**Introduction to Transaction**

A transaction is a sequence of operations performed (using one or more SQL statements) on a database as a single logical unit of work. The effects of all the SQL statements in a transaction can be either all committed (applied to the database) or all rolled back (undone from the database). A database transaction must be atomic, consistent, isolated and durable. Later we have discussed these four points.

To understand the concept of a transaction, consider a banking database. Suppose a bank customer transfers money from his savings account (SB a/c) to his overdraft account (OD a/c), the statement will be divided into four blocks:

* Debit SB a/c.
* Credit OD a/c.
* Record in Transaction Journal
* End Transaction

The SQL statement to debit SB a/c is as follows :

UPDATE sb\_accounts  
SET balance = balance - 1000  
WHERE account\_no = 932656 ;

The SQL statement to credit OD a/c is as follows :

UPDATE od\_accounts  
SET balance = balance + 1000  
WHERE account\_no = 933456 ;

The SQL statement for record in transaction journal is as follows :

INSERT INTO journal VALUES   
(100896, 'Tansaction on Benjamin Hampshair a/c', '26-AUG-08' 932656, 933456, 1000);

The SQL statement for End Transaction is as follows :

COMMIT WORK;

A database transaction must be atomic, consistent, isolated and durable. Bellow we have discussed these four points.

**Atomic :** A transaction is a logical unit of work which must be either completed with all of its data modifications, or none of them is performed.

**Consistent :** At the end of the transaction, all data must be left in a consistent state.

**Isolated :** Modifications of data performed by a transaction must be independent of another transaction. Unless this happens, the outcome of a transaction may be erroneous.

**Durable :** When the transaction is completed, effects of the modifications performed by the transaction must be permanent in the system.

Often these four properties of a transaction are acronymed as **ACID**.

We have explained the above four properties of a translation with the following example :

Example of Fund Transfer

* Transaction to transfer $1000 from account X to account Y:
  1. read(X)
  2. X = X –1000
  3. write(X)
  4. read(Y)
  5. Y = Y + 1000
  6. write(Y)
* Atomicity requirement — if the transaction fails after step 3 and before step 6, the system should ensure that its updates are not reflected in the database, else an inconsistency will result.
* Consistency requirement – the sum of X and Y is unchanged by the execution of the transaction.
* if between steps 3 and 6, another transaction is allowed to access the partially updated database, it will see an inconsistent database (the sum X + Y will be less than it should be).
  1. Isolation can be ensured trivially by running transactions serially, that is one after the other.
  2. However, executing multiple transactions concurrently has significant benefits, as we will see later.
* Durability requirement — once the user has been notified that the transaction has completed (i.e., the transfer of the $1000 has taken place), the updates to the database by the transaction must persist despite failures.

Beginning a Transaction

A transaction is beginning to initiate the execution of multiple SQL statements. Beginning of a transaction guarantees the atomicity of a transaction. After beginning, either it can be committed to making the modifications permanent or rolled back to undo the changes to leave the database unaltered.

Committing a Transaction

By committing a transaction, it is closed explicitly and modifications performed by the transaction is made permanent.

Rolling Back a Transaction

By rolling back a transaction, a transaction is explicitly closed and any modifications made by the transaction is discarded.

**COMMIT command**

COMMIT command is used to permanently save any transaction into the database.

When we use any DML command like INSERT, UPDATE or DELETE, the changes made by these commands are not permanent, until the current session is closed, the changes made by these commands can be rolled back.

To avoid that, we use the COMMIT command to mark the changes as permanent.

Following is commit command's syntax,

**COMMIT;**

**ROLLBACK command**

This command restores the database to last commited state. It is also used with SAVEPOINT command to jump to a savepoint in an ongoing transaction.

If we have used the UPDATE command to make some changes into the database, and realise that those changes were not required, then we can use the ROLLBACK command to rollback those changes, if they were not commited using the COMMIT command.

Following is rollback command's syntax,

**ROLLBACK TO savepoint\_name;**

**SAVEPOINT command**

SAVEPOINT command is used to temporarily save a transaction so that you can rollback to that point whenever required.

Following is savepoint command's syntax,

**SAVEPOINT savepoint\_name;**