ключа. Задаётся пользователем или автоматически генерируется системой.
RDB$CONST_NAME_UQ	CHAR(31)	Имя ограничения первичного или уникального ключа, на которое ссылается предложение REFERENCES в данном ограничении.
RDB$MATCH_OPTION	CHAR(7)	Не используется. Текущим значением является FULL во всех случаях.
RDB$UPDATE_RULE	CHAR(11)	Действия по ссылочной целостности, применимые к данному внешнему ключу, когда изменяется первичный (уникальный) ключ родительской таблицы: RESTRICT, NO ACTION, CASCADE, SET NULL, SET DEFAULT.
RDB$DELETE_RULE	CHAR(11)	Действия по ссылочной целостности, применимые к данному внешнему ключу, когда удаляется первичный (уникальный) ключ родительской таблицы: RESTRICT, NO ACTION, CASCADE, SET NULL, SET DEFAULT.

RDB$RELATION_CONSTRAINTS

Описание всех ограничений на уровне таблиц: первичного, уникального, внешнего ключей, ограничений CHECK, NOT NULL.

Таблица D.27. Описание столбцов таблицы RDB$RELATION_CONSTRAINTS

Наименование столбца	Тип данных	Описание
RDB$CONSTRAINT_NAME	CHAR(31)	Имя ограничения на уровне таблицы, заданное пользователем или автоматически присвоенное системой.
RDB$CONSTRAINT_TYPE	CHAR(11)	Содержит название типа ограничения: PRIMARY KEY, UNIQUE, FOREIGN KEY, CHECK, NOT NULL.
RDB$RELATION_NAME	CHAR(31)	Имя таблицы, к которой применяется это ограничение.
RDB$DEFERRABLE	CHAR(3)	В настоящий момент во всех случаях NO.
RDB$INITIALLY_DEFERRED	CHAR(3)	В настоящий момент во всех случаях NO.
RDB$INDEX_NAME	CHAR(31)	Имя индекса, который поддерживает это ограничение (содержит NULL, если ограничением является CHECK или NOT NULL).

RDB$RELATION_FIELDS

Характеристики столбцов таблиц и представлений.

Таблица D.28. Описание столбцов таблицы RDB$RELATION_FIELDS

Наименование столбца	Тип данных	Описание
RDB$FIELD_NAME	CHAR(31)	Имя столбца.
RDB$RELATION_NAME	CHAR(31)	Имя таблицы (представления), где присутствует описываемый столбец.
RDB$FIELD_SOURCE	CHAR(31)	Содержит имя домена (определённого пользователем или созданного автоматически системой), на котором основывается данный столбец.
RDB$QUERY_NAME	CHAR(31)	В настоящей версии системы не используется.
RDB$BASE_FIELD	CHAR(31)	Только для представления. Имя столбца из базовой таблицы
RDB$EDIT_STRING	VARCHAR(127)	Не используется.

Наименование столбца	Тип данных	Описание
RDB$FIELD_POSITION	SMALLINT	Позиция столбца в таблице или представлении. Нумерация начинается с 0.
RDB$QUERY_HEADER	BLOB TEXT	Не используется.
RDB$UPDATE_FLAG	SMALLINT	Указывает, является ли столбец обычным столбцом (значение 1) или вычисляемым (значение 0).
RDB$FIELD_ID	SMALLINT	В настоящей версии системы в точности соответствует значению в столбце RDB$FIELD_POSITION.
RDB$VIEW_CONTEXT	SMALLINT	Для столбца представления это внутренний идентификатор базовой таблицы, откуда приходит это поле.
RDB$DESCRIPTION	BLOB TEXT	Примечание к столбцу таблицы или представления.
RDB$DEFAULT_VALUE	BLOB BLR	Записанное в двоичном виде (BLR) значение по умолчанию — предложение DEFAULT, если оно присутствует при описании столбца таблицы (представления).
RDB$SYSTEM_FLAG	SMALLINT	Указывает, определено пользователем (значение 0) или системой (значение 1 или выше).
RDB$SECURITY_CLASS	CHAR(31)	Может ссылаться на класс безопасности, определённый в RDB$SECURITY_CLASSES для применения ограничений управления доступом для всех пользователей этого столбца.
RDB$COMPLEX_NAME	CHAR(31)	Не используется.
RDB$NULL_FLAG	SMALLINT	Указывает, допускает ли столбец значения NULL (значение NULL) или не допускает (значение 1).
RDB$DEFAULT_SOURCE	BLOB TEXT	Исходный текст предложения DEFAULT, если присутствует.
RDB$COLLATION_ID	SMALLINT	Идентификатор последовательности сортировки в составе набора символов для столбца не по умолчанию.
RDB$GENERATOR_NAME	CHAR(31)	Имя внутреннего генератора для реализации identity столбца.

Наименование столбца	Тип данных	Описание
RDB$IDENTITY_TYPE	SMALLINT	В настоящее время для identity столбца всегда хранит значение 0 (GENERATED BY DEFAULT) и NULL для других типов столбцов. В будущем этот столбец будет иметь возможность хранить значение 1 (GENERATED ALWAYS), когда этот тип идентификационных столбцов будет поддерживаться Firebird.

RDB$RELATIONS

Хранит некоторые характеристики таблиц и представлений.

Таблица D.29. Описание столбцов таблицы RDB$RELATIONS

Наименование столбца	Тип данных	Описание
RDB$VIEW_BLR	BLOB BLR	Для представления содержит на языке BLR спецификации запроса. Для таблицы в поле содержится NULL.
RDB$VIEW_SOURCE	BLOB TEXT	Для представления содержит оригинальный исходный текст запроса на языке SQL (включая пользовательские комментарии). Для таблицы в поле содержится NULL.
RDB$DESCRIPTION	BLOB TEXT	Произвольный текст примечания к таблице (представлению).
RDB$RELATION_ID	SMALLINT	Внутренний идентификатор таблицы (представления).
RDB$SYSTEM_FLAG	SMALLINT	Указывает, создана ли таблица (представление) пользователем (значение 0) или системой (значение 1 или выше).
RDB$DBKEY_LENGTH	SMALLINT	Общая длина ключа. Для таблицы это 8 байтов. Для представления это 8, умноженное на количество таблиц, на которые ссылается представление.
RDB$FORMAT	SMALLINT	Внутреннее использование.
RDB$FIELD_ID	SMALLINT	Количество столбцов в таблице (представлении).
RDB$RELATION_NAME	CHAR(31)	Имя таблицы или представления.

Наименование столбца	Тип данных	Описание
RDB$SECURITY_CLASS	CHAR(31)	Может ссылаться на класс безопасности, определённый в таблице RDB$SECURITY_CLASSES для применения ограничений управления доступом для всех пользователей этой таблицы (представления).
RDB$EXTERNAL_FILE	VARCHAR(255)	Полный путь к внешнему файлу данных, если таблица описана с предложением EXTERNAL FILE.
RDB$RUNTIME	BLOB	Описание метаданных таблицы. Внутреннее использование для оптимизации.
RDB $EXTERNAL_DESCRIPTION	BLOB	Произвольное примечание к внешнему файлу таблицы.
RDB$OWNER_NAME	CHAR(31)	Имя пользователя — владельца (создателя) таблицы или представления.
RDB$DEFAULT_CLASS	CHAR(31)	Класс безопасности по умолчанию. Применяется, когда новый столбец добавляется в таблицу.
RDB$FLAGS	SMALLINT	Внутренние флаги.
RDB$RELATION_TYPE	SMALLINT	Тип описываемого объекта: • 0 – постоянная таблица созданная пользователем или системная таблица; • 1 – представление; • 2 – внешняя таблица; • 3 – виртуальная таблица (таблицы мониторинга MON$, псевдотаблицы безопасности SEC $); • 4 – GTT уровня соединения (PRESERVE ROWS); • 5 – GTT уровня транзакции (DELETE ROWS).

RDB$ROLES

Определение ролей.

Таблица D.30. Описание столбцов таблицы RDB$ROLES

Наименование столбца	Тип данных	Описание
RDB$ROLE_NAME	CHAR(31)	Имя роли.
RDB$OWNER_NAME	CHAR(31)	Имя пользователя-владельца роли.
RDB$DESCRIPTION	BLOB TEXT	Произвольный текст примечания к роли.
RDB$SYSTEM_FLAG	SMALLINT	Системный флаг.
RDB$SECURITY_CLASS	CHAR(31)	Может ссылаться на класс безопасности, определённый в таблице RDB$SECURITY_CLASSES для применения ограничений управления доступом для всех пользователей этой роли.

RDB$SECURITY_CLASSES

Списки управления доступом.

Таблица D.31. Описание столбцов таблицы RDB$SECURITY_CLASSES

Наименование столбца	Тип данных	Описание
RDB$SECURITY_CLASS	CHAR(31)	Имя класса безопасности.
RDB$ACL	BLOB ACL	Список управления доступом, связанный с классом безопасности. Перечисляет пользователей и их полномочия.
RDB$DESCRIPTION	BLOB TEXT	Произвольный текст примечания к классу безопасности.

RDB$TRANSACTIONS

Состояние транзакций при обращении к нескольким базам данных.

Таблица D.32. Описание столбцов таблицы RDB$TRANSACTIONS

Наименование столбца	Тип данных	Описание
RDB$TRANSACTION_ID	INTEGER	Уникальный идентификатор отслеживаемой транзакции.
RDB$TRANSACTION_STATE	SMALLINT	Состояние транзакции: • 0 — зависшая; • 1 — подтверждённая; • 2 — отменённая.

Наименование столбца	Тип данных	Описание
RDB$TIMESTAMP	TIMESTAMP	Не используется.
RDB $TRANSACTION_DESCRIPTION	BLOB	Описывает подготовленную транзакцию к нескольким базам данных. Используется в случае потери соединения, которое не может быть восстановлено.

RDB$TRIGGER_MESSAGES

Сообщения триггеров.

Таблица D.33. Описание столбцов таблицы RDB$TRIGGER_MESSAGES

Наименование столбца	Тип данных	Описание
RDB$TRIGGER_NAME	CHAR(31)	Имя триггера, с которым связано данное сообщение.
RDB$MESSAGE_NUMBER	SMALLINT	Номер сообщения в пределах одного триггера (от 1 до 32767).
RDB$MESSAGE	VARCHAR(1023)	Текст сообщения триггера.

RDB$TRIGGERS

Описания триггеров.

Таблица D.34. Описание столбцов таблицы RDB$TRIGGERS

Наименование столбца	Тип данных	Описание
RDB$TRIGGER_NAME	CHAR(31)	Имя триггера.
RDB$RELATION_NAME	CHAR(31)	Имя таблицы или представления, для которого используется триггер. Если триггер применяется не к событию таблицы, а к событию базы данных, то в этом поле находится NULL.
RDB$TRIGGER_SEQUENCE	SMALLINT	Последовательность (позиция) триггера. Ноль обычно означает, что последовательность не задана.
RDB$TRIGGER_TYPE	BIGINT	Событие, на которое вызывается триггер: • 1 — BEFORE INESRT; • 2 — AFTER INSERT; • 3 — BEFORE UPDATE;

Наименование столбца	Тип данных	Описание
		• 4 — AFTER UPDATE; • 5 — BEFORE DELETE; • 6 — AFTER DELETE; • 17 — BEFORE INSERT OR UPDATE; • 18 — AFTER INSERT OR UPDATE; • 25 — BEFORE INSERT OR DELETE; • 26 — AFTER INSERT OR DELETE; • 27 — BEFORE UPDATE OR DELETE; • 28 — AFTER UPDATE OR DELETE; • 113 — BEFORE INSERT OR UPDATE OR DELETE; • 114 — AFTER INSERT OR UPDATE OR DELETE; • 8192 — ON CONNECT; • 8193 — ON DISCONNECT; • 8194 — ON TRASACTION START; • 8195 — ON TRANSACTION COMMIT; • 8196 — ON TRANSACTION ROLLBACK. Для DDL триггеров тип триггера получается путём побитового ИЛИ над фазой события (0 - BEFORE, 1 - AFTER) и всех перечисленных типов событий: • CREATE TABLE — `0x0000000000004002`; • ALTER TABLE — `0x0000000000004004`; • DROP TABLE — `0x0000000000004008`; • CREATE PROCEDURE — `0x0000000000004010`; • ALTER PROCEDURE — `0x0000000000004020`; • DROP PROCEDURE — `0x0000000000004040`; • CREATE FUNCTION — `0x0000000000004080`; • ALTER FUNCTION — `0x0000000000004100`; • DROP FUNCTION — `0x0000000000004200`; • CREATE TRIGGER — `0x0000000000004400`;

Наименование столбца	Тип данных	Описание
		• ALTER TRIGGER — 0x0000000000004800; • DROP TRIGGER — 0x0000000000005000; • CREATE EXCEPTION — 0x0000000000014000; • ALTER EXCEPTION — 0x0000000000024000; • DROP EXCEPTION — 0x0000000000044000; • CREATE VIEW — 0x0000000000084000; • ALTER VIEW — 0x0000000000104000; • DROP VIEW — 0x0000000000204000; • CREATE DOMAIN — 0x0000000000404000; • ALTER DOMAIN — 0x0000000000804000; • DROP DOMAIN — 0x0000000001004000; • CREATE ROLE — 0x0000000002004000; • ALTER ROLE — 0x0000000004004000; • DROP ROLE — 0x0000000008004000; • CREATE INDEX — 0x0000000010004000; • ALTER INDEX — 0x0000000020004000; • DROP INDEX — 0x0000000040004000; • CREATE SEQUENCE — 0x0000000080004000; • ALTER SEQUENCE — 0x0000000100004000; • DROP SEQUENCE — 0x0000000200004000; • CREATE USER — 0x0000000400004000; • ALTER USER — 0x0000000800004000; • DROP USER — 0x0000001000004000; • CREATE COLLATION — 0x0000002000004000; • DROP COLLATION — 0x0000004000004000;

Наименование столбца	Тип данных	Описание
		• ALTER CHARACTER SET — 0x0000008000004000; • CREATE PACKAGE — 0x0000010000004000; • ALTER PACKAGE — 0x0000020000004000; • DROP PACKAGE — 0x0000040000004000; • CREATE PACKAGE BODY — 0x0000080000004000; • DROP PACKAGE BODY — 0x0000100000004000; • CREATE MAPPING — 0x0000200000004000; • ALTER MAPPING — 0x0000400000004000; • DROP MAPPING — 0x0000800000004000; • ANY DDL STATEMENT — 0x7FFFFFFFFFFDFFE. Например, триггер BEFORE CREATE PROCEDURE OR CREATE FUNCTION будет иметь тип 0x0000000000004090, AFTER CREATE PROCEDURE OR CREATE FUNCTION — 0x0000000000004091, BEFORE DROP FUNCTION OR DROP EXCEPTION — 0x0000000000044200, AFTER DROP FUNCTION OR DROP EXCEPTION — 0x0000000000044201, BEFORE DROP TRIGGER OR DROP DOMAIN — 0x0000000001005000, AFTER DROP TRIGGER OR DROP DOMAIN — 0x0000000001005001.
RDB$TRIGGER_SOURCE	BLOB TEXT	Хранит исходный код триггера в PSQL.
RDB$TRIGGER_BLR	BLOB BLR	Хранит триггер в двоичном коде BLR.
RDB$DESCRIPTION	BLOB TEXT	Текст примечания триггера.
RDB$TRIGGER_INACTIVE	SMALLINT	Указывает, является ли триггер в настоящее время неактивным (1) или активным (0).
RDB$SYSTEM_FLAG	SMALLINT	Признак — триггер определён пользователем (0) или системой (1 или выше).

Наименование столбца	Тип данных	Описание
RDB$FLAGS	SMALLINT	Внутреннее использование.
RDB$VALID_BLR	SMALLINT	Указывает, остаётся ли текст триггера корректным после последнего изменения триггера при помощи оператора ALTER TRIGGER.
RDB$DEBUG_INFO	BLOB	Содержит отладочную информацию о переменных, используемых в триггере.
RDB$ENGINE_NAME	CHAR(31)	Имя движка для использования внешних триггеров. Обычно UDR.
RDB$ENTRYPOINT	CHAR(255)	Имя точки входа в библиотеке, где находится этот триггер.

RDB$TYPES

Описание перечислимых типов данных.

Таблица D.35. Описание столбцов таблицы RDB$TYPES

Наименование столбца	Тип данных	Описание
RDB$FIELD_NAME	CHAR(31)	Имя перечисляемого типа. Совпадает с именем столбца, для которого определён данный перечислимый тип.
RDB$TYPE	SMALLINT	Задаёт идентификатор для типа. Последовательность чисел является уникальной для каждого отдельного перечислимого типа: • 0 — таблица; • 1 — представление; • 2 — триггер; • 3 — вычисляемый столбец; • 4 — проверка; • 5 — процедура.
RDB$TYPE_NAME	CHAR(31)	Текстовое представление для перечислимого типа.
RDB$DESCRIPTION	BLOB TEXT	Произвольный текст примечания к перечислимому типу.
RDB$SYSTEM_FLAG	SMALLINT	Признак: определён пользователем (значение 0) или системой (значение 1 или выше).

RDB$USER_PRIVILEGES

Полномочия пользователей системы.

Таблица D.36. Описание столбцов таблицы RDB$USER_PRIVILEGES

Наименование столбца	Тип данных	Описание
RDB$USER	CHAR(31)	Пользователь, роль или объект которому предоставляется данное полномочие.
RDB$GRANTOR	CHAR(31)	Имя пользователя, предоставляющего полномочие.
RDB$PRIVILEGE	CHAR(6)	Привилегия, предоставляемая в полномочии: • A – all (все привилегии); • S – select (выборка данных); • I – insert (добавление данных); • U – update (изменение данных); • D – delete (удаление строк); • R – reference (внешний ключ); • X – execute (выполнение); • G – usage (использование); • M – membership (членство).
RDB$GRANT_OPTION	SMALLINT	Содержит ли полномочие авторизацию WITH GRANT OPTION: • 0 – не содержит; • 1 – содержит.
RDB$RELATION_NAME	CHAR(31)	Имя объекта (таблица, роль, процедура) на который предоставляется полномочие.
RDB$FIELD_NAME	CHAR(31)	Имя столбца, к которому применяется привилегия на уровне столбца (только привилегии UPDATE и REFERENCES).
RDB$USER_TYPE	SMALLINT	Идентифицирует тип пользователя (или объекта), которому предоставляется привилегия: • 1 – представление; • 2 – триггер;

Наименование столбца	Тип данных	Описание
		• 5 – процедура; • 8 – пользователь; • 13 – роль; • 15 – функция; • 18 – пакет.
RDB$OBJECT_TYPE	SMALLINT	Идентифицирует тип объекта, к которому предоставляется привилегия: • 0 – таблица; • 1 – представление; • 2 – триггер; • 5 – процедура; • 7 – исключение; • 8 – пользователь; • 9 – домен; • 11 – набор символов; • 13 – роль; • 14 – генератор (последовательность); • 15 – функция; • 16 – BLOB фильтр; • 17 – сортировка; • 18 – пакет.

RDB$VIEW_RELATIONS

Описывает представления.

Таблица D.37. Описание столбцов таблицы RDB$VIEW_RELATIONS

Наименование столбца	Тип данных	Описание
RDB$VIEW_NAME	CHAR(31)	Имя представления.
RDB$RELATION_NAME	CHAR(31)	Имя таблицы, представления или хранимой процедуры на которое ссылается данное представление.
RDB$VIEW_CONTEXT	SMALLINT	Псевдоним (контекст), используемый для ссылки на столбец представления. Имеет то же значение, что и псевдоним, используемый в самом тексте представления на языке BLR в операторе запроса этого представления.

Наименование столбца	Тип данных	Описание
RDB$CONTEXT_NAME	CHAR(255)	Текстовый вариант псевдонима, указанного в столбце RDB $VIEW_CONTEXT.
RDB$CONTEXT_TYPE	SMALLINT	Тип контекста: • 0 – таблица; • 1 – представление; • 2 – хранимая процедура.
RDB$PACKAGE_NAME	CHAR(31)	Имя пакета для упакованной хранимой процедуры.

Приложение E: Описания таблиц мониторинга

СУБД Firebird предоставляет возможность отслеживать работу с конкретной базой данных, выполняемую на стороне сервера. Для этих целей используются таблицы мониторинга (имеют префикс имени MON$). Эти таблицы являются виртуальными в том смысле, что до обращения к ним со стороны пользователя, никаких данных в них не записано. Они заполняются данными только в момент запроса пользователя (в том числе, поэтому на такие таблицы бесполезно пытаться создавать триггеры). При этом описания этих таблиц в базе данных присутствуют постоянно.

Ключевым понятием функции мониторинга является снимок активности. Снимок представляет собой текущее состояние базы данных, содержащее множество информации о самой базе данных, активных соединениях, пользователях, транзакциях, подготовленных и выполняемых запросах и т.д.

Снимок создаётся при первой выборке из любой таблице мониторинга и сохраняется до конца текущей транзакции, чтобы запросы к множеству таблиц (например, главная-подчинённая) всегда возвращал непротиворечивые данные.

Другими словами таблицы мониторинга ведут себя подобно snapshot table stability ("consistency") транзакции, даже если родительская транзакция была запущена с меньшим уровнем изоляции.

Для обновления снимка, текущая транзакция должна быть завершена и таблицы мониторинга должны быть запрошены в новом контексте транзакции.

Безопасность:

- Полный доступ ко всей информации, предоставляемой таблицами мониторинга, имеют SYSDBA и владелец базы данных;
- Обычные пользователи ограничены информацией о собственных соединениях, другие соединения невидимы для них.

> **Примечание**
>
> Частый сбор информации с помощью таблиц мониторинга в сильно нагруженной среде может негативно отразится на производительности системы.

Таблица E.1. Таблицы мониторинга

Таблица	Содержание
MON$ATTACHMENTS	Сведения о текущих соединениях с базой данных.
MON$CALL_STACK	Обращения к стеку активными запросами хранимых процедур и триггеров.

Таблица	Содержание
MON$CONTEXT_VARIABLES	Сведения о пользовательских контекстных переменных.
MON$DATABASE	Сведения о базе данных, с которой выполнено соединение.
MON$IO_STATS	Статистика по вводу-выводу.
MON$MEMORY_USAGE	Статистика использования памяти.
MON$RECORD_STATS	Статистика на уровне записей.
MON$STATEMENTS	Подготовленные к выполнению операторы.
MON$TABLE_STATS	Статистика на уровне таблиц.
MON$TRANSACTIONS	Запущенные на выполнение транзакции.

MON$ATTACHMENTS

Сведения о текущих соединениях с базой данных.

Таблица E.2. Описание столбцов таблицы MON$ATTACHMENTS

Наименование столбца	Тип данных	Описание
MON$ATTACHMENT_ID	INTEGER	Идентификатор соединения.
MON$SERVER_PID	INTEGER	Идентификатор серверного процесса.
MON$STATE	SMALLINT	Состояние соединения: • 0 — бездействующее; • 1 — активное.
MON$ATTACHMENT_NAME	VARCHAR(255)	Строка соединения — полный путь к файлу и имя первичного файла базы данных.
MON$USER	CHAR(31)	Имя пользователя, соединённого с базой данных.
MON$ROLE	CHAR(31)	Имя роли, указанное при соединении. Если роль во время соединения не была задана, поле содержит текст NONE.
MON$REMOTE_PROTOCOL	VARCHAR(10)	Имя удалённого протокола.
MON$REMOTE_ADDRESS	VARCHAR(255)	Удалённый адрес (адрес и имя сервера).
MON$REMOTE_PID	INTEGER	Идентификатор удалённого клиентского процесса.

Наименование столбца	Тип данных	Описание
MON$CHARACTER_SET_ID	SMALLINT	Идентификатор набора символов в соединении.
MON$TIMESTAMP	TIMESTAMP	Дата и время начала соединения.
MON $GARBAGE_COLLECTION	SMALLINT	Флаг сборки мусора.
MON$REMOTE_PROCESS	VARCHAR(255)	Полный путь к файлу и имя программного файла, выполнившего данное соединение.
MON$STAT_ID	INTEGER	Идентификатор статистики.
MON$CLIENT_VERSION	VARCHAR(255)	Версия клиентской библиотеки.
MON$REMOTE_VERSION	VARCHAR(255)	Версия удалённого протокола.
MON$REMOTE_HOST	VARCHAR(255)	Имя удалённого хоста.
MON$REMOTE_OS_USER	VARCHAR(255)	Имя пользователя в операционной системе.
MON$AUTH_METHOD	VARCHAR(255)	Метод проверки подлинности, используемый при подключении.
MON$SYSTEM_FLAG	SMALLINT	Флаг того, что подключение системное: • 0 — пользовательское подключение; • 1 — системное подключение.

Таблицы мониторинга доступны только для чтения. Однако в сервер встроен механизм для удаления (и только удаления) записей в таблице MON$ATTACHMENTS, что позволяет, закрыть соединение с базой данных.

Примечание

- Вся текущая активность в удаляемом соединении немедленно прекращается, и все активные транзакции откатываются;
- Закрытое соединение вернёт приложению ошибку с кодом isc_att_shutdown;
- Последующие попытки использовать это соединение (т.е. использовать его handle в API-вызовах) вернут ошибки;
- Завершение системных соединений (MON$SYSTEM_FLAG = 1) невозможно. Сервер пропустит системные подключения затронутые оператором DELETE FROM MON $ATTACHMENTS.

Примеры:

Пример E.1. Получение сведений о клиентских приложениях

```
SELECT MON$USER, MON$REMOTE_ADDRESS, MON$REMOTE_PID, MON$TIMESTAMP
FROM MON$ATTACHMENTS
```

```
WHERE MON$ATTACHMENT_ID <> CURRENT_CONNECTION
```

Пример E.2. Отключение всех соединений, за исключением своего

```
DELETE FROM MON$ATTACHMENTS
WHERE MON$ATTACHMENT_ID <> CURRENT_CONNECTION
```

MON$CALL_STACK

Обращения к стеку запросами хранимых процедур, хранимых функций и триггеров.

Таблица E.3. Описание столбцов таблицы MON$CALL_STACK

Наименование столбца	Тип данных	Описание
MON$CALL_ID	INTEGER	Идентификатор обращения.
MON$STATEMENT_ID	INTEGER	Идентификатор верхнего уровня оператора SQL — оператора, инициировавшего цепочку обращений. По этому идентификатору можно найти запись об активном операторе в таблице MON$STATEMENTS.
MON$CALLER_ID	INTEGER	Идентификатор обращающегося триггера, хранимой функции или хранимой процедуры.
MON$OBJECT_NAME	CHAR(31)	Имя объекта PSQL.
MON$OBJECT_TYPE	SMALLINT	Тип объекта PSQL: • 2 — триггер; • 5 — хранимая процедура; • 15 — хранимая функция.
MON$TIMESTAMP	TIMESTAMP	Дата и время старта обращения.
MON$SOURCE_LINE	INTEGER	Номер исходной строки оператора SQL, выполняющегося в настоящий момент.
MON$SOURCE_COLUMN	INTEGER	Номер исходного столбца оператора SQL, выполняющегося в настоящий момент.
MON$STAT_ID	INTEGER	Идентификатор статистики.
MON$PACKAGE_NAME	CHAR(31)	Имя пакета для упакованных процедур/функций.

> **Примечание**
>
> В стек вызовов не попадёт информация о вызовах при выполнении оператора EXECUTE STATEMENT.

Примеры:

Пример E.3. Получение стека вызовов для всех подключений кроме своего

```
WITH RECURSIVE
  HEAD AS (
    SELECT
      CALL.MON$STATEMENT_ID, CALL.MON$CALL_ID,
      CALL.MON$OBJECT_NAME, CALL.MON$OBJECT_TYPE
    FROM MON$CALL_STACK CALL
    WHERE CALL.MON$CALLER_ID IS NULL
    UNION ALL
    SELECT
      CALL.MON$STATEMENT_ID, CALL.MON$CALL_ID,
      CALL.MON$OBJECT_NAME, CALL.MON$OBJECT_TYPE
    FROM MON$CALL_STACK CALL
      JOIN HEAD ON CALL.MON$CALLER_ID = HEAD.MON$CALL_ID
  )
SELECT MON$ATTACHMENT_ID, MON$OBJECT_NAME, MON$OBJECT_TYPE
FROM HEAD
  JOIN MON$STATEMENTS STMT ON STMT.MON$STATEMENT_ID = HEAD.MON$STATEMENT_ID
WHERE STMT.MON$ATTACHMENT_ID <> CURRENT_CONNECTION
```

MON$CONTEXT_VARIABLES

Сведения о пользовательских контекстных переменных.

Таблица E.4. Описание столбцов таблицы MON$CONTEXT_VARIABLES

Наименование столбца	Тип данных	Описание
MON$ATTACHMENT_ID	INTEGER	Идентификатор соединения. Содержит корректное значение только для контекстных переменных уровня соединения, для переменных уровня транзакции устанавливается в NULL.
MON$TRANSACTION_ID	INTEGER	Идентификатор транзакции. Содержит корректное значение только для контекстных переменных уровня транзакции, для переменных уровня соединения устанавливается в NULL.
MON$VARIABLE_NAME	VARCHAR(80)	Имя контекстной переменной.

Наименование столбца	Тип данных	Описание
MON$VARIABLE_VALUE	VARCHAR(255)	Значение контекстной переменной.

Примеры:

Пример E.4. Получение всех сессионных контекстных переменных для текущего подключения

```
SELECT VAR.MON$VARIABLE_NAME, VAR.MON$VARIABLE_VALUE
FROM MON$CONTEXT_VARIABLES VAR
WHERE VAR.MON$ATTACHMENT_ID = CURRENT_CONNECTION
```

MON$DATABASE

Сведения о базе данных, с которой выполнено соединение.

Таблица E.5. Описание столбцов таблицы MON$DATABASE

Наименование столбца	Тип данных	Описание
MON$DATABASE_NAME	VARCHAR(255)	Полный путь и имя первичного файла базы данных или псевдоним базы данных.
MON$PAGE_SIZE	SMALLINT	Размер страницы файлов базы данных в байтах.
MON$ODS_MAJOR	SMALLINT	Старшая версия ODS.
MON$ODS_MINOR	SMALLINT	Младшая версия ODS.
MON $OLDEST_TRANSACTION	INTEGER	Номер старейшей заинтересованной транзакции — OIT, Oldest Interesting Transaction.
MON$OLDEST_ACTIVE	INTEGER	Номер старейшей активной транзакции — OAT, Oldest Active Transaction.
MON$OLDEST_SNAPSHOT	INTEGER	Номер транзакции, которая была активной на момент старта транзакции OAT, транзакция OST — Oldest Snapshot Transaction.
MON$NEXT_TRANSACTION	INTEGER	Номер следующей транзакции.
MON$PAGE_BUFFERS	INTEGER	Количество страниц, выделенных в оперативной памяти для кэша.
MON$SQL_DIALECT	SMALLINT	SQL диалект базы данных: 1 или 3.
MON$SHUTDOWN_MODE	SMALLINT	Текущее состояние останова (shutdown) базы данных:

Наименование столбца	Тип данных	Описание
		• 0 — база данных активна (online); • 1 — останов для нескольких пользователей (multi-user shutdown); • 2 — останов для одного пользователя (single-user shutdown); • 3 — полный останов (full shutdown).
MON$SWEEP_INTERVAL	INTEGER	Интервал чистки (sweep interval).
MON$READ_ONLY	SMALLINT	Признак, является база данных только для чтения, read only, (значение 1) или для чтения и записи, read-write (0).
MON$FORCED_WRITES	SMALLINT	Указывает, установлен ли для базы режим синхронного вывода (forced writes, значение 1) или режим асинхронного вывода (значение 0).
MON$RESERVE_SPACE	SMALLINT	Флаг, указывающий на резервирование пространства.
MON$CREATION_DATE	TIMESTAMP	Дата и время создания базы данных.
MON$PAGES	BIGINT	Количество страниц, выделенных для базы данных на внешнем устройстве.
MON$STAT_ID	INTEGER	Идентификатор статистики.
MON$BACKUP_STATE	SMALLINT	Текущее физическое состояние backup: • 0 — нормальное; • 1 — заблокированное; • 2 — слияние (объединение).
MON$CRYPT_PAGE	BIGINT	Количество зашифрованных страниц.
MON$OWNER	CHAR(31)	Владелец базы данных.
MON$SEC_DATABASE	CHAR(7)	Отображает, какой тип базы данных безопасности используется: • Default — база данных безопасности по умолчанию, т.е. `security3.fdb`; • Self — в качестве базы данных безопасности используется текущая база данных; • Other — в качестве базы данных безопасности используется

Наименование столбца	Тип данных	Описание
		другая база данных (не сама и не `security3.fdb`).

MON$IO_STATS

Статистика по вводу-выводу.

Таблица E.6. Описание столбцов таблицы MON$IO_STATS

Наименование столбца	Тип данных	Описание
MON$STAT_ID	INTEGER	Идентификатор статистики.
MON$STAT_GROUP	SMALLINT	Группа статистики: • 0 — база данных (database); • 1 — соединение с базой данных (connection); • 2 — транзакция (transaction); • 3 — оператор (statement); • 4 — вызов (call).
MON$PAGE_READS	BIGINT	Количество прочитанных (read) страниц базы данных.
MON$PAGE_WRITES	BIGINT	Количество записанных (write) страниц базы данных.
MON$PAGE_FETCHES	BIGINT	Количество загруженных в память (fetch) страниц базы данных.
MON$PAGE_MARKS	BIGINT	Количество отмеченных (mark) страниц базы данных.

Счётчики этой таблицы являются накопительными и накапливают информацию по каждой из групп статистики.

MON$MEMORY_USAGE

Статистика использования памяти.

Таблица E.7. Описание столбцов таблицы MON$MEMORY_USAGE

Наименование столбца	Тип данных	Описание
MON$STAT_ID	INTEGER	Идентификатор статистики.
MON$STAT_GROUP	SMALLINT	Группа статистики: • 0 — база данных (database);

Наименование столбца	Тип данных	Описание
		• 1 — соединение с базой данных (connection); • 2 — транзакция (transaction); • 3 — оператор (statement); • 4 — вызов (call).
MON$MEMORY_USED	BIGINT	Количество используемой памяти, байт. Информация о высокоуровневом распределении памяти, выполненной сервером из пулов. Может быть полезна для отслеживания утечек памяти и чрезмерного потребления памяти в соединениях, процедурах и т.д.
MON$MEMORY_ALLOCATED	BIGINT	Количество памяти, выделенной ОС, байт. Информация о низкоуровневом распределении памяти, выполненном менеджером памяти Firebird — объем памяти, выделенный операционной системой, что позволяет контролировать физическое потребление памяти. Обратите внимание, не все записи этого столбца имеют ненулевые значения. Малые выделения памяти здесь не фиксируются, а вместо этого добавляются к пулу памяти базы данных. Только MON$DATABASE (MON$STAT_GROUP = 0) и связанные с выделением памяти объекты имеют ненулевое значение.
MON$MAX_MEMORY_USED	BIGINT	Максимальное количество байт, используемое данным объектом.
MON $MAX_MEMORY_ALLOCATED	BIGINT	Максимальное количество байт, выделенное ОС данному объекту.

Примечание

Счётчики, связанные с записями уровня базы данных MON$DATABASE (MON$STAT_GROUP = 0), отображают выделение памяти для всех соединений. В архитектурах Classic и SuperClassic нулевые значения счётчиков обозначают, что в этих архитектурах нет общего кэша.

Примеры:

Пример E.5. Получение 10 запросов потребляющих наибольшее количество памяти

```
SELECT STMT.MON$ATTACHMENT_ID, STMT.MON$SQL_TEXT, MEM.MON$MEMORY_USED
```

```
FROM MON$MEMORY_USAGE MEM
   NATURAL JOIN MON$STATEMENTS STMT
ORDER BY MEM.MON$MEMORY_USED DESC
FETCH FIRST 10 ROWS ONLY
```

MON$RECORD_STATS

Статистика на уровне записей.

Таблица E.8. Описание столбцов таблицы MON$RECORD_STATS

Наименование столбца	Тип данных	Описание
MON$STAT_ID	INTEGER	Идентификатор статистики.
MON$STAT_GROUP	SMALLINT	Группа статистики: • 0 — база данных (database); • 1 — соединение с базой данных (connection); • 2 — транзакция (transaction); • 3 — оператор (statement); • 4 — вызов (call).
MON$RECORD_SEQ_READS	BIGINT	Количество последовательно считанных записей (read sequentially).
MON$RECORD_IDX_READS	BIGINT	Количество записей, прочитанных при помощи индекса (read via an index).
MON$RECORD_INSERTS	BIGINT	Количество добавленных записей (inserted records).
MON$RECORD_UPDATES	BIGINT	Количество изменённых записей (updated records).
MON$RECORD_DELETES	BIGINT	Количество удалённых записей (deleted records).
MON$RECORD_BACKOUTS	BIGINT	Количество возвращённых в базу данных записей (backed out records).
MON$RECORD_PURGES	BIGINT	Количество удалённых ненужных записей (purged records).
MON$RECORD_EXPUNGES	BIGINT	Количество вычищенных средствами сборки мусора записей (expunged records).
MON$RECORD_LOCKS	BIGINT	Количество записей прочитанных с использованием предложения WITH LOCK.

Наименование столбца	Тип данных	Описание
MON$RECORD_WAITS	BIGINT	Количество попыток обновления/ модификации/блокировки записей принадлежащих нескольким активным транзакциям. Транзакция находится в режиме WAIT.
MON$RECORD_CONFLICTS	BIGINT	Количество неудачных попыток обновления/модификации/ блокировки записей принадлежащих нескольким активным транзакциям. В таких ситуациях сообщается о конфликте обновления (UPDATE CONFLICT).
MON $BACKVERSION_READS	BIGINT	Количество прочитанных версий при поиске видимых версий записей.
MON$FRAGMENT_READS	BIGINT	Количество прочитанных фрагментов записей.
MON$RECORD_RPT_READS	BIGINT	Количество повторно прочитанных записей.

Счётчики этой таблицы являются накопительными и накапливают информацию по каждой из групп статистики.

MON$STATEMENTS

Подготовленные к выполнению операторы.

Таблица E.9. Описание столбцов таблицы MON$STATEMENTS

Наименование столбца	Тип данных	Описание
MON$STATEMENT_ID	INTEGER	Идентификатор оператора.
MON$ATTACHMENT_ID	INTEGER	Идентификатор соединения.
MON$TRANSACTION_ID	INTEGER	Идентификатор транзакции.
MON$STATE	SMALLINT	Состояние оператора: • 0 — бездействующий (idle); • 1 — выполняемый (active); • 2 — приостановленный (stalled).
MON$TIMESTAMP	TIMESTAMP	Дата и время старта оператора.
MON$SQL_TEXT	BLOB TEXT	Текст оператора на языке SQL.
MON$STAT_ID	INTEGER	Идентификатор статистики.
MON$EXPLAINED_PLAN	BLOB TEXT	План оператора в explain форме.

Состояние оператора STALLED — это состояние "приостановлено". Возможно для запроса, который начал своё выполнение, ещё не завершил его, но в данный момент не выполняется. Например, ждёт входных параметров или очередного фетча (fetch) от клиента.

Таблицы мониторинга доступны только для чтения. Однако в сервер встроен механизм для удаления (и только удаления) записей в таблице MON$STATEMENTS, что позволяет завершить активный запрос.

> **Примечание**
>
> - Попытка отмены запросов не выполняется, если в соединении в настоящее время нет никаких выполняющихся операторов.
> - После отмены запроса вызов API-функций execute/fetch вернёт ошибку с кодом isc_cancelled.
> - Последующие запросы в данном соединении не запрещёны.

Примеры:

Пример E.6. Отображение активных запросов за исключением тех, что выполняются в своём соединении

```
SELECT ATT.MON$USER, ATT.MON$REMOTE_ADDRESS, STMT.MON$SQL_TEXT, STMT.MON$TIMESTAMP
FROM MON$ATTACHMENTS ATT
  JOIN MON$STATEMENTS STMT ON ATT.MON$ATTACHMENT_ID = STMT.MON$ATTACHMENT_ID
WHERE ATT.MON$ATTACHMENT_ID <> CURRENT_CONNECTION
  AND STMT.MON$STATE = 1
```

Пример E.7. Отмена всех активных запросов для заданного соединения

```
DELETE FROM MON$STATEMENTS
WHERE MON$ATTACHMENT_ID = 32
```

MON$TABLE_STATS

Статистика на уровне таблицы.

Таблица E.10. Описание столбцов таблицы MON$TABLE_STATS

Наименование столбца	Тип данных	Описание
MON$STAT_ID	INTEGER	Идентификатор статистики.
MON$STAT_GROUP	SMALLINT	Группа статистики: • 0 — база данных (database); • 1 — соединение с базой данных (connection); • 2 — транзакция (transaction); • 3 — оператор (statement);

Наименование столбца	Тип данных	Описание
		• 4 — вызов (call).
MON$TABLE_NAME	CHAR(31)	Имя таблицы.
MON$RECORD_STAT_ID	INTEGER	Ссылка на MON$RECORD_STATS.

Примеры:

Пример E.8. Получение статистики на уровне записей по каждой таблицы для своего соединения

```
SELECT
    t.mon$table_name,
    r.mon$record_inserts,
    r.mon$record_updates,
    r.mon$record_deletes,
    r.mon$record_backouts,
    r.mon$record_purges,
    r.mon$record_expunges,
    ------------------------
    r.mon$record_seq_reads,
    r.mon$record_idx_reads,
    r.mon$record_rpt_reads,
    r.mon$backversion_reads,
    r.mon$fragment_reads,
    ------------------------
    r.mon$record_locks,
    r.mon$record_waits,
    r.mon$record_conflicts,
    ------------------------
    a.mon$stat_id
FROM
    mon$record_stats r
    JOIN mon$table_stats t ON r.mon$stat_id = t.mon$record_stat_id
    JOIN mon$attachments a ON t.mon$stat_id = a.mon$stat_id
WHERE
      a.mon$attachment_id = CURRENT_CONNECTION
```

MON$TRANSACTIONS

Описывает запущенные на выполнение транзакции

Таблица E.11. Описание столбцов таблицы MON$TRANSACTIONS

Наименование столбца	Тип данных	Описание
MON$TRANSACTION_ID	INTEGER	Идентификатор (номер) транзакции.
MON$ATTACHMENT_ID	INTEGER	Идентификатор соединения.
MON$STATE	SMALLINT	Состояние транзакции:

Наименование столбца	Тип данных	Описание
		• 0 — бездействующая; • 1 — активная.
MON$TIMESTAMP	TIMESTAMP	Дата и время старта транзакции.
MON$TOP_TRANSACTION	INTEGER	Идентификатор (номер) транзакции верхнего уровня.
MON $OLDEST_TRANSACTION	INTEGER	Номер старейшей заинтересованной транзакции — OIT, Oldest Interesting Transaction.
MON$OLDEST_ACTIVE	INTEGER	Номер старейшей активной транзакции — OAT, Oldest Active Transaction.
MON$ISOLATION_MODE	SMALLINT	Режим (уровень) изоляции: • 0 — consistency (snapshot table stability); • 1 — concurrency (snapshot); • 2 — read committed record version; • 3 — read committed no record version.
MON$LOCK_TIMEOUT	SMALLINT	Время ожидания: • -1 — бесконечное ожидание (wait); • 0 — транзакция no wait; • другое число — время ожидания в секундах (lock timeout).
MON$READ_ONLY	SMALLINT	Признак, является ли транзакцией только для чтения, read only (значение 1) или для чтения и записи, read-write (0).
MON$AUTO_COMMIT	SMALLINT	Признак, используется ли автоматическое подтверждение транзакции auto-commit (значение 1) или нет (0).
MON$AUTO_UNDO	SMALLINT	Признак, используется ли автоматическая отмена транзакции auto-undo (значение 1) или нет (0).
MON$STAT_ID	INTEGER	Идентификатор статистики.

Примеры:

Пример E.9. Получение всех подключений, которые стартовали Read Write транзакции с уровнем изоляции выше Read Commited.

```
SELECT
```

```
    DISTINCT a.*
FROM
    mon$attachments a
    JOIN mon$transactions t ON a.mon$attachment_id = t.mon$attachment_id
WHERE
  NOT(t.mon$read_only = 1 AND t.mon$isolation_mode >= 2);
```

Приложение F: Псевдотаблицы безопасности

Псевдо таблицы безопасности имеют префикс имени SEC$. Они отображают данные из текущей базы данных безопасности. Эти таблицы являются виртуальными в том смысле, что до обращения к ним со стороны пользователя, никаких данных в них не записано. Они заполняются данными только в момент запроса пользователя. При этом описания этих таблиц в базе данных присутствуют постоянно. В этом смысле эти псевдо-таблицы подобны таблицам семейства MON$, используемых для мониторинга сервера.

Безопасность:

- Полный доступ ко всей информации, предоставляемой таблицами безопасности, имеют SYSDBA и владелец базы данных;
- Обычные пользователи ограничены информацией о самих себе, другие пользователи невидимы для них.

> **Важно**
>
> Эти функции во многом зависят от плагина управления пользователями. Имейте в виду, что некоторые опции игнорируются при использовании устаревшего плагина управления пользователями.

Таблица F.1. Псевдотаблицы безопасности

Таблица	Содержание
SEC$GLOBAL_AUTH_MAPPING	Сведения о глобальных отображениях.
SEC$USERS	Список пользователей в текущей базе данных безопасности.
SEC$USER_ATTRIBUTES	Сведения о дополнительных атрибутах пользователей.

SEC$GLOBAL_AUTH_MAPPING

Сведения о глобальных отображениях.

Таблица F.2. Описание столбцов таблицы SEC$GLOBAL_AUTH_MAPPING

Наименование столбца	Тип данных	Описание
SEC$MAP_NAME	CHAR(31)	Имя глобального отображения.

Наименование столбца	Тип данных	Описание
SEC$MAP_USING	CHAR(1)	Является ли аутентификация общесерверной (S) или обычной (P).
SEC$MAP_PLUGIN	CHAR(31)	Имя плагина аутентификации, из которого происходит отображение.
SEC$MAP_DB	CHAR(31)	Имя базы данных, в которой прошла аутентификация. Из неё происходит отображение.
SEC$MAP_FROM_TYPE	CHAR(31)	Тип объекта, который будет отображён.
SEC$MAP_FROM	CHAR(255)	Имя объекта, из которого будет произведено отображение.
SEC$MAP_TO_TYPE	SMALLINT	Тип объекта, в который будет произведено отображение: • 0 — USER; • 1 — ROLE.
SEC$MAP_TO	CHAR(31)	Наименование объекта, в который будет произведено отображение (имя пользователя или роли).
RDB$SYSTEM_FLAG	SMALLINT	Является ли отображение системным: • 0 — определено пользователем; • 1 — определено системой.
RDB$DESCRIPTION	BLOB TEXT	Произвольное текстовое описание порядка сортировки.

SEC$USERS

Список пользователей в текущей базе данных безопасности.

Таблица F.3. Описание столбцов таблицы SEC$USERS

Наименование столбца	Тип данных	Описание
SEC$USER_NAME	CHAR(31)	Имя пользователя.
SEC$FIRST_NAME	VARCHAR(32)	Первое имя (имя).
SEC$MIDDLE_NAME	VARCHAR(32)	Среднее имя (отчество).
SEC$LAST_NAME	VARCHAR(32)	Последнее имя (фамилия).
SEC$ACTIVE	BOOLEAN	Флаг активности пользователя.

Наименование столбца	Тип данных	Описание
SEC$ADMIN	BOOLEAN	Отражает, имеет ли пользователь права RDB$ADMIN в базе данных безопасности.
SEC$DESCRIPTION	BLOB TEXT	Комментарий к пользователю.
SEC$PLUGIN	CHAR(31)	Имя плагина управления пользователями, с помощью которого был создан данный пользователь.

SEC$USER_ATTRIBUTES

Сведения о дополнительных атрибутах пользователей.

Таблица F.4. Описание столбцов таблицы SEC$USER_ATTRIBUTES

Наименование столбца	Тип данных	Описание
SEC$USER_NAME	CHAR(31)	Имя пользователя.
SEC$KEY	VARCHAR(31)	Имя атрибута.
SEC$VALUE	VARCHAR(255)	Значение атрибута.

Приложение G: Наборы символов и порядки сортировки

Таблица G.1. Наборы символов и порядки сортировки

Название	ID	Байтов на символ	Порядок сортировки	Язык
ASCII	2	1	ASCII	Английский
BIG_5	56	2	BIG_5	Китайский, Вьетнамский, Корейский.
CP943C	68	2	CP943C	Японский.
			CP943C_UNICODE	Японский.
CYRL	50	1	CYRL	Русский.
			DB_RUS	Русский dBase.
			PDOX_CYRL	Русский Paradox.
DOS437	10	1	DOS437	Английский — США.
			DB_DEU437	Немецкий dBase.
			DB_ESP437	Испанский dBase.
			DB_FIN437	Финский dBase.
			DB_FRA437	Французский dBase.
			DB_ITA437	Итальянский dBase.
			DB_NLD437	Голландский dBase.
			DB_SVE437	Шведский dBase.
			DB_UK437	Английский (Великобритания) dBase.
			DB_US437	Английский (США) dBase.
			PDOX_ASCII	Кодовая страница Paradox — ASCII.
			PDOX_SWEDFIN	Шведский / Финский Paradox.
			PDOX_INTL	Международный английский Paradox.
DOS737	9	1	DOS737	Греческий.
DOS775	15	1	DOS775	Балтийский.

Название	ID	Байтов на символ	Порядок сортировки	Язык
DOS850	11	1	DOS850	Латинский I (нет символа Евро).
			DB_DEU850	Немецкий.
			DB_ESP850	Испанский.
			DB_FRA850	Французский.
			DB_FRC850	Французский — Канада.
			DB_ITA850	Итальянский.
			DB_NLD850	Голландский.
			DB_PTB850	Португальский — Бразилия.
			DB_SVE850	Шведский.
			DB_UK850	Английский — Великобритания.
			DB_US850	Английский — США.
DOS852	45	1	DOS852	Латинский II.
			DB_CSY	Чешский dBase.
			DB_PLK	Польский dBase.
			DB_SLO	Словацкий dBase.
			PDOX_CSY	Чешский Paradox.
			PDOX_HUN	Венгерский Paradox.
			PDOX_PLK	Польский Paradox.
			PDOX_SLO	Словацкий Paradox.
DOS857	46	1	DOS857	Турецкий.
			DB_TRK	Турецкий dBase.
DOS858	16	1	DOS858	Латинский I с символом Евро.
DOS860	13	1	DOS860	Португальский.
			DB_PTG860	Португальский dBase.
DOS861	47	1	DOS861	Исландский.
			PDOX_ISL	Исландский Paradox.
DOS862	17	1	DOS862	Иврит.
DOS863	14	1	DOS863	Французский — Канада.
			DB_FRC863	Французский dBase — Канада.

Название	ID	Байтов на символ	Порядок сортировки	Язык
DOS864	18	1	DOS864	Арабский.
DOS865	12	1	DOS865	Скандинавские.
			DB_DAN865	Датский dBase.
			DB_NOR865	Норвежский dBase.
			PDOX_NORDAN4	Paradox Норвегия и Дания.
DOS866	48	1	DOS866	Русский.
DOS869	49	1	DOS869	Современный греческий.
EUCJ_0208	6	2	EUCJ_0208	Японские EUC.
GB_2312	57	2	GB_2312	Упрощенный китайский (Гонконг, Корея).
GB18030	69	4	GB18030	Китайский.
			GB18030_UNICODE	Китайский.
GBK	67	2	GBK	Китайский.
			GBK_UNICODE	Китайский.
ISO8859_1	21	1	ISO8859_1	Латинский I.
			DA_DA	Датский.
			DE_DE	Немецкий.
			DU_NL	Голландский.
			EN_UK	Английский, Великобритания.
			EN_US	Английский, США.
			ES_ES	Испанский.
			ES_ES_CI_AI	Испанский не чувствительный к регистру символов и к акцентам (ударению).
			FI_FI	Финский.
			FR_CA	Французский, Канада.
			FR_FR	Французский.
			FR_FR_CI_AI	Французский не чувствительный к регистру символов и к акцентам (ударению).
			IS_IS	Исландский.

Название	ID	Байтов на символ	Порядок сортировки	Язык
			IT_IT	Итальянский.
			NO_NO	Норвежский.
			PT_PT	Португальский.
			PT_BR	Португальский, Бразилия.
			SV_SV	Шведский.
ISO8859_2	22	1	ISO8859_2	Латинский 2 — Центральная Европа (хорватский, чешский, венгерский, польский, румынский, сербский, словацкий, словенский).
			CS_CZ	Чешский.
			ISO_HUN	Венгерский.
			ISO_PLK	Польский.
ISO8859_3	23	1	ISO8859_3	Латинский 3 — Южная Европа (мальтийский, эсперанто).
ISO8859_4	34	1	ISO8859_4	Латинский 4 — Северная Европа (эстонский, латвийский, литовский, гренландский, саамский).
ISO8859_5	35	1	ISO8859_5	Кириллица (русский).
ISO8859_6	36	1	ISO8859_6	Арабский.
ISO8859_7	37	1	ISO8859_7	Греческий.
ISO8859_8	38	1	ISO8859_8	Иврит.
ISO8859_9	39	1	ISO8859_9	Латинский 5.
ISO8859_13	40	1	ISO8859_13	Латинский 7 — Балтика.
			LT_LT	Литовский.
KOI8R	63	1	KOI8R	Русский. Словарное упорядочение.
			KOI8R_RU	Русский.
KOI8U	64	1	KOI8U	Украинский. Словарное упорядочение.
			KOI8U_UA	Украинский.
KSC_5601	44	2	KSC_5601	Корейский.
			KSC_DICTIONARY	Корейский — словарный порядок сортировки.
NEXT	19	1	NEXT	Кодирование NeXTSTEP.

Название	ID	Байтов на символ	Порядок сортировки	Язык
			NXT_DEU	Немецкий.
			NXT_ESP	Испанский.
			NXT_FRA	Французский.
			NXT_ITA	Итальянский.
			NXT_US	Английский, США.
NONE	0	1	NONE	Нейтральная кодовая страница. Перевод в верхний регистр выполняется только для кодов ASCII 97–122. Постарайтесь сделать так, чтобы этот набор символов никогда не появлялся в столбцах ваших баз данных.
OCTETS	1	1	OCTETS	Двоичные символы.
SJIS_0208	5	2	SJIS_0208	Японский.
TIS620	66	1	TIS620	Тайский.
			TIS620_UNICODE	Тайский.
UNICODE_FSS	3	3	UNICODE_FSS	UNICODE
UTF8	4	4	UTF8	UNICODE 4.0.
			USC_BASIC	UNICODE 4.0.
			UNICODE	UNICODE 4.0.
			UNICODE_CI	UNICODE 4.0. Не чувствительна к регистру символов.
			UNICODE_CI_AI	UNICODE 4.0. Не чувствительна к регистру символов и к акцентам (ударению).
WIN1250	51	1	WIN1250	ANSI — Центральная Европа.
			BS_BA	Боснийский.
			PXW_CSY	Венгерский.
			PXW_HUN	Венгерский — словарная сортировка.
			PXW_HUNDC	Польский.
			PXW_PLK	Словацкий.
			PXW_SLOV	Словенский.
			WIN_CZ	Чешский.

Название	ID	Байтов на символ	Порядок сортировки	Язык
			WIN_CZ_CI	Чешский без различия строчных и прописных букв.
			WIN_CZ_CI_AI	Чешский без различия строчных и прописных букв, нечувствительный к знакам ударения.
WIN1251	52	1	WIN1251	ANSI кириллица.
			WIN1251_UA	Украинский.
			PXW_CYRL	Paradox кириллица (русский).
WIN1252	53	1	WIN1252	ANSI — Латинский I.
			PXW_INTL	Английский интернациональный.
			PXW_INTL850	Paradox многоязыковый Латинский I.
			PXW_NORDAN4	Норвежский и датский.
			PXW_SPAN	Paradox испанский.
			PXW_SWEDFIN	Шведский и финский.
			WIN_PTBR	Португальский, Бразилия.
WIN1253	54	1	WIN1253	ANSI греческий.
			PXW_GREEK	Paradox греческий.
WIN1254	55	1	WIN1254	ANSI турецкий.
			PXW_TURK	Paradox турецкий.
WIN1255	58	1	WIN1255	ANSI иврит.
WIN1256	59	1	WIN1256	ANSI арабский.
WIN1257	60	1	WIN1257	ANSI балтийский.
			WIN1257_EE	Эстонский. Словарное упорядочение.
			WIN1257_LT	Литовский. Словарное упорядочение.
			WIN1257_LV	Латвийский. Словарное упорядочение.
WIN1258	65	1	WIN1258	Вьетнамский.

Алфавитный указатель

A

B

C

S

T

U

V

W

Y