THE INTERBASE & FIREBIRD DEVELOPER MAGAZINE

Full Text Search in DBMS

Object-oriented development and RDBMS, Part 2

JayBird 2.0, a JCA/JDBC driver for Firebird

How I started to work with MS SQL Server and ADO

TPC based tests for InterBase & Firebird

#3
2005
www.ibdeveloper.com

Keep on watching

www.ibdeveloper.com

We're preparing
special
Christmas
surprises and bonuses
for all InterBase
and Firebird fans!

THE INTERBASE & FIREBIRD DEVELOPER MAGAZINE

Credits

Alexey Kovyazin,
Chief Editor

Dmitri Kouzmenko,
Editor

Helen Borrie,
Editor

Noel Cosgrave,
Sub-editor

Lev Tashchilin,
Designer

Natalya Polyanskaya,
Blog editor

Editorial Office

IBase IBDeveloper, office 5,
1-st Novokuznetsky lane, 10

zip: 115184

Moscow, Russia
Phone: +7095 6869763
Fax: +7095 9531334

Email: ibdeveloper@ibdeveloper.com

www.ibdeveloper.com

Contents

Back to the future

You know, the flow of life is like a spiral – old ideas become new with each convolution. Basic, apparently modern ideas were allive in Plato's ideal world of ideas. Today we have a good chance to prove spiral theory in the software development area – many ideas have been resurrected this year. I'd like to devote the first part of this editor's note to several new ideas with old histories.

Yukon and others

It was about 25 years ago when multi-generational (sometimes called multi-version for marketing reasons, I suppose) architecture (MGA) was implemented in the first versions of InterBase, and now it is "reinvented" by Microsoft. The newest version of MSSQL (Yukon) has multi-version abilities for records reading.

Well, one of the database market leaders decided to use the approach that was used for years in InterBase and Firebird. Fans of locking servers frequently proclaimed MGA wrong and useless, yet it is clear that multi-version capabilities are valuable and useful for database development, especially for combined OLTP+OLAP systems. Of course, Yukon is only the first step. Microsoft folks need to study many things before they turn MSSQL into a full-scale multi-version engine (well, in every joke lies a piece of truth :).

Multi-version architecture is rising in popularity each year: it was implemented in the MySQL InnoDB storage engine, in Linter, and it seems like many servers are on the same course, too. I suppose that we will be seeing more and more multi-version implementations before long.

Vulcan

You know that Vulcan is a revolutionary new architectural prototype for Firebird development. Although based on the principles of the original InterBase architecture, it is implemented using modern techniques.

All community members are anticipating the SMP support, scalability and easy extensibility that the Vulcan implementation will bring to subsequent releases of Firebird. We can also look forward to the release of Vulcan itself, which is already SMP-enabled, in quarter 1 of 2006.

We intend to test Beta version of Vulcan once it will be available among other servers with tests based on TPC-R and TPC-C. TPC-C is especially interested because it implies intensive use of SMP capabilities. Of course, at this point there is still a lot of work needed to make Vulcan stable and convenient enough for all users, but I believe it is only a question of time.

Issue 3 is out

Well, let's get back to the present. Issue 3 of "The InterBase and Firebird Developer Magazine" is out. "3" is a magical number and, with a third issue, we can say that we are on the right road. Growing interest and increasing readership instill in us some confidence that our magazine is a good and relevant idea.

What is in this issue?

I got an email from a reader who noted that there was no need to describe articles – if readers are too lazy to read articles, they certainly won't read the editor's note :)

The only thing I'd like to say, therefore, is that we've published a printed version of the third issue simultaneously so, if you prefer to read it beside the Christmas tree, you can. Back issues are available too - for details visit http://ibdeveloper.com/paper-version/.

In this issue we've added some comics and interviews to make it more fun and friendly. Hope you'll enjoy it!

Community

Actually, I am less than happy with the community feedback. It seems people just have no opinions about the articles. Please do not hesitate to post your comments in the blog at www.ibdeveloper.com or send them to readers@ibdeveloper.com. Your thoughts and suggestions are very important for our authors and editors!

Firebird Conference and Borland Conference

In November we've had two major events of the year – the Firebird Conference and the Borland Conference (with a track devoted to InterBase). We plan to publish special issues devoted to these events – watch the news!

Happy New Year!

This is the last issue of our magazine in 2005, so it is a good time to wish you Merry Christmas and Happy New Year! See you in 2006!

Sincerely yours,

Alexey Kovyazin,

Chief editor

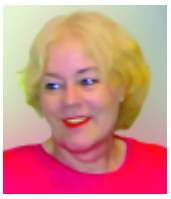
Buy paper versions

#1

#3

Connecting Remotely

*by **Helen Borrie**, helebor@tpg.com.au*



The rave of the month has been, of course, the Firebird Conference. For the first time, the conference moved out of Germany and was staged in the Czech Republic, in beautiful, historic Prague. The blogs and photo albums tell it all: the fun bits as well as the serious bits. There was plenty of beer—a traditional element, given that the two preceding conferences took place in the homeland of German hops!—not to mention cake and cuckoo clocks and confrontations with armed soldiers and hotel maids.

People came from far and wide—Brazil, South Africa, Russia, Central Europe, Japan, North America, even New Zealand, which is about as far away from Prague as you can go and still have running tapwater. It was great to have Russians there for the first time: it's been so hard in the past to get them to Germany.

The reasons for choosing Prague were several. Our past venue in Fulda had been great but it was also expensive, especially for visitors from outside the Euro currency zone. There had been persistent difficulties for some would-be participants to get visas. Prague appeared a good candidate to address both of those problems, with the added benefit of Very Cheap Beer, or so we were told by the locals. Was it true? Apparently—if the volume of empty beer bottles visible in nearly every photo is any kind of indicator.

Well, the upshot of it all was that a good time was had by all. The wish to return to Prague for next year's conference was unanimous—as long as it were not convened at Hotel Olsanka!

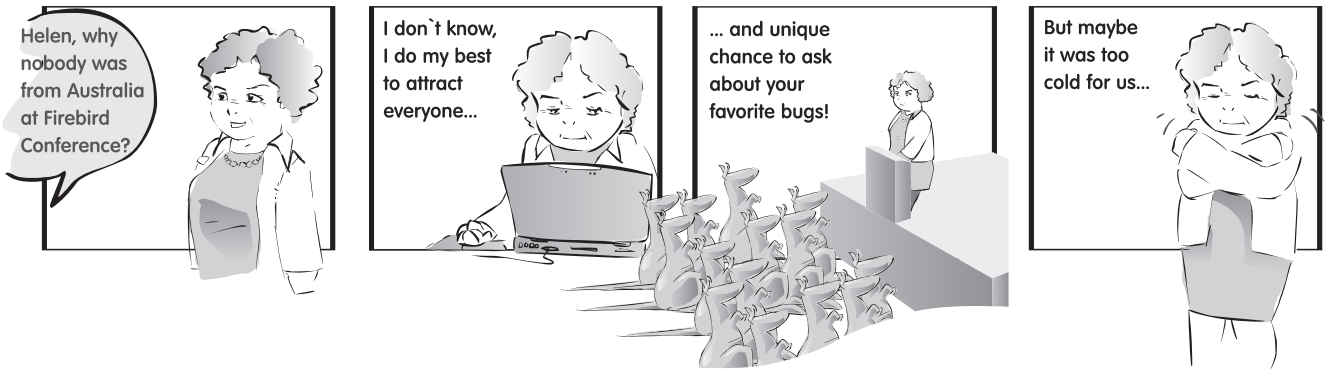
Now, who am I to be waxing lyrical about the Prague conference? I wasn't there. Too sad! There were many moments during those four days that I wished it were otherwise. What made it "happen" for me were the blogs. Martijn Tonies, from the Netherlands, started blogging the conference before it even began. Thanks to Holger Klemt and his team of techos from Germany, the conference attendees had Internet by wireless in one and only one place—the bar. So, while I was visualising lonely Martijn slaving over a hot keyboard in his hotel room at every spare moment, in fact he was slaving over a cool beer and a hot keyboard in the bar at every spare moment! It was a good formula.

Stefan Heymann, from Germany, also blogged, although not with the intensity and frequency exhibited by Martijn. Make a note, all you bloggers, for next year: there is no such thing as Too Many Blogs.

Another conference tradition is for Lucas Franzen to kidnap my coffee plunger and force me to pay a ridiculous price at his mad auction to get it back in order to survive the rest of my journey with proper coffee to drink...While in New Zealand at mid-year, I bought a replica of the Famous Coffee Plunger, intending to put it up for the auction, with the hopeful intention of being permitted to keep mine. I totally forgot to mail it over for the auction. Mea culpa! But it will keep for next year.

Amazingly, Luc's famous auction—which nobody can deny is a total rip-off—netted about 3000 Euros for the Firebird Foundation's coffers. That translates neatly into a grant allocation for a part-time QA person for Firebirds 2 and 3. Deep pockets are a wonderful thing for open source software development.

Oh and—yes, it's true, Prague in November is too cold for this sub-tropical dweller. I don't own an overcoat and I live in a part of Australia where a winter coat would be a collector's item if it were available at all and would be priced accordingly. Add the shortcomings of my cold-climate wardrobe to the overwhelming cost of airfares from here to anywhere—the speaker's fee would have got me about as far as Tokyo, one way—and one has fairly compelling reasons to stay put.



Full-text search in RDBMS

by Roman Rokytskyy, rrokytskyy@acm.org

What is full-text search

The full-text search came into our lives together with the Web and Internet. Catalogs like Yahoo! are quite often inconvenient when looking for a particular piece of information that does not belong to a single category and are just useless when information was simply not added to the catalog. Google revolutionized the search industry by improving the quality of the search, usually showing the answer to the asked question in the first items returned in the result set.

So, what do people expect from a full-text search? Three main characteristics come to mind:

- Query language with text-specific predicates, not only Boolean ones. People expect to have the possibility of performing a search for documents containing some specific phrase, or documents where some words are close together.
- Fuzzy matching. People expect that not only words specified in the query will be matched, but also variations of them will be considered, e.g. word stemming, plurals, thesaurus search and it would be nice if search engine could anticipate common spelling errors.
- Result scoring. People expect that results of the query are returned in the correct order, i.e. the most relevant documents are returned first.

This article aims to explain the common issues that arise with implementing full-text search support in databases and reviews the current situation with full-text search in various RDBMS systems and Firebird in particular.

Search and indexing

When we talk about search engines, "full-text search" is used as a synonym for a corresponding index. But in fact we do not need an index for this. One could create a function that would go through each document and check whether it satisfies the specified query or not.

The index speeds up the search. The faster we get results, the happier we are. Two main approaches to the text indexing exist: an inverted file index and a signature file index. Below are short descriptions of both approaches. The reader can find more detailed information in [1].

Inverted file index

The inverted file index has two parts – a vocabulary, containing all distinct values being indexed (i.e. words, when we talk about the text documents) and an inverted list, a mapping between the vocabulary entries and the documents containing those entries. A query is evaluated by obtaining the inverted lists for each term from the query and then either merging them for disjunctive queries or intersecting them for conjunctive queries. The final set contains "pointers" to the documents that satisfy the specified query. Additional structures allow the search algorithm to rank the documents in the result set.

Signature file index

The signature file index groups various indexing approaches that have in common the fixed-length signature of each document. The simplest approach is that, for each document, a fixed-width bitstring of length w is assigned. Each word that appears in the document is hashed to determine the bit in the bit string that should be set to 1. It can and will happen that two different words can set the same bit: in this case, no additional action is taken. The query terms are also hashed using the same algorithm and then those documents whose signatures have the same bits set as in the query become candidates for retrieval. Later, each document is processed to discover any false matchesthat were caused by the fixed length of the bit string. Various modifications are possible in order to improve querying performance, a good overview of which can be found in [1] as well.

Anatomy of the text search engine

The inverted file approach is believed to produce the more compact and faster index compared to the signature file index and this approach is the basis for the Lucene search engine. We use Lucene as a model of a text search engine. Below you will find a description of its main components; later, we will use the classification of components and algorithms for RDBMS search solutions.

Lucene

The Lucene search engine indexes documents. Each document is an entity that consists of one or more text fields. When the document is added to the index, its fields are usually split into indexable terms which are generated by a "tokenizer" component. Union of all terms in the index forms an inverted file dictionary. In order to simplify the explanation we will suppose that our document contains only one field.

Constructing an index

The process of adding a new document to an index consists of a number of steps. First, the indexing component iterates through all fields in the document passed into it and calls the tokenizing component, which in turn splits the text of each field into set of terms and performs additional steps, like converting words into singular form or extracting word stems. Later, the tokenized terms are stored in the index together with the pointer to the field to which it belongs, and the position of the term in a field.

Let's consider following short documents:

"Fascinating creatures, phoenixes, they can carry immensely heavy loads, their tears have healing powers and they make highly faithful pets." (J.K.Rowling, Harry Potter)

"Fascinating Large Pack Animal: this creature can carry large and heavy pieces of equipment; can be equipped with large carry packs" (Slightly modified phrase from a web site dedicated to one computer game).

"The Healing Power of Pets reveals the completeness of animals and their ability to heal our lives... A captivating and heartwarming look at the fascinating role of pets and relationships in health and healing." (Amazon.com editorial review page on a book "Healing Powers of Pets" by Dr. Marry Becker)

After passing these sentences through the tokenizer, the set of tokens is added to the dictionary. The inverted index can be represented by a table similar to the one in Illustration 1. The left column contains terms, the right column a list of the documents in which each term was found.

Term	Documents
ability	3
animal	2,3
can	1,2
...	...

***llustration 1**: Inverted index table*

Already this table can be used to return a set of documents that satisfy the specified full-text query. The application can combine predicates with AND and OR operators, which is internally converted into operations over the sets of document IDs. But the answer to a query is barely usable by a human. For a reasonably large document index, thousands of matches will be returned.

The main task of the indexing component, therefore, is to construct structures that not only answer the question of which documents contain terms from the query, but that also help to rank the documents in the result.

A structure called a "term frequency histogram", an ordered list of <term, frequency> pairs for each indexed document, plays a major role in document ranking. It can be imagined as a spreadsheet whose first column contains terms from the index dictionary, whose first row contains document IDs, and the rest of whose cells contain the frequency of the term's appearance in the document. This frequency is computed from the number of times the specific term appears in the document.

An example of the term frequency histogram can be found in Illustration 2, in the "term frequency" column that contains term frequency values for each indexed document.

Another structure, called "document frequency", is built, containing the number of documents that contain the specific term. Document frequency allows an estimate of the "selectivity" of the term to be made.

It is clear that the term frequency histogram also works as an inverted index, so there is no need to keep a separate table for this purpose.

It is also worth mentioning that the indexing component also builds a "term proximity table", which stores the position of the term in the document. This structure is used by phrase queries, but it is too big to be shown on the illustration.

Searching

Searching is done in two steps. In the first step, the query is parsed and converted into a tree of atomic queries. It is unlike databases, with each atomic query returning a score that rates how well that particular query matches a document.

The following atomic queries are supported:

- "term query", the most widely used query. Each word in the term query is processed with the tokenizer, which converts the word into its "normal" form; the query matches documents containing the specified term;
- "fuzzy query" matches documents containing terms for which the similarity is greater than some previously specified value. The similarity measure in Lucene is based on the Levenshtein algorithm, but other similarity measures like SOUNDEX can be used;
- "wildcard query" matches documents with terms that in turn match the specified wildcard pattern; this query can be very slow, especially when the "*" wildcard is used at the beginning of the term;
- "prefix query" is a special case of the wildcard query that has room for additional optimization; consequently, it has been extracted into separate class;
- "phrase query" matches documents that contain a particular sequence of terms. Another variant of this query, called "sloppy phrase query", also matches sequences that might contain other terms between the specified ones;
- "boolean query" combines other atomic queries with Boolean operators.

The simplest implementation can combine the scores obtained from the atomic queries into a base score and use that as the basis for sorting the result set. This would provide much better results than simply matching on the inverted file index, but would still leave much to be desired with regard to relevance. The reason is that words are simply not equally relevant. To the best knowledge of the author there is no "silver bullet" that solves the question of which word is more relevant to the context in comparison to the others.

Statistics play a great role here. The following scoring factors have been determined to improve the match relevancy:

- "term frequency" – score factor based on the number of times the specified term appears in the document. The more often term appears in the document, the better that document corresponds to the topic. The corresponding data structure, created during the addition of a document to the index, was described earlier.
- "document frequency" – score factor based on the number of documents that contain the term. It is used to determine term relevance, since terms that occur in fewer documents are usually better indicators for the topic. Because a smaller value means better selectivity, an inverted value, also computed when adding a document to the index, is usally used.
- "length norm" – normalization factor for the "term frequency" score, based on the total number of tokens in a document. A match in a large document is less precise than the match in a short document, so the higher "term frequency" score is needed for a document to become relevant. It is computed during the addition of the document to the index.
- "coord score" – score factor based on the fact that the more terms in the

query matching the terms in the document, the better the match is; it is computed dynamically for each query.

- "sloppy match" – score factor applicable only to phrase searches; the closer the proximity of the terms from the query to one another in the document, the better the match. It is computed dynamically for each query, but requires a term proximity table that is created during the addition of the document to the index

The search component applies these scoring factors to the outcome of each query and later recombines the results into an overall document score. This document score is the final score that is used to sort documents in the result set.

Let's return to our example, the inverted index for which is presented on Illustration 2, and consider two queries. The first one should search for all documents containing words "healing" and "abilities", the second one searches for the phrase "fascinating creatures". Let's compute the rank of each document for both queries.

The first query consists of two atomic term queries combined with a Boolean query. First, let's process words with the imaginary tokenizer that was also used to construct the term frequency histogram. Our words after processing are "heal" and "ability". The rank of document k in our case can be computed as the sum of the term ranks normalized by coord score, where the rank of the term i in document k is proportional to the term frequency and inversely proportional to the number of terms in the document and the document frequency of the term.

The coord score is computed and the number of term matches in the document divided by the maximum number of matches in all documents.

The formula used in Lucene gives ranks r1=0.06, r2=0 and r3=0.47, which completely corresponds to our expectations – the third document matches our query the best, since it talks about the abilities of pets to heal our life; the first one talks about healing powers, which semantically is quite similar to the healing abilities, but because there's no 100% word match, it gets lower rank; and finally the second document does not have any term match, so it will not be displayed in

Term	Term frequency			Doc freq
	Doc 1	Doc 2	Doc 3	
ability			1	1
animal		1	1	2
can	1	2		2
captivating			1	1
carry		1	1	2
completeness			1	1
creature	1	2		2
equipment		1		1
equipped		1		1
faithful	1			1
fascinating	1	1	1	3
have	1			1
heal	1		3	2
health			1	1
heartwarming			1	1
heavy	1	1		2
highly	1			1
immensely	1			1
large		3		1
life			1	1
load	1			1
look			1	1
make	1			1
pack		2		1
pet			2	2
phoenix	1			1
piece		1		1
power	1		1	2
relationship			1	1
reveal			1	1
role			1	1
tear	1			1
Length norm	**16**	**16**	**18**	

Illustration 2.: *Term frequency histogram for the sentences mentioned above.*

the result at all. The formula is constructed to give r=1 for an exact match.

The second query is a phrase query, which matches only those documents that contain all terms specified in a query string, and therefore the third document will be filtered before the ranking begins. We will use term score and sloppy phrase score to construct the document ranking. First, we compute the term rank, as in the previous case, and then combine it with the sloppy phrase score. That score is considered to be 1 when the search terms are found within the specified maximum distance between words and 0 if one of the terms is not found, or the distance is greater than the maximum allowed. In our case, we use maximum distance 4, to include also the second document.

This approach gives us ranks r1=0.37 and r2=0.21. For the sake of completeness we define r3=0. These results are in full agreement with our expectations.

It is also worth mentioning that, normally, the sloppy phrase scoring factor is limited to a relatively short distance between terms, usually 2. A scoring factor of two here would effectively filter out the second document, since the phras "fascinating creatures" does not occur in it.

Finally, a few words about the Google ranking algorithm, which caused a revolution in the Internet search engines. During some university research, Larry Page and Sergey Brin suggested that information on an Internet website is more relevant if many other sites contain hypertext links on it. This fact introduced an additional scoring factor "page rank" that is proportional to the number of back-links to the web page; in essence, the more popular the page is, the more people have referenced the page and the more relevant it is.

This approach works very well when used with hypertext documents. However, it is useless when no back-links exist conceptually in the target domain, such as an RDBMS system. Recently, Google started a new service called "Google Database", by which people can store their information and let Google index it. It will be an interesting task to discover its behavior in case of simple text files and to compare the document ranking algorithm with Lucene, for example.

Text search in RDBMS: Case studies

Creating a usable full-text search solution for an RDBMS is challenging task. To start with, full-text search systems work with the "document" concept. A way must be devised to map relations in an RDBMS to documents in the full-text search engine. The designer of the system must decide on the level at which the mapping is performed and what the result of the search is to be. Furthermore, the search engines that are widely used to index web pages are using a "single writer" strategy. In other words, there is a crawler component populating an index and a search engine accessing the index in read-only mode. This is a complete contrast to a database engine, where transactions write to the database and run in concurrent mode.

Neither the relational data model nor SQL is of much use for full-text search, though it is relatively easy to build the inverted file index and corresponding data structures that were described above. The main issue is not the task of populating the index, but the querying itself, especially when terms are combined with the AND operator. The ranking of the results is also an issue. We recommend to the reader an article "Integrating Structured Data and Text", published in the Intelligent Enterprise magazine, as a good overview of a pure SQL approach to the text search [2].

The absence of an acceptable solution that uses relations and SQL does not mean that full-text search in databases is not possible. On the contrary, most database vendors have implemented this feature in their products. Some third-party products exist for Firebird, too, though they do not compare favourably with the solutions from the commercial database vendors from the perspective of usability.

Below we summarize the most notable approaches that are used in the products available on the market.

Oracle Text

Oracle Text was a part of a separate product interMedia. Since Oracle version 9 it has shipped with the Standard Edition of the server. The Oracle Text feature is completely integrated with the database and supports replication, parallel execution, etc.

Creating a full-text index in Oracle is done by issuing the CREATE INDEX command and specifying the INDEXTYPE IS CTXSYS.CONTEXT clause. The CONTEXT index type corresponds to the general-purpose index described in previous section. Oracle Text has two additional index types: CTXCAT, an index "designed specifically for eBusiness catalogs" and CTXRULE for building classification or routing applications. The CONTEXT index type also allows an application to specify the synchronization preference: manually, on commit, or at regular intervals. It also allows the use of a transactional text index, by which information becomes searchable right after inserting or updating. The CTXCAT index type is optimized for short documents and is always up-to-dat.

Illustration 3.: *Example of CREATE INDEX statement in Oracle*

```
CREATE INDEX description_idx ON product_information(product_description)
INDEXTYPE IS CTXSYS.CONTEXT
PARAMETERS ('sync (every "SYSDATE+5/1440")');
```

Illustration 3 shows an example of the CREATE INDEX command for a "product_description" column of the "product_information" table that is to be updated every five minutes.

Oracle supports following query types:

- "Exact match". Corresponds to the "term query", without the tokenizer in our classification; matches only those documents that contain exactly the specified term;
- "Word positioning". Corresponds to the "phrase query" and "sloppy phrase"

query; it matches documents that contain a specified phrase, words near to each other, but also searching for documents that contain the discrete words of the phrase in the same sentence or paragraph.

- "Inexact match". Sort of combination of the "predicate query" and "fuzzy query" with different matching algorithms, like "SOUNDEX", stemming, prefix matching, thesaurus matching.

- "Intelligent match". This corresponds to the "fuzzy query" with the matching algorithm backed by some knowledgebase that can determine ontological similarity of the query terms and terms found in documents.

- "Boolean combination". Corresponds to "Boolean query" in our classification, allows other queries to be combined by means of AND, OR, NOT operators.

The query specification (Illustration 4) is somewhat unusual and requires explanation.

Illustration 4.: *Example of a full-text query in Oracle*

```
SELECT
    score(1), product_id, product_name
FROM product_information
WHERE
     CONTAINS (product_description, 'monitor NEAR high resolution', 1) > 0
 ORDER BY score(1) DESC;
```

First, the CONTAINS query operator. It takes two mandatory parameters – column name and query expression. It returns a value greater than 0 for all records where the specified column matches the query expression. The third optional parameter assigns a label to the score returned for each record which can be later accessed by the SCORE function, which takes the label as parameter. Applications can use multiple CONTAINS clauses in one statement and combine corresponding scores with boost factors or other arithmetic to obtain better document ranking.

The technical side of the story looks not so bright. In fact, despite the white papers, even in Oracle 10g full-text search works rather more like an add-on than an architectural feature. First, the full-text search parameters can be changed only via procedure calls from the CTXSYS schema. The actual syntax for the call is a mix of Oracle SQL and the component-specific commands. The full-text index itself is a potential bottleneck. As you can see from [3], when index synchronization is set to "sync on commit", Oracle effectively serializes all transactions. Additionally, the synchronization happens only after commit, causing a gap between when documents that were added or updated become visible to other transactions and when they become searchable. And, in the worst case, error/failure in the full-text index has no influence on the outcome of the transaction.

Microsoft SQL Server

Microsoft took a different approach. The core search functionality is provided by the Microsoft Search (MSSearch) technology that is also used by Microsoft Exchange and Microsoft SharePoint Portal Server. MSSearch builds, maintains and queries full-text indexes stored in the file system (as opposed to inside SQL Server). The logical and physical storage unit used for full-text indexes by MSSearch is a catalog. A full-text catalog contains one or more full-text indexes per database—one full-text index may be created per table in SQL Server and may include one or more columns from that table in the index. Each table may belong to only one catalog, and only one index may be created on each table [4]. The querying mechanism supports searching for words or phrases, words in close proximity to each other and inflectional forms of verbs and nouns.

Querying involves an OLEDB access to the MSSearch component and returns a result set containing the ID of the document and its rank. Illustration 5 shows two examples of using the full-text search in Microsoft SQL Server 2000 (see [4] for details).

Illustration 5.: *Querying MSSearch component and joining it with the table from the database in SQL Server 2000*

```
-- [APPROACH 1:]
-- most expensive: select all, then join and filter
SELECT ARTICLES_TBL.Author, ARTICLES_TBL.Body, ARTICLES_TBL.Dateline,
   FT_TBL.[rank]
FROM FREETEXTTABLE(Articles, Body, 'Ichiro') AS FT_TBL
INNER JOIN Articles AS ARTICLES_TBL
ON FT_TBL.[key] = ARTICLES_TBL.ArticleID
WHERE ARTICLES_TBL.Category = 'Sports'

-- [APPROACH 2:]
-- works, but can backfire and become slow or return inaccurate results:
-- perform filtering via Full-Text and only extract key and rank
-- (processing done at web server level)
SELECT [key], [rank]
FROM CONTAINSTABLE(Articles, *, 'FORMSOF(INFLECTIONAL('Ichiro')
      AND "sports"')
```

In the imminent Microsoft SQL Server 2005 release, the full-text search component was improved with respect to index maintenance, e.g., the full-text search index is backed up together with the database backup, which was not the case before. Microsoft also claims to have improved the actual indexing performance in the new version.

Netfrastructure

Netfrastructure is not positioned as pure database system, but as a web application development environment. Netfrastructure is designed and developed by Jim Starkey, the person who created InterBase, but targets a different application domain. Unfortunately, not much information about its full-text search feature can be found in public sources [5]. However, the approach used there is quite interesting and worth discussing here. All of the detail below was obtained either from posts by Jim Starkey in Firebird-Architect list or from the private correspondence with him.

There is no separate command to create a full-text search repository, since it comes into existence with the creation of the database. The only thing required is to define the fields of the tables as searchable, as shown on Illustration 6.

Illustration 6.: *Definition of the searchable column in Netfrastructure*

```
CREATE TABLE product_information(
  id INTEGER NOT NULL PRIMARY KEY,
  product_description VARCHAR(2000) SEARCHABLE);
```

Adding the SEARCHABLE keyword after the text or CLOB column definition is enough to tell the engine to index the contents of the field on each INSERT or UPDATE. The added documents are directly searchable within the same transaction and rollback removes added items from the index. Currently development is done to allow applications to move the updating of the indexes to a separate thread. In this case, added documents might not be directly searchable in the current transaction.

There are two options for the full-text query. The first uses the MATCHING operator within the SELECT statement (Illustration 7); the second one requires use of the API and, according to Jim Starkey, is exactly what application developers are interested in.

Illustration 7.: *Example of a SELECT-based full-text search in Netfrastructure*

```
SELECT * FROM product_information
WHERE
   product_description MATCHING '+monitor +"high resolution"';
```

The most interesting part of the full-text search using the API is that the search query returns all hits in all tables and all fields. Each hit is ranked by the scoring algorithm (currently hardcoded) and is returned in the ResultList structure, representing a list of java.sql.ResultSet instances, each of them containing one record.

Additionally, a ResultList object provides schema and table names and the score of the hit. The key message here is that people looking for something in the database are not interested in hits in a single table, but most likely in all tables. Naturally, the search API allows the search scope to be limited to specified tables.

An example of using the Netfrastructure search API from Java is shown in Illustration 8.

llustration 8.: *Example of using search API from Java in Netfrastructure*

```
Statement stmt = connection.createStatement();
ResultList rl = stmt.search("+monitor +\"high resolution\"");
while (rl.next()) {
  String tableName = rl.getTableName();
  double score = rl.getScore();
  ResultSet rs = rl.fetchRecord();
  ... // process ResultSet object
}
```

Stops words in Netfrastructure are indexed but skipped during the scan. After a hit, the search engine goes back and checks that the word was actually there. The phrase "Alexander the Great" scans for "Alexander" and "Great" with an intervening word. When it finds an instance, it checks for "the" in between.

Netfrastructure supports the following atomic queries:

- Term or list of terms. Hitting more terms in the query is better than hitting few. Upper case letter matches uppercase letter; lower case letter matches either case;
- Specifying the required term and the term that should not be contained in a document. This is somewhat similar to the "Boolean query" described above, especially if we consider that the simple list of terms is combined with OR operator;
- Phrase search, which was explained above;
- Wildcard and prefix queries.

In addition to the standard scoring factors already mentioned, Netfrastructure supports "distance from the beginning of the document". No information about the reasons for this factor was found, however.

MySQL

MySQL's solution is somewhat similar to the solution used in Oracle, but has also some interesting differences. It must be mentioned that full-text search is supported only on MyISAM tables, which is pretty strong limitation.

An application can define a full-text index for a selected table and list of columns to index (Illustration 9). The FULLTEXT "constraint" takes the list of columns that will be automatically indexed.

Illustration 9.: *Example of creating the full-text index for two columns in MySQL*

```
CREATE TABLE articles (
     id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
     title VARCHAR(200),
     body TEXT,
     FULLTEXT (title,body)
);
```

MySQL supports all basic query types except the fuzzy queries. A typical MySQL query using full text search (query 1 in Illustration 10) performs only record filtering according to the specified query. In order to obtain the score of the hit, quite a strange construct is used where the MATCHING ... AGAINST clause is used in the column list section of the SELECT statement as well as in its WHERE clause (query 2 on Illustration 10). The query optimizer detects such a usage pattern and the actual full-text query is executed only once.

Illustration 10.: *Example full-text search queries in MySQL*

```
-- query 1
SELECT * FROM articles
    WHERE MATCH (title,body) AGAINST ('database');

-- query 2
SELECT      id, body,
    MATCH (title,body) AGAINST
     'Security implications of running MySQL as root')
       AS score
FROM
      articles
WHERE
    MATCH (title,body) AGAINST
      ('Security implications of running MySQL as root');
```

The most interesting feature used in MySQL is so-called "query expansion". It is quite useful when the search query is short or when the query terms contain spelling errors. The idea is to execute two full-text queries. The first one is matched against the specified query and its top results are used as a query for the second search. Documentation contains the following example that describes the fuzzy matching with the spelling corrections:

"Another example could be searching for books by Georges Simenon about Maigret, when a user is not sure how to spell "Maigret". A search for "Megre and the reluctant witnesses" finds only "Maigret and the Reluctant Witnesses" without query expansion. A search with query expansion finds all books with the word "Maigret" on the second pass.".

Happy New Year Wishes

Marina Novikova:
What do you think were the greatest FB project achievements in 2005 in general?

Helen Borrie:
Getting the Fb 2.0 Beta out was more of an achievement than it is givencredit for. The guys have done a fantastic job of cleaning up some serious messes and making a lot of things work better. The record of serious, thoughtful work is there for all to see, whether you read the core devs' reports at the website:
http://firebird.sourceforge.net/index.php?op=devel?sub=engine
or whether you test Fb 2.0 against predecessors and competitors.

Paul Ruizendaal:
We have achieved so much in 2005, it is hard to pick something. Perhaps our greatest achievement is that we are now recognised as one of a few leading open source databases, winning prestigious projects. Mostly that is because of all the great stuff we as a community have done, but in part it is also because we have gained a lot of self confidence: we radiate success. People see that and try our code because of it.

Vlad Horsun:
There have been many events this year, but speaking about achievements the greatest one is probably the release of the first beta, and, I hope, the second beta release before the end of this year.

Dmitry Yemanov:
It was a hard work on making v2.0 stable and moving to the Beta stage. We have released nothing major this year, but a lot of internal work has been done and people may see the results in our field-test versions.

Alex Peshkov:
Foremost it is stable Vulcan's work with huge databases in the SMP mode in SAS.

Firebird: Present and the Future

Full-text search support in open-source databases an exception rather than a commodity. MySQL provides a solution similar to the one used in Oracle, but PostgreSQL and Firebird do not have any built-in solutions. Existing third-party components have limited capabilities, especially when it comes to the level of integration with the engine and the querying capabilities.

It is clear that Firebird needs full-text search capability, though it is not yet clear how to implement them. Until the issue is solved by the Firebird team, let's try to define the possible solutions that applications can use.

Fuzzy word matching

Before we start thinking about the full-text search approach, let's check whether we need it at all. Those applications looking for better fuzzy matching capabilities in short text fields do not need true full-text search engine. Not only is it a feature overkill but it slows down the database. Short sentences and product names are not good candidates for full-text search algorithms.

Below is the short description of two products that have been available on the market for some time and can be directly used with Firebird. Neither of them provides hit scoring, but limits the search to "match"/"no match" behavior. Because of the querying and ranking limitation we do not regard them as full-text search solution , but as a fuzzy word matching components.

FastTextSearch for InterBase/Firebird

FastTextSearch is a set of UDFs and stored procedures to perform full-text search within text and BLOB fields. The architecture is conceptually similar to the one used in Lucene. However, due to limited querying capabilities and absence of hit ranking, we did not include this component in our case studies.

The FastTextSearch component constructs a search repository using a proprietary binary index format. Since Firebird does not provide any mechanism for engine callback from within UDF functions, it has to use BLOB fields, the content of which is processed by the proprietary UDF.

Illustration 11.: *Initial steps to enable the full-text search for a table CUSTOMER*

```
-- step 1
ALTER TABLE customer ADD fts_id INTEGER;
CREATE INDEX customer_idx_fts ON customer(fts_id);

-- step 2
CREATE TRIGGER t_bi_customer_fts FOR customer ACTIVE BEFORE INSERT AS
    DECLARE VARIABLE parser INTEGER;
    DECLARE VARIABLE i INTEGER;
BEGIN
    IF (EXISTS(SELECT * FROM ts$opt WHERE enable > 0)) THEN BEGIN
        parser = parser_create();
        i = parser_add(parser, new.CUSTOMER);
        i = parser_add(parser, new.CONTACT_FIRST);
        i = parser_add(parser, new.CONTACT_LAST);
        i = parser_add(parser, new.ADDRESS_LINE1);
        i = parser_add(parser, new.ADDRESS_LINE2);
        i = parser_add(parser, new.CITY);
        i = parser_add(parser, new.COUNTRY);

        EXECUTE PROCEDURE ts$update(new.FTS_ID, 'customer', :parser)
            RETURNING_VALUES new.FTS_ID;

        i = parser_free(parser);
    END
END
```

These functions are shipped in a precomit has to use BLOB fields, the content of which is processed by the proprietary UDF.

Because of the restrictions described, the FastTextSeach component requires changes to the existing schema. First of all, in all tables that are to be indexed, a new integer field has to be added. This field will contain a "document ID" that is generated by the indexing component when the document is added to the index. Next, the application has to define triggers that perform mapping from the table record to an abstract, multi-field document.

1 Even though the "full-text search" term is used in their descriptions.

The parser_add UDF parsers the content of the specified column and registers it under the ID that was generated by the parser_create UDF. The stored procedure call adds the preprocessed document to the index and returns its ID for storing in the column fts_id.

Querying capabilities of the FastTextSearch component are rather limited. It supports only term queries, with terms that can be combined with either AND or OR Boolean operators; the terms matching can be either prefix or SOUNDEX and the the setting applies to all terms in the query (Illustration 12). FastTextSearch does not provide hit scoring, either, limiting the result to "match"/"no match" only.

Illustration 12.: *Example of full-text query using the FastTextSearch component*

```
SELECT c.* FROM
   ts$select_or('customer', 'tech corp', 0) ts
 LEFT JOIN
   customer c ON c.fts_id = ts.obj_id
```

IBObjects Full-Text Search Module

IBObjects [8] has also a module for fuzzy text matching with the features similar to the FastTextSearch component, but using different technology. Like the rest of the IBObjects components, it is limited to the Delphi/C++ Builder development environment and Windows platform. Efforts have been made to port IBObjects to Kylix to run it under Linux, but no official release is available.

Similar to the FastTextSearch component, IBObjects FTS requires changes to the existing database schema. The distribution contains a handy GUI that simplifies this task and hides all complexities from the user. Unlike the FastTextSearch component, that performs text processing and index maintenance within the same request, IBObjects FTS works more elegantly. Triggers that were defined in the previous step populate an auxiliary table with the changed information and generate a Firebird event about the change. This event is processed by a standalone index synchronization component, that fetches the information about the changed tables, processes the content of the text fields (including BLOB SUB_TYPE 1 fields) and updates the indexes.

The most elegant thing is the usage of the event facility of Firebird, which effectively decouples the transaction in which tables are modified and index population and, since events are only posted if the transaction in which they are created is committed, it saves CPU cycles in case of rollback. The drawback of this scheme is that, because the full-text query will not "see" updates performed in the same transaction, changes will be searchable only after commit.

IBObjects FTS components supports exact and partial word matching, exact and partial metaphone code matching, SOUNDEX code matching, synonym and antonym matching.

Searching the database is performed in a way similar to the FastTextSearch and MS SQL Server approaches, but offers a separate Delphi component for these purposes. Internally the full-text query is evaluated against the index tables and then joined with the original table using the primary key column defined during index creation. IBObjects FTS does not provide hit ranking, only "match"/"no match" hits.

Integration with external tools

Another possible approach for Firebird users is integration with third-party tools. The author does not know about any such projects, but the development, at least theoretically, is not complicated. The only requirement is the ability to execute queries against external data sources, which is only available currently in Oracle-mode Firebird using external stored procedures capabilities [9].

In this case application can use scheme similar to the IBObjects FTS, where a stand-alone synchronization engine would listen for the events generated by the update triggers, would fetch the modified data and feed them to an external full-text search engine, say, MSSearch via the COM interface, for example. The querying should be performed via the external procedure that returns the table primary key and the score, which the application can join with the target table using syntax similar to the one used in MS SQL Server, shown in Illustration 5.

2 In old versions of Firebird three triggers has to be defined, BEFORE INSERT, BEFORE UPDATE and BEFORE DELETE. Since Firebird 1.5 application can use universal triggers and put all logic into one PSQL block.

3 Warning: The C/C++ external procedure interface is not yet stable, significant changes are possible in the future, especially when this feature will be ported to Firebird main code base. Therefore, those who will start development against the existing external procedure facility, be prepared to update the interface part.

About the author

Roman Rokytskyy works as a senior consultant in Germany. Since 2001 he has been partcipating in the Firebird project, primarily in the JCA/JDBC driver subproject, but is also involved in other areas of Firebird development. Information retrieval, full-text search in particular, is one of the author's interests, that has been applied in practice during the development of a high-performance content store for spatially distributed information.

References

1. Justin Zobel, Alaister Moffat, and Kotagiri Ramamohanarao. Inverted File versus Signature Files for Text Indexing. ACM Transactions on Database Systems (TODS), Volume 23 , Issue 4 (December 1998), Pages: 453 – 490

2. David Grossman and Ophir Frieder. Integrating Structured Data and Text.
Intelligent Enterprise Magazine, September 18, 2001, Vol 4 No 14
http://www.intelligententerprise.com/010918/414analytic1_1.jhtml
and October 24, 2001, Vol 4 No 16 http://www.intelligententerprise.com/011024/416analytic1_2.jhtml.

3. Oracle Text 10g. Technical Overview. http://www.oracle.com/technology/products/text/x/10g_tech_overview.html

4. Andrew B. Cencini. Building Search Applications for the Web Using Microsoft SQL Server 2000 Full-Text Search.
Microsoft Corporation, December 2002.
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsql2k/html/sql_fulltextsearch.asp

5. Netfrastructure Inc., http://netfrastructure.com/

6. MySQL 5.0 Reference Manual. Chapter 12.7. Full-Text Search Functions.
http://dev.mysql.com/doc/refman/5.0/en/fulltext-search.html

7. FastTextSearch for InterBase. http://www.textolution.com/ftsib.asp

8. IBObjects. Full Text Search Module. http://www.ibobjects.com/ibofts.html

9. Fyracle: Oracle-mode Firebird. http://www.janus-software.com/fb_fyracle.html

New Cache Options In InterBase 7.5

by Bill Todd,
billtodd@name.org

Write Caching

Before I discuss write caching I have to explain the confusing and inconsistent terminology that surrounds it. Nowhere will you find an option to turn write caching on or off. If you look at the General tab of the Database Properties dialog in IBConsole you will see an option called Forced Writes. Setting forced writes to true disables write caching by forcing all writes to disk immediately. If you examine the documentation for gfix in the Operations Guide you will see that it has a -write switch that can be followed by either async or sync. The sync option turns on synchronous writes, which is the same as turning off write caching. The async option enables asynchronous writes which turns on write caching. In other words:

Write caching on = forced writes off = asynchronous writes

Write caching off = forced writes on = synchronous writes

Because the terminology is inconsistent and confusing I will talk about turning forced writes on or off. Just be aware that how you do that and what it is called varies depending on whether you use IBConsole or gfix. By the way, just to add to the confusion, write caching is on by default on Linux and Solaris and off by default on Windows.

Write caching is a two edged sword. It can improve performance because pages are written to the operating system's write cache and the operating system writes the data to disk at some later time. Take a simple case of a database where one page holds, on average, 50 records. If you update 20 of the rows on that page the page will be written to disk once with write caching enabled. With write caching disabled the page will be written to disk at least once for each transaction and perhaps more often than that.

The disadvantage of write caching, and it is a big one, is that you are much more likely to have a corrupt database if the database server crashes with write caching on than with it off. The reason is that all high-end relational databases have complex internal structures that include pointers to other structures. One example for InterBase is a table with a blob field. If you insert a new row the row will contain a pointer to the page where the blob is stored. Now suppose that write caching is on and that the page containing the new row is written to disk then the server crashes before the blob page is written. The database now contains a pointer that points to an object that does not exist.

Why can that happen with write caching on but not with write caching off? InterBase uses a system call "careful writes". Careful writing means that InterBase writes the object being pointed to before it writes the pointer. With write caching off the pages are always written to disk in the order that InterBase writes them which is in careful write order. With write caching on InterBase writes the pages to the operating system write cache. InterBase has no control over the order in which the operating system cache manager will write the pages to disk. In Firebird and in InterBase versions prior to 7.5 you have only two choices. Turn write caching off, be safe, and put up with the slower performance or turn write caching on to get the improved performance and hope that the server does not crash. In InterBase 7.5 you have two new options.

Group Commit

With forced writes enabled you are guaranteed that all user updates that are part of a transaction are on disk and have been written in careful order when the transaction is marked as committed. With force writes off you are not guaranteed that writes will take place in careful order and you are not guaranteed that all changes are on disk when the transaction is marked committed.

When a transaction commits with forced writes and group commit enabled, the pages that have been updated by the transaction are passed to a background cache writer thread in careful write order. As soon as the updated pages have been passed to the cache writer thread the commit returns. This lets the client that committed the transaction continue working without waiting for the disk I/O to take place. It also lets other transactions that are waiting for the committed transaction's locks to be released to continue. The writer thread writes the pages to disk in the order it received the pages (which is careful write order). Since all writes are done in careful write order you get the same protection from corruption that you get with forced writes only but you get better performance because the user does not have to wait for the disk I/O to take place. The only risk is that if the server crashes you will lose any changes made by transactions that were committed but which the writer thread has not yet written to disk.

That seems like a recipe for disaster because a transaction can be marked as committed even though it has not been written to disk. It seems that, if the server crashes, the changes made by the transaction will be lost, whilst InterBase will think the transaction committed so it will not rollback the transaction when the server restarts.

Fortunately, that is not what happens. The state of transactions is tracked on the transaction inventory pages (TIP). However, there are two copies of the TIP, one in memory and one on disk. With group commit on, when a transaction commits it is marked as committed in the in-memory copy of the TIP (called the transaction inventory page cache or TPC). The transaction is not marked as committed in the on-disk copy of the TIP until after the writer thread has finished writing all of the transaction's pages to disk. If the server crashes, the in-memory TPC is lost. When IB restarts it looks at the on-disk TIP, sees that the transaction is still

active and rolls it back.

To enable group commit execute:

```
ALTER DATABASE
   SET GROUP COMMIT
```

To disable group commit use:

```
ALTER DATABASE
   SET NO GROUP COMMIT
```

Although you can have group commit set with forced writes off it makes no sense to do so. The background writer will be writing to the operating system cache, which cannot preserve careful write order or guarantee when the updated pages will actually be written to disk.

Database Flush

If you want better performance than forced writes with group commit offers and you are willing to take some additional risk, then database flush may be right for your environment. Database flush is an option you can use when forced writes are off.

The problem with turning forced writes off is that there is no way to predict when the operating system will actually flush writes in its cache to disk. This means that there is no way to estimate how many writes you may lose if the database server crashes.

The new database flush option lets you tell InterBase when it should flush all cached writes to disk. You do this by setting the flush interval. The safest flush interval is zero. When the flush interval is set to zero InterBase flushes the cache to disk each time a transaction commits. If multiple transactions commit at the same time the cache is flushed once after all of the transactions have committed. This is fairly safe because the cache is flushed immediately after each commit. Nevertheless, because careful write order is not used, a crash while the cache manager is writing can still corrupt your database.

For better performance you can set the database flush interval to a positive integer that is the number of seconds between flushes. In this mode the InterBase cache writer thread ignores transactions and tells the operating system to flush the cache every N seconds. This is almost as risky as running with forced writes off and database flush disabled. The only advantage is that you can be sure the cache is flushed to disk at least every N seconds.

To enable database flush, first make sure that forced writes are off for the database; then execute the following command:

```
ALTER DATATBASE
   SET FLUSH INTERVAL 5
```

where 5 is the number of seconds between requests to flush the cache. You can set the flush interval to any value you choose.

To disable database flush use the following command:

```
ALTER DATABASE
   SET NO FLUSH INTERVAL
```

Which Option Should You Use?

You now have five options for controlling physical writes to disk. The following two options are very safe because they use careful write order and because they take place under transaction control. Any transaction that has not been completely written to disk before a server crash will be rolled back when InterBase restarts.

- Forced writes on (also known as synchronous writes)
- Forced writes + group commit

The last three options carry much more risk because careful write order is not used and the changes made by a single transaction may be partially written to disk. This can cause logical or physical database corruption.

- Forced writes off + database flush after each commit
- Forced writes off + database flush at a fixed interval
- Forced writes off

Choosing the best option is easy. Pick the safest option that gives you the performance you need. If choose any option that does not include forced writes on, make sure your server is as stable as possible by using a dedicated database server in a physically secure location with a UPS. This will reduce the chance of a database crash.

If you use user defined function libraries, and particularly if you write your own UDFs, test them very careful-

ly. Buggy UDFs are the leading cause of InterBase server crashes.

Database Linger

If at least one user is always connected to your database or if your client application(s) use connection pooling so that one or more pooled connections are always connected to the database, there is no reason to use database linger.

When the last user disconnects from a database the database is removed from memory. This causes several problems. First, it may not give the garbage collector thread enough time to delete old record versions that have been queued for deletion. Second, when the first user connects to the database, memory for the cache must be allocated and initialized and the database metadata must be loaded into memory. This makes the first connection much slower than subsequent connections.

The first user to connect is not the only one that will pay a performance penalty. Consider the case where user A runs a SELECT statement. When the SELECT is executed, all of the required index and data pages are read from disk and placed in the cache. If user B runs the same SELECT he/she will get better performance because the index and data pages will be read from the cache instead of from disk. Now, suppose that users A and B disconnect and the database is removed from memory. A few seconds latter user C connects and has to wait while the cache is allocated and the metadata loaded. Next user D connects and runs the same SELECT that was run earlier by users A and B. However, user D does not get the fast execution that user B experienced because the newly created cache is empty. All of the index and data pages must be read from disk again.

Database linger keeps the database in memory for a specified period of time after the last user disconnects. It has the same effect as always keeping a connection open but it is safer because there is no risk of leaving a transaction open.

To enable database linger:

```
ALTER DATABASE SET LINGER INTERVAL 600
```

This command tells InterBase to keep the database in memory for 600 seconds (10 minutes) after the last user disconnects. Of course you can use any linger interval you choose.

There are two ways to disable database linger.

```
ALTER DATABASE SET NO LINGER INTERVAL
ALTER DATABASE SET LINGER INTERVAL 0
```

If your environment is such that all of the users might disconnect from the database at the same time, enable database linger. It costs nothing and will improve performance any time all users disconnect for a short period of time.

New Fields in RDB$DATABASE

You can determine the settings for any of the new cache attributes by querying the RDB$DATABASE system table. The RDB$FLUSH_INTERVAL field contains the current database flush interval. RDB$LINGER_INTERVAL contains the database linger value.

The RDB$GROUP_COMMIT field shows whether group commit is enabled or not. A "Y" in RDB$GROUP_COMMIT means group commit is on, while an "N" shows that group commit is off.

Bill Todd is a member of Team Borland supporting InterBase and Delphi on Borland's Internet newsgroups. He provides InterBase design, consulting and training services to clients throughout the world.

You can contact Bill at contact@dbginc.com

Object-oriented development and RDBMS, Part 2

By Vladimir Kotlyarevsky
vlad@contek.ru

Links between objects

(See also [5], the "Links" chapter)

Links as object attributes

In my opinion, the traditional approach to designing relational databases leads to some problems in establishing connections between the relations.

1. A relational connection is poorly documented (or not documented at all) "in itself." That is to say, if there is no FK, then it would be impossible to figure out whether there is a relationship or not, without project documentation. (Here, of course, one can argue that if there is no FK, then there is no a relational connection. I do not think it is quite true, since a link is always something more than just an FK, and FK is nothing but a constraint). If there is an FK, then a relationship is needed, though its essence is not always clear (unless the attribute names and FK name are chosen properly). As you know, self-descriptiveness is one of essential characteristics of a good programming style.

2. To open a relationship (i.e. to access a tuple from a certain relation via a link to it) one would need to apply a method, specific to this particular relationship. A relationship in an ordinary object model means that a field of an object contains a pointer (link) to another object.

For example, if all classes of a system are inherited from the MyObject class, which has the "Name" property, (or they realize one appointed interface, or the language supports RTTI, etc) then, with the help of a link to an unknown object (using the target polymorphism), we can figure out at least its "Name." In a case where RTTI is supported, we can find out everything else. We can get all this information with the help of a single link. What does a relational model offer us in such cases?

Let us use an order-document as an example. One of the fields in this document is a link to a customer. How, using a standard projecting method, can we figure out the customer's name (the most frequently used attribute)? To do that, one would need to know where and how the customer list is stored, how the relationship between an "order" and a "customer" is organized.

Moreover, one must be sure that such a relationship exists ?, and, in addition, be able to create some kind of expression for opening that relationship--a relational join , a stored procedure call or something else. Such things are usually hardcoded for each particular case, i.e., the relationship is defined during the projection process and implemented during programming. A particular linkage is given substance via a FK and some SQL-expressions that are encoded somewhere for the purposes of sampling and storing the object. Worse, the linkage might be substantiated by no other means than a lookup-control anchored to a record set field. in a user interface.

The problem becomes even more obvious in another example, where it is not even clear where the link actually leads. Consider an accounting system based on transactions. The transactions are stored in a separate table; one record contains information about one transaction. Each transaction has the "amount" attribute (money, goods, etc) and several analytical characteristics. Each characteristic field is a link to a certain directory. At that, one does not know what that directory is, and it is not clear whether it is at the stage of projecting or of programming. It also depends on several additional conditions, which occur in run-time only.

So what should we do in such cases, when a standard FK cannot be created? How can one figure out the name of the object to which the link refers? It is possible, if it is specified at the stage of projection. It can be done in the following way: get to the account description and see what analytical accounting object type is stored in this transaction field. After that, find out what table this object belongs to, and then figure out the name of the field in this table, in which the object's name is stored. In this case, there is some self-descriptiveness--a description of analytical characteristics in account parameters. The downside is that this self-descriptiveness is too problem-oriented. What's to be done with other links in the database pertaining to these transactions? The common practice is to do nothing. The content of a "select" is left to accomplish the relationship…

The first two problems have a simple solution which, if incomplete, is at least simple and convenient. All links to objects are created only through the "OBJECTS" table and have a "TOID" type. Indeed, if each object has a corresponding record in OBJECTS, then why not? Furthermore, in the "Classes" table, each attribute of "link type" should have a distinct set rules described, which would limit the number of objects the link may refer to (for example, "objects of the listed types only," or "only the objects from a certain folder").

1. Such a link is better documented since, even using just this link (type OID value), you can easily figure out the type of the object it is linked to when referring to OBJECTS.

Type ClassId, if the id link to the object is known:

```
select ClassId
      from OBJECTS
      where OID = :id
Name of the type:
select o.ClassId, o1.name as ClassName
      from OBJECTS o, OBJECTS o1
      where o.ClassId = o1.OID and OID = :id
```

Happy New Year Wishes

Marina Novikova:
Who has done his best to push the project forward? Why?

Helen Borrie:
They have all done their best and should be rightly proud of their year's work. Milan Babuskov, Marius Popa and Mauricio Longo deserve special commendation for their efforts to "get Firebird out there". And let's not forget to raise our hats to Paul Vinkenoog, whose solitary efforts have turned the Firebird docs system (originally developed by David Jencks) into a very usable system that anyone can use. But I particularly want to highlight the largely unheralded efforts of Claudio, whose job it is to scrutinise every bit of code that comes through and to get into debates (usually private, occasionally public, often tortuous) about pieces that don't measure up to standard. Claudio spends many hours also writing detailed and precise arguments - in English, not his native language, and often with a lot of wry wit.

Why? The efforts of most of the core devs and the driver developers are very visible. Claudio has an essential, but horrible, job to do and he does it very, very well. He never stops, because the code never stops. Sometimes he has even had to put up with an insulting response from some programmer who believes himself to be above criticism. Where does one go for recourse in such a situation?

Paul Ruizendaal:
We *all* have done our best to push the project forward. Who has done his best the most? I don't know. Much effort occurs in areas that I am not so familiar with. From my personal perspective I am most impressed by Dmitry Yemanov. Keeping many highly creative, highly talented, highly opinionated developers focused is a tough job. Linus

2. To open a link, a simple call and a link value would be enough:

```
select o.Name, o.Description, o.ClassId
    from OBJECTS o
    where o.OID = :id
```

If the link is not created yet (for example, a user is creating a new object and is about to enter a link), the rules of its creation can be got in "Classes."

External links between objects

(See [5], the "Links" chapter)

In the previous chapter, we considered the cases where a relationship (link) is an object attribute. However, there are links which, generally, are not a part of an object.

1. A relationship of "what entailed what" type. For example, a document entails a whole chain of actions and documents, which together constitute a business-process. However, the same documents and actions occur in completely different business-processes as well, and those can be independent units. In this case, it does not make a sense to create a field-link to an entailed object in each object: first, their types may differ; second, there can be several of them.

2. Purely logical entering of something to somewhere. For example, the files are included in catalogs, but nevertheless are self-dependent objects, and their meaning (as a rule) does not depend on a catalog changing.

3. And so on ?

This problem is solved by creating an extra link table.

```
create table links(
  link_type TOID,
  left TOID,
  right TOID,
  Constraint LINKS_PK primary key (link_type, left, right))
```

This is an almost common table for organizing a M:M relationship. The "left" and "right" are the links to the left and right sides of the relationship, i.e., the two objects between which the relationship is established. I call it "almost common" because there is a difference. First, we have a common object set of all types, and there is the only way to refer to them (the "OBJECTS" table and the unique-keys system). Secondly, there is the link_type field. The first thing allows us to use the "links" table for any external relationships between any objects. Contrast that with the standard relational approach, which requires creation of separate tables for each M:M relation, because key data types are not standardized. If they are standardized (for example, integer only), their values remain unique only within the limits of the tables. The second (link_type field) enables the type of a relationship to be defined, as well as rules for its creation.

What does all this mean for us? Let us consider the known method of realizing a tree-like structure in a relational table (id, parent_id, described, for example, in [6]). Using the "Objects" and "links" tables, you can easily get this structure with the help of the following query (for a certain relationship type):

```
select o.OID as id,
    l.left as parent_id,
    o.name,
    o.description
    from objects o, links l
    where o.OID = l.right and l.link_type = :link_type
```

Such a structure is useful and simple, and, besides, there are many visual components which visually realize the tree-like presentation.

However, it is often necessary to realize a more complex relationship system

between objects. So, the "links" table (this is where the most interesting things begin to happen ?) enables almost any kind of trees and object networks to be defined. Why? To explain this, let us consider an example with the "order" object.

- First, to organize a **global logical object catalog** in a database, it is desirable to simultaneously store all objects of "order" type in:
  - the "Orders" folder.
  - the folder of the manager who created it and holds the business-process concerned with it.
  - the folder with all company's documents over the period of the current month.
  - the personal folder of the clerk responsible for this particular order.
  - somewhere else.
- Secondly, consider the need to present the business process (BP) to which the order belongs as a column and handle it as a single whole. For example,
  - BP began with a customer contract about permanent services during a year.
  - As a consequence, the document named "delivery schedule" was created.
  - Then, according to the contract and schedule, the order appears,
  - All this is followed by invoice, shipping, etc.

Now let us describe how such problems can be solved. It seems to me that, at least in systems dealing with documents circulation, such tasks are quite widespread.

The main object catalog

The first task is accomplished right away because, in contrast to the id, parent_id structure, our structure allows use of an unlimited number of links to one and the same object (due to the fact that the notions of object and links are separated both logically and physically). Let's enter the "Logic Folder" folder type with ClassId = 100; also, let the Order type have ClassId = 200. Then the record set in our structure, which realizes the problem specification (though with no records in the Classes folder), will look as the follows:

Objects

OID	Name	Description	ClassId
0	Root	Root folder	100
1000	PrivateFolders	Private folders of the members	100
1001	Documents	All documents of the company	100
1002	Smith	All orders of the clients	100
1003	February2002	Mr. Smith's (manager) private folder of	100
1004	March2002	All documents obtained during February, 2002	100
1005	Roberts	All documents obtained during March, 2002	100
1006	1001	Mr. Roberts' (clerk) private folder	100
1007	AnotherFolder	Just a folder	100
2001	Order0001	Our order	100

Happy New Year Wishes

Torvalds once described it as "herding cats". I think Dmitry has done an excellent job of keeping the world's most talented database engineering team focused.

Vlad Horsun:
All did their best :) Why? Because it is pleasant for us ;)

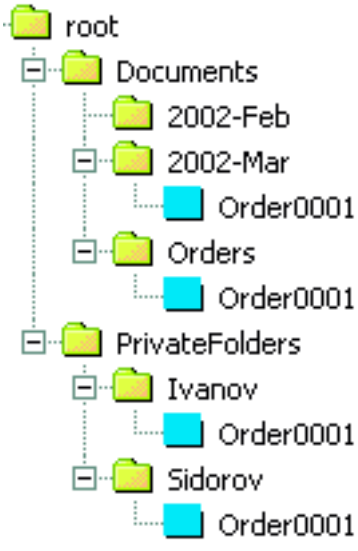
Dmitry Yemanov:
IMO, Vlad Horsun was the most active core developer during this year. Arno has done his best creating more efficient index implementation which eliminates a few major performance issues, he's also been excellent in the optimizer improvements, as always. And I highly appreciate efforts of Adriano dos Santos Fernandes in regard to the INTL issues.

Alex Peshkov:
Certainly Jim. One of the reasons is that he can develop the project without digressing to other things.

Link_type	Left	Right	Link Content
0	0	1000	The PrivateFolders folder is included in root
0	0	1001	The Documents folder is included in root
0	0	1007	The AnotherFolder folder is included in root
0	1000	1003	The Smith folder is included in PrivateFolders
0	1000	1006	The Roberts folder is included in PrivateFolders
0	1001	1004	The February2002 folder is included in Documents
0	1001	1005	The March2002 folder is included in Documents
0	1001	1002	The Orders folder is included in Documents
0	1003	2001	Order0001 is included in Smith's private folder
0	1006	2001	Order0001 is included in Roberts' private folder
0	1005	2001	Order0001 is included in the March2002 folder
0	1002	2001	Order0001 is included in the Orders folder

The content of any folder with a known OID is selected by the query:

```
select o.*
    from objects o, links l
    where l.link_type = 0 and
    o.OID = l.right and
    l.left = :folder_id
```



The whole tree-like structure looks as follows :

There is actually only one "Order0001" object in the "Objects" table, while there are several links in different folders referring to it.

A smarter simple directory, hierarchical

At the very beginning of the article, I promised to consider a method of building of a directory that would be more sophisticated than the simplest one. It can be got from OBJECTS by a query:

```
select OID, Name, Description
    from OBJECTS
    where ClassId = :LookupId
```

The result will be all objects of a certain type from the "OBJECTS" table. Slightly changing this query (considering that folder's ClassID equals 100, and the link type is link_type) "logical entering" equals 0, let us write

```
select o.OID, o.Name, o.Description
    from OBJECTS o
    inner join Links l on l.right = o.OID and
    l.link_type = 0
    l.left = :LookupRoot and
    where o.ClassId = :LookupClassId or
    o.ClassId = 100
```

This query will return the first level of the hierarchical directory, which begins with the OID LookupRoot folder and contains all the sub-objects (in the catalog hierarchy) with ClassId = LookupClassId. The query to open any first-level folder, will be similar, but LookupRoot should be substituted by the OID of that folder.

It is a matter for the computer to display the result of this query in the dialog window in a usual form (with folders opening by double-clicking and selectable objects inside these folders).

For example, this dialog may be exactly the same as a standard "Open file" system dialog. Given this dialog and a controlling run-time object (I call it DirectoryBrowser), we will get a universal user interface for most of the directories in the system.

Actually, this is true not only for the directories. In fact, we will get a universal API and user interface for browsing a catalog of any objects in the database, which is functionally similar to the Win dows Shell API (IShellFolder, IShellBrowser etc) bundled with Windows Explorer. You can also try to embed your mechanism into Windows Explorer, and make the objects accessible from anormal Explorer window. Without naming names, I know who tried to do it, and even succeeded in it!?

A new directory type will be created by two clicks, setting LookupRoot and LookupClassId for it (it is possible to use several "ClassId"s, if the directory can contain several types). For example, a contractor list may contain both suppliers and purchasers (which can be divided into other directories).

About how to create a directory, which requires an extra field (in addition to those stored in Objects) for sampling I will tell later, in the chapter about realization of access library organization.

Pathname in a Catalog

Since we can build a structure of catalog type in our database, it would be reasonable to use another "good old" abstraction: we can access the objects with the help of a pathname. For our example, the path to the Order0001 object (actually, it is one of several possible ways) can be written as /Documents/Orders/Order0001 or /PrivateFolders/Smith/Order0001. Indeed, in a catalog, the path to a certain object, which starts from a particular unit (root or something else), unambiguously identifies an object on condition that we provided for uniqueness of the object names within the catalog. The expression "unambiguously identify an object by a path" in our case means that, with the help of such a path, one can find the OID of the object. Indeed, there can be many paths to one and the same object (since you can have as many links to the object as you want); each path would point to only one object.

Uniqueness of the names within a catalog can be provided by imposing a constraint on the "Links" table with the help of a trigger that responds when link_type = 0.

We will not discuss here the well-known problem concerning the impossibility to fully support such restrictions with the help of a trigger, especially because we can avoid this problem in our case.?

There is no need to describe in detail the benefits of using of pathnames instead of OIDs (or, more precisely, together with them), since there are many of them and they are obvious.

Finding an object OID by a path can be realized easily in a stored database procedure, which would provide you with a possibility to use the pathnames in queries and other stored procedures and triggers, thus increasing database's logicality and cohesion. To realize such a procedure in Interbase, you would need to write a simple UDF for the path line analysis; for MS SQL, TransactSQL's capabilities would be enough.

Other types of links

Using the features provided by the "Links" table, you can build a lot of useful structures for joining database objects, in addition to the described global logical catalog. For example, you can solve a problem of a linked business process (BP), which is described at the end of the previous chapter. The object type defining a certain BP type (for example, the already-described long-term attendance), in this case, is a set of possible links between document types, which can be created in its context. The object, which describes a particular BP (attendance contract #10), contains a set of the existing links between the already created documents in the BP chain, and also is a central link during a visual presentation of the chain of documents and objects in the general network, i.e. parts of this BP. In this case we would need a new link type (link_type), with its own creation rules and restrictions, for example, link_type = 1.

And so on. Everyone can create their own types. I can suggest the following two: a type for class hierarchy presentation, and a type for account hierarchy from the accounting card and some other directories.

The Links table can be used for links of the "master-detail" type, if a "detail"set is an object list. For example, the table part of a bank extract is a set of pay-sheets. The advantage linking such links relationships via the Links table is that you would not need to realize the full function and field

sets (master_id, etc) thousands of times for each master-detail link. Instead, a standard, already realized and debugged relation method is used.

The access restriction system

Because calling any database object is accomplished via the "Objects" table, you also can realize an access restriction system at the document level (or at the record level), which is a "sweet dream" for many programmers who develop document-oriented databases. Indeed, this system will be independent from object types, i.e. when adding a new type, a user will not have to change the system.

Here I will briefly describe two methods: the first is simple and fast, the second is more complex and flexible. They do not guarantee you absolute security (they can be omitted at the SQL level), but under these circumstances a hacker would not benefit from it. Generally speaking, these methods provide an application developer with a measure of convenience: one can quickly and directly figure out whether a user has particular rights on a certain object by, for example, adding a simple BOOL function check_access(char* user, int object, int right) or something along those lines.

Thus:

1) A standard structure of users and groups support list is created:

```
create table users (
    UID integer not null primary key,
    name varchar(32) unique);
create table groups (
    UID integer not null primary key,
    name varchar(32) unique);

/* uniqueness of users' and groups' names is to your
taste */
create table users_groups(
    UID integer not null,
    GID integer not null,
    constraint users_groups_pk primary key (UID, GID));
create procedure uid_by_name(name varchar(32))
  returns (uid integer)
  as
  begin
     select uid from users where name = :name into :uid;
  end
```

Two fields ("read_group integer" and "write_group integer" are added to the "Objects" table. These fields are the links to the "groups" field.

A presentation is created

```
create view s_objects as
    select o.*
    from objects o, users_groups ug
    where (o.read_group = ug.gid and
    ug.uid = uid_by_name(CURRENT_USER));
grant select on s_objects to public;
```

"Objects" becomes inaccessible to everything but the "s_objects" presentation. This presentation provides a user with rights for reading all the objects whose read_group matches the one entered by the current user for this particular object. "Before insert," "before delete," and "before update" triggers are created for the "objects" table for checking access rights according to CURRENT_USER and write_group. The triggers' source text is trivial and is not included to the present article. You would also need to create "before update" triggers for each table containing extra object attributes. Those triggers will be absolutely identical. The "before update" trigger should do nothing but update the "objects.change_date" field, thus calling access rights checking by the "objects before update" trigger.

Happy New Year Wishes

Marina Novikova:
What could be done better (Is it possible to improve this in 2006)?

Helen Borrie:
I hope that Dmitry and the team will get a good, clear run for the final Fb 2.0 release and the subsequent Fb 2-Vulcan merger work. I really really hope that our international field-tester groups will become better-coordinated as there will be a lot of new things to test.

The Foundation has some funds in hand for grants to help one or two people who have some serious time to contribute to testing.

I hope we soon find some good, experienced testers to spearhead the QA effort.

Paul Ruizendaal:

Everything! We have a roadmap chockful of improvements to the code base, our "google footprint" still is not as large as that of Oracle, our documentation set could be better, we could have prebuilt binaries for more platforms, etc., etc.

Two big themes in 2006 should be documentation and applications. I am impressed by IBPhoenix' donation of their doc set to the Firebird Foundation and hope that preliminary docs for the community to work with and on will be available soon.

Packaged applications are important to further grow the user base. If Firebird gets strongly associated - perhaps even bundled - with exciting ready-to-use (open source) applications, our user base will grow and grow.

My personal opinion is that midmarket and corporate business applications (such as ERP, groupware, Web2.0, accounting, etc.) are the most important for Firebird.

2) The essence of the second variant is almost the same, but it is based on the well-known ACL algorithm. An ACL (access control list) enables you to specify a whole list of different access rights for different users and is applicable to many multi-user file systems. Thus, instead of the "read_group" and "write_group" field, an additional table is created:

```
create table ACL( id integer not null primary key,
   acl_id integer not null, /*acl number*/
   uid integer not null, /*link to a user*/
   right integer not null /*identifier of a certain access right */
   ))
```

The "acl_id integer not null" field is added to the "Objects" table as a link to acl.acl_id. The query

```
select a.uid, a.right
   from acl a, objects o
   where acl_id = o.acl_id and o.OID = :oid
```

will return the ACL for any of the objects from objects. The number of the lists themselves may be significantly lower than the number of the objects, due to the fact that several objects may share a single list, the way some file systems, such as NTFS, do. The access control itself is accomplished in almost the same way as in the first example, except that the "s_objects" presentation and access rights checking triggers become more complex and slower. I will not cite a definition of the s_objects presentation. Even though it is quite complex, it is problem-dependent, i.e. it depends on the access rights set specified in the particular system.

Advantages and disadvantages. The advantages of the first way are simplicity of realization and speed of the relational sampling with s_objects, since s_objects is a simple link, and its sampling is not significantly slower than sampling directly from objects. The disadvantage is the low flexibility (the features provided by this method are not always enough). The second method is much more flexible, but it is harder to realize. Moreover, using it in sampling with s_objects would possibly be complicated due to a slower read rights verification procedure. In conclusion, the common disadvantages of both methods are:

a) necessity to create triggers for every new additional attributes storage table

b) neither way guarantees reading security for the additional tables of object attributes if such a reading passes over s_objects. However, it is possible to take certain extra actions, so that reading of attributes tables would be of no use to malefactors. For example, you can store one or two essential items in objects, without which most of the objects would serve no purpose.

Bibliography

1. Mapping Objects to Relational Databases - White Paper.
Scott W. Ambler. 26-FEB-1999
http://www.AmbySoft.com/mappingObjects.pdf

2. "A Description of the Ultima-S System" by Vladimir Ivanov.
http://ivn73.tripod.com/ultima_overview.htm

3. The Design of a Robust Persistence Layer For Relational Databases -
Scott W. Ambler http://www.ambysoft.com/persistenceLayer.pdf. 28-NOV-2000

4. "Natural Keys vs. Artificial Keys". Anatoliy Tentser. July 20, 1999.
http://www.ibase.ru/devinfo/NaturalKeysVersusAtrificialKeysByTentser.html

5. A Database as Objects Storage. Anatoliy Tentser.
http://www.compress.ru/Article.asp?id=2006

6."Tree-Type (Hierarchical) Data Structures in Relational Databses",
Dmitriy Kuzmenko, iBase -
http://www.ibase.ru/devinfo/treedb.htm. and other articles about trees and objects at http://www.ibase.ru/develop.htm

Happy New Year Wishes

Vlad Horsun:

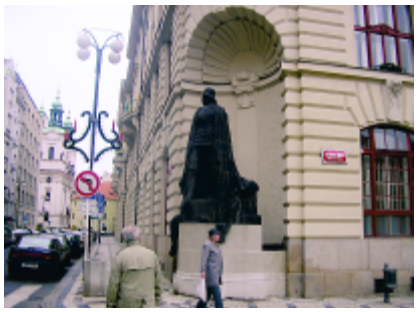
In my opinion, beta-testing might have begun half a year earlier. I hope we shall reduce a release-cycle next year. And the roadmap from DY confirms it :)

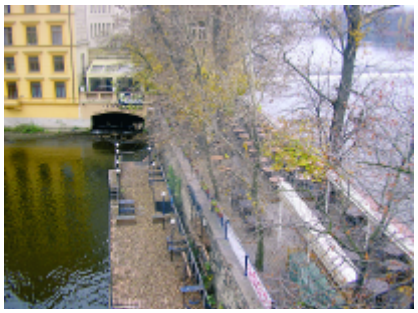
Dmitry Yemanov:
First, we could move faster. The v2.0 Alpha testing stage was performed not as active as it could be. Second, we need to learn making decisions in time. Too many things were discussed but an agreement has been defered till it's too late to do something in the current version.

These issues are hopefully to be improved over the next years.

Alex Peshkov:
I think Vulcan was divided for nothing. If firebird/Vulcan were not divided, we would have a single system of better quality.

This year's Firebird Conference has taken place at the Hotel Olsanka in Prague, Czech Republic.





Here you can find a set of nice photos made by Dmitry Kouzmenko, Serg Vostrikov and Mary Novikova.

Happy New Year Wishes

Marina Novikova:
If you remember the horoscopes, the symbol of year 2006 is a dog.

Helen Borrie:
Does that include all canines? :-)

Marina Novikova:
Will Firebird be a lucky dog? Why?

Helen Borrie: Let's hope it won't be a "dog" (slang term: something that functions badly and runs slowly, as in "It's a real DOG!" Why? I doubt that dogs have anything to do with luck! I remember the response given by an Australian entrepreneur in an interview some years ago. He was asked how important luck had been in his success. He replied, "Luck has been very important. And the harder I work, the luckier I get."

Paul Ruizendaal: I looked it up and my Chinese horoscope says: "People born in the Year of the Dog possess the best traits of human nature. They have a deep sense of loyalty, are honest, and inspire other people's confidence because they know how to keep secrets. But Dog People are somewhat selfish, terribly stubborn, and eccentric.

They care little for wealth, yet somehow always seem to have money. They can find fault with many things and are noted for their sharp tongues. Dog people make good leaders."

Yeah, I think it fits: Firebird is a lucky dog.

Vlad Horsun: Firebird will be a lucky Bird :) Because there are so many peoples loving it. And they love it in Dog's year and in all other years :)

Dmitry Yemanov: Not sure about dogs, as I don't trust the horoscopes, but Firebird was quite lucky and successful since its beginning. I don't expect anything to change here.

Happy New Year Wishes

Marina Novikova:
What was your most important contribution to Firebird in 2005?

Helen Borrie: Just being there, I guess, nothing extraordinary. Converting all of the Firebird 1.5 and Firebird 2.0 release notes to DocBook XML was a big job, worth doing. It is going to simplify translation and updating, besides making the contents of the release notes directly available for "lifting" into the manual-style documentation that the doc team is developing. The documentation task is enormous and the new sources will vastly lower the mountain top.

Paul Ruizendaal: Donating Ъ800 to the Firebird Foundation. I hope it will be more in the future.

Vlad Horsun: I like my implementation of GLOBAL TEMPORARY TABLEs though it has not come in this release :) Also I participated in implementation of External Engines support together with Eugeney Putilin and Roman Rokytskyy. From those features which have included in FB2, it would be desirable to note EXECUTE BLOCK, removal of the requirement of exclusive connections at time of FK creation and the advanced garbage collector. But year is not completed yet, so there will be new important changes :) And I do not know what from listed above is most important :)

Dmitry Yemanov: Did I really contribute anything? Are you sure? Ah, that roadmap thingy, you're right :-) Getting serious, I'm glad that a lot of bugs died with my direct assistance. There weren't many new features by me this year, although I consider my parser changes and optimizer improvements useful enough.

Alex Peshkov: Probably renewed DB security structure and support of this structure in the engine and gsec. Also the release of beta1 for Amd64. This 64-bit structure certainly has future.

The third worldwide Firebird Conference is in session.

Happy New Year Wishes

Marina Novikova: *If you were Santa Claus, what present would you give to Sparky and each member of the Team (any preferences)?*

Helen Borrie: Sparky's present is to have a long holiday and one doesn't need Santa to help with that. I'm still vacuuming red, yellow and orange feathers up from strange places in my house, where they were blown by cooling fans. I think Santa and his elves got it right when they built their toy factory at the North Pole. :-)

I hope each of the other team members gets the top items from his or her Christmas list. My top item? a winning Lotto ticket.

Paul Ruizendaal: Each member of the Team? That is ten's, perhaps even hundred's of people, the way I see it. That list is too long for this interview.

To Sparky? Perhaps a pair of sunglasses so that he can be a real cool dude during his global celebrity tour in 2006. Or should that be 'she' and 'dudette'?

Another nice gift would be airplane tickets for the family reunion, as Sparky consists of 4 brothers and sisters now -- scattered around the world. I hope Jason has enough spare room to host the party.

Vlad Horsun: To Dmitry - multicore multiprocessor 128bits notebook for development and entertainments. To Nickolay - a little bit "a smoke of fatherland" though I am not sure, that it is pleasant to him. To Jim - books "How to win friends", "How to work in the team" and a some valeriana. I would like to present something to Ann, Helen, Arno, Alex, Claudio and the rest, but mine fantasy is exhausted now :) I like all these people and very glad to work with them.

Dmitry Yemanov: The present is obvious: my live body at the next conference :-)

Faces
of
Firebird
Conference 2005 ...

Happy New Year Wishes

Marina Novikova:
How are you going to celebrate Christmas and New Year's Eve?

Helen Borrie: Christmas I shall spend with my husband and children in Auckland, New Zealand. After Christmas, I'm going down to my mother in the South Island (Christchurch). My father died in June, unexpectedly. His computer is much better than her old 486, so I'll be setting up Dad's machine for Mum to use.

For New Year, I'll be back at home but I have no plans. If it's a hot day on Dec. 31, I'll probably stay up all night with the garden hose ready, in case the local drunks do their usual thing and ignite fireworks in the eucalyptus forest that surrounds my house.

Paul Ruizendaal: 2005 has been a long hard slug for both my wife and myself. We will probably be in a very sunny place, doing absolutely nothing. Not quite sure where that will be, it seems to be rainy season in all the well know places. Perhaps it will be Zanzibar.

Vlad Horsun: I'll celebrate at home with my family. Last times I looked under a X-mas tree basically to put a gift, I do not think, that this time will be exception ;)

Dmitry Yemanov: Don't know yet. Usually I'm told about the upcoming New Year in the middle of December when it's too late to schedule something unusual :-) Most probably, I will celebrate it with my girlfriend and friends at home. Yeah, a small Xmas tree, candles, champagne and the stuff.

Alex Peshkov: I have no plans yet. Most likely I will celebrate it with my family.

Happy New Year Wishes

Marina Novikova:
What would/would not you like to see under the X-mas tree?

Helen Borrie: I expect our 24-year-old tabby cat, Peggotty will be under the tree, as usual. Not too many broken light-bulbs, I hope. A nice pile of books for me to read. Hint to Santa: "Thud" by Terry Pratchett?

Paul Ruizendaal: Would like to see: lot's of presents. Would not like to see: lot's of needles.

Dmitry Yemanov: Is Santa reading this? :-) Well, any presents are always welcome, I don't have any preferences. For me, it's more about kindness rather than about the material stuff.

Marina Novikova:
When will you start working on Firebird in 2006 (will you have long NY vacations or not)?

Helen Borrie: No NY vacations. In Australia, the time from Dec. 24 to January 2 is when life stands still while we eat, drink and be merry. A lot of people take annual vacations in January but, for me, it will be all over by NY and I'll be back to the desk.

Paul Ruizendaal: Probably January 1st. The best ideas pop up in my mind when doing absolute nothing and then I can't wait to get started. One thing that intrigues me is how to best implement materialized views for Firebird.

Vlad Horsun: I think January, 2-nd i'll not sustain and turn on my computer :)

Dmitry Yemanov: Since January, 3, as always. The second day is usually spent asleep in attempt to recover from the active celebration :-)

Alex Peshkov: I hope by January, 3 I will have come to consciousnesss after the NY and start working on the project.

... happy faces

Happy New Year Wishes

Marina Novikova:
That's the place for your NY wishes to the Firebird community: ...

Helen Borrie: I wish everyone a great time! In Russia, I think you have all of your holidays in January. Keep warm!

I wish everyone the best possible year in If Dmitry has his way, you will all be very busy.

Paul Ruizendaal: I wish all Firebird community members health and happiness for themselves and for their loved ones, onto the seventh generation at least (more if their OAT is ancient).

Vlad Horsun: I wish the community to grow, take pleasure from Firebird and new releases in new year!

Dmitry Yemanov: I wish everyone to live in peace, enjoy the life and be happy with Firebird.

JayBird 2.0, a JCA/JDBC driver for Firebird

by Roman Rokytskyy,
JayBird team
rrokytskyy@acm.org

JayBird was started by David Jencks in 2001 and Alejandro Alberola wrote a first implementation of the wire protocol. The goal of the project was to provide better JDBC driver than the InterClient that we inherited from Borland. It took us more than 2 years to complete the first release. Currently the release cycle is approximately 18 months but point releases are published more often, about once every three months. (There is no rule, except "fix quick, release often"). Today we have seen ~130,000 downloads since the project started and more than 1,300 members in Firebird-Java list.

Architecturally, JayBird consists of three layers: GDS, JCA and JDBC. The GDS layer is an abstraction of the physical interface to the server and represents the ibase.h file translated to Java. The JCA layer is responsible primarily for transaction management, including support for XA transactions. The JDBC layer implements the JDBC 3.0 specification. Two additional components, a connection pooling framework and database management were added later to expose Firebird-specific features and to provide efficient resource management classes in Java.

We have two main GDS implementations: pure Java and a JNI-based one. The pure Java implementation connects to the Firebird server via a TCP socket and talks Firebird's wire protocol directly. This is the fastest way to connect to the remote server from Java. The JNI-based implementation was created primarily to support local IPC connections (~30-40% faster compared to TCP local loopback, on AS3AP benchmark) and Firebird Embedded (up to 2x speed-up on the same benchmark).

Two additional GDS implementations exist, one supporting Oracle-mode Firebird via its API library and one supporting the Vulcan client library, but they are not part of JayBird distribution.

JayBird 2.0 introduced full support of updatable result sets, which allows seamless integration with GUI applications that allow in-place editing as well as integration with the javax.sql.RowSet implementations from Sun Microsystems and Oracle. Another feature added in JayBird 2.0 is full support of the Services API, which allows invoking backup and restore procedures from within the Java VM, performing user management and database maintenance tasks and collecting database statistics.

The main changes in new version are not visible, however. JayBird 2.0 features a new pluggable architecture with runtime plug-in discovery, and other refactorings that were made to increase code maintainability and stability in all places of the driver.

What's coming next?

Gabriel Reid has already working version of event support for Firebird, which did not make it into JayBird 2.0 due to a release schedule. It will be committed to CVS in next days/weeks and we will start releasing JayBird 2.1. We also have a pending change to support multiple client libraries on the JNI level, contributed by Evgeny Putilin and Vlad Horsun, which has been already included in Oracle-mode Firebird, but has not yet been committed to JayBird due to release schedule.

Ludovic Orban is helping us to achieve better XA specification support. JayBird 2.2 will target XA optimization to reduce the number of transactions needed when multiple transaction branches of the same global transaction are executed against the same resource. Depending on the transaction coordinator involved, the expected gain is one database page flush on commit instead of five. Also we plan to add read-only optimization that will use one-phase commit if no changes were made to the database within the specified transaction branch.

Steven Jardine is pursuing a goal to improve the code layout and build process by migrating from Ant build scripts to Maven 2.0 project management. We hope to finish this task before the 2.1 release.

Upcoming versions of JayBird will feature support for the INSERT... RETURNS statements introduced in Firebird 2.0 and we hope, as well, to see Java Stored Procedures, which are currently supported in Oracle-mode Firebird, merged with the main Firebird tree.

How I started to work with MS SQL Server and ADO

By Vladimir Kotlyarevsky
vlad@contek.ru

MS SQL Server – what are you talking about?!

The title of this article might seem surprising for an IB developer magazine. However, with six years' InterBase experience before I started with MS SQL Server in 2002, I thought an account of my thoughts and experiences in moving from IB to MS SQL might make an interesting read. I will try to describe the features of MS SQL that were strange for me as an IB programmer, features that appeared to be very suitable and some that I found to be awful.

Of course, I understand that any direct comparison of "IB vs. MS SQL" is a somewhat dangerous topic for an author since he risks being beaten by the fans from both sides. Please keep in mind that this paper is just an experiential account by one application programmer, and does not pretend to be either a complete description or a methodologically correct comparison.

My account encompasses not just the actual servers but also development tools, data access APIs and techniques.

Many important features of SQL Server are left beyond this article because I have not used them and cannot say much about them (full text search, replication, Data Transformation Services, Analysis Services, etc.). Many things in ADO that have changed in ADO.NET are not covered. This article is not to be taken as a complete SQL Server guide ? but merely as "one developer's perspective".

When I use the mnemonic "IB" in this story, I usually mean InterBase up to version 7.x (sometimes it is Firebird 1.0, but it will be highlighted). SQL Server stands for MS SQL Server 2000.

How could this happen? Background

In 2002, in my work with the Contek-Soft software company, I began trying to develop a kind of three-tier framework or "techniques toolbox" for our future multi-user applications. I called it "Kelly" – don't ask me why, I just liked this word ?.

The initial Kelly attempts were done using IB 6.x as the RDBMS and an application server (middle-tier) written in Delphi 7. It ran as COM+ application, had COM API for remote client applications, and used FIBPlus for data access. It was a tried and tested approach for me, but it had few important disadvantages.

I could not use COM+ transaction management, I could not pass datasets into the VBScript ActiveScripting engine, which I planned to use in both application server and client, and I had troubles with passing datasets from application server to clients. I also had to implement my own database connection pool for the application server, not a very complicated task, but annoying.

All of the MS how-to samples about COM+ and ActiveScripting used MS ADO and OLE DB for data access.

For those who work with Borland products rather than with MS ones: ADO and OLE DB drivers are somewhat similar, respectively, to the Borland BDE plus the VCL data access components and SQL links. ADO is a very simple, high-level data access interface for application programming, mostly for use with VB.

You use the same ADO components for any data source, be it ORACLE, MS SQL or dbf. OLE DB is s rather low-level, complex but powerful COM API, intended for C++ developers. Every RDBMS requires its own OLE DB driver. ADO uses OLE DB drivers to access data on different RDBMSes, but the application programmer does not need to know how it does this. One just has to specify the data source type in the connection string.

I knew that an implementation of the OLE DB driver for IB already existed – IBProvider http://www.ibprovider.com so I decided to check what ADO was. I found that:

1. ADO was well supported by COM+.

2. All ADO objects were implemented as ActiveX components. This meant I could use them in Delphi, any ActiveScript, MS Office VBA (which was very nice too) and any other language supporting COM and ActiveX.

3. ADO's Recordset object had well-implemented serialization, which would allow me to pass data sets over the network easily.

I found that, though it was still in in the beta stage, IBProvider had already become a product with a quality which was (and still is) rare in our times. Congratulations and respect to its authors.

So the next iteration of Kelly used IBProvider and ADO. And it really worked nicely!

I was absolutely happy when I had written my first test in VBScript for the application server API. The API under test returned an ADO recordset. I could call it remotely just from VBScript and process the result recordset in VBScript or Excel VBA without any problems.

COM+ native connection pooling just worked: I need to do nothing more with it. The COM+ transaction coordinator (MS DTC) worked too, so I no longer needed to be concerned with managing databases transactions and connections on the application server. Anyone who has implemented these things in a three-tier environment would understand my happiness ?.

Enter MS SQL Server

At this point, our company had secured a large contract for accounting software development. We decided to implement it using our new three-tier Kelly techniques.

One of the customer requirements was to use MS SQL Server 2000 as the RDBMS. It was just their corporate standard, they had licenses, support and trained administrators, and were not going to change that.

I decided to install and take a first look at the Developer version of MS SQL Server 2000 from our MSDN subscription. The Developer version is the full one with the same feature list as the

Enterprise version, and it's free for development purposes only, not for commercial use.

Installation was quick and easy, just few simple questions in wizard screens that I didn't have to think much about. It seemed to me not much more complex than installing InterBase or Firebird.

Tools

Enterprise Manager

After installing, I made a quick tour of the tools from the SQL Server set.

SQL Server Enterprise Manager (EM) was a strange, heavy and slow application, mostly for database administrators. The DBA can use it to start and stop the server, manage server parameters, backup/restore databases, attach/detach databases, shrink databases, manage security, and to manage and perform replication and other administrative tasks. You can edit stored procedures there, but only in a small modal dialog. God knows why ?.

The closest thing in IB to compare with EM is InterBase Server Manager. My summary assessment is that developers would not usually need this tool. Oh, no – there is one single function in EM that I use sometimes – "extract db metadata", which creates an SQL script, just like in IB. EM would be useful for newbie SQL Server DBAs, but an experienced DBA doesn't usually need any visual tool, since he knows SQL commands. In SQL Server, anything a DBA has to do can be done using Transact-SQL.

Query Analyzer

Query Analyzer is quite a nice tool for the developer. In short, it is just a multi-window MDI-style isql, which also has database metadata tree pane. This is the main and SQL tool for SQL Server developer and almost the only one. For comparison with IB, QA is something like IBConsole. But for IB we also have well-known excellent IBExpert (http://www.ibexpert.com/), which has similar look, though is much superior.

Concerning third-party tools for SQL Server, incidentally, as far as I know there are only a few of them, much fewer than for IB.

SQL Profiler

SQL Profiler is a wonderful tool! It is what I always dreamed about when working with IB. Its function is to show a full SQL trace for a selected SQL Server instance through a server-side interface. Because of this server-side interface it can show queries from all working clients. It seems to be incredibly useful and suitable tool.

IB has no comparable tools. Well, there are FIB+ and IBX SQLMonitor, but they are built into client libraries and hence work only from the client side. only, tracing queries just from this client, and requiring special client builds. The SQL Server Profiler traces queries from all clients, yours or third party, independently of what data access library is used. You can use it not only to develop and debug your own applications but also to see how some interesting third-party software works with its database.

Index Tuning Wizard

This tool makes suggestions about how to changing database index structures based on query statistics from running applications. My own humble opinion is – "just throw it away". Its advice is useless for experienced developers and will damage the brains of newbies ?.

Documentation

The SQL Server online documentation set is called Books Online. It consists of a number of "chm" (html help) files, through a single header. Documentation is of quite high quality, full of samples and hyperlinks. It is comfortable to use, both as online help and as a guide to study offline. The IB Langref, DataDefinitionGuide, etc. are really good books, often more interesting to read than fiction literature. As books, by comparison, I consider them superior to Books Online. They are not very comfortable for use as an online reference, though. The format and content organiization of the IB documentation are its main disadvantage, and I don't regard it as very important.

Transact-SQL

Batches, SET and others

The SQL Server SQL dialect is called Transact-SQL, or just T-SQL.

The first surprise for an IB programmer is a Transact-SQL command, called batch, may contain many statements, much like a stored procedure. It can contain any variety of different statements, including mixed DDL and DML. It can even contain several different SELECT statements, each capable of returning results to a client. This feature is often very handy when you need to execute a complex statement and don't want to create a SP or when you need to retrieve some complex object containing several data sets. I must say that, in comparison with IB, this feature is a pleasant one.

Editor's note: Firebird 2.0 introduces very similar feature namely EXECUTE BLOCK

The second surprise: an assignment statement in T-SQL requires the SET keyword, just like in old Basic ? - example "SET @TMP = 10". Furthermore, any variable in T-SQL has to begin with the character @. Example – DECLARE @TMP INT. I don't know why, but I suspect Microsoft is just too lazy to rewrite the language parser ?. It's an ugly feature, I think – but it doesn't hurt.

Third surprise – you can write something like

```
SELECT 'test'
```

without any FROM clause! – and client will get a recordset with one record, containing "test" ?. Or you can write

```
SELECT @a = 10, @b = 'test'
```

– and the statement will just assign '10' to @a variable and "test" to @b variable, without returning anything to client. Looks very strange for IB developer, does

not conform to ANSI SQL standards, but – why not if it doesn't hurt?

Auto incrementation

There are no objects or functions like the IB "GENERATOR". Creating an autoincrement field in the table is easy: in the "CREATE TABLE" statement you just define something like "ID INT IDENTITY(1,1)", where IDENTITY(1,1) means that server will automatically insert incrementing integer values in this field, with seed 1 and step 1. Seems to be simpler than creating generator and trigger in IB, doesn't it?

However, if you need even slightly smarter behavior, this simplicity becomes a problem. For example, once I needed to create a field which should be incremented by every second record, because in my DB structure, the "accounting entry" entity was stored as two records (debit and credit) in one table and had to have the same "entry id" attribute. With the IB "GENERATOR" implementation it would be trivial just to call "GEN_ID" to generate the next id and then insert two records with the same entry id. It is impossible to use the SQL Server IDENTITY function to implement such behaviour. I had to implement my own T-SQL function that worded similarly to the IB "GEN_ID", but with serious limitations.

My opinion - the IB "GENERATOR" is very useful feature with a wide functionality that is absent in SQL Server.

Select form select

In 2002 I was very glad to see that T-SQL supported SELECT FROM SELECT, but now the latest Firebird versions support this, too (Editor's note: Derived tables in Firebird 2.0). Also T-SQL supports UPDATE FROM – a useful operation that allows you to update one table with the data from another. It is useful for complex batch updates; I used it, for example, in a data warehouse project for batch updating of fact tables.

Stored procedures

A stored procedure in T-SQL cannot be used in a SELECT statement. The only thing you can do with a SP is to EXEC it. This does not mean that SPs cannot return data sets – they can, and more, they can return several different data sets. But they are returned in such way that they cannot be used in SELECT, they are just immediately passed to calling client. The only way to return a data set from a SP in T-SQL batch is to us temporary tables – see below.

Since a SP in T-SQL cannot be used in expressions and queries, T-SQL has a thing known as a "function" that is somewhat like a SP, but returns a single value and can be used in any expression. There is also the "table function", a function that returns a data set, or a "table", in an in-memory table variable. These functions can be used in SELECT FROM clause. The main limitation of functions is that they cannot change data in the database or change its state in any way. That is to say, you can employ a function to do SELECTs and any calculations, i.e., retrieve data, but you cannot do an UPDATE, an INSERT or even an EXECUTE PROCEDURE.

Instead of beautiful InterBase FOR SELECT statement in T-SQL you must use more complex CURSOR processing. The usage is "more complex" in that you must declare a cursor, fetch the first record, do a fetch loop, close the cursor, deallocate the cursor – many lines of ugly code instead of the simple FOR SELECT. However T-SQL CURSOR is a bit more flexible which can be useful in some rare special cases. (Editor note: IB and Firebird also have a form of explicit cursor syntax in PSQL, never documented by Borland, which makes the FOR UPDATE OF CURRENT syntax available for very fast execution of DML on single or multiple tables. By comparison with the clumsiness of cursor processing in T-SQL, it is supremely elegant. This pearl of performance has been greatly enhanced in Firebird 2.0.)

Temporary tables

Temporary tables are a nice feature, if you use them properly. A local temporary table can be created in the scope of the connection and in the scope of a SP. This means the temporary table is visible in that scope and is automatically dropped when it goes out of scope (the connection is closed or the SP finishes execution, respectively).

Temporary tables are created with the same CREATE TABLE statement as usual

tables, except the name of temporary table must start with #. Temporary tables are useful when you need to simplify and/or optimize a complex query, dividing it into two or more simple and fast queries that use result sets returned by previous queries. They can be used also when you want to return a data set to the current batch from a SP. To illustrate this, you can create a temporary table with necessary structure, then call a SP that fills that table, then process the data in the table as required and drop the table when done – all in a single batch.

Temporary tables are stored in a special, separate database (named tempdb) so they don't affect the size and structure of the working database.

My conclusion on temporary tables is that it is useful feature, if used with caution. "Caution" here means that you use them only when there is no other way, since creating the table and filling it with data take significant time. I'd wish this feature existed in IB, not for returning datasets from SP – IB have better solution (SUSPEND) for that – but for complex query optimization.

In T-SQL there are also global temporary tables – the same as the local ones just described except that, once created, they are visible to all connections. I have never used them and have no experience or idea on how to use them. Some dynamic data exchange between two different connections may be?

Error handling

An awful thing is that there is no structured error handling in T-SQL. The InterBase "WHEN .. DO" statement might appear to have some disadvantages, but IB programmers just don't understand their luck! Whatever its warts, it is much better than nothing at all. The only thing you can do in T-SQL is to check the value of a return error code by way of the @@ERROR system function. Moreover, if some error is raised somewhere in a SP or a trigger (e.g. primary key violation) it doesn't cause code execution to be stopped (except for very serious , usually system-level, errors)! If you need to stop it you must watch for @@ERROR values and stop execution by hand.

Transactions in SP and triggers

In T-SQL you can manage transactions from inside your SP or trigger code. That is to say, you can issue "BEGIN TRANSACTION", "COMMIT TRANSACTION" or "ROLLBACK TRANSACTION" anywhere. for my part, I have never used this feature, perhaps because of my previous experience and habits with IB.

T-SQL also claims to support nested transactions. It looks like very nice feature until you find that it does not work as one would expect. Here are few quotations from SQL Server Books Online:

1. "Committing inner transactions is ignored by Microsoft® SQL Server™."

2. "It is not legal for the transaction_name parameter of a ROLLBACK TRANSACTION statement to refer to the inner transactions of a set of named nested transactions. transaction_name can refer only to the transaction name of the outermost transaction."

3. "If a ROLLBACK WORK or ROLLBACK TRANSACTION statement without a transaction_name parameter is executed at any level of a set of nested transaction, it rolls back all the nested transactions, including the outermost transaction."

This just means that T-SQL nested transactions are not nested transactions ? - at least in the sense you might expect.

Triggers

Triggers in T-SQL can be only AFTER and INSTEAD OF. And this does not mean AFTER (or INSTEAD OF) the row insert, update or delete, but after the statement completes. In other words, unlike IB triggers , T-SQL triggers are statement level triggers, not record level.

Furthermore, in the statement level trigger you deal with a set of changed (inserted, deleted) records, so there are no NEW and OLD context variables available in T-SQL triggers. Instead there are virtual tables called INSERTED and DELETED. The INSERTED table contains new records (inserted records as well as the new versions of updated ones), the DELETED table contains old records (deleted and old versions of updated).

What do I think about all this? Well, in some special conditions, if used properly and coded carefully, T-SQL set-oriented triggers can work faster than the record-oriented triggers in IB. What are these conditions? An example is where your software frequently makes updates or inserts that affect multiple records, say, hundreds or more. I refer to such conditions as "special" because I think that OLTP usually deals with single records, and when you need to do some massive batch update you would usually write a special procedure for it, switching triggers off before running the update and turning them back on afterwards. In my view, an OLTP solution that does frequent massive batch updates as a part of normal processing is bad practice. For OLAP or similar solutions that require massive batch updates, it is much easier and more common to create special procedures with no triggers at all and a minimum of constraints, to increase update speed.

At the same time T-SQL set-oriented triggers are much more difficult to write and their source code is usually anything but obvious.

My conclusion about triggers is that IB triggers are more developer-friendly and their usability is better than in MS SQL Server. As for speed, in common conditions the speed is the same.

Functions

T-SQL has very wide set of ready to use functions, in contrast to IB, where it is supposed that the developer can implement anything he needs via UDF. String, date and time, mathematical, security, metadata functions, even system configuration functions, all make your life easier, so you usually don't need to write UDFs. If you still need some function that does not exist in the standard library you usually can write this function using T-SQL itself. For example, while in IB I used my own UDF for special string parsing, in MS SQL Server I could write this function in T-SQL.

SQL Server also comes with a lot of system stored procedures, mostly for administrative needs. Using these SPs, a DBA can configure most server and database parameters, replication, see different DB statistics and current state, etc. There are even a few procedures for sending email from a T-SQL batch, calling OLE automation objects and XML processing. It's my opinion that email and OLE functions are obviously redundant features in SQL Server but – why not if does not hurt? ?

In T-SQL you can deal with different databases within one batch. One uses so-called "fully qualified names" in the form database.owner.object_name. For example, you can issue a SELECT statement that joins tables from different databases. With the "Linked Server" feature (see below) or the OpenDataSource built-in function, you can even do heterogeneous queries with tables from different SQL Server instances on different hosts or even from a different RDBMS – SQL Server and IB, or Oracle, or dbf, for example. From my point of view it is a very useful feature when you need to integrate different data sources.

Almost any administrative task in MS SQL Server can be done using T-SQL. In my opinion, this makes it so much more comfortable and "natural" than the IB Services API.

Overall impression about T-SQL: it seems more powerful and flexible than IB SQL, thanks to its built-in function library and its ability to mix DDL and DML. At the same time it looks more "dirty" because of its SET, @ and some other atavisms. The IB SQL dialect presents as more concise and clear, though less flexible.

Miscellaneous server features

The feature of MS SQL Server that I like most of all and that is absent in IB is heterogeneous (distributed) queries through the Linked Server or the OPENDATASOURCE built-in function. Linked Server is any OLE DB or ODBC data source, that is preconfigured (connection string specified) and registered in SQL Server instance by DBA. After a Linked Server is registered, its tables are available to T-SQL queries. The T-SQL query optimizer can even use some schema information presented by the Linked Server OLE DB driver to optimize heterogeneous joins.

Functionality similar to Linked Server

can be obtained using the OPENDATASOURCE and OPENROWSET built-in functions, without registering a Linked Server. These functions can be useful for data sources that are used infrequently.

As for me – I often use heterogeneous queries in T-SQL to load data from external data sources. For example, recently we needed to load few hundred thousand records from several Excel files. Direct data loading using Excel VBA worked very slowly, so we tried a T-SQL INSERT FROM SELECT statement using OPENDATASOURCE through MS Jet OLE DB 4 driver for the FROM clause and found that it worked about 100 times faster. Using this INSERT FROM SELECT, we even could filter out some unnecessary records by joining the Excel "table" with an SQL table to plug in the filter criteria.

The feature of MS SQL Server that I hate most of all is the locking. It locks data whenever you read or write. It has no record versioning engine like IB. Almost all RDBMSes behave this way, but happy IB developers don't even think about how bad their lives could be if IB behaved like that. It is a really bad, very annoying feature that must be kept in mind from the very beginning when you design and write your application.

Example – if a client runs some heavy query, say an annual report that takes a minute or two to generate, nobody can write to the pages where the report data is located until the transaction in which the report is running has completed.

How do people work around this? I was very interested in that, too. One time, I ran the SQL Profiler and watched how one well-known, modern and respected accounting software running on MS SQL Server generated its reports. Easy – it just executed queries in dirty read mode. I think you understand well what that means. GIGO – "garbage in – garbage out". Usually everything is OK, but one day you find that Total line in your report or document that is not equal to sum of its lines…

The next version of MS SQL Server – YUKON – is promised to have a record versioning engine. If it does – fine. But IB has had this engine for more than 20 years. I would be surprised if, at least for the next 2-3 years, MS is not fixing bugs every day in YUKON versioning. So IB is still far ahead.

Query execution speed

I don't present any test results here since my tests were not standard and I didn't propose to publish them. The following is my private opinion based on my own assessment, which I am not going to argue about. My estimation is that, starting from millions of records in main tables, MS SQL Server is faster by two or three times on simple queries, especially with GROUP BY and aggregations. This is an average and very rough estimation. In some conditions, IB ran faster.

Optimizer and complex queries

Again, no test results, for the same as before. Query optimizer quality is a very difficult thing to assess and compare at all. In my opinion, SQL Server optimizer is "smarter", since it usually makes a rather good plan for complex queries which IB optimizer cannot manage with. But sometimes (not often) this "smartness" fails and it generates terrible plans for quite simple queries. I don't know why since I don't know how it works internally. As far as I know, RDBMS query optimizer logic is still closer to art than to engineering…

If you meet this situation in SQL Server, you can prompt the optimizer for a better plan using query hints right in the query text – how to make join, how to order, etc. For example, for a join you can prompt it to use one of LOOP | HASH | MERGE | REMOTE as the join method. There is no opportunity to give the full execution plan to the SQL Server optimizer like in IB. Usually, though, query hints are enough and work fine.

The SQL Server optimizer uses histogram index statistics, not just the single scalar value as IB does. It is likely that this helps it to make better decisions about the execution plan in some cases. However, if some table has a clustered index (another thing that is absent in IB), the optimizer will use it without paying attention to other indexes.

In SQL Server you can split your database into several files, as you can in IB. But what you can do in SQL Server (but not in IB) is specify particular files for specific tables and indices. This is done just in CREATE TABLE statement. Sometimes it can be useful. For example, if you have one very fast but not large hard disk and another one that is large, but slow, you can place your main table(s) on the fast disk and other tables on the slow one.

Backup and restore

They just work. At least I've never heard about an "unrestorable backup" on SQL Server, but it's something that sometimes happens on IB. I'm not sure why, but I think it's because a SQL Server backup file is completely different from an IB one. Internally, it is just a compressed database, not metadata and data exported in generic format like in IB.

There is also "differential backup" in SQL Server – it creates not a complete database backup, but just the changes from some point. I haven't used this feature, but I think it must be useful for large databases, when you just have not enough space for, say, complete "six revolver" backups. (Editor note: Firebird 2.0 has an incremental backup facility. Its name is Nbackup.)

SQL Server also has a "shrink database" function to remove empty pages from database files. If you often do batch DELETEs, this function can decrease database size by up to two or three times. In IB you can achieve the same result with backup/restore.

SQL Server allows a client to cancel a long-running batch. It is a useful feature, not available in Firebird or older versions of IB.

SQL Server has Windows-based authentication along with its own internal autherntication, useful if your application works in a LAN with Windows Domain or Active Directory installed. It simplifies the lives of developers (they don't need to think much about login dialogs and authentication) and users (they don't need to enter a password to connect to SQL Server). The only person who would possibly be not too happy is Domain admin ?, because he will have a bit more work administering users and groups for your application.

Data Access

At the time of writing, the main data access library for SQL Server was MS ADO 2.5, working over the SQL Server OLE DB driver. ADO is built as a set of ActiveX components, allowing it to be used in any language supporting ActiveX (COM). In our applications we used it in Delphi, Office VBA, VBScript and JScript (MS ActiveScripting engines). In comparison with the Delphi IBExpress or or FIBPlus data access libraries, this is an obvious advantage of ADO.

In ADO, the data access and data representation components are more separated than in the VCL and, in my opinion, this makes it better. ADO has a Command component that executes SQL statements on the server and a Recordset component which represents data sets returned by the server. In our multi-tier application this pattern was more comfortable for us than IBDataSet or FIBDataSet, which incorporate data access and data representation logic in single object.

Next, ADO Recordset component can marshal itself by value transparently for COM applications. This means that if your application server has a COM interface with a method returning an ADO Recordset and you call this method from client application, local or remote, you can use usual method call semantics.

Here's a small code sample in Delphi:

Server:

```
TMyServerObject = class(TAutoObject, IMyServerObject)
function GetSomeData: Recordset;
end;

function TmyServerObject GetSomeData: Recordset;
begin
  Result := CreateOleObject('ADODB.Recordset');
  //fill recordset with data here
end;
```

Client:

```
Var
rs: Recordset;
serverObject: IMyServerObject;
begin
serverObject := CreateRemoteComObject('remote_host',
MyServerObjectGUID,) as IMyServerObject;
rs := serverObject.GetSomeData();
while not rs.Eof do begin
  //do anything here
  Rs.MoveNext();
end;
end;
```

If you need to run MS Excel (or any other OLE server) from your application and to pass it a recordset that it should process in VBA or just add data to its sheet, you do almost the same – just pass the ADO recordset as a parameter of the method call. Behind the scenes, it performs transparent marshalling – it serializes the data into a binary stream, sends the stream on to another process or to a remote host, then deserializes the stream back into a Recordset, all without you needing to be concerned about any of it. – but you don't need to care about all this. Thus, you can use the Recordset to implement the "Data Transfer Object" pattern. The VCL ClientDataSet has similar functionality but, of course, you can use it only in Delphi.

You can also construct an ADO Recordset and fill it with data by hand, without any database. You just define fields, open it, insert data rows and it works.

If you want to use VCL DataAware controls with an ADO Recordset returned by an application server (or received by some other means), you can use the VCL ADODataSet and its Recordset property: just assign your recordset to the ADODataSet.Recordset and it turns into usual VCL DataSet that you can attach to a VCL DataSource and any Tdatasource compatible data aware control.

As for speed, in general ADO works more slowly than IBExpress or FIBPlus on such typical tasks as looping through a dataset and doing something with its fields. How much slower depends on the actual task. The thing that seriously decreases speed for the ADO Recordset is that it gets and sets field values as OleVariant data type. In our code there are two often-used low-level library functions (copy one recordset to another) where, in order to optimize performance, we had to process data at the OLE DB level, without any OleVariant conversions. It works two or three times faster. Other code works normally with ADO Recordsets.

And the last words about ADO: since there is an excellent InterBase OLE DB driver (IBProvider, http://www.ibprovider.com), you can use ADO to access InterBase databases in just the same way SQL Server programmers do it with SQL Server.

SQL-DMO – SQL Server Data Manipulation Objects library is another thing that I would like to have in IB. SQL-DMO is COM library, which presents SQL Server database objects as a hierarchical set of collections of tables, views, SPs, roles, etc. A Table object in turn contains collections of fields, triggers, constraints, etc. That seems familiar, doesn't it? Right – this is just what most RDBMS administrative and development tools look like. So, if you want to create some administrative or development tool for SQL Server, you don't need to parse system tables as you do in IB: SQL-DMO does it for you.

In our applications we use SQL-DMO to create databases from our own XML schema and, even more importantly, to alter databases when versions of software and database schema are changed. Why is the latter important? Because you can create a database using a simple SQL script but, to alter an existing database, you must determine differences between old and new metadata. That's a rather complex task when using plain SQL scripts and a very simple one with SQL-DMO and structured schema files.

Unfortunately, unlike ADO, SQL-DMO does not work over OLE DB. It is a SQL Server specific library so, sadly, you cannot use it for IB ?.

Conclusion

Both IB and MS SQL Server are good and of sufficiently high quality to use in real industrial applications. Both have advantages and disadvantages, which you should keep in mind when choosing an RDBMS and designing your application.

For my part, I use the following very rough recommendations for the choice (if there are no external reasons to choose one and not the other):

1. Estimated database size 10Gb and more – probably SQL Server, due to its better performance in query processing on large tables.

2. Estimated database size is less than 10Gb – probably InterBase, due to its non-locking versioning engine and therefore simplified application design.

For 10Gb and larger databases you also have to pay serious attention to the type of application you need – OLTP or OLAP or a mixture. As I said earlier, SQL Server does not like mixed types. since writers block readers and vice versa (well you are also able to work in dirty read mode if you aren't afraid of GIGO). So if you are going to create some mixed type of application, you should consider using separate databases for OLTP and OLAP query processing or using full-blown OLAP software, such as MS Analysis Services (OLAP server), for example, which is included in the MS SQL Server Enterprise version.

Incidentally, you can use Analysis Services as an OLAP solution for IB, too.

The conclusion is very simple and far from being something new – the developer should not be a fanatic follower of a single RDBMS, but should be open to choosing the one that best fits his abilities and the customer's requirements and problems.

TPC based tests for InterBase & Firebird

By Alexey Kovyazin
test@ibdeveloper.com

What is the TPC?

I expect most database developers know what TPC is. On their site (www.tpc.org), they state "the TPC is a non-profit corporation founded to define transaction processing and database benchmarks and to disseminate objective, verifiable TPC performance data to the industry."

In plain language, these folks investigate the limits of database servers+hardware configurations and provide everyone with results. The results represent an independent and objective appraisal of a database's performance

TPC methodology

How do they estimate performance? Well, it is a very interesting question with a very interesting answer. If you ever think about assessing the performance of any computer system you'd probably agree we usually have grades like these: "Wow, it is fast!" and "Damn, it is too slow" (and several slightly different variations).

TPC's more sophisticated approach is related to business: they estimate the cost of business operations.

They established a set of standard business operations for this purpose - "transactions" (yes, TPC uses the term "transaction" in a business context – as equivalent to "a business operation", not to be confused with database transactions).

Measurement techniques include databases, SQL scripts and tools to create/populate the databases and run the SQL scripts. The idea is to count how many transactions can be performed on particular software+hardware configuration and then divide the count of transactions by the cost of the hardware and software.

Our TPC based tests

If you take a look at TPC results you will observe that only large companies participate in them. The reason is that full-scale TPC testing is very expensive. Moreover, its target is high-productivity systems for the world's largest companies, As an indication, the unit of measurement for OLTP test TPC-C is a "tpcm" –a million business transactions).

With Firebird and InterBase we tend to serve the low-end to middle segments of the database market, so it is reasonable to ask how TPC could benefit our databases.

First, TPC publishes all its test SQL scripts and the source code for the tools. They developed database structures and SQL scripts according to standards for the enterprise-level DBMS. Examining these databases and scripts reveals a well-known structure – customers, orders, orders lines and so on. Almost every database developer is familiar with such things.

So, we decided to get the TPC databases, scripts and tools, adapt them to InterBase and Firebird and see just what this is all about and how it might be useful.

TPC-R

The TPC Benchmark™R (TPC-R) is a decision support benchmark, but which allows additional optimizations based on advanced knowledge of the queries. It consists of a suite of business-oriented queries and concurrent data modifications

Now TPC-R is marked as obsolete at tpc.org, but we thought it is still looked pretty useful for our goals. TPC-R contains the kinds of "heavy queries" which are hard for optimizers.

We plan to use TPC-R to test optimizer workings and the performance of access methods in all new versions of InterBase and Firebird. The goal is to check that new versions will be at least not worse than the old.

Also TPC-R generates high load to a system and can be considered useful as a stability test.

In fact, the test based on TPC-R for InterBase or Firebird contains a database creation script, a tool for populating the test database and 22 scripts. We measure execution time of for these queries and consider the query plans generated for them.

TPC-R is already showing very interesting results which are available at http://ibdeveloper.com/tests/tpc-r/

The toolkit for TPC-R is also available where.

TPC-C

TPC-C is an on-line transaction processing benchmark. It is potentially much more useful because it enables us to test the new SMP capabilities in InterBase 7.x and the forthcoming Vulcan (and Firebird 3.0, of course).

It simulates the work of large number of clients inserting, updating and deleting records in several warehouses.

You can see some TPC-C based test results here: http://ibdeveloper.com/tests/tpc-c/

The toolkit for TPC-C is available here also available where.

The TPC based testing program

So, IBDeveloper magazine hereby announces the start of its TPC based test program. We will test each new release of InterBase and Firebird, publish the results of testing along with commentaries.

The toolkit for performing our tests are available for anyone interested in reproducing them. What's more, we are calling for testers – if you would like to help us by carrying out tests on your real-world hardware we will assist you and publish the results on our test results page.

Thanks

The initial efforts for porting TPC-R and TPC-C for InterBase and Firebird were down to Aleksey Karyakin, developer of mature ODBC-driver Gemini (www.ibdatabase.com) and one of the former Yaffil developers. Currently we (the IBDeveloper team and Aleksey Karyakin) are continuing our work on the tests.

SQLLY DEVELOPMENT

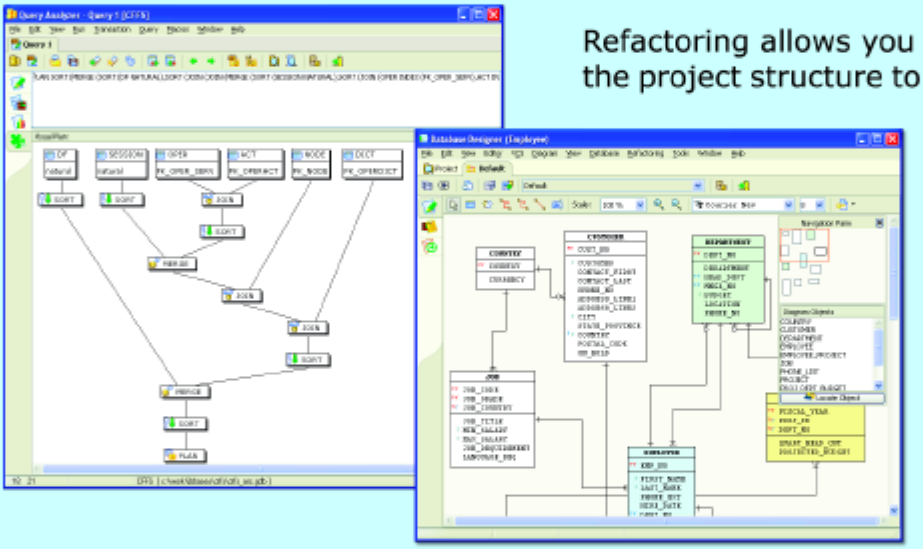
WWW.SQLLY.COM

INTERBASE/FIREBIRD DEVELOPMENT STUDIO

Interbase/Firebird Development Studio is a power-packed, fully-loaded solution that most database programmers and administrators can only dream about—at a price you can afford.

All stages of your database development, from design to deployment to maintenance, are easy, flexible, and powerful.

With Interbase/Firebird Development Studio, you get the most powerful tools for the first time ever—dynamic errors highlighting and refactoring of SQL code.



Refactoring allows you to quickly and easily change the project structure to accommodate new features, requirements, or external systems.

And, at last, you can now develop error-free code quickly and easily—when code errors highlight dynamically.

Our exclusive Database Designer presents the next step in the evolution of database development.

IBADMIN 4 FOR LINUX

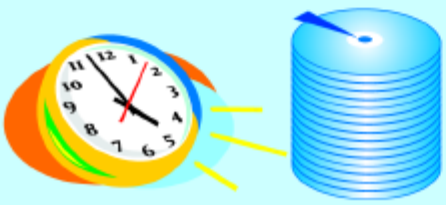
SuSE LINUX Inc.
THE LINUX EXPERTS

Simple but well packaged solution for managing Interbase/Firebird databases under Linux.

Aimed with deployment support features like Database Comparer, SQL Debugger and more.

Full support for Server Management is also available.

TIME TO BACKUP



"Time to backup" is a suite of programs that allow to set up a scheduled backup of Interbase or Firebird databases on multiple hosts.

Scheduler services can be managed remotely from Windows or Linux machine.

THE INTERBASE & FIREBIRD DEVELOPER MAGAZINE

www.ibdeveloper.com