15 Firebird Anti-patterns,

or

don’t do it!

Alexey Kovyazin, Firebird Foundation

15 Firebird antipatterns

1) Multiple parallel queries to MON$
2) Slow Dashboard Loading
3) Loading Unnecessary Records
4) Excessive Querying on Scroll
5) Unnecessary Automatic Refreshes
6) Frequent Record Updates
7) Using write transactions for read-only selects
8) Use LIKE :param
9) Not closing transactions for read-only operations
10) Query Parameterization Issues
11) Wrong Integrity Checking: triggers/CHECKs instead of Primary Key
12) ID generation with MAX()
13) Inefficient GUID Usage
14) Inefficient Computed Fields
15) Error Suppression Without Logging

1. Multiple parallel queries to MON$

- Very popular mistake — trigger OnConnect, query to MON$ATTACHMENTS to select user’s details for audit purposes, or calculate number of connection for licensing purposes.

1. Multiple parallel queries to MON$ — why is it bad?

- MON$ tables are virtual tables which are stored in fbNN_mon_xx system files, with performance statistics, etc
  - File >1Gb means you are using it too much
- They are designed for system administrators usage only — i.e., 1-2 parallel queries, exclusively for administrators
- 200+ connections with parallel queries to MON$ will slow down Firebird very significantly, and 500+ simultaneous queries will «hang» Firebird with high chances

1. Multiple parallel queries to MON$ — Solutions

- Don’t use MON$ for non-administrative tasks, i.e., to count or audit, avoid using them in OnConnect
  - For audit purposes
    - Use Context Variables like CURRENT_USER, CURRENT_TIMESTAMP, etc
      - https://firebirdsql.org/file/documentation/html/en/refdocs/fblangref50/firebird-50-language-reference.html#fblangref50-contextvars
    - Use Audit — native Firebird feature, much more powerful than triggers
  - For licensing purposes — use user’s context variables

2. Slow Dashboard Loading

- Loading comprehensive dashboards or scoreboards that sum all orders and invoices for the last month or year **during application startup**, or updating some metrics every minute or more often.

```
SELECT
  SUM(total_sales) as yearly_sales,
  COUNT(DISTINCT customers) as customer_count,
  AVG(order_value) as avg_order_value
FROM orders
WHERE order_date BETWEEN '2025-01-01' AND '2025-01-01';
```

2. Slow Dashboard Loading — why is it bad?

- Users must wait several seconds to see company-wide statistics before they can start their actual work, such as creating invoices or performing other specific tasks they're paid to do.
- From Firebird point of view — in order to constantly run many parallel queries, to retrieve the massive amounts of data, sort/group them, Firebird will intensively use multiple CPU cores, reading from disk, cache, memory dedicated for sorting (and sometimes sorting goes to disk).
  - It is like building report several times per minute!

2. Slow Dashboard Loading — Solutions

1) Decrease the number of users who will see dashboards:
    1) Usually Dashboard is required only for analysts and managements, exclude it from general application load
    2) Make loading of dashboard on startup/for some form optional, disabled by default
    3) Load dashboard data by explicit button click, not at startup (i.e., make it as a report)
2) Calculate dashboard data with 1 process by schedule (i.e., robot) and store them into the simple table ready to be retrieved by simple query (without necessity to read, sort/group large amounts of data)
3) Use triggers to aggregate data and store them ready for use
4) Use replica database to calculate dashboards data (and all heavy reports too)

3. Loading Unnecessary Records

Loading all data without filtering into the grid when opening an application or form, regardless of whether it contains hundreds of thousands of records.

```
procedure TDataForm.LoadAllRecords;
begin
  FDQuery1.SQL.Text := 'SELECT * FROM large_table';
  FDQuery1.Open;
  // Loads entire table into memory
  DBGrid1.DataSource.DataSet := FDQuery1;
end;
```

3. Loading Unnecessary Records — why is it bad?

- Despite the grid only showing 50 records, users must scroll through thousands of records instead of using search functionality.
- In 99% of cases users need the very narrow subset of data: the most recent sales records, for example.
- From Firebird point of view:
  - Every opening requires reading, storing in cache, and transfers of thousands of records through network
  - If you keep dataset open (in Delphi), Firebird keeps buffers, sorted records in temp space (if ORDER BY, GROUP BY, etc) until the closing of dataset

3. Loading Unnecessary Records — Solutions

1) Limit the number of records with FIRST/SKIP/ROWS

2) Limit the number of records with some criteria, for example, shows records created/changed during recent 3 days

3) Close queries as soon as possible

4. Excessive Querying on Scroll

- Executing queries on scroll events. For example, when displaying data in a grid or table, performing a separate query FOR EACH record (i.e., requesting Details in master-detail)

```
procedure TForm1.GridScrolled(Sender: TObject);
begin
  // query for each row
  FDQuery2.SQL.Text :=
    'SELECT additional_info FROM details ' +
    'WHERE id = ' + IntToStr(CurrentRowId);
  FDQuery2.Open;
end;
```

4. Excessive Querying on Scroll — why is it bad?

- Performing a separate query FOR EACH record in dynamic grid forces Firebird to process thousands of tiny queries, unnecessarily consuming CPU resources.
- From Firebird point of view
  - Many (thousands per second) small queries will create significant CPU load, because even if query shows 0ms in statistics, it requires to be prepared, executed, result to be transferred, etc

4. Excessive Querying on Scroll — Solutions

- Load multiple rows at once

```
procedure TForm1.LoadDetailsInBatch;
begin
  FDQuery2.SQL.Text :=
    'SELECT id, additional_info FROM details WHERE id IN (:ids)';
  FDQuery2.Params.ArraySize := GridVisibleRows.Count;
  // Batch load multiple rows at once
  for i := 0 to GridVisibleRows.Count - 1 do
    FDQuery2.Params.ParamByName('ids').AsIntegers[i] :=
      GridVisibleRows[i].ID;
  FDQuery2.Execute;
end;
```

4. Excessive Querying on Scroll — Solutions-2

1) Enhance the main query for grid to execute detailed query as part of it
2) Add explicit button to load details for visible part of grid
3) Add delay to execute the query to receive details, to prevent immediate queries during the scrolling
4) Don’t enable on-scroll loading of details for all users by default
5) Prepare queries

5. Unnecessary Automatic Refreshes

- Refreshing grid data automatically at minimal intervals in every client application, with this feature enabled by default.

5. Unnecessary Automatic Refreshes — why is it bad?

- This results in hundreds of client connections running almost identical queries to retrieve the same records.
- Where it happens: automatic refreshes for schedules, or select for queue positions, or search for «nearest slot», etc
- From Firebird point of view
  - Combination of loading dashboards and on-scroll events: many mid-size queries create load on the system

5. Unnecessary Automatic Refreshes — Solutions

1) Increase interval!

2) Implement explicit (user-triggered) refreshes

3) Use selective refresh of data set based on actual data changes (streaming or triggers or event+trigger)

6. Frequent Record Updates

- **Frequently updating the same record in different transactions, creating numerous record versions.**
- **How to check it Max Versions:**

```
gstat -r -t TableName -user SYSDBA -pass masterkey > stat1.txt
TRDT (161)
    Primary pointer page: 977, Index root page: 978
    Total formats: 1, used formats: 1
    Average record length: 22.50, total records: 4
    Average version length: 10.59, total versions: 444, max versions: 148
    Average fragment length: 0.00, total fragments: 0, max fragments: 0
    Average unpacked length: 58.00, compression ratio: 2.58
    Pointer pages: 1, data page slots: 1
    Data pages: 1, average fill: 76%
    Primary pages: 1, secondary pages: 0, swept pages: 0
    Empty pages: 0, full pages: 0
```

6. Frequent Record Updates — why is it bad?

- A record with dozens of versions can significantly degrade performance, a record with thousands can become a blocker.
- From Firebird point of view: record versions chain should be reconstructed to identify the proper version of the specific transaction, it requires numerous read operations
  - As a result, garbage collection becomes much slower

6. Frequent Record Updates — Solutions

1) Migrate to Firebird 4+, there is intermediate garbage collection

2) Don’t keep long running writeable transations, organize proper garbage collection

3) For Firebird <4, consider to use DELETE+INSERT instead of UPDATE

7. Using write transactions for read-only selects — why is it bad?

- Using write transactions for read-only selects leads to many unnecessary writes of header pages
- Using write transactions for read-only operations is inefficient (large TIP during commit creates additional load on server)

7. Using write transactions for read-only selects — Solutions

- Use separate read-only transaction for operations which do not change data
  - Firebird is one of a few databases which allow to open several transactions in the frames of single connection.
  - Global Temporary tables are available for use in read-only transactions

8. Use LIKE :param

- The following query with parameter will not use index for the fieldName (even if index exists):

  SELECT * FROM Table1 WHERE fieldName LIKE :p

8. Use LIKE :param — why is it bad?

- Since LIKE allows wildcard search (%), which can replace any number of symbols, Firebird cannot determine in advance, will the parameter value be suitable for index search, and disable index usage.
- Usually developers try to workaround it with explicit parameter value in the query text:
  - fieldName LIKE «Alex%» - possible to use index
  - fieldName LIKE «%Alex» - not possible to use standard index
  - fieldName LIKE «%Alex%» - not possible to use index at all
- It leads to other problems (see #10 below)

8. Use LIKE :param — solutions-1

1. Use STARTING WITH for known string prefixes

When your search value never starts with a wildcard %, prefer `STARTING WITH` over `LIKE`:

```
WHERE fieldName STARTING WITH :param1
```

2. Optimize bidirectional string searches

For strings with known prefix or suffix patterns, use reversed index:

```
-- Create reversed index
CREATE INDEX ixreverse1 ON TABLE1 COMPUTED BY (REVERSE(fieldName));
-- Query using both directions
WHERE fieldName STARTING WITH :param1
   OR reverse(fieldName) STARTING WITH reverse(:param2)
```

8. Use LIKE :param — solutions-2

3. Implement progressive search strategy

For strings that appear at start/end/middle (but not simultaneously):

1. First try fast indexed search with STARTING WITH
2. If no results found, fall back to slower LIKE search

4. Word-Based Search Optimization

When searching for complete words (delimited by spaces, commas, etc.):

- Create a separate word-ID mapping table
- Search through the mapping table instead of the original text

5. For comprehensive full-text search capabilities:

- Consider using IBSurgeon Full Text Search UDR
- This open-source solution provides advanced text search functionality

9. Not closing transactions for read-only operations — why is it bad?

- Keeping transactions open for extended periods could force Firebird to maintain numerous back versions for potential old snapshot transactions

9. Not closing transactions for read-only operations — Solutions

- Use read-only transactions where it is possible, and close writeable transactions as soon as possible
- Use modern Firebird versions (4+) to reduce impact of records versions chains
- Implement proper sweep

10. Query Parameterization Issues

Avoiding prepared queries and parameterization, instead embedding parameter values directly in query text.

```
FDQuery1.SQL.Text :=
  'SELECT * FROM users WHERE name = ''' +
  EditUsername.Text + '''';
FDQuery1.Open;
```

10. Query Parameterization Issues — why is it bad?

- This practice reduces performance for repeated queries
  - Every query with embedded parameters values should be prepared as new
  - Preparation can be long and time consuming for large tables
- Complicates problem analysis
  - It is difficult to group queries by text
- Creates SQL injection vulnerabilities

10. Query Parameterization Issues — Solutions

- Use parameters!

```
FDQuery1.SQL.Text :=
  'SELECT * FROM users WHERE name = :username';
FDQuery1.ParamByName('username').AsString :=
  EditUsername.Text;
FDQuery1.Open;
```

11. Wrong Integrity Checking: triggers/CHECKs instead of Primary Key

- Using triggers or CHECK instead of Primary Keys for database integrity checks.
  - Yes, it happens!

11. Wrong Integrity Checking: triggers/CHECKs instead of Primary Key — why is it bad?

- This ignores that Primary Key validation occurs within a special reading mode of current record versions (independent of the users' transaction isolation mode)
- Making PK checks with triggers in user transactions increases the possibility of duplication and unnecessarily complicates logic.
- Replication will not work without Primary or Unique Key for the table

11. Wrong Integrity Checking: triggers/CHECKs instead of Primary Key — solutions

- Use primary keys
- Avoid redundant integrity checks
- Keep database logic simple

12. ID generation with MAX()

- Using MAX(id)+1 for new identifiers is unreliable and inefficient.

```
INSERT INTO users (id, name)
VALUES ((SELECT MAX(id) + 1 FROM users),
  'John Doe');
```

12. ID generation with MAX() — why is it bad?

- Using MAX(id)+1 instead of sequences (generators) for new identifiers.
- MAX(id)+1 doesn’t guarantee uniqueness with common transaction parameters - two parallel transactions could receive the same MAX() value.
  - Combination of Max()+1 and CHECK(select if unique) also does not work!

12. ID generation with MAX() — solution

```
-- Use generator/sequence!
CREATE GENERATOR gen_user_id;


-- Use generator for ID generation
INSERT INTO users (id, name)
VALUES (
  GEN_ID(gen_user_id, 1),
  'John Doe' );
```

13. Inefficient GUID Usage — why is it bad?

- Using system-generated GUIDs instead of gen_uuid() can impact index performance
  - System-generated GUID is highly randomized

13. Inefficient GUID Usage

- Use gen_uuid() function
- Consider to use BIGINT instead
- In version 6 there will be UUID v7

14. Inefficient Computed Fields

- Using computed fields with SELECTs to other tables significantly decreases the performance of simple SELECT operations.

```
CREATE TABLE orders (
  id INTEGER,
  total_amount COMPUTED BY (
    (SELECT SUM(item_price) FROM order_items
     WHERE order_items.order_id = orders.id)));
```

14. Inefficient Computed Fields — why is it bad?

- Computed fields are calculated on the fly, and they are not supposed to implement complex logic, and can significantly complicate optimization efforts
- It strengthens relationships between tables
- It makes sense to use computed fields only for lightweight calculations with fields of the table, like concantenation

14. Inefficient Computed Fields - Solutions

```
CREATE TABLE orders (
  id INTEGER PRIMARY KEY,
  cached_total_amount DECIMAL(10,2));

CREATE TRIGGER update_order_total
BEFORE INSERT OR UPDATE ON orders
AS
BEGIN
  NEW.cached_total_amount = (
    SELECT SUM(item_price)
    FROM order_items
    WHERE order_items.order_id = NEW.id
  );
END;
```

15. Error Suppression Without Logging

- Don’t suppress Firebird errors and warning without logging!

```
try
  FDQuery1.Open;
except
  // Silent failure
end;
```

15. Error Suppression Without Logging — why is it bad?

- Hiding errors prevents proper diagnosis and debugging. Proper error logging is crucial for understanding and resolving issues quickly.

15. Error Suppression Without Logging — Solutions?

```
try
  FDQuery1.Open;
except
  on E: Exception do
  begin
    // Comprehensive logging
    Logger.Error('Database connection failed: ' + E.Message);
    ShowMessage('Unable to connect to database. Please contact support.');
    // Log additional context
    Logger.LogStackTrace(E);
  end;
end;
```

Thank you, and what next?

- Send your questions to ak@ib-aid.com
- Download text summary (link in the description of video)
- Become Firebird Supporter (from EUR10/month) and participate in closed advanced webinars!
  - https://store.firebirdsql.org/p/firebird-associate-donation-eur/