# Managing Database Transactions

Typical database applications may be accessible to hundreds/thousands of users simultaneously e.g. in the case of online applications. In most cases its possible for many users to request access to the same database tables simultaneously. Some means of protecting the data from conflicting access must be used.

Example: Suppose two online users search for open seats on the same flight. Independently of each other they are searching the same database. Seat B4 is indicated as open. User 1 books the seat. User 2 waits some time and then books the same seat. It is now double-booked.

An associated database problem is that of ensuring that related database updates occur together as one operation or else not at all.

Example: Transfer funds from A to B:

Nor mal Operation Faulty Operation

Debit €100 from A Debit €100 to A

time

Credit €100 to B Server Crashes

B is not credited

Normal Operation is performed as one 'operation' (see START TRANSACTION below).

Database Transactions are a mechanism used to isolate individual database operations and provide recovery checkpoints so that the database state can be restored easily in the event of failure. A database transaction has the following properties (aka ACID properties):

Atomic: Each transaction is considered to be a single, self-contained database operation (as above)

Consistent: A transaction must leave the database in a consistent state

Isolated: Transactions must operate in isolation from each other. Each one must operate independently and without knowledge of other transactions.

Durable: The effects of a transaction persist in the database i.e. has a permanent effect on the database.

Normally each SQL statement executed in Workbench is considered a separate transaction. However a sequence of statements can be combined with the "START TRANSACTION" instruction. A transaction ends with a COMMIT or ROLLBACK statement. If all operations complete successfully then a COMMIT will apply the changes permanently to the database. On the other hand if anything goes wrong (e.g. data validation error, database error, server error etc) then a ROLLBACK statement reverses any changes made and leaves the database the way it was before the START TRANSACTION statement.

Example: Funds transfer

START TRANSACTION

UPDATE Account

SET Balance = Balance – 100

WHERE Name = "A";

SELECT Balance FROM Accounts

WHERE Name = "A"

IF Balance < 0 ROLLBACK

UPDATE Account

SET Balance = Balance+100

WHERE Name = "B";

COMMIT;

The COMMIT/ROLLBACK of transactions provides us with an atomic mode of operation. Either the whole transaction is executed successfully or none of it is performed. Permanent changes to the data in the database are held off until the COMMIT statement is executed.

To improve performance each transaction is executed concurrently in a multi-threaded environment. If each transaction uses different sets of data then there will be no interference and all transactions can operate independently. However if transactions reference the same data then the usual practice is for the database server to lock the records being updated until the new data is committed. This has the effect of delaying other transactions which read/write the same table rows. The database remains consistent.

If ALL transactions just READ the same data then no record locking is required and no transaction is delayed. If only one transaction writes to the rows then locking occurs. Sometimes for the sake of improved performance you can tolerate reading inconsistent date (dirty reads). The effect may not be important to the user, for example the seat they want is booked but becomes free later – they just choose another seat in the meantime. You can specify the 'isolation level' for each transaction. The default is 'serialised'.

Isolation Levels:

1) Read Uncommitted: A transaction may read data that is modified but not yet committed by another transaction. (Least safe level)

2) Read Committed: Only committed data is read. Possibility exists that the data is later modified.

3) Repeatable Read: Data records read by the transaction are guaranteed to exist if they are re-read. New records may not exist but won't be returned.

4) Serialized: Transactions are executed one after another with no interference. (Safest but possibly slowest level)