Transaction

A **transaction** is a single logical unit of work which accesses and possibly modifies the contents of a database. Transactions access data using read and write operations. .

Each transaction begins with a specific task and ends when all the tasks in the group successfully complete. If any of the tasks fail, the transaction fails. Therefore, a transaction has only two results: **success** or **failure**.

Incomplete steps result in the failure of the transaction. A database transaction, by definition, must be atomic, consistent, isolated and durable. These are popularly known as ACID properties.

How to implement Transactions using SQL?
Following commands are used to control transactions. It is important to note that these statements cannot be used while creating tables and are only used with the DML Commands such as – INSERT, UPDATE and DELETE.

1. BEGIN TRANSACTION: It indicates the start point of an explicit or local transaction.
Syntax:
BEGIN TRANSACTION transaction_name ;

2. SET TRANSACTION: Places a name on a transaction.
Syntax:
SET TRANSACTION [READ WRITE | READ ONLY];

3. COMMIT: If everything is in order with all statements within a single transaction, all changes are recorded together in the database is called **committed**. The COMMIT command saves all the transactions to the database since the last COMMIT or ROLLBACK command.
Syntax: COMMIT;

Student				
Rol_No	Name	Address	Phone	Age
1	Ram	Delhi	9455123451	18
2	Ramesh	Gurgaon	9652431543	18
3	Sujit	Rohtak	9156253131	20
4	Suresh	Delhi	9156768971	18
3	Sujit	Rohtak	9156253131	20
2	Ramesh	Gurgaon	9652431543	18

Following is an example which would delete those records from the table which have age = 20 and then COMMIT the changes in the database.
Queries:

DELETE FROM Student WHERE AGE = 20;

COMMIT;

Output:
Thus, two rows from the table would be deleted and the SELECT statement would look like,

Rol_No	Name	Address	Phone	Age
1	Ram	Delhi	9455123451	18
2	Ramesh	Gurgaon	9652431543	18
4	Suresh	Delhi	9156768971	18
2	Ramesh	Gurgaon	9652431543	18

4. ROLLBACK: If any error occurs with any of the SQL grouped statements, all changes need to be aborted. The process of reversing changes is called **rollback**. This command can only be used to undo transactions since the last **COMMIT or ROLLBACK** command was issued.

Syntax: ROLLBACK;
Example: From the above example **Sample table1**,
Delete those records from the table which have age = 20 and then ROLLBACK the changes in the database.
Queries: DELETE FROM Student WHERE AGE = 20;
ROLLBACK;

Student				
Rol_No	Name	Address	Phone	Age
1	Ram	Delhi	9455123451	18
2	Ramesh	Gurgaon	9652431543	18
3	Sujit	Rohtak	9156253131	20
4	Suresh	Delhi	9156768971	18
3	Sujit	Rohtak	9156253131	20
2	Ramesh	Gurgaon	9652431543	18

Advantages of COMMIT and ROLLBACK:

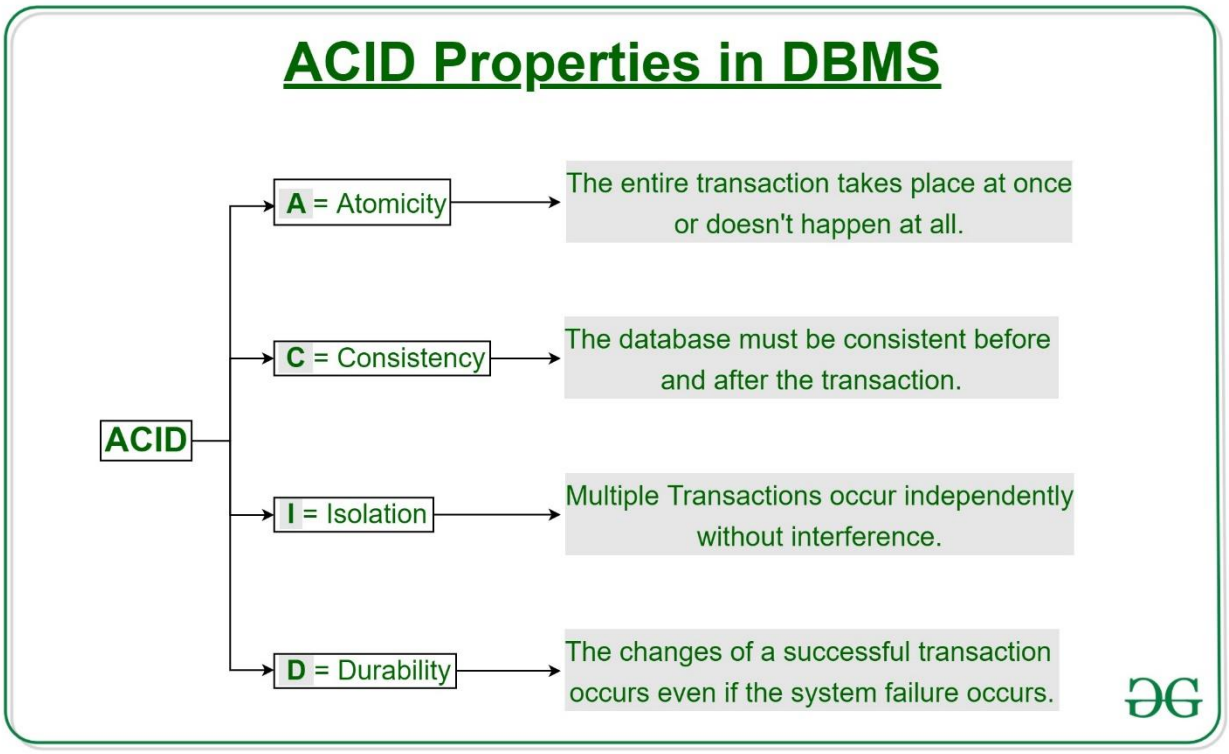
- Ensure data consistency
- Preview data changes before making changes permanent
- Group logically related operations

Transaction Control

- Savepoints:
  - Savepoints are sort of markers that help in splitting a long transaction into smaller units by setting some checkpoints.
  - By setting savepoints within a long transaction, you can roll back to a checkpoint if required.
  - This is done by issuing the SAVEPOINT command.
  - The general syntax for the SAVEPOINT command is:

SAVEPOINT < savepoint_name >;

ACID PROPERTIES:



Atomicity

By this, we mean that either the entire transaction takes place at once or doesn't happen at all. **There is no midway** i.e., transactions do not occur partially. Each transaction is considered as one unit and either runs to completion or is not executed at all. It involves the following two operations. —**Abort**: If a transaction aborts, changes made to database are not visible. —**Commit**: If a transaction commits, changes made are visible. Atomicity is also known as the **'All or nothing rule'**. Consider the following transaction **T** consisting of **T1** and **T2**: Transfer of 100 from account **X** to account **Y**.

Before: X : 500	Y: 200
Transaction T	
T1	**T2**
Read (X) X: = X – 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

If the transaction fails after completion of **T1** but before completion of **T2.**(say, after **write(X)** but before **write(Y)**), then amount has been deducted from **X** but not added to **Y**. **This results in an inconsistent database state**. Therefore, the transaction must be executed in entirety in order to ensure correctness of database state.

Consistency

This means that integrity constraints must be maintained so that the database is consistent before and after the transaction. It refers to the correctness of a database. Referring to the example above,
The total amount before and after the transaction must be maintained. Total **before T** occurs = **500 + 200 = 700**.
Total **after T occurs** = **400 + 300 = 700**.
Therefore, database is **consistent**. Inconsistency occurs in case **T1** completes but **T2** fails. As a result, T is incomplete

Isolation

This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of database state. Transactions occur independently without interference. Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed. This property ensures that the execution of transactions concurrently will result in a state that is equivalent to a state achieved these were executed serially in some order.

Durability:

This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a system failure occurs. These updates now become permanent and are stored in non-volatile memory. The effects of the transaction, thus, are never lost.