Subject: DBMS Semester-III

Chapter No.4

Structured Query language(SQL)

Introduction to SQL

Structure Query Language (SQL) is a database query language used for storing and managing data in Relational DBMS. SQL was the first commercial language introduced for E.F Codd's **Relational** model of database. Today almost all RDBMS (MySql, Oracle, Infomix, Sybase, MS Access) use **SQL** as the standard database query language. SQL is used to perform all types of data operations in RDBMS.

SQL Command

SQL defines following ways to manipulate data stored in an RDBMS.

DDL: Data Definition Language

Command	**Description**
create	to create new table or database
alter	for alteration
truncate	delete data from table
drop	to drop a table

rename	to rename a table

This includes changes to the structure of the table like creation of table, altering table, deleting a table etc.

All DDL commands are **auto-committed**. That means it saves all the changes permanently in the database.

SQL: create command

Creating a Database

To create a database in RDBMS, **create** command is used. Following is the syntax,

```
CREATE DATABASE <DB_NAME>;
```

Example for creating Database

```
CREATE DATABASE Test;
```

The above command will create a database named **Test**, which will be an empty schema without any table.

To create tables in this newly created database, we can again use the create command.

Creating a Table

create command can also be used to create tables. Now when we create a table, we have to specify the details of the columns of the tables too. We can specify the **names** and **datatypes** of various columns in the create command itself.

Following is the syntax,

```
CREATE TABLE <TABLE_NAME>
(
   column_name1 datatype1,
   column_name2 datatype2,
   column_name3 datatype3,
   column_name4 datatype4
);
```

create table command will tell the database system to create a new table with the given table name and column information.

Example for creating Table

```
CREATE TABLE Student(
   student_id INT(10),
   name VARCHAR(100),
   age INT(10));
```

The above command will create a new table with name **Student** in the current database with 3 columns, namely student_id, name and age. Where the column student_id will only store integer, name will hold upto 100 characters and age will again store only integer value.

Most commonly used datatypes for Table columns

Here we have listed some of the most commonly used datatypes used for columns in tables.

Datatype	Use
INT	used for columns which will store integer values.
FLOAT	used for columns which will store float values.
DOUBLE	used for columns which will store float values.
VARCHAR	used for columns which will be used to store characters and integers, basically a string.
CHAR	used for columns which will store char values (single character).
DATE	used for columns which will store date values.
TEXT	used for columns which will store text which is generally long in length. For example, if you create a table for storing profile information of a social networking website, then for **about me** section you can have a column of type TEXT.

SQL: ALTER command

alter command is used for altering the table structure, such as,

- to add a column to existing table
- to rename any existing column

- to change datatype of any column or to modify its size.
- to drop a column from the table.

ALTER Command: Add a new Column

Using ALTER command we can add a column to any existing table. Following is the syntax,

```
ALTER TABLE table_name ADD(
   column_name datatype);
```

Here is an Example for this,

```
ALTER TABLE student ADD(
   address VARCHAR(200)
);
```

The above command will add a new column address to the table **student**, which will hold data of type varchar which is nothing but string, of length 200.

ALTER Command: Add multiple new Columns

Using ALTER command, we can even add multiple new columns to any existing table. Following is the syntax,

```
ALTER TABLE table_name ADD(
   column_name1 datatype1,
   column-name2 datatype2,
   column-name3 datatype3);
```

Here is an Example for this,

```
ALTER TABLE student ADD(
   father_name VARCHAR(60),
```

```
   mother_name VARCHAR(60),
   dob DATE);
```

The above command will add three new columns to the **student** table

ALTER Command: Modify an existing Column

ALTER command can also be used to modify data type of any existing column. Following is the syntax,

```
ALTER TABLE table_name modify(
   column_name datatype
);
```

Here is an Example for this,

```
ALTER TABLE student MODIFY(
   address varchar(300));
```

Remember we added a new column address in the beginning? The above command will modify the address column of the **student** table, to now hold upto 300 characters.

ALTER Command: Rename a Column

Using ALTER command you can rename an existing column. Following is the syntax,

```
ALTER TABLE table_name RENAME
   old_column_name TO new_column_name;
```

Here is an example for this,

```
ALTER TABLE student RENAME
   address TO location;
```

The above command will rename address column to location.

ALTER Command: Drop a Column

ALTER command can also be used to drop or remove columns. Following is the syntax,

```
ALTER TABLE table_name DROP(
   column_name);
```

Here is an example for this,

```
ALTER TABLE student DROP(
   address);
```

The above command will drop the address column from the table **student**.

TRUNCATE command

TRUNCATE command removes all the records from a table. But this command will not destroy the table's structure.

```
TRUNCATE TABLE table_name
```

Here is an example explaining it,

```
TRUNCATE TABLE student;
```

The above query will delete all the records from the table **student**.

DROP command

DROP command completely removes a table from the database. This command will also destroy the table structure and the data stored in it. Following is its syntax,

```
DROP TABLE table_name
```

Here is an example explaining it,

```
DROP TABLE student;
```

RENAME query

RENAME command is used to set a new name for any existing table. Following is the syntax,

```
RENAME TABLE old_table_name to new_table_name
```

Here is an example explaining it.

```
RENAME TABLE student to students_info;
```

The above query will rename the table **student** to **students_info**.

Data Manipulation Language

DML commands are used for manipulating the data stored in the table and not the table itself.

Command	Description
insert	to insert a new row
update	to update existing row
delete	to delete a row
select	To select row

Data Manipulation Language (DML) statements are used for managing data in database. DML commands are not auto-committed. It means changes made by DML command are not permanent to database, it can be rolled back.

Talking about the Insert command, whenever we post a Tweet on Twitter, the text is stored in some table, and as we post a new tweet, a new record gets inserted in that table.

INSERT command

Insert command is used to insert data into a table. Following is its general syntax,

```
INSERT INTO table_name VALUES(data1, data2, ...)
```

Lets see an example,

Consider a table **student** with the following fields.

s_id	name	age

```
INSERT INTO student VALUES(101, 'Adam', 15);
```

table.

s_id	name	Age
101	Adam	15

Insert value into only specific columns

We can use the INSERT command to insert values for only some specific columns of a row. We can specify the column names along with the values to be inserted like this,

```
INSERT INTO student(id, name) values(102, 'Alex');
```

The above SQL query will only insert id and name values in the newly inserted record.

Insert NULL value to a column

Both the statements below will insert NULL value into **age** column of the **student** table.

```
INSERT INTO student(id, name) values(102, 'Alex');
```

```
INSERT INTO Student VALUES(102,'Alex', null);
```

and the other column is set to null.

S_id	S_Name	age
101	Adam	15
102	Alex	

Let's take an example of a real-world problem. These days, Facebook provides an option for **Editing** your status update, how do you think it works? Yes, using the **Update** SQL command.

Let's learn about the syntax and usage of the UPDATE command.

UPDATE command

UPDATE command is used to update any record of data in a table. Following is its general syntax,

```
UPDATE table_name SET column_name = new_value WHERE some_condition;
```

WHERE is used to add a condition to any SQL query, we will soon study about it in detail.

Lets take a sample table **student**,

student_id	name	Age
101	Adam	15
102	Alex	
103	chris	14

```
UPDATE student SET age=18 WHERE student_id=102;
```

S_id	S_Name	Age
101	Adam	15
102	Alex	18
103	chris	14

In the above statement, if we do not use the WHERE clause, then our update query will update age for all the columns of the table to **18**.

DELETE command

DELETE command is used to delete data from a table.

Following is its general syntax,

```
DELETE FROM table_name;
```

Let's take a sample table **student**:

s_id	name	age
101	Adam	15
102	Alex	18
103	Abhi	17

Delete all Records from a Table

```
DELETE FROM student;
```

The above command will delete all the records from the table **student**.

Delete a particular Record from a Table

In our **student** table if we want to delete a single record, we can use the WHERE clause to provide a condition in our DELETE statement.

```
DELETE FROM student WHERE s_id=103;
```

The above command will delete the record where s_id is 103 from the table **student**.

S_id	S_Name	age
101	Adam	15
102	Alex	18

Isn't DELETE same as TRUNCATE

TRUNCATE command is different from DELETE command. The delete command will delete all the rows from a table whereas truncate command not only deletes all the records stored in the table, but it also re-initializes the table(like a newly created table).

For eg: If you have a table with 10 rows and an **auto_increment** primary key, and if you use DELETE command to delete all the rows, it will delete all the rows, but will not re-initialize the primary key, hence if you will insert any row after using the DELETE command, the auto_increment primary key will start from 11. But in case of TRUNCATE command, primary key is re-initialized, and it will again start from 1.

What are SQL Functions?

SQL provides many built-in functions to perform operations on data. These functions are useful while performing mathematical calculations, string concatenations, sub-strings etc. SQL functions are divided into two categories,

1. Aggregate Functions

Aggregate Functions

These functions **return a single value** after performing calculations on a group of values. Following are some of the frequently used Aggregrate functions.

AVG() Function

Average returns average value after calculating it from values in a numeric column.

Its general **syntax** is,

```
SELECT AVG(column_name) FROM table_name
```

Using AVG() function

Consider the following **Emp** table

Eid	name	Age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query to find average salary will be,

```
SELECT avg(salary) from Emp;
```

Result of the above query will be,

avg(salary)
8200

COUNT() Function

Count returns the number of rows present in the table either based on some condition or without condition.

Its general **syntax** is,

```
SELECT COUNT(column_name) FROM table-name
```

Using COUNT() function

Consider the following **Emp** table

Eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000

405	Tiger	35	8000

SQL query to count employees, satisfying specified condition is,

```
SELECT COUNT(name) FROM Emp WHERE salary = 8000;
```

Result of the above query will be,

count(name)
2

Example of COUNT(distinct)

Consider the following **Emp** table

Eid	**Name**	**age**	**salary**
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000

405	Tiger	35	8000

SQL query is,

```
SELECT COUNT(DISTINCT salary) FROM emp;
```

Result of the above query will be,

count(distinct salary)
4

FIRST() Function

First function returns first value of a selected column

Syntax for FIRST function is,

```
SELECT FIRST(column_name) FROM table-name;
```

Using FIRST() function

Consider the following **Emp** table

Eid	**name**	**age**	**salary**
401	Anu	22	9000
402	Shane	29	8000

403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query will be,

```
SELECT FIRST(salary) FROM Emp;
```

and the result will be,

first(salary)
9000

LAST() Function

LAST function returns the return last value of the selected column.

Syntax of LAST function is,

```
SELECT LAST(column_name) FROM table-name;
```

Using LAST() function

Consider the following **Emp** table

Eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query will be,

```
SELECT LAST(salary) FROM emp;
```

Result of the above query will be,

last(salary)
8000

MAX() Function

MAX function returns maximum value from selected column of the table.

Syntax of MAX function is,

```
SELECT MAX(column_name) from table-name;
```

Using MAX() function

Consider the following **Emp** table

Eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query to find the Maximum salary will be,

```
SELECT MAX(salary) FROM emp;
```

Result of the above query will be,

MAX(salary)
10000

MIN() Function

MIN function returns minimum value from a selected column of the table.

Syntax for MIN function is,

```
SELECT MIN(column_name) from table-name;
```

Using MIN() function

Consider the following **Emp** table,

Eid	name	age	Salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

SQL query to find minimum salary is,

```
SELECT MIN(salary) FROM emp;
```

Result will be,

MIN(salary)
6000

SUM() Function

SUM function returns total sum of a selected columns numeric values.

Syntax for SUM is,

```
SELECT SUM(column_name) from table-name;
```

Using SUM() function

Consider the following **Emp** table

Eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000

405	Tiger	35	8000

SQL query to find sum of salaries will be,

```
SELECT SUM(salary) FROM emp;
```

Result of above query is,

SUM(salary)
41000

SQL JOIN

SQL Join is used to fetch data from two or more tables, which is joined to appear as single set of data. It is used for combining column from two or more tables by using values common to both tables.

Types of JOIN

Following are the types of JOIN that we can use in SQL:

- Inner
- Outer
- Left
- Right

Cross JOIN or Cartesian Product

This type of JOIN returns the cartesian product of rows from the tables in Join. It will return a table which consists of records which combines each row from the first table with each row of the second table.

Cross JOIN Syntax is,

```
SELECT column-name-list
FROM
table-name1 CROSS JOIN table-name2;
```

Example of Cross JOIN

Following is the **class** table,

ID	**NAME**
1	Abhi
2	adam
4	Alex

and the **class_info** table,

ID	**Address**
1	DELHI
2	MUMBAI
3	CHENNAI

Cross JOIN query will be,

```
SELECT * FROM
class CROSS JOIN class_info;
```

The resultset table will look like,

ID	NAME	ID	Address
1	Abhi	1	DELHI
2	Adam	1	DELHI
4	Alex	1	DELHI
1	Abhi	2	MUMBAI
2	Adam	2	MUMBAI
4	Alex	2	MUMBAI
1	Abhi	3	CHENNAI
2	Adam	3	CHENNAI

4	Alex	3	CHENNAI

As you can see, this join returns the cross product of all the records present in both the tables.

INNER Join or EQUI Join

This is a simple JOIN in which the result is based on matched data as per the equality condition specified in the SQL query.

Inner Join Syntax is,

```
SELECT column-name-list FROM

table-name1 INNER JOIN table-name2

WHERE table-name1.column-name = table-name2.column-name;
```

Example of INNER JOIN

Consider a **class** table,

ID	**NAME**
1	abhi
2	adam
3	Alex

4	Anu

and the **class_info** table,

ID	**Address**
1	DELHI
2	MUMBAI
3	CHENNAI

Inner JOIN query will be,

```
SELECT * from class INNER JOIN class_info where class.id = class_info.id;
```

The resultset table will look like,

ID	**NAME**	**ID**	**Address**
1	Abhi	1	DELHI
2	Adam	2	MUMBAI
3	Alex	3	CHENNAI

Natural JOIN

Natural Join is a type of Inner join which is based on column having same name and same datatype present in both the tables to be joined.

The syntax for Natural Join is,

```
SELECT * FROM
table-name1 NATURAL JOIN table-name2;
```

Example of Natural JOIN

Here is the **class** table,

ID	NAME
1	abhi
2	Adam
3	Alex
4	Anu

and the **class_info** table,

ID	Address
1	DELHI

2	MUMBAI
3	CHENNAI

Natural join query will be,

```
SELECT * from class NATURAL JOIN class_info;
```

The result set table will look like,

ID	NAME	Address
1	Abhi	DELHI
2	Adam	MUMBAI
3	Alex	CHENNAI

In the above example, both the tables being joined have **ID** column(same name and same datatype), hence the records for which value of **ID** matches in both the tables will be the result of Natural Join of these two tables.

OUTER JOIN

Outer Join is based on both matched and unmatched data. Outer Joins subdivide further into,

1. Left Outer Join
2. Right Outer Join
3. Full Outer Join

LEFT Outer Join

The left outer join returns a result set table with the **matched data** from the two tables and then the remaining rows of the **left** table and null from the **right** table's columns.

Syntax for Left Outer Join is,

```
SELECT column-name-list FROM

table-name1 LEFT OUTER JOIN table-name2

ON table-name1.column-name = table-name2.column-name;
```

To specify a condition, we use the ON keyword with Outer Join.

Left outer Join Syntax for **Oracle** is,

```
SELECT column-name-list FROM

table-name1, table-name2 on table-name1.column-name = table-name2.column-name(+);
```

Example of Left Outer Join

Here is the **class** table,

ID	NAME
1	abhi
2	adam
3	alex

4	anu
5	ashish

and the **class_info** table,

ID	**Address**
1	DELHI
2	MUMBAI
3	CHENNAI
7	NOIDA
8	PANIPAT

Left Outer Join query will be,

```
SELECT * FROM class LEFT OUTER JOIN class_info ON (class.id = class_info.id);
```

The resultset table will look like,

ID	**NAME**	**ID**	**Address**

1	Abhi	1	DELHI
2	Adam	2	MUMBAI
3	Alex	3	CHENNAI
4	Anu	null	Null
5	Ashish	null	Null

RIGHT Outer Join

The right outer join returns a resultset table with the **matched data** from the two tables being joined, then the remaining rows of the **right** table and null for the remaining **left** table's columns.

Syntax for Right Outer Join is,

```
SELECT column-name-list FROM

table-name1 RIGHT OUTER JOIN table-name2

ON table-name1.column-name = table-name2.column-name;
```

Right outer Join Syntax for **Oracle** is,

```
SELECT column-name-list FROM

table-name1, table-name2

ON table-name1.column-name(+) = table-name2.column-name;
```

Example of Right Outer Join

Once again the **class** table,

ID	NAME
1	abhi
2	adam
3	alex
4	anu
5	ashish

and the **class_info** table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI

7	NOIDA
8	PANIPAT

Right Outer Join query will be,

```
SELECT * FROM class RIGHT OUTER JOIN class_info ON (class.id = class_info.id);
```

The resultant table will look like,

ID	NAME	ID	Address
1	abhi	1	DELHI
2	adam	2	MUMBAI
3	alex	3	CHENNAI
Null	null	7	NOIDA
Null	null	8	PANIPAT

Full Outer Join

The full outer join returns a resultset table with the **matched data** of two table then remaining rows of both **left** table and then the **right** table.

Syntax of Full Outer Join is,

```
SELECT column-name-list FROM
table-name1 FULL OUTER JOIN table-name2
ON table-name1.column-name = table-name2.column-name;
```

Example of Full outer join is,

The **class** table,

ID	**NAME**
1	Abhi
2	Adam
3	Alex
4	Anu
5	Ashish

and the **class_info** table,

ID	**Address**
1	DELHI

2	MUMBAI
3	CHENNAI
7	NOIDA
8	PANIPAT

Full Outer Join query will be like,

```
SELECT * FROM class FULL OUTER JOIN class_info ON (class.id = class_info.id);
```

The resultset table will look like,

ID	NAME	ID	Address
1	abhi	1	DELHI
2	adam	2	MUMBAI
3	alex	3	CHENNAI
4	anu	null	Null
5	ashish	null	Null

Null	null	7	NOIDA
Null	null	8	PANIPAT

SET Operations in SQL

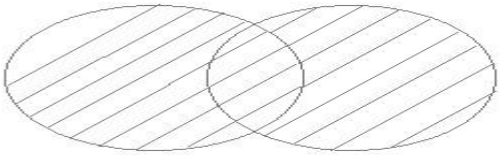
SQL supports few Set operations which can be performed on the table data. These are used to get meaningful results from data stored in the table, under different special conditions.

4 different types of SET operations.

1. UNION
2. UNION ALL
3. INTERSECT
4. MINUS

UNION Operation

UNION is used to combine the results of two or more SELECT statements. However it will eliminate duplicate rows from its result set. In case of union, number of columns and datatype must be same in both the tables, on which UNION operation is being applied.



Example of UNION

The **First** table,

ID	Name
1	Abhi
2	adam

The **Second** table,

ID	Name
2	Adam
3	Chester

Union SQL query will be,

```
SELECT * FROM First
UNION
SELECT * FROM Second;
```

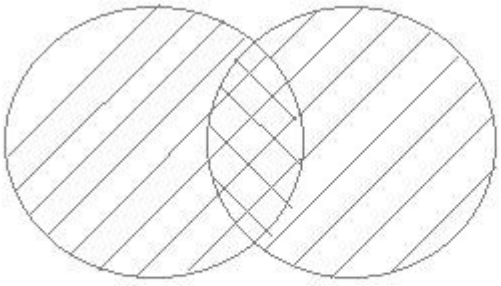
The result set table will look like,

ID	NAME

1	Abhi
2	Adam
3	Chester

UNION ALL

This operation is similar to Union. But it also shows the duplicate rows.



Example of Union All

The **First** table,

ID	**NAME**
1	Abhi
2	Adam

The **Second** table,

ID	NAME
2	Adam
3	Chester

Union All query will be like,

```
SELECT * FROM First
UNION ALL
SELECT * FROM Second;
```

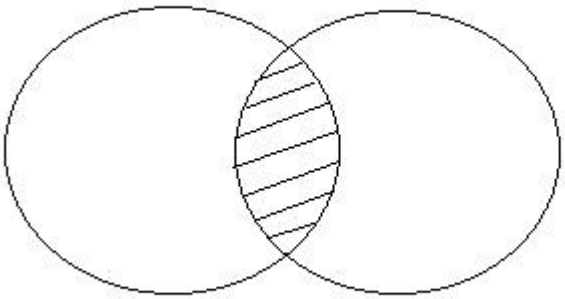
The resultset table will look like,

ID	NAME
1	Abhi
2	Adam
2	Adam
3	Chester

INTERSECT

Intersect operation is used to combine two SELECT statements, but it only returns the records which are common from both SELECT statements. In case of **Intersect** the number of columns and datatype must be same.

NOTE: MySQL does not support INTERSECT operator.



Example of Intersect

The **First** table,

ID	NAME
1	Abhi
2	Adam

The **Second** table,

ID	NAME
2	Adam

3	Chester

Intersect query will be,

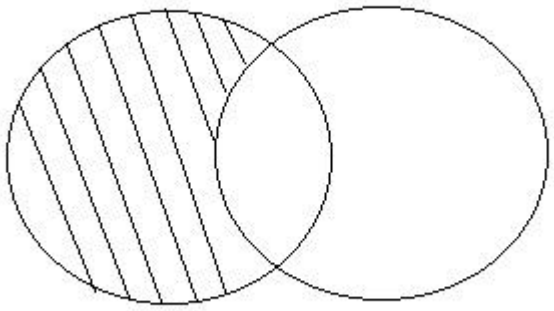
```
SELECT * FROM First
INTERSECT
SELECT * FROM Second;
```

The result set table will look like

ID	NAME
2	adam

MINUS

The Minus operation combines results of two SELECT statements and return only those in the final result, which belongs to the first set of the result.



Example of Minus

The **First** table,

ID	NAME
1	Abhi
2	Adam

The **Second** table,

ID	NAME
2	Adam
3	Chester

Minus query will be,

```
SELECT * FROM First
MINUS
SELECT * FROM Second;
```

The result set table will look like,

ID	NAME
1	abhi

SQL VIEW

A VIEW in SQL is a logical subset of data from one or more tables. View is used to restrict data access.

Syntax for creating a View,

```
CREATE or REPLACE VIEW view_name
  AS
  SELECT column_name(s)
  FROM table_name
  WHERE condition
```

As you may have understood by seeing the above SQL query, a view is created using data fetched from some other table(s). It's more like a temporary table created with data.

Creating a VIEW

Consider following **Sale** table,

Oid	order_name	previous_balance	customer
11	ord1	2000	Alex
12	ord2	1000	Adam
13	ord3	2000	Abhi
14	ord4	1000	Adam

15	ord5	2000	Alex

SQL Query to Create a View from the above table will be,

```
CREATE or REPLACE VIEW sale_view

AS

SELECT * FROM Sale WHERE customer = 'Alex';
```

The data fetched from SELECT statement will be stored in another object called **sale_view**. We can use CREATE and REPLACE seperately too, but using both together works better, as if any view with the specified name exists, this query will replace it with fresh data.

Displaying a VIEW

The syntax for displaying the data in a view is similar to fetching data from a table using a SELECT statement.

```
SELECT * FROM sale_view.
```

Oid	**order_name**	**previous_balance**	**customer**
11	ord1	2000	Alex
15	ord5	2000	Alex

Force VIEW Creation

FORCE keyword is used while creating a view, forcefully. This keyword is used to create a View even if the table does not exist. After creating a force View if we create the base table and enter values in it, the view will be automatically updated.

Syntax for forced View is,

```
CREATE or REPLACE FORCE VIEW view_name AS
   SELECT column_name(s)
   FROM table_name
   WHERE condition;
```

Update a VIEW

UPDATE command for view is same as for tables.

Syntax to Update a View is,

```
UPDATE view-name SET VALUE
WHERE condition;
```

NOTE: If we update a view it also updates base table data automatically.

Read-Only VIEW

We can create a view with read-only option to restrict access to the view.

Syntax to create a view with Read-Only Access

```
CREATE or REPLACE FORCE VIEW view_name AS
   SELECT column_name(s)
   FROM table_name
   WHERE condition WITH read-only;
```

The above syntax will create view for **read-only** purpose, we cannot Update or Insert data into read-only view. It will throw an **error**.

TCL: Transaction Control Language

These commands are to keep a check on other commands and their affect on the database. These commands can annul changes made by other commands by rolling the data back to its original state. It can also make any temporary change permanent.

Command	Description
Commit	to permanently save

Rollback	to undo change
Savepoint	to save temporarily

Commit, Rollback and Savepoint SQL commands

Transaction Control Language(TCL) commands are used to manage transactions in the database. These are used to manage the changes made to the data in a table by DML statements. It also allows statements to be grouped together into logical transactions.

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 realize that those changes were not required, then we can use the ROLLBACK command to rollback those changes, if they were not committed 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;
```

In short, using this command we can **name** the different states of our data in any table and then rollback to that state using the ROLLBACK command whenever required.

Using Savepoint and Rollback

Following is the table **class**,

Id	**name**
1	Abhi
2	Adam
4	Alex

Lets use some SQL queries on the above table and see the results.

```
INSERT INTO class VALUES(5, 'Rahul');


COMMIT;


UPDATE class SET name = 'Abhijit' WHERE id = '5';
```

```
SAVEPOINT A;

INSERT INTO class VALUES(6, 'Chris');

SAVEPOINT B;

INSERT INTO class VALUES(7, 'Bravo');

SAVEPOINT C;

SELECT * FROM class;
```

NOTE: SELECT statement is used to show the data stored in the table.

The resultant table will look like,

Id	**Name**
1	Abhi
2	Adam
4	Alex
5	Abhijit

6	Chris
7	Bravo

Now let's use the ROLLBACK command to roll back the state of data to the **savepoint B**.

```
ROLLBACK TO B;


SELECT * FROM class;
```

Now our **class** table will look like,

Id	**name**
1	Abhi
2	Adam
4	Alex
5	Abhijit
6	Chris

Now let's again use the ROLLBACK command to roll back the state of data to the **savepoint A**

```
ROLLBACK TO A;


SELECT * FROM class;
```

Now the table will look like,

Id	Name
1	Abhi
2	Adam
4	Alex
5	Abhijit

DQL: Data Query Language

Data query language is used to fetch data from tables based on conditions that we can easily apply.

Command	Description
select	retrieve records from one or more table

Using the WHERE SQL clause

WHERE clause is used to specify/apply any condition while retrieving, updating or deleting data from a table. This clause is used mostly with SELECT, UPDATE and DELETE query.

When we specify a condition using the WHERE clause then the query executes only for those records for which the condition specified by the WHERE clause is true.

Syntax for WHERE clause

Here is how you can use the WHERE clause with a DELETE statement, or any other statement,

```
DELETE FROM table_name WHERE [condition];
```

The WHERE clause is used at the end of any SQL query, to specify a condition for execution.

Example

Consider a table **student**,

s_id	name	age	Address
101	Adam	15	Chennai
102	Alex	18	Delhi
103	Abhi	17	Banglore
104	Ankit	22	Mumbai

Now we will use the SELECT statement to display data of the table, based on a condition, which we will add to our SELECT query using WHERE clause.

Let's write a simple SQL query to display the record for student with s_id as 101.

```
SELECT s_id,
   name,
   age,
   address
   FROM student WHERE s_id = 101;
```

Following will be the result of the above query.

s_id	Name	age	address
101	Adam	15	Noida

Applying condition on Text Fields

In the above example we have applied a condition to an integer value field, but what if we want to apply the condition on name field. In that case we must enclose the value in single quote ' '. Some databases even accept double quotes, but single quotes is accepted by all.

```
SELECT s_id,
   name,
   age,
   address
   FROM student WHERE name = 'Adam';
```

Following will be the result of the above query.

s_id	name	age	address
101	Adam	15	Noida

Operators for WHERE clause condition

Following is a list of operators that can be used while specifying the WHERE clause condition.

Operator	Description
=	Equal to
!=	Not Equal to
<	Less than
>	Greater than
<=	Less than or Equal to

>=	Greate than or Equal to
BETWEEN	Between a specified range of values
LIKE	This is used to search for a pattern in value.
IN	In a given set of values

SQL LIKE clause

LIKE clause is used in the condition in SQL query with the WHERE clause. LIKE clause compares data with an expression using wildcard operators to match pattern given in the condition.

Wildcard operators

There are two wildcard operators that are used in LIKE clause.

- **Percent sign %**: represents zero, one or more than one character.
- **Underscore sign _**: represents only a single character.

Example of LIKE clause

Consider the following **Student** table.

s_id	**s_Name**	**age**
101	Adam	15

102	Alex	18
103	Abhi	17

```
SELECT * FROM Student WHERE s_name LIKE 'A%';
```

The above query will return all records where **s_name** starts with character 'A'.

s_id	s_Name	age
101	Adam	15
102	Alex	18
103	Abhi	17

Using _ and %

```
SELECT * FROM Student WHERE s_name LIKE '_d%';
```

The above query will return all records from **Student** table where **s_name** contain 'd' as second character.

s_id	s_Name	age

101	Adam	15

Using % only

```
SELECT * FROM Student WHERE s_name LIKE '%x';
```

The above query will return all records from **Student** table where **s_name** contain 'x' as last character.

s_id	s_Name	age
102	Alex	18

SQL ORDER BY Clause

Order by clause is used with SELECT statement for arranging retrieved data in sorted order. The **Order by** clause by default sorts the retrieved data in ascending order. To sort the data in descending order DESC keyword is used with Order by clause.

Syntax of Order By

```
SELECT column-list|* FROM table-name ORDER BY ASC | DESC;
```

Using default Order by

Consider the following **Emp** table,

eid	name	Age	Salary

401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	10000
405	Tiger	35	8000

```
SELECT * FROM Emp ORDER BY salary;
```

The above query will return the resultant data in ascending order of the **salary**.

eid	**Name**	**Age**	**Salary**
403	Rohan	34	6000
402	Shane	29	8000
405	Tiger	35	8000
401	Anu	22	9000

404	Scott	44	10000

Using Order by DESC

Consider the **Emp** table described above,

```
SELECT * FROM Emp ORDER BY salary DESC;
```

The above query will return the resultant data in descending order of the **salary**.

eid	Name	age	Salary
404	Scott	44	10000
401	Anu	22	9000
405	Tiger	35	8000
402	Shane	29	8000
403	Rohan	34	6000

SQL Group By Clause

Group by clause is used to group the results of a SELECT query based on one or more columns. It is also used with SQL functions to group the result from one or more tables.

Syntax for using Group by in a statement.

```
SELECT column_name, function(column_name)
FROM table_name
WHERE condition
GROUP BY column_name
```

Example of Group by in a Statement

Consider the following **Emp** table.

Eid	Name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000
404	Scott	44	9000
405	Tiger	35	8000

Here we want to find **name** and **age** of employees grouped by their **salaries** or in other words, we will be grouping employees based on their salaries, hence, as a result, we

will get a data set, with unique salaries listed, along side the first employee's name and age to have that salary.

group by is used to group different row of data together based on any one column.

SQL query for the above requirement will be,

```
SELECT name, age
FROM Emp GROUP BY salary
```

Result will be,

Name	**age**
Rohan	34
Shane	29
Anu	22

Example of Group by in a Statement with WHERE clause

Consider the following **Emp** table

Eid	**Name**	**age**	**salary**
401	Anu	22	9000
402	Shane	29	8000

403	Rohan	34	6000
404	Scott	44	9000
405	Tiger	35	8000

SQL query will be,

```
SELECT name, salary
FROM Emp
WHERE age > 25
GROUP BY salary
```

Result will be.

Name	**salary**
Rohan	6000
Shane	8000
Scott	9000

You must remember that Group By clause will always come at the end of the SQL query, just like the Order by clause.

SQL HAVING Clause

Having clause is used with SQL Queries to give more precise condition for a statement. It is used to mention condition in Group by based SQL queries, just like WHERE clause is used with SELECT query.

Syntax for HAVING clause is,

```
SELECT column_name, function(column_name)
FROM table_name
WHERE column_name condition
GROUP BY column_name
HAVING function(column_name) condition
```

Example of SQL Statement using HAVING

Consider the following **Sale** table.

oid	order_name	previous_balance	customer
11	ord1	2000	Alex
12	ord2	1000	Adam
13	ord3	2000	Abhi
14	ord4	1000	Adam

15	ord5	2000	Alex

Suppose we want to find the **customer** whose **previous_balance** sum is more than **3000**.

We will use the below SQL query,

```
SELECT *

FROM sale GROUP BY customer

HAVING sum(previous_balance) > 3000
```

Result will be,

oid	**order_name**	**previous_balance**	**customer**
11	ord1	2000	Alex

The main objective of the above SQL query was to find out the name of the customer who has had a **previous_balance** more than **3000**, based on all the previous sales made to the customer, hence we get the first row in the table for customer Alex.

DISTINCT keyword

The distinct keyword is used with SELECT statement to retrieve unique values from the table. Distinct removes all the duplicate records while retrieving records from any table in the database.

Syntax for DISTINCT Keyword

```
SELECT DISTINCT column-name FROM table-name;
```

Example using DISTINCT Keyword

Consider the following **Emp** table. As you can see in the table below, there is employee **name**, along with employee **salary** and **age**.

In the table below, multiple employees have the same salary, so we will be using DISTINCT keyword to list down distinct salary amount, that is currently being paid to the employees.

Eid	Name	age	salary
401	Anu	22	5000
402	Shane	29	8000
403	Rohan	34	10000
404	Scott	44	10000
405	Tiger	35	8000

```
SELECT DISTINCT salary FROM Emp;
```

The above query will return only the unique salary from **Emp** table.

salary
5000

8000	
10000	

SQL AND & OR operator

The AND and OR operators are used with the WHERE clause to make more precise conditions for fetching data from database by combining more than one condition together.

AND operator

AND operator is used to set multiple conditions with the WHERE clause, alongside, SELECT, UPDATE or DELETE SQL queries.

Example of AND operator

Consider the following **Emp** table

eid	**Name**	**age**	**salary**
401	Anu	22	5000
402	Shane	29	8000

403	Rohan	34	12000
404	Scott	44	10000
405	Tiger	35	9000

```
SELECT * FROM Emp WHERE salary < 10000 AND age > 25
```

The above query will return records where **salary** is less than **10000** and **age** greater than **25**. Hope you get the concept here. We have used the AND operator to specify two conditions with WHERE clause.

eid	**name**	**age**	**salary**
402	Shane	29	8000
405	Tiger	35	9000

OR operator

OR operator is also used to combine multiple conditions with WHERE clause. The only difference between AND and OR is their behavior.

When we use AND to combine two or more than two conditions, records satisfying all the specified conditions will be there in the result.

But in case of OR operator, at least one condition from the conditions specified must be satisfied by any record to be in the result set.

Example of OR operator

Consider the following **Emp** table

eid	name	age	salary
401	Anu	22	5000
402	Shane	29	8000
403	Rohan	34	12000
404	Scott	44	10000
405	Tiger	35	9000

```
SELECT * FROM Emp WHERE salary > 10000 OR age > 25
```

The above query will return records where **either** salary is greater than 10000 **or** age is greater than 25.

402	Shane	29	8000
403	Rohan	34	12000
404	Scott	44	10000

405	Tiger	35	9000

SQL Constraints

SQL Constraints are rules used to limit the type of data that can go into a table, to maintain the accuracy and integrity of the data inside table.

Constraints can be divided into the following two types,

1. **Column level constraints:** Limits only column data.
2. **Table level constraints:** Limits whole table data.

Constraints are used to make sure that the integrity of data is maintained in the database. Following are the most used constraints that can be applied to a table.

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT

NOT NULL Constraint

NOT NULL constraint restricts a column from having a NULL value. Once **NOT NULL** constraint is applied to a column, you cannot pass a null value to that column. It enforces a column to contain a proper value.

One important point to note about this constraint is that it cannot be defined at table level.

Example using NOT NULL constraint

```
CREATE TABLE Student(s_id int NOT NULL, Name varchar(60), Age int);
```

The above query will declare that the **s_id** field of **Student** table will not take NULL value.

UNIQUE Constraint

UNIQUE constraint ensures that a field or column will only have unique values.
A **UNIQUE** constraint field will not have duplicate data. This constraint can be applied at column level or table level.

Using UNIQUE constraint when creating a Table (Table Level)

Here we have a simple CREATE query to create a table, which will have a column **s_id** with unique values.

```
CREATE TABLE Student(s_id int NOT NULL UNIQUE, Name varchar(60), Age int);
```

The above query will declare that the **s_id** field of **Student** table will only have unique values and wont take NULL value.

Using UNIQUE constraint after Table is created (Column Level)

```
ALTER TABLE Student ADD UNIQUE(s_id);
```

The above query specifies that **s_id** field of **Student** table will only have unique value.

Primary Key Constraint

Primary key constraint uniquely identifies each record in a database. A Primary Key must contain unique value and it must not contain null value. Usually Primary Key is used to index the data inside the table.

Using PRIMARY KEY constraint at Table Level

```
CREATE table Student (s_id int PRIMARY KEY, Name varchar(60) NOT NULL, Age int);
```

The above command will creates a PRIMARY KEY on the s_id.

Using PRIMARY KEY constraint at Column Level

```
ALTER table Student ADD PRIMARY KEY (s_id);
```

The above command will creates a PRIMARY KEY on the s_id.

Foreign Key Constraint

FOREIGN KEY is used to relate two tables. FOREIGN KEY constraint is also used to restrict actions that would destroy links between tables. To understand FOREIGN KEY, let's see its use, with help of the below tables:

Customer_Detail Table

c_id	Customer_Name	address
101	Adam	Noida
102	Alex	Delhi
103	Stuart	Rohtak

Order_Detail Table

Order_id	Order_Name	c_id
10	Order1	101

11	Order2	103
12	Order3	102

In **Customer_Detail** table, **c_id** is the primary key which is set as foreign key in **Order_Detail** table. The value that is entered in **c_id** which is set as foreign key in **Order_Detail** table must be present in **Customer_Detail** table where it is set as primary key. This prevents invalid data to be inserted into **c_id** column of **Order_Detail** table.

If you try to insert any incorrect data, DBMS will return error and will not allow you to insert the data.

Using FOREIGN KEY constraint at Table Level

```
CREATE table Order_Detail(
    order_id int PRIMARY KEY,
    order_name varchar(60) NOT NULL,
    c_id int FOREIGN KEY REFERENCES Customer_Detail(c_id)
);
```

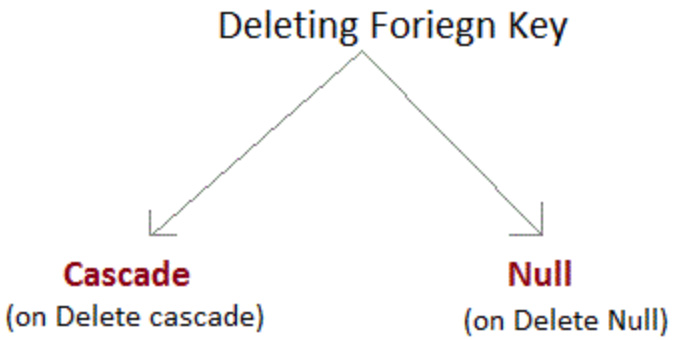
In this query, **c_id** in table Order_Detail is made as foriegn key, which is a reference of **c_id** column in Customer_Detail table.

Using FOREIGN KEY constraint at Column Level

```
ALTER table Order_Detail ADD FOREIGN KEY (c_id) REFERENCES
Customer_Detail(c_id);
```

Behaviour of Foriegn Key Column on Delete

There are two ways to maintin the integrity of data in Child table, when a particular record is deleted in the main table. When two tables are connected with Foriegn key, and certain data in the main table is deleted, for which a record exits in the child table, then we must have some mechanism to save the integrity of data in the child table.



1. **On Delete Cascade :** This will remove the record from child table, if that value of foriegn key is deleted from the main table.
2. **On Delete Null :** This will set all the values in that record of child table as NULL, for which the value of foriegn key is deleted from the main table.
3. If we don't use any of the above, then we cannot delete data from the main table for which data in child table exists. We will get an error if we try to do so.

```
ERROR : Record in child table exist
```

CHECK Constraint

CHECK constraint is used to restrict the value of a column between a range. It performs check on the values, before storing them into the database. Its like condition checking before saving data into a column.

Using CHECK constraint at Table Level

```
CREATE table Student(
```

```
    s_id int NOT NULL CHECK(s_id > 0),
    Name varchar(60) NOT NULL,
    Age int
);
```

The above query will restrict the **s_id** value to be greater than zero.

Using CHECK constraint at Column Level

```
ALTER table Student ADD CHECK(s_id > 0);
```

SQL Alias - AS Keyword

Alias is used to give an alias name to a table or a column, which can be a resultset table too. This is quite useful in case of large or complex queries. Alias is mainly used for giving a short alias name for a column or a table with complex names.

Syntax of Alias for table names,

```
SELECT column-name FROM table-name AS alias-name
```

Following is an SQL query using **alias**,

```
SELECT * FROM Employee_detail AS ed;
```

Syntax for defining alias for columns will be like,

```
SELECT column-name AS alias-name FROM table-name;
```

Example using alias for columns,

```
SELECT customer_id AS cid FROM Emp;
```

Example of Alias in SQL Query

Consider the following two tables,

The **class** table,

ID	Name
1	abhi
2	adam
3	alex
4	anu
5	ashish

and the **class_info** table,

ID	Address
1	DELHI
2	MUMBAI
3	CHENNAI
7	NOIDA
8	PANIPAT

Below is the Query to fetch data from both the tables using SQL Alias,

```
SELECT C.id, C.Name, Ci.Address from Class AS C, Class_info AS Ci where C.id = Ci.id;
```

and the resultset table will look like,

ID	Name	Address
1	abhi	DELHI
2	adam	MUMBAI

3	alex	CHENNAI

SQL Alias seems to be quite a simple feature of SQL, but it is highly useful when you are working with more than 3 tables and have to use JOIN on them.

DCL: Data Control Language

Data control language are the commands to grant and take back authority from any database user.

Command	**Description**
Grant	grant permission of right
Revoke	take back permission.

Using GRANT and REVOKE

Data Control Language(DCL) is used to control privileges in Database. To perform any operation in the database, such as for creating tables, sequences or views, a user needs privileges. Privileges are of two types,

- **System:** This includes permissions for creating session, table, etc and all types of other system privileges.
- **Object:** This includes permissions for any command or query to perform any operation on the database tables.

In DCL we have two commands,

- GRANT: Used to provide any user access privileges or other priviliges for the database.
- REVOKE: Used to take back permissions from any user.

Allow a User to create session

When we create a user in SQL, it is not even allowed to login and create a session until and unless proper permissions/priviliges are granted to the user.

Following command can be used to grant the session creating priviliges.

```
GRANT CREATE SESSION TO username;
```

Allow a User to create table

To allow a user to create tables in the database, we can use the below command,

```
GRANT CREATE TABLE TO username;
```

Provide user with space on tablespace to store table

Allowing a user to create table is not enough to start storing data in that table. We also must provide the user with priviliges to use the available tablespace for their table and data.

```
ALTER USER username QUOTA UNLIMITED ON SYSTEM;
```

The above command will alter the user details and will provide it access to unlimited tablespace on system.

NOTE: Generally unlimited quota is provided to Admin users.

Grant all privilege to a User

sysdba is a set of priviliges which has all the permissions in it. So if we want to provide all the privileges to any user, we can simply grant them the sysdba permission.

```
GRANT sysdba TO username
```

Grant permission to create any table

Sometimes user is restricted from creating come tables with names which are reserved for system tables. But we can grant privileges to a user to create any table using the below command,

```
GRANT CREATE ANY TABLE TO username
```

Grant permission to drop any table

As the title suggests, if you want to allow user to drop any table from the database, then grant this privilege to the user,

```
GRANT DROP ANY TABLE TO username
```

To take back Permissions

And, if you want to take back the privileges from any user, use the REVOKE command.

```
REVOKE CREATE TABLE FROM username
```

Scalar Functions

Scalar functions return a single value from an input value. Following are some frequently used Scalar Functions in SQL.

UCASE() Function

UCASE function is used to convert value of string column to Uppercase characters.

Syntax of UCASE,

```
SELECT UCASE(column_name) from table-name;
```

Using UCASE() function

Consider the following **Emp** table

eid	name	age	salary
401	anu	22	9000
402	shane	29	8000
403	rohan	34	6000
404	scott	44	10000
405	Tiger	35	8000

SQL query for using UCASE is,

```
SELECT UCASE(name) FROM emp;
```

Result is,

UCASE(name)
ANU
SHANE

ROHAN
SCOTT
TIGER

LCASE() Function

LCASE function is used to convert value of string columns to Lowecase characters.

Syntax for LCASE is,

```
SELECT LCASE(column_name) FROM table-name;
```

Using LCASE() function

Consider the following **Emp** table

eid	name	age	salary
401	Anu	22	9000
402	Shane	29	8000
403	Rohan	34	6000

404	SCOTT	44	10000
405	Tiger	35	8000

SQL query for converting string value to Lower case is,

```
SELECT LCASE(name) FROM emp;
```

Result will be,

LCASE(name)
anu
shane
rohan
scott
tiger

MID() Function

MID function is used to extract substrings from column values of string type in a table.

Syntax for MID function is,

```
SELECT MID(column_name, start, length) from table-name;
```

Using MID() function

Consider the following **Emp** table

eid	name	age	salary
401	anu	22	9000
402	shane	29	8000
403	rohan	34	6000
404	scott	44	10000
405	Tiger	35	8000

SQL query will be,

```
SELECT MID(name,2,2) FROM emp;
```

Result will come out to be,

MID(name,2,2)

nu
ha
oh
co
ig

ROUND() Function

ROUND function is used to round a numeric field to number of nearest integer. It is used on Decimal point values.

Syntax of Round function is,

```
SELECT ROUND(column_name, decimals) from table-name;
```

Using ROUND() function

Consider the following **Emp** table

eid	name	age	salary
401	anu	22	9000.67

402	shane	29	8000.98
403	rohan	34	6000.45
404	scott	44	10000
405	Tiger	35	8000.01

SQL query is,

```
SELECT ROUND(salary) from emp;
```

Result will be,

ROUND(salary)
9001
8001
6000
10000
8000