1 Database Transactions

Database transaction is collection of SQL queries which forms a logical one task. For transaction to be completed successfully all SQL queries has to run successfully. Database transaction executes either all or none, so for example if your database transaction contains 4 SQL queries and one of them fails then change made by other 3 queries will be rolled back. This way your database always remain consistent whether transaction succeeded or failed. Transaction is implemented in database using SQL keyword transaction, commit and rollback. Commit writes the changes made by transaction into database and rollback removes temporary changes logged in transaction log by database transaction.

2 ACID Property of Transactions

- **Atomicity**
  A transaction must be an atomic unit of work; either all of its data modifications are performed, or none of them is performed.
- **Consistency**
  When completed, a transaction must leave all data in a consistent state. In a relational database, all rules must be applied to the transaction's modifications to maintain all data integrity. All internal data structures, such as B-tree indexes or doubly-linked lists, must be correct at the end of the transaction.
- **Isolation**
  Modifications made by concurrent transactions must be isolated from the modifications made by any other concurrent transactions. A transaction either recognizes data in the state it was in before another concurrent transaction modified it, or it recognizes the data after the second transaction has completed, but it does not recognize an intermediate state. This is referred to as serializability because it results in the ability to reload the starting data and

replay a series of transactions to end up with the data in the same state it was in after the original transactions were performed.

- **Durability**
  After a transaction has completed, its effects are permanently in place in the system. The modifications persist even in the event of a system failure.

3 Transaction Statement

3.1 Explicit Transactions

Explicitly start a transaction through an API function or by issuing the Transact-SQL BEGIN TRANSACTION statement.

3.2 Auto commit Transactions

The default mode for the Database Engine. Each individual Transact-SQL statement is committed when it completes. You do not have to specify any statements to control transactions.

3.3 Implicit Transactions

Set implicit transaction mode on through either an API function or the Transact-SQL SET IMPLICIT_TRANSACTIONS ON statement. The next statement automatically starts a new transaction. When that transaction is completed, the next Transact-SQL statement starts a new transaction.

3.4 COMMIT

If a transaction is successful, commit it. A COMMIT statement guarantees all of the transaction's modifications are made a permanent part of the database. A COMMIT also frees resources, such as locks, used by the transaction.

3.5 ROLLBACK

If errors occur in a transaction, or if the user decides to cancel the transaction, then roll the transaction back. A ROLLBACK statement backs out all modifications made in the transaction by returning the data to the state it was in at the start of the transaction. A ROLLBACK also frees resources held by the transaction.

3.6 Example

```
--drop table MyTable;
create table MyTable (ID int, Status varchar(10));
insert into MyTable (ID, Status) values (2,'Start');
begin transaction;
insert into MyTable (ID, Status) values (4,'Rollback');
rollback;
begin transaction;
insert into MyTable (ID, Status) values (5,'Commit');
insert into MyTable (ID, Status) values (6,'Commit');
insert into MyTable (ID, Status) values (7,'Commit');
insert into MyTable (ID, Status) values (8,'Commit');
insert into MyTable (ID, Status) values (9,'End');
commit;
select * from MyTable;
```

3.6.1 Multi-statement Transactions

To make transactions a little more useful you really need to put two or more statements in them. These are called Explicit Transactions. For example,

```
BEGIN TRAN

UPDATE  MyTable
SET     status = 'multi-1'
WHERE   id = '5'

UPDATE  MyTable
SET     status = 'multi-2'
WHERE   id = '6'

COMMIT TRAN
```

Note that we have a BEGIN TRAN at the beginning and a COMMIT TRAN at the end. These statements start and complete a transaction. Everything inside these statements is considered a logical unit of work. If the system (Note: change statement to system for clarity) fails after the first update, neither update statement will be applied when SQL Server is restarted. The log file will contain a BEGIN TRAN but no corresponding COMMIT TRAN.

3.6.2 Rolling Back

You can also roll back a transaction if it doesn't do what you want. Consider the following transaction:

```
BEGIN TRAN

 UPDATE  MyTable
 SET     status = 'commit'
 WHERE   id = '5'

 UPDATE  MyTable
 SET     id = '0'
 WHERE   status = 'commit'

IF @@ROWCOUNT = 5 – check result and now change it to @@ROWCOUNT = 3
  COMMIT TRAN
ELSE
  ROLLBACK TRAN
```

Suppose that for whatever reason, the second update statement should update exactly five rows. If @@ROWCOUNT, which hold the number of rows affected by each statement, is five then the transaction commits otherwise it rolls back. The ROLLBACK TRAN statement "undoes" all the work since the matching BEGIN TRAN statement. It will not perform either update statement. Note that Query Analyzer will show you messages indicating that rows were updated but you can query the database to verify that no actual data modifications took place.

Also see this links for more examples

http://www.codeproject.com/Articles/4451/SQL-Server-Transactions-and-Error-Handling