# A not-so-very technical discussion of Multi Version Concurrency Control

A comment on a discussion between IBM and Oracle sales departments with regard to the pros and cons of multi-version concurrency control. IBM DB2 is a classic example of the database system with pessimistic locking, while Oracle uses record versions to provide better concurrency in conditions involving both readers and writers. Since Firebird also uses record versions, we respond to the critique from IBM and show how Firebird would behave in such conditions.

## Table of Contents



## Origins of conflict

In February 2002 Oracle published a “Technical Comparison of Oracle Database vs. IBM DB2 UDB: Focus on Performance” white paper where they claimed to have better architecture in Oracle 9i compared to IBM DB2 UDB V7.2. In August 2002 IBM published “A Technical Discussion of Multi Version Read Consistency” white paper claiming that Oracle multi-version concurrency is not better than the approach used in IBM DB2, but requires many workarounds to achieve needed results.

Traditionally, the problem of concurrency is solved using locking. If A needs access to resource N, it locks it; after use the lock is released. If B wants to access resource N while A is using it, it must wait. It is clear that such approach may give very poor results when the locks are applied at a very high level – consider the example of two editors editing different chapters in a big MS Word document. MS Word blocks access to the document file at the file system level. While the first editor is able to modify the document, the second must wait until the first one finishes editing. And this is correct, since the second editor does not know what changes were made by the first one in general. However, MS Word gives an option to open the document in read-only mode, allowing the second editor to read the chapter, and plan what to change on the “secondary storage”, read “using a pen and a sheet of paper”. When the first editor finishes editing, the second editor re-opens the latest version of the document in a read-write mode and “applies” the changes noted on the paper.

**DB2 Magazine:** *“...an application ported directly from Oracle to DB2 UDB may experience deadlocks that it did not have previously”*

In its white paper Oracle claims that IBM DB2 UDB V7.2 EEE, which uses locking as in the example above, has poor concurrency, citing the “Oracle to DB2 Porting Guide”: *“As a result of different concurrency controls in Oracle and DB2 UDB, an application ported directly from Oracle to DB2 UDB may experience deadlocks that it did not have previously. As DB2 UDB acquires a share lock for readers, updaters may be blocked where that was not the case using Oracle. A deadlock occurs when two or more applications are waiting for each other but neither can proceed because each has locks that are required by others. The only way to resolve a deadlock is to roll back one of the applications.”*[1]. In response, IBM claims that Oracle's multi-version architecture does not solve the problem, since now the database engine has to do much more I/O to access needed record versions and the disk space for record versions is limited, and, when it is filled completely, transactions are rolled back with a ORA-1555 "Snapshot too old" message. IBM also claims that approach used in Oracle gives incorrect results under some conditions and additional programming is needed to solve the issue.

## Firebird case

InterBase, the predecessor of Firebird, was among the first commercial

1 Oracle to DB2 Porting Guide, page 47, http://www.db2udb.net/guide/program/text/oraclev3.pdf

databases to implement multi-version concurrency control (MVCC)[2]. This makes the behavior of Firebird close to Oracle, however with a notable difference – Firebird is naturally multi-versioned, while Oracle acquired this feature in Oracle 7.x. Until than it had an architecture similar to IBM DB2. Firebird simply does not have the negative issues emphasized in the both white papers, while using all advantages of MVCC.

### Concept

So how does it work? The main idea was already presented when we talked about MS Word opening a file in read-only mode, but there are some important details. As the name implies, each record in the system might have multiple versions visible to different transactions. When a transaction modifies a record, a new version is written to the database, and a previous version, representing only the difference between the version of the record that was read by the transaction and the new value of the record, is written as a back version of that record.

Multi-generational architecture allows different transactions to avoid conflicts between readers and writers. Reading transaction can always see consistent view of the database regardless of the write operations that happen concurrently.

How does the system know which version is visible to which transaction? When a transaction starts, it receives a singly incrementing number. This number uniquely identifies the transaction within the system during the lifespan of the database since the last restore. Every change that is done in the database is "signed" by the transaction number. When a record is read on behalf of some transaction, the database system compares the "signature" of the record with a transaction number. If the "signature" belongs to a transaction that was committed when the current transaction started, that version is returned to the application. Otherwise, the database engine computes the needed version using the current record state and the back versions of that record without regard to the locks that the writing transaction has.

This is very simplified description of what happens in Firebird, for more technical details please read the Firebird for the Database Expert: Episode 4- OAT, OIT, & Sweep article[3]. Ann W. Harrison provides an excellent description with examples that illustrate the whole complexity of this issue.

### Similarities and differences

The description above should be enough to see that Firebird functions similarly to Oracle 9i.

- Multi-generational architecture allows different transactions to avoid conflicts between readers and writers. The reading transaction can always see a consistent view of the database regardless of the write operations that are happening concurrently. IBM DB2 can provide

2 According to Ann W. Harrison, first was Rdb/ELN released in 1984 by DEC, second was InterBase, both designed by Jim Starkey. Later DEC decided to push Rdb/VMS, which had the same API, but was implemented completely different, so InterBase can be considered the first database using MVCC that survived to our days.

3 http://www.ibphoenix.com/main.nfs?a=ibphoenix&page=ibp_expert4

...the description ... is enough to see that Firebird functions similar to Oracle 9i...

But unlike Oracle, Firebird cannot produce something similar to the ORA-1555 "Snapshot too old". This is one of the consequences of the overall goal to be a DBA-free database.

such level of concurrency only sacrificing the database consistency and using dirty reads.

- The mechanism of back versions in Firebird is similar to the rollback segments used in Oracle for the same purposes. Both systems are optimistic, in other words, they assume that, in most cases, an application will not need previous versions of the records. The optimization is performed to give the best performance to the most likely case.

But unlike Oracle, Firebird cannot produce anything similar to the ORA-1555 "Snapshot too old". There is no need to estimate the size of the rollback segments as described in the IBM white paper, since all information needed for rollback operations and computing previous record versions is stored inside the database itself and the database file grows automatically if more space is needed.

However, the approach used in Firebird has its price. What Oracle solves by rolling the rollback segments over, and which finally leads to the ORA-1555 "Snapshot too old" error, Firebird must handle differently.

The first issue is long record version chains. Oracle drops rollback segments when they get too large. Firebird never drops a back version if it could be seen by any running transaction. As a result, a long-lived transaction blocks the removal of back versions of all records, causing the database to grow and performance to deteriorate. The performance cost is due both to the decreased density of valid data and to the cost of checking whether any back versions of records can be deleted.

A second issue is the cost of removing back versions. Oracle's back versions are in a separate segment. Firebird's back versions are in the database, so they must be removed one at a time, as they are encountered by subsequent transactions.

A third issue is the cost of a rollback. When a transaction inserts, updates, or deletes a record, Firebird changes the database immediately, relying on the back versions as an undo log. A failed transaction's work remains in the database and must be removed when it is found.

Firebird successfully handles these cases without user intervention. Its behavior is controlled by a few parameters, like “sweep interval”. However detailed discussion is out of the scope of this paper: please see Firebird documentation for more details.

...Example on Illustration 1 is used to demonstrate weakness of Oracle 9i...

So, how does it apply to Firebird? It will not work, Firebird reports an error on step 6.

It is worth mentioning one very nice “consequence” of the fact that there is no recovery log. Firebird has to take additional care to keep the database file in a consistent state – if a crash happens, there is no other place where information can be recovered except the database file itself. This is achieved using the careful write technique – Firebird writes data onto disk in such a manner that, at every single moment, the database file is consistent. The careful writes feature is something that really makes the life of the end-user easier. In addition to automated database housekeeping, Firebird has also

| *Time* | *Transaction 1* | *Transaction 2* |
|---|---|---|
| 1. | Begin transaction | |
| 2. | | Begin transaction |
| 3. | Select available seats on flight ABC111.<br>See seat 23F is the last seat available<br>Reserve this seat. | |
| 4. | | Select available seats on flight ABC111. Also sees 23F as Oracle will go to the rollback segment to get the old version of that block. |
| 5. | Commit Transaction. | |
| 6. | | Reserve this seat. |
| 7. | | Commit Transaction.<br>Successful but now the flight is oversold. |

*Illustration 1: Example IBM used to show incorrect logic in Oracle 9i version control.*

automated crash recovery – a truly DBA-free database engine.

The next critique of Oracle's versioning mechanism is what IBM calls an ability to see current data. The example on Illustration 1 is used to demonstrate the weakness of Oracle 9i.

So, how does it apply to Firebird? It will not work. Firebird reports an error on step 6. The logic is quite simple in this case. At the beginning of the operation, both transactions saw a record version signed by a transaction, let's say, 120. When transaction 1 committed on step 5, the new record version was signed with a number of transaction 1, lets say, 125. Now, if transaction 2 tries to update the same record, it will find that the version of the record is no longer 120, but 125, and will report an error to the application. The update operation will not succeed.

Furthermore, the same error will be reported if step 6 happens before step 5, but after step 3. It is also possible to tell transaction 2 to wait until transaction 1 finishes and then decide the outcome of the operation. If transaction 2 is lucky and transaction 1 is rolled back (for example, the customer booking a seat in transaction 1 changed his mind), it will successfully book the seat for the second customer. In case of IBM DB2, the lock conflict would have happened already in step 4, since transaction 2 would try to lock a record that had already been modified by transaction 1. The change of mind by the first customer does not help the second one. The application has to re-read the table and check for a new seat for the booking.

...multi-version concurrency control, if implemented correctly, provides a superior concurrency in cases when update conflicts are rare compared to traditional pessimistic locking schemes.

## *Conclusion*

From the above it is clear that multi-version concurrency control, if implemented correctly, provides a superior concurrency in cases when

update conflicts are rare compared to traditional pessimistic locking schemes. It is also clear that there are cases when pessimistic locking will perform better. However, the claim made by IBM that multi-version concurrency control is not used in most database systems is no longer true since Microsoft has decided to switch to MVCC in the next version of SQL Server (code name Yukon). Now two of three biggest commercial database vendors use MVCC. In fact, the versioning mechanism used in Yukon is almost an exact copy of the mechanism used in Firebird. It took almost 20 years for other software vendors to find out that MVCC is great approach to handle concurrent access to the database.

## Acknowledgments

The author is grateful to Ann W. Harrison and Helen Borrie for their comments and help during the preparation of this paper.

## References

1. A Technical Discussion of Multi Version Read Consistency, By IBM Software Group, Toronto Laboratory August 2002, ftp://ftp.software.ibm.com/software/data/pubs/papers/readconsistency.pdf
2. Technical Comparison of Oracle Database vs. IBM DB2 UDB: Focus on Performance, An Oracle White Paper, February 2002

## About the Author

Roman Rokytskyy is one of the Firebird Project members, leader of the JayBird subproject, the JCA/JDBC driver for Firebird.