Transaction

A transaction is a single task with a number of specific characteristics. An application can perform one or more operations, within the context of a transaction, each of which must be completed in sequence.

One of the main tools used by relational databases to maintain data integrity is the transaction. A transaction is a task with a number of specific characteristics:

1. An application can perform one or more operations within the context of a transaction, each of which must be completed in sequence. An operation consists of, as a rule, one SQL statement, such as SELECT, INSERT, UPDATE, or DELETE.
2. The changes performed by the transaction can be committed if all of the operations in the transaction are completed. Until the results of the transaction are committed, the changes made to the database are invisible to other users.
3. A transaction can also be rolled back. In this case, as far as other database users are concerned, the data never changed.

Because of these characteristics, transactions ensure that complex operations on the database are performed completely. Transactions provide complete protection against operations not being completely processed, therefore ensuring data integrity.

A transaction can be in one of the following four states:

1. in limbo
2. committed
3. rolled back
4. active

Transaction mask

A transaction mask is an array of two bit pairs that represents the state of all transactions starting with the oldest interesting and ending with the next transaction. The oldest interesting transaction is the first transaction in the database after transaction zero) whose state is not committed. Transaction zero is the system transaction and is always active. The next transaction is the transaction after the one that started most recently.

In the Classic architecture, each connection maintains its own copy of the transaction mask. In shared server architectures, each server maintains a single copy of the transaction mask. In Classic, and in particular on machines with memory sizes that were typical in the early 90's, you could eat up a lot of memory describing a system that had a few hundred thousand transactions between the oldest interesting and the next transaction, even if you only use two bits per transaction.

Transaction number column

For every table you create, including system tables, Firebird/InterBase® maintains an extra column for the transaction number. When you insert or update a column as part of a transaction, the transaction number is written to this column, so that Firebird/InterBase® knows which transaction is controlling that row of the table. Even when you delete a row as part of the transaction, the number is written to the row until the transaction is committed or rolled back, in case there is a problem, or in case the transaction is a lengthy one.

The Firebird/InterBase® versioning engine uses this transaction number to ensure that each user receives a consistent view of the database at a moment in time. This is known as a *repeatable read*.

Active transactions

A transaction is active, if one of the following conditions is true:

- The transaction has not yet started.
- The transaction has started but not yet completed.
- The transaction has started, could not however complete successfully, due to for example, a system crash or communication problems etc.

The actual status of each transaction is recorded in the TIP (Transaction Inventory Page). In fact, the only alteration that occurs when a transaction is committed is the alteration to the status in the TIP from active to committed.

Transactions in limbo

Firebird/InterBase®'s transaction mechanism, like most databases, can only handle transactions within a single database. However within an embedded SQL application, Firebird/InterBase® can perform operations on more than one database at a time.

With a logical transaction that spans databases, Firebird/InterBase® handles the operations within each database as separate transactions, and sequences them using a two-phase commit model, to ensure that both transactions complete or that neither completes. When Firebird/InterBase® is ready to commit or rollback such a multidatabase transaction, it first changes the transaction status from active to limbo. It then performs the commit or rollback operation. Finally the transaction status is changed from limbo to committed.

Transactions in limbo are transactions that have been started by the PREPARE command within the framework of a two-phase commit. The transaction may or may not still be running. This transaction may become relevant at any point in time and all changes made so far may be committed or rolled back. Such alterations made by such transactions can neither be examined or ignored; they can neither be defined as executed or aborted. They can therefore not simply be removed from the

database.

However for a database backup to be fully performed without aborting, such transactions in limbo need to be ignored in the backup. Only those most recent, committed transactions are backed up. It allows a database to be backed up, before recovering corrupted transactions. Generally in limbo transactions should be recovered before a backup is performed.

Note: BDE clients use only single-database transactions, even if the client application accesses two or more databases. Embedded SQL and Firebird/InterBase® API provide methods for programming distributed transactions.