How Firebird transactions work

Dmitry Kuzmenko
www.IBSurgeon.com
www.ib-aid.com
Thank you!
IBSurgeon

- Tools and consulting
- Platinum Sponsor of Firebird Foundation
- Founded in 2002: 12 years of Firebird and InterBase recoveries and consulting
- Based in Moscow, Russia, worldwide 24x7 support
Agenda

What is transaction? Why we need it?
How we will present about transactions
Records and versions
Transactions and record versions
Transaction Inventory
Record visibility in transactions
Transaction Markers and their evaluation
Some conclusions
What is transaction?

- Transaction as a general concept of any dynamic system
- “Classic” example
  - begin
    - -- move money from account1 to account2
    - Decrease account1
    - Increase account2
  - end – commit/rollback

- Transaction Managers
Database transaction definition

• a unit of work performed against a database, and treated in a coherent and reliable way independent of other transactions.

• A database transaction, by definition, must be Atomic, Consistent, Isolated and Durable
In ideal world

Insert into T1(i1) values (100);

SELECT i1 FROM T1

Insert into T1(i1) values (200);

only serial operations
In real world

- **Tx14**: 
  - **t0** nowait
  - **t1** UPDATE T1
  - **t2** UPDATE T1
  - **t3** commit

- **Tx11**: 
  - **t0** commit
  - **t1** INSERT T1

- **Tx20**: 
  - **t0** rollback
  - **t1** UPDATE T1
The ultimate purpose of transaction:

• Concurrent execution of operations should lead to the exactly the same result as sequential execution of operations.

*In simple words: each transaction should run as the only transaction.*

For each [snapshot] transaction Firebird engine should maintain a stable view of the database.
How Firebird does implement stable view for each transactions?
How we will present about transactions
How we will present about transactions

Transaction's result

Transaction's result
How we will present about transactions
How we will present about transactions

Operation in the frames of transaction

Insert into T1(i1) values (100);

commit
How we will present about transactions

Result of operation

```
Insert into T1(i1)
values (100);
```

```
SELECT i1
FROM T1
```

snapshot
Now let's start...

Basics your [probably] know:
- Everything in the database is done within transaction
- Each transaction gets its own incremented number
1, 2, 3, … etc
- Firebird is a multi-version engine (each record in Firebird can have versions)
Record versions is a key thing for understanding transactions' work in Firebird.
How record versions appear

```
Insert into T1(i1) values (100);
```
How record versions appear

```
insert into T1(i1) values (100);
```

```
select i1 from T1
```

```
commit
```
How record versions appear

**Tx10**
- Commit
- Insert into T1(i1) values (100);

**Tx50**
- Commit
- SELECT i1 FROM T1
- i1
- 100

**Tx60**
- Commit
- UPDATE T1
  - SET i1=200
- new version!

t0 t1 t2 t3 t4 t5 t6 t7 t8 t9 t10 t11 t12
How record versions appear

- **Tx10**: Insert into T1(i1) values (100);
  - t0: Insert 
  - t1: Commit

- **Tx50**: SELECT i1 FROM T1
  - t5: SELECT 100
  - t6: Commit

- **Tx60**: UPDATE T1 SET i1=200
  - t7: UPDATE
  - t8: COMMIT
  - t9: SELECT 200
  - t10: SELECT 100
  - t11: SELECT 200
  - t12: COMMIT
How it works?
Each record version has transaction #

<table>
<thead>
<tr>
<th>N on page</th>
<th>Transaction number</th>
<th>Datafield1, datafield2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50</td>
<td>100</td>
</tr>
<tr>
<td>N</td>
<td>Tx</td>
<td>Data</td>
</tr>
<tr>
<td>---</td>
<td>----</td>
<td>------</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>100</td>
</tr>
</tbody>
</table>

...
TR50

read

<table>
<thead>
<tr>
<th>N</th>
<th>Tx</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
<td>100</td>
</tr>
</tbody>
</table>

TR60

write
Some intermediate conclusions

1. No “locks” are placed on the record
2. There can be a lot of committed versions for one record
3. Versions may be needed or not. If not, they can be considered as “garbage”.
4. Only one non-committed version can exist for the record
   (2 active transactions can’t update the same record)
How server knows about transactions states? Is transaction Active or not?

• TIP – Transaction Inventory Pages
  • Linear list of transaction states, from 1 to last transaction number
  • Stored in the database
  • Limitation — 2 billions of transactions
Transaction states

- Each transaction is represented in Transactions Inventory by its state:
  - 00 – Active
  - 01 – Committed
  - 10 – Rolled back
  - 11 – Limbo (distributed 2-phase transactions)

```
<table>
<thead>
<tr>
<th>Tx №</th>
<th>Tx state</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>committed</td>
</tr>
<tr>
<td>11</td>
<td>committed</td>
</tr>
<tr>
<td>12</td>
<td>committed</td>
</tr>
<tr>
<td>13</td>
<td>rolled back</td>
</tr>
<tr>
<td>14</td>
<td>committed</td>
</tr>
<tr>
<td>15</td>
<td>committed</td>
</tr>
<tr>
<td>16</td>
<td>committed</td>
</tr>
<tr>
<td>17</td>
<td>rolled back</td>
</tr>
<tr>
<td>18</td>
<td>active</td>
</tr>
<tr>
<td>19</td>
<td>committed</td>
</tr>
<tr>
<td>20</td>
<td>active</td>
</tr>
</tbody>
</table>
```
TIP

**Tx10**

- Insert into T1(i1) values (100);

**Tx50**

- SELECT i1 FROM T1
  - i1: 100

**Tx60**

- **UPDATE T1**
  - SET i1=200
- **SELECT i1 FROM T1**
  - i1: 200

**Tx** | **State**
---|---
10 | Committed
50 | Active
60 | Active
Transaction isolation levels
Isolation levels in Firebird

<table>
<thead>
<tr>
<th>Isolation levels in Firebird</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ COMMITED</td>
</tr>
<tr>
<td>SNAPSHOT</td>
</tr>
<tr>
<td>SNAPSHOT WITH TABLE STABILITY</td>
</tr>
</tbody>
</table>
**Snapshot**

- **Tx 10**
  - Snapshot
  - SELECT FROM T1
  - Commit

- **Tx 51**
  - Insert into T1(i1) values (200);
  - Rollback

- **Tx 52**
  - Insert into T1(i1) values (100);
  - Commit
Read Committed

Tx 10

read committed

SELECT i1
FROM T1

i1

100

Tx 15

Insert into T1(i1)
values (100);

commit
Read Committed transactions “see” global TIP. That’s why they can read committed changes of other transactions.

Snapshot copies TIP on its start. It does not see any changes made by other committed transactions after snapshot start.
TIP for Read Committed

```
Insert into T1(i1)
values (100);
```

```
SELECT i1
FROM T1
```

```
SELECT i1
FROM T1
```

<table>
<thead>
<tr>
<th>Tx</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Active</td>
</tr>
<tr>
<td>15</td>
<td>Active</td>
</tr>
<tr>
<td>10</td>
<td>Active</td>
</tr>
<tr>
<td>15</td>
<td>Commited</td>
</tr>
</tbody>
</table>
TIP for snapshot

**Tx 10**
- SELECT FROM T1
- snapshot

**Tx 51**
- Insert into T1(i1) values (200);
- rollback

**Tx 52**
- Insert into T1(i1) values (100);
- commit

<table>
<thead>
<tr>
<th>Tx</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Active</td>
</tr>
<tr>
<td>51</td>
<td>Active</td>
</tr>
<tr>
<td>52</td>
<td>Active</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tx</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Active</td>
</tr>
<tr>
<td>51</td>
<td>Rollback</td>
</tr>
<tr>
<td>52</td>
<td>Commited</td>
</tr>
</tbody>
</table>
Each transaction can see:

- Own created records and versions
  - Insert, Update, Delete
- If it is **Read Committed**, it can see every changes that was made by committed transactions, because it looks into global TIP
- If it is **Snapshot**, it can see own changes and record versions committed to the moment of its start, because it looks into its own copy of TIP
Record versions visibility
How we will present about records

Each record can have versions, created by different transactions

Record 10

Tx 10 100 → Tx 20 200 → Tx 30 555
How we will present about records

Record 10

Tx 10 100 → Tx 20 200 → Tx 30 555

Compact representation

R10

Tx 10 → Tx 20 → Tx 30
3 rules of record visibility

1) For each snapshot transaction engine maintains stable view of database
2) Transaction can not see record versions created by another active transaction
3) Transaction should walk backversions chain looking for committed backversion
Ex: record versions visibility for Tx20

Snapshot isolation, copy of TIP for Tx20

<table>
<thead>
<tr>
<th>Tx No</th>
<th>Tx state</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>committed</td>
</tr>
<tr>
<td>11</td>
<td>committed</td>
</tr>
<tr>
<td>12</td>
<td>committed</td>
</tr>
<tr>
<td>13</td>
<td>committed</td>
</tr>
<tr>
<td>14</td>
<td>active</td>
</tr>
<tr>
<td>15</td>
<td>committed</td>
</tr>
<tr>
<td>16</td>
<td>committed</td>
</tr>
<tr>
<td>17</td>
<td>rolled back</td>
</tr>
<tr>
<td>18</td>
<td>active</td>
</tr>
<tr>
<td>19</td>
<td>committed</td>
</tr>
<tr>
<td>20</td>
<td>active</td>
</tr>
<tr>
<td>...</td>
<td>active</td>
</tr>
</tbody>
</table>

Record versions or versions chain

R1: Tx 18 → Tx 16
R2: Tx 12
R3: Tx 20 → Tx 11 → Tx 10
R4: Tx 14
R5: Tx 25 → Tx 14 → Tx 12

Tx 20 can see

Tx 20 can see
• In order to figure out which record version is visible, every transaction **must read** TIP

• TIP can contain up to 2 Billion transactions

• *So each transaction should read up to 2 billions of transactions!* - Damn, that's why Firebird is slow! (it's a joke)
TIP (example)

We need a way to separate old, not interesting transactions from currently active part of TIP

• For this purpose engine maintains Oldest Interesting Transaction marker, or OIT
### TIP (example)

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>96</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>112</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>144</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Green** indicates the transaction is committed.
- **Yellow** indicates the transaction is active.
- **Red** indicates the transaction is rolled back.

**15** transaction number
Transaction markers

firebird>gstat -h A.FDB
Database header page information:
Flags 0
Generation 6
System Change Number 0
Page size 4096
ODS version 12.0
Oldest transaction 1
Oldest active 2
Oldest snapshot 2
Next transaction 3
Sequence number 0
Next attachment ID 3
4 markers

• Transaction markers are key characteristics of TIP and transaction mechanism

  – Let's see what they mean and how they evaluated:
    • NEXT — next transaction
    • OAT — Oldest Active
    • OST — Oldest Snapshot
    • OIT — Oldest Interesting
NEXT

• NEXT is the simplest — it's the most recent transaction

• NEXT number is written on header page
OAT - Oldest Active Transaction

**OAT** is the first transaction in TIP which state is “active”

**Evaluation:**
- Scan TIP starting from current OAT value looking for “active” transaction
- Save found value in transaction's lock data
- Save found value as new OAT marker

**OAT is really an oldest active transaction**
OAT evaluation example

• Sample of transactions flow and evaluation of OAT
Problems indicated by OAT

• Where to look?
  • NEXT — OAT > (number of connections * number of transaction)

• What it means?
  • Long running transaction which makes Firebird to think that record versions are still needed
- **Oldest Snapshot Transaction (OST) marker** is the value of the OAT recorded when oldest of currently active transactions was started.
- Get min value of stored in transactions lock's data.
- Save found value as new OST marker.

```
Tx 1, OAT = 1
Tx 2, OAT = 1
Tx 3, OAT = 1
Tx 4, OAT = 2
Tx 5, OAT = 4
```

```
OST = 1
OST = 2
```
Oldest Snapshot Transaction (OST) marker is the value of the OAT when oldest of currently active transactions was started.

OST value often is not an alive transaction.
OST marker defines a garbage collection threshold: records, created by transactions $\geq$ OST can not be garbage collected

Long running transactions will “stuck” OST and delay GC
OST and Read Committed transactions

- Read Committed transaction don't require stable snapshot of database
- Oldest Active value for Read Committed transaction is an own number of such transaction
- Read Committed Readonly transaction can't create record versions, is pre-committed at start and have no impact on OST

Read Committed Readonly transaction could run forever and do not delay garbage collection
The longer transaction lasts, the higher chance to create potentially useless (potential garbage) versions.

```
Select * from rdb$database
```
Problems indicated by OST

Where to look

(OST-OIT) > sweep interval

What it means

– Autosweep does not work (if sweep interval >0)
– Some records need garbage collection
Problems caused by long running transactions

• Direct
  • Loss of performance due to more record versions: i.e., queries become slower
    • More indexed reads
    • More data page reads
      • 1.5mln versions ~30mb per record

• Indirect
  • After transaction’s end its versions become garbage, and garbage collection mechanism tries to gather it
  • Due to long transaction OST stuck, so autosweep (if it is not disabled) tries to start at unpredictable moment (and ends without success)
    • GC and sweep can consume a lot of resources
    • Unpredictable moment can occur at high load time
Oldest Interesting Transaction

- Oldest Interesting Transaction (OIT) marker is necessary to know to separate old not active part of TIP from currently used active part
- OIT points before a first transaction in TIP which state is not committed
- Evaluation:
  - Scan TIP starting from current OIT value looking for first not committed transaction
TIP size

- TIP to be copied is NEXT - OIT
- Size of active part of the TIP in bytes is \( \frac{\text{Next} - \text{OIT}}{4} \)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page size</td>
<td>4096</td>
</tr>
<tr>
<td>Forced Write</td>
<td>ON</td>
</tr>
<tr>
<td>Dialect</td>
<td>3</td>
</tr>
<tr>
<td>OnDiskStructure</td>
<td>11.2</td>
</tr>
<tr>
<td>Attributes</td>
<td>force write</td>
</tr>
<tr>
<td>Sweep interval</td>
<td>20000</td>
</tr>
<tr>
<td>Oldest transaction</td>
<td>2147483644</td>
</tr>
<tr>
<td>Oldest snapshot</td>
<td>2147483645</td>
</tr>
<tr>
<td>Oldest active</td>
<td>2147483645</td>
</tr>
<tr>
<td>Next transaction</td>
<td>2147483646</td>
</tr>
<tr>
<td>Sweep gap (active - oldest)</td>
<td>1</td>
</tr>
<tr>
<td>TIP size</td>
<td>131073 pages, 524292 kilobytes</td>
</tr>
</tbody>
</table>
### Database info

<table>
<thead>
<tr>
<th>Database name</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Creation date</td>
<td>05.06.2003 10:02:19</td>
</tr>
<tr>
<td>Statistics date</td>
<td>31.08.2006 18:11:32</td>
</tr>
<tr>
<td>Page size</td>
<td>8192</td>
</tr>
<tr>
<td>Forced Write</td>
<td>ON</td>
</tr>
<tr>
<td>Dialect</td>
<td>1</td>
</tr>
<tr>
<td>OnDiskStructure</td>
<td>10.0</td>
</tr>
<tr>
<td>Attributes</td>
<td>force write</td>
</tr>
<tr>
<td>Sweep interval</td>
<td>0</td>
</tr>
<tr>
<td>Oldest transaction</td>
<td>534249471</td>
</tr>
<tr>
<td>Oldest snapshot</td>
<td>429490176</td>
</tr>
<tr>
<td>Oldest active</td>
<td>534249472</td>
</tr>
<tr>
<td>Next transaction</td>
<td>534249481</td>
</tr>
<tr>
<td>Sweep gap (snapshot - oldest)</td>
<td>-104759295</td>
</tr>
<tr>
<td>TIP size</td>
<td>16305 pages, 130440 kilobytes</td>
</tr>
<tr>
<td>Snapshot TIP size</td>
<td>10 transactions, 8 kilobytes</td>
</tr>
<tr>
<td>Active transactions</td>
<td>9, 0% of daily average</td>
</tr>
<tr>
<td>Transactions per day</td>
<td>451224, for 1184 days</td>
</tr>
</tbody>
</table>
Problems indicated by OIT

Where to look
OIT- OST
Problem
Big size of TIP
— Global, and, specifically copies of TIP for snapshots
Ideal transactions flow

Short transactions does not stuck OIT or OAT or OST, and avoid problems related with it.

Oldest transaction X-1
Oldest active X
Oldest snapshot X
Next transaction X+1
Summary

• Make write (for INSERT/UPDATE/DELETE) transactions as short as possible
• Use Read Commited Read-Only transactions for SELECTs
• Troubleshoot transactions with HQbird toolset: MonLogger, IBAnalyst and FBDataGuard
Thank you!

• Questions? support@ib-aid.com