Effective development of Firebird database applications in Delphi/Lazarus

Dmitrii Kuzmenko, IBSurgeon
Firebird Conference 2019
Berlin, 17-19 October

IBPhoenix
YOUR PREMIER SOURCE OF FIREBIRD SUPPORT

IBSurgeon

MOSCOW EXCHANGE

Fast Reports
Reporting must be fast!

IB Expert

RED SOFT
List of topics

• Client Library
• Delphi, Lazarus
• Unicode
• Transactions
• Other
  – Reports
  – Stored Aggregates
  – Explicit locks
  – Data Editing
Client and server, 32 and 64 bit

Application 32-bit

fbclient.dll 32-bit

Server, 32-bit or 64-bit

TCP

Embedded

Application 64-bit

fbclient.dll 64-bit

Firebird Client or Embedded

One capacity

Application

Driver

Firebird Client or Embedded
Exe + dll

1.5-2.5

App

fbembed.dll
client + engine

DB

3.0

App

fbclient.dll

engine12.dll

DB
Embedded - Development and usage

• Development is uncomfortable
  – ServerMode=SuperClassic, not SuperServer
  – Better use normal Firebird server installation

• 32bit or 64bit – to your taste

• Usage – single-user applications. Not multi-user
  – SuperClassic mode allows to work with one database for several applications on the same computer
Delphi

- FireDAC
- IBX
- dbExpress
- IBOObjects
- FIBPlus – discontinued
- IBDAC, UNIDAC
- ...

- in the beginning of the 2000 there were around 40 different drivers and component sets
Delphi

- Delphi – 32bit IDE
- Can compile 32bit and 64bit applications
- so, you need to have 32bit and 64bit fbclient.dll installed (by instclient i f)
- instclient i g - may also be needed for IBX (gds32.dll)
Delphi - FireDAC

see Firebird documentation
Firebird 3.0 Developer’s Guide
Developing Firebird Applications in Delphi

FDTransaction.Params
dbExpress – handles?

- Cannot set transaction parameters
- TDBXTransaction exists, but useless
- Cannot switch between transactions

```cpp
transaction1.BeginTransaction;
... transaction2.BeginTransaction;
... here you can not return to transaction1 context, you can only call it’s commit/rollback.
```
Lazarus

• Components

• IBX2
  – contains FB 3 API wrapper

• ZeosDBO

• FBLib
Client libraries

- Lazarus – 64bit IDE
- all components need 64bit fbclient.dll
- use instreg i f
- or put fbclient to the lazarus and application folder
Lazarus - IBX
IBX
ZeosDBO

- No Transaction component
- if not IConnection.InTransaction then
  IConnection.BeginTransaction;
  if IConnection.InTransaction then
  IConnection.CommitTransaction;

- Default to BDE – autocommit, etc...
• `ZConnection.TransactIsolationLevel := tiNone;`
  `ZConnection.Properties.Add('isc\_tpb\_concurrency');`
  `ZConnection.Properties.Add('isc\_tpb\_wait');`
  `ZConnection.Properties.Add ('lc\_ctype=ISO8859\_1');`
  `ZConnection.Connect;`
Working with components

• Table component – only for reference tables
• “Live dataset” does not exist
  – Select, insert, update, delete, refreshRow

• Main problem – invalidation of a cursor after commit/rollback
  – you may use ClientDataSet (briefcase model)
Components summary

• Better use components with transaction control, with ability to set Firebird specific connection and transaction parameters, and with Services API support

• Universal components (including ODBC) – for universal (different SQL servers) development
Unicode

• Delphi up to 2007
  – non-unicode

• Delphi 2009 and higher
  – unicode, UTF-16, UTF-8

• Lazarus
  – unicode, UTF-8
Unicode

• database in country charset
  – ISO8859_1, WIN1252 – Western Europe
    • Danmark, Germany, Netherlands, UK, Spain, Finland, France, Iceland, Italy, Notway, Portugal, Brasil, Sweden
  – ISO8859_2, WIN1250 – Central Europe
    • Czech, Hungary, Poland
  – ISO8859_4 – Northern Europe
    • Estonia, Latvia, Lithuania, Greenland

• database in unicode (UTF8)
Charsets

• **Connection charset** – only one particular charset
• **Database charset** – each character or text blob column can have it’s own charset

! “database charset” is a default charset that will be used when new character/blob column is created, and no charset is specified. In this sense database has no “charset”.

```
 rdb$database.rdb$character_set_name
 rdb$character_sets.rdb$default_collate_name – from 2.5
```

• Example:
  if connection charset is ISO8859_1, and database have character columns of ISO8859_2, there must be “conversion table” from the column charset to connection charset
Is unicode is bigger?

**Test:**
X1251 varchar(30) character set win1251
XUTF8 varchar(30) character set UTF8
100k records.

Win1251 is a single-byte character set. So, 30 characters = 30 bytes
UTF8 is dynamic character set. Russian characters here occupy 2 bytes per each, latin characters are 1 byte per character.
So, let us fill data with the national (russian) characters only.

<table>
<thead>
<tr>
<th>Table</th>
<th>Records</th>
<th>RecLength</th>
<th>Data Pages</th>
<th>Size, mb</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1251</td>
<td>100000</td>
<td>28.86</td>
<td>852</td>
<td>6.66</td>
</tr>
<tr>
<td>XUTF8</td>
<td>100000</td>
<td>49.01</td>
<td>1094</td>
<td>8.55</td>
</tr>
</tbody>
</table>

UTF8 storage takes ~30% more than single-byte character set
To and back

• Connection win1251 – database win1251

• Connection utf8 – database win1251
  – only win1251 characters will pass

• Connection win1251 – database utf8
  – only win1251 characters will pass

• As a transition – first use UTF8 connection charset, next upgrade your database
How to upgrade to UTF8?

• Only by pumping data to the new UTF8 DB.
  – take script from the db (isql –x –ch nnn)
    • -ch is mandatory. otherwise you may get garbage instead of literals and comments
  – check any specification of “character set”
  – check any “collation” specification, decide what to specify
    – UTF8 (USC_BASIC), UNICODE, UNICODE_CI, UNICODE_CI_AI
  – check column sizes that are close to 32k bytes (32k/4=8k)
  – create database
  – pump data from original database to the new one
    (don’t worry to use non-unicode pump, just use same character set for both connections, data will be converted to unicode automatically)
Upper

• Old style
  – select * from table
    where upper(name) = ‘STRING’

• New style
  – declare column as
    name varchar(30) collate unicode_ci
  – select * from table
    where name = ‘string'
iOS, Android?

• Only UTF8 databases.
Working with transactions

- Worst example – MastApp. One transaction for everything CommitRetaining
- Do not use CommitRetaining (or RollbackRetaining)
- Use as many transactions components, as you need
- Do not use IBTransaction.Active:=True/False
  - use StartTransaction, Commit and Rollback methods. Active:=False is equal to Rollback (by default)
- Do not start transactions too often
- Avoid long running transactions
Transaction defaults

- IBX – write wait snapshot
- FIBPlus – write no wait read_committed rec_version
- FireDAC - write no wait read_committed
- Others - ? check by yourself
Main performance issues

• Transaction control
  – garbage, slowness

• Query optimization
  – bad SQL, no index, redundant index

• firebird.conf optimization

• Hardware optimization
Sequential transactions

Always test your application in the multi-user mode
Gstat -h

• Database header page information:
  • Flags 0
  • Checksum 12345
  • Generation 112431494
  • Page size 8192
  • ODS version 11.1
  • Oldest transaction 100 x-1
  • Oldest active 101 x
  • Oldest snapshot 101 x
  • Next transaction 102 x+1
  • Bumped transaction 1
  • Sequence number 0
  • Next attachment ID 0
  • Implementation ID 16
  • Shadow count 0
  • Page buffers 0
  • Next header page 0
  • Database dialect 1
  • Creation date Jun 5, 2011 10:02:19
  • Attributes force write

• Variable header data:
  • Sweep interval: 20000
  • *END*
Ideal transaction control
Two ways to almost ideal transaction control

1. Long read-only read-committed (until Firebird 4.0) and short write
2. Short read and write
Long read-only RC and short write

```sql
 INSERT INTO T1(i1) VALUES (100);
 SELECT i1 FROM T1
 WHERE i1 = 100;
 UPDATE T1 SET i1 = 200;
 SELECT i1 FROM T1
 WHERE i1 = 200;
```
Short read and write

- **Tx60**
  - Select * from T1

- **Tx65**
  - Update T1 set i1=200;

- **Tx67**
  - Insert into T1 values (200);

- Get records into app cache
- Update cache or re-read all records
- Update cache or re-read all records
Pro & Contra

Long read-only RC and short writes

+ easy to implement read and update logic
- requires support from drivers/components (2 transactions or 2 connections)
+ more convenient for client-server
- less convenient for multi-tier and stateless applications

Short read and writes

- hard to implement sophisticated caching
+ works with any data access drivers/components
- less convenient for client-server
+ more convenient for multi-tier and stateless applications
WHY NOTHING IS PERFECT
Exceptions from ideal transaction control

- Reports
- Product balances
- Explicit record locking
- Data editing
- Robots
Reports

• Need data consistency
• Long queries
• Complex reports read the same data several times
Reports - snapshot

No difference – wait/nowait (except concurrency), read/write
Heavy report example – OAT stuck
How to workaround long report problem

• Most reports does not need real-time data

• Change logic of data processing

• Scaling
  – Replication
  – Transferring data to another DB with Execute Statement On External
  – Nbackup
Change logic of data processing – Stored aggregates

- 1 order - ~10 goods
- 100 orders per day
- $100 \times 10 = 1000$ records per day
- 365000 records per year

- Store “order_total” in ORDERS table – 10 times less records
- **Pro:** less records, faster queries
  - No update conflicts if there is no concurrent order editing
- **Con:** additional field in ORDERS
If you want to go further...

• To store sum by day, month, ...
• Updates by triggers “in place” won’t work – too high possibility of lock conflicts
• Solution? Regular updates
  – Regular procedure must be run in exclusive mode
    • Using generator
    • Using consistency isolation mode
  – By schedule (at night)
Goods balances – update locks

• Change AMOUNT while order is processed

• Insert - set AMOUNT = AMOUNT - new.INORDER
• Delete - set AMOUNT = AMOUNT + new.INORDER
• Update - set AMOUNT = AMOUNT + new.INORDER – old.INORDER

• There may be conflicts when 2 people sell same good_id
  – Long transaction will lock all concurrent order processing
  – Short transactions have less chances to get update conflict, and may be retried
Update locks

- Item1
- Item2
- Item3

Order 1
- ok for the order, while only 1 user may edit 1 order

Order 2

Goods
- not ok for goods, since same item may be sold in 2 (or more) orders, that may be edited same time
ORDER 55 included good with id 1000

ORDER 34 edited good with id 1000

update GOODS
set amount=amount-x
where good_id = 1000

update GOODS
set amount=amount-x
where good_id = 1000

Editing one order by 2 users is a rare case, but using same item is not rare
Goods balance - solution

• CREATE TABLE MOVEMENTS (GOOD INTEGER NOT NULL REFERENCES GOODS, AMOUNT INTEGER NOT NULL)

• CREATE TABLE GOODS_AMOUNTS_AGG (GOOD INTEGER NOT NULL REFERENCES GOODS, AMOUNT INTEGER NOT NULL)

• On insert delete and update MOVEMENTS do
  • INSERT INTO GOODS_AMOUNT_AGG (GOOD, AMOUNT) VALUES
    – (NEW.GOOD, NEW.AMOUNT);
    – (NEW.GOOD, NEW.AMOUNT-OLD.AMOUNT);
    – (OLD.GOOD, -OLD.AMOUNT);
CREATE VIEW GOODS_AMOUNT (GOOD, AMOUNT) AS
SELECT GOOD, SUM(AMOUNT)
FROM GOODS_AMOUNT_AGG
GROUP BY GOOD

CREATE PROCEDURE GOODS_AMOUNT_ROLL_UP AS
DECLARE GOOD INTEGER;
DECLARE TOTAL INTEGER;
BEGIN
FOR SELECT GOOD, SUM(AMOUNT)
FROM GOODS_AMOUNT_AGG
GROUP BY GOOD
HAVING COUNT(*)>1 — interested of 2 or more records
INTO :GOOD, :TOTAL
DO
BEGIN
DELETE FROM GOODS_AMOUNT_AGG
WHERE GOOD=:GOOD;
INSERT INTO GOODS_AMOUNT_AGG
(GOOD, AMOUNT) VALUES(:GOOD, :TOTAL);
END
END

Run procedure in concurrency (or consistency)
Exclusive document editing

• Goal – implement exclusive changes
• Rollbacks are not welcome
• Need explicit record locking
How to implement explicit record locking

• Blank update in long transaction
  – Or SELECT … FOR UPDATE WITH LOCK

• Flags at business logic level
Blank update

First update creates record version, preventing other transactions to update this record

Pro: Easy to implement

Con: Trigger fires on update, 2\textsuperscript{nd} update causes update “in-place”
SELECT ... FOR UPDATE WITH LOCK

• Same as blank update
• Can lock several records
• Locks record on fetch
  – Result returns one record per one fetch (no buffering)
• Useless for aggregates (SUM, AVG, COUNT, ...)

Firebird Conference 2019, Berlin
• Locking in the versioning server is not normal
• It maybe not enough to choose appropriate transaction isolation level
Flags at business logic level

• Add User and TimeStamp fields, or create additional table
• When you want to “lock”, write USER and CURRENT_TIMESTAMP in short transaction

• if user <> myself then
  if TimeStamp is far then
    UPDATE set User, TimeStamp
  else Fail(“locked by user User at TimeStamp”)
else
  UPDATE set TimeStamp

• Additional table need to be cleared (disconnected apps)
Data editing

• Application is used by operator not in the way developer designed it
• Badly designed data editing can be a problem
Data editing: wrong scenario

Open form
StartTransaction;
IBDataSet.Open;
IBDataSet.Edit;
// entering edit mode

Wait for user input
on Save button
Commit;
on Cancel button
Rollback;

Update t1 set
i1=...
What is that?
Lunch time!
Data editing: Solution

- Open form
- StartTransaction;
- Fill controls
- Commit;
- Wait for user

User presses Save button:
- StartTransaction;
  - IBDataSet.Edit; or IBQuery1.Prepare
- Fill data from controls
  - IBDataSet.Post; or IBQuery1.ExecSQL;
- Commit;
Data editing: Solution

Open form
StartTransaction;
Fill controls
Commit;

on Save button:
StartTransaction;
IBDataSet.Edit; or IBQuery1.Prepare
Fill data from controls
IBDataSet.Post; or IBQuery1.ExecSQL;
Commit;

Select * from T1

Insert into T1 values (200);
Robot rules

Reading robots

• Use read-only ReadCommitted

• Try to do work in one transaction, if possible

• Multi-tier - connection and transaction pooling

• Goals
  – Do not stuck OAT
  – Do not advance Next too much

Writing robots

• Do not keep attachment open
  – attach, do work, close;

• Keep transactions short

• Try to do work in one transaction, if possible

• Goals
  – Do not stuck OAT
Tools to monitor transaction markers

- **gstat –h**
- **mon$transactions** (Firebird 2.1 and higher)
- **Trace** (Firebird 2.5 and higher)
- **HQBird DataGuard**
- **FBScanner**
- **IBAnalyst** (gstat visual)
About IBSurgeon

IBSurgeon

- Tools and consulting
- Platinum Sponsor of Firebird Foundation
- Founded in 2002: 17 years of Firebird and InterBase recoveries and consulting
- Based in Moscow, Russia

www.ib-aid.com  www.ibsurgeon.com