Transactions in SQL

A transaction is a sequential group of database manipulation operations, which is performed as if it were one single work unit. In other words, a transaction will never be complete unless each individual operation within the group is successful. If any operation within the transaction fails, the entire transaction will fail.

Practically, you will club many SQL queries into a group and you will execute all of them together as a part of a transaction.

Properties of Transaction

The transaction contains mainly four properties, which referred to as **ACID** property. Now, we are going to discuss the ACID property in detail. The ACID property stands for:

1. Atomicity
2. Consistency
3. Isolation
4. Durability

**Atomicity:** This property ensures that all statements or operations within the transaction unit must be executed successfully. Otherwise, if any operation is failed, the whole transaction will be aborted, and it goes rolled back into their previous state. It includes features:

* COMMIT statement.
* ROLLBACK statement.
* Auto-commit setting.
* Operational data from the INFORMATION\_SCHEMA tables.

**Consistency:** This property ensures that the database changes state only when a transaction will be committed successfully. It is also responsible for protecting data from crashes. It includes features:

* InnoDB doublewrite buffer.
* InnoDB crash recovery.

**Isolation:** This property guarantees that each operation in the transaction unit operated independently. It also ensures that statements are transparent to each other. It includes features:

* SET ISOLATION LEVEL statement.
* Auto-commit setting.
* The low-level details of InnoDB locking.

**Durability:** This property guarantees that the result of committed transactions persists permanently even if the system crashes or failed. It includes features:

* Write buffer in a storage device.
* Battery-backed cache in a storage device.
* Configuration option innodb\_file\_per\_table.
* Configuration option innodb\_flush\_log\_at\_trx\_commit.
* Configuration option sync\_binlog.

In MySQL, the transactions begin with the statement **BEGIN WORK** and end with either a **COMMIT** or a **ROLLBACK** statement. The SQL commands between the beginning and ending statements form the bulk of the transaction.

These two keywords **Commit** and **Rollback** are mainly used for MySQL Transactions.

* When a successful transaction is completed, the COMMIT command should be issued so that the changes to all involved tables will take effect.
* If a failure occurs, a ROLLBACK command should be issued to return every table referenced in the transaction to its previous state.

**EXAMPLES:**

Create a table with table name account\_holder

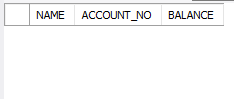
CREATE TABLE account\_holder(

NAME VARCHAR(255),

ACCOUNT\_NO INT(255),

BALANCE INT(255)

);



Inserting data into table using transaction

start transaction;

insert into account\_holder values('Anji','11111','10000');

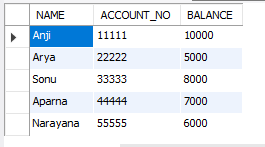
insert into account\_holder values('Arya','22222','5000');

insert into account\_holder values('Sonu','33333','8000');

insert into account\_holder values('Aparna','44444','7000');

insert into account\_holder values('Narayana','55555','6000');

commit;



COMMIT STATEMENT IN TRANSACTION:

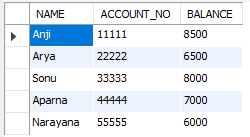
**COMMIT STATEMENT commits** the current **transaction**, making its changes permanent.

START Transaction;

UPDATE account\_holder SET BALANCE=BALANCE- 1500 where ACCOUNT\_No = 11111;

UPDATE account\_holder SET BALANCE=BALANCE+ 1500 where ACCOUNT\_No = 22222;

COMMIT;



ROLLBACK STATEMENT IN TRANSACTION:

ROLLBACK rolls back the current **transaction**,

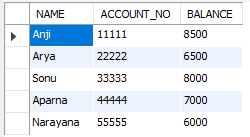
START Transaction;

UPDATE account\_holder SET BALANCE=BALANCE- 1500 where ACCOUNT\_No = 11111;

UPDATE account\_holder SET BALANCE=BALANCE+ 1500 where ACCOUNT\_No = 22222;

ROLLBACK;

COMMIT;



NOTE : HERE THERE IS NO CHANGES IN THE TRANSACTION BECAUSE OF ROLLBACK STATEMENT IN THE TRANSACTION (update is done but it is back to original state because of rollback)

The SAVEPOINT Command

A SAVEPOINT is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction.

The syntax for a SAVEPOINT command is as shown below.

SAVEPOINT SAVEPOINT\_NAME;

**EXAMPLE**

start transaction;

insert into account\_holder values('A','77777',2000);

insert into account\_holder values('B','88888',9000);

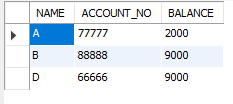
SAVEPOINT save\_point;

insert into account\_holder values('C','99999',1000);

ROLLBACK TO save\_point;

insert into account\_holder values('D','66666',9000);

commit;



AUTO-COMMIT STATEMENT IN TRANSACTION:

**Auto**-**commit** mode means that when a statement is completed, the method **commit** is called on that statement **automatically**. **Auto**-**commit** in effect makes every SQL statement a transaction. The **commit** occurs when the statement completes or the next statement is executed, whichever comes first.

 An SQL statement executed in autocommit mode cannot be rolled back.

Auto-commit Commands :

There are four Auto-commit commands that exist in SQL, they are:

1. **SET AUTOCOMMIT ON –**  
   By executing this particular command, the auto-commit status turned to be ON, if it is OFF initially. This command is itself executed by the SQL server initially. After execution, the commit statement itself gets executed after each SQL statement. Here each statement is a separate transaction.
2. **SET AUTOCOMMIT OFF –**  
   This instruction is just the reverse of the first one. After executing this, the auto-commit status is changed to OFF. Now, the user needs to explicitly mention the commit statements, wherever required. Here Commit() or rollback() completes a transaction.
3. **SET AUTOCOMMIT INT\_VALUE –**  
   This instruction is used to limit the number of statements, which itself gets auto-committed by the server. For example, after the execution of the below command, the auto-commit remains ON for every 9 transactions, and then it turned OFF by itself. SET AUTOCOMMIT 9
4. **SHOW AUTOCOMMIT –**  
   This statement is used to determine the current status of the auto-commit so that the database users can change it according to the requirements. Hence, this statement has only two possible outcomes, either ON or OFF.

**EXAMPLE**

**WHEN SET AUTOCOMMIT OFF**

start transaction;

insert into account\_holder values('A','77777',2000);

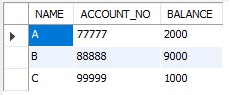
commit;

set autocommit=ON;

insert into account\_holder values('B','88888',9000);

insert into account\_holder values('C','99999',1000);

ROLLBACK;



**WHEN SET AUTOCOMMIT OFF**

start transaction;

insert into account\_holder values('A','77777',2000);

commit;

set autocommit=OFF;

insert into account\_holder values('B','88888',9000);

insert into account\_holder values('C','99999',1000);

ROLLBACK;



NOTE : Observe the difference between AUTOCOMMIT ON AND OFF.

**SYNTAX FOR SHOW AUTOCOMMIT**

SELECT @@AUTOCOMMIT;

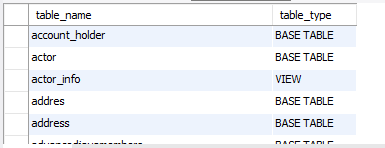


Means AUTOCOMMIT is in ON.

**Operational data from the INFORMATION\_SCHEMA tables.**

SELECT table\_name,table\_type

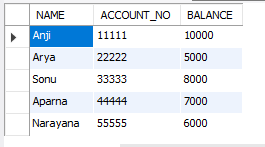
FROM information\_schema.tables;



TRANSACTION ISOLATION LEVEL statement.

* **Dirty Read –** A Dirty read is the situation when a transaction reads a data that has not yet been committed. For example, Let’s say transaction 1 updates a row and leaves it uncommitted, meanwhile, Transaction 2 reads the updated row. If transaction 1 rolls back the change, transaction 2 will have read data that is considered never to have existed.
* **Non Repeatable read –** Non Repeatable read occurs when a transaction reads same row twice, and get a different value each time. For example, suppose transaction T1 reads data. Due to concurrency, another transaction T2 updates the same data and commit, Now if transaction T1 rereads the same data, it will retrieve a different value.
* **Phantom Read –** Phantom Read occurs when two same queries are executed, but the rows retrieved by the two, are different. For example, suppose transaction T1 retrieves a set of rows that satisfy some search criteria. Now, Transaction T2 generates some new rows that match the search criteria for transaction T1. If transaction T1 re-executes the statement that reads the rows, it gets a different set of rows this time.

READ UNCOMITTED

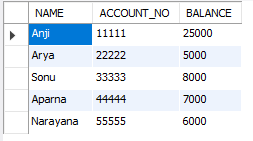


START TRANSACTION;

update account\_holder set BALANCE=25000 where ACCOUNT\_NO=11111;

SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

select \*from account\_holder ;



READ COMMITTED SNAPSHOT ON

TRANSACTION-1

START TRANSACTION;

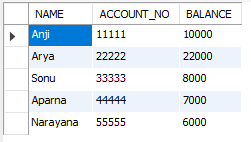
UPDATE account\_holder

SET

BALANCE = 22000

WHERE

ACCOUNT\_NO = 22222;



TRANSACTION-2

start transaction;

SELECT BALANCE from account\_holder where ACCOUNT\_NO = 22222;

COMMIT;



NOTE : Here transaction-2 excutes after transaction-1 because transaction-1 is blocked without commit.

In select query it will take only commited values of table. If any transaction is opened and incompleted on table in others sessions then select query will wait till no transactions are pending on same table.

READ REPEATABLE

select query data of table that is used under transaction of isolation level "Repeatable Read" can not be modified from any other sessions till transcation is completed.

TRANSACTION IN ONE QUERY TAB

START TRANSACTION;

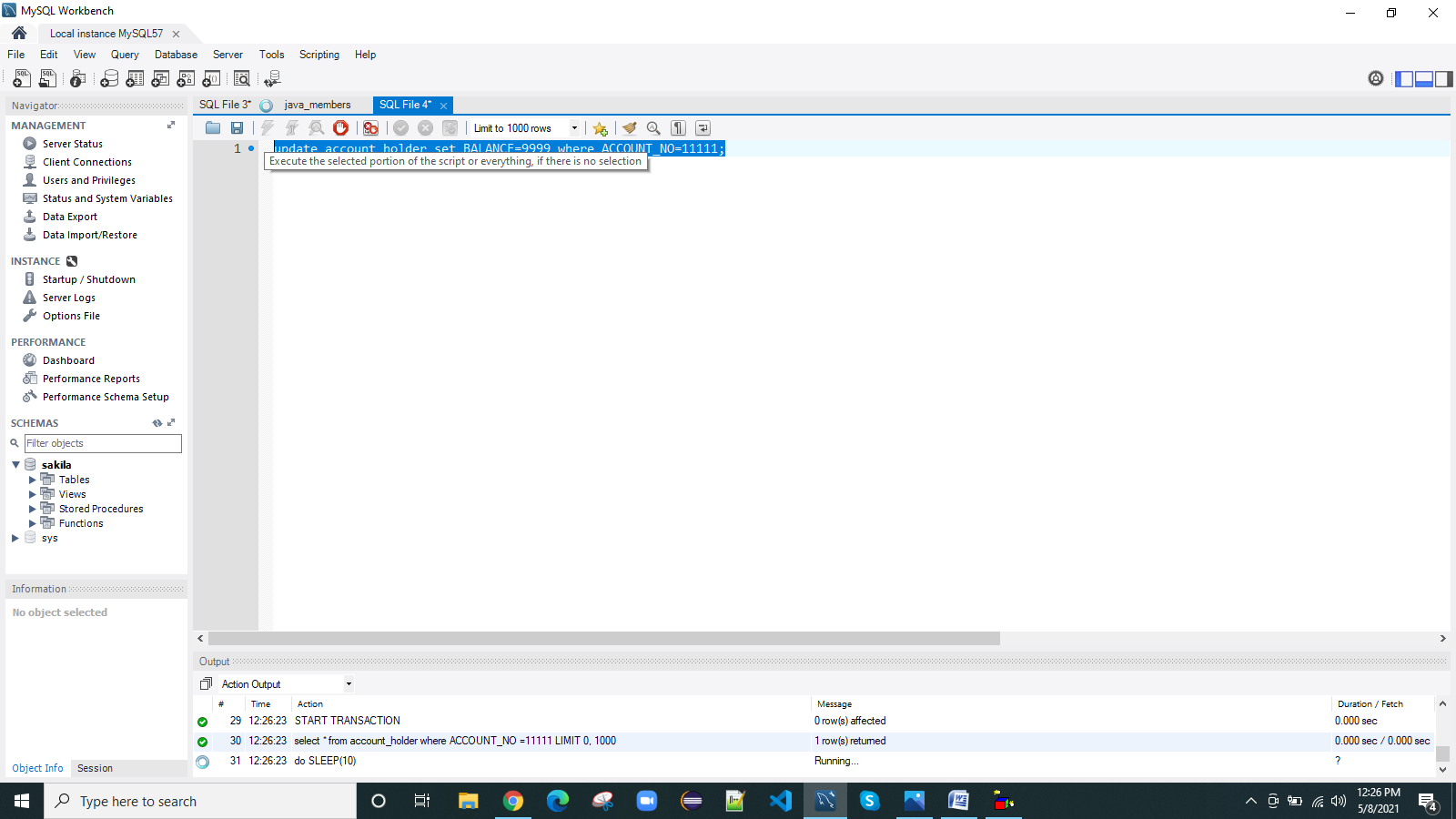
select \* from account\_holder where ACCOUNT\_NO =11111;

do SLEEP(10);

rollback;

TRANSACTION IN ONE ANOTHER-QUERY TAB

update account\_holder set BALANCE=9999 where ACCOUNT\_NO=11111;



NOTE: UPDATEDS MAY NOT OCCUR BUT INSERTION OCCURES AT SAME TIME IN THE TABLE

## Serializable

Serializable Isolation is similar to Repeatable Read Isolation but the difference is it prevents Phantom Read. This works based on range lock.This is the Highest isolation level. A *serializable* execution is guaranteed to be serializable. Serializable execution is defined to be an execution of operations in which concurrently executing transactions appears to be serially executing.

