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

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.

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

DML: Data Manipulation Language

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

DML commands are not auto-committed. It means changes are not permanent to database, they can be rolled back.

Command	Description
insert	to insert a new row
update	to update existing row
delete	to delete a row
merge	merging two rows or two tables

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

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.

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

DDL Commands:

1. create command

create is a DDL SQL command used to create a table or a database in relational database management system.

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,

    name VARCHAR(100),

    age INT);
```

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.

If you are currently not logged into your database in which you want to create the table then you can also add the database name along with table name, using a dot operator `.`

For example, if we have a database with name **Test** and we want to create a table **Student** in it, then we can do so using the following query:

```
CREATE TABLE Test.Student(

   student_id INT,

   name VARCHAR(100),

   age INT);
```

Most commonly used data types for Table columns

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

Data type	**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`.

2. 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 data type 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: Add Column with default value

`ALTER` command can add a new column to an existing table with a default value too. The default value is used when no value is inserted in the column. Following is the syntax,

```
ALTER TABLE table_name ADD(

    column-name1 datatype1 DEFAULT some_value

);
```

Here is an Example for this,

```
ALTER TABLE student ADD(

    dob DATE DEFAULT '01-Jan-99'

);
```

The above command will add a new column with a preset default value to the table **student**.

`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**.

3. SQL Truncate, Drop or Rename a Table

In this we will learn about the various DDL commands which are used to re-define the tables.

`TRUNCATE` command

`TRUNCATE` command removes all the records from a table. But this command will not destroy the table's structure. When we use `TRUNCATE` command on a table its (auto-increment) primary key is also initialized. Following is its syntax,

```
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**.

In DML commands, we will study about the `DELETE` command which is also more or less same as the `TRUNCATE` command. We will also learn about the difference between the two in that tutorial.

`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;
```

The above query will delete the **Student** table completely. It can also be used on Databases, to delete the complete database. For example, to drop a database,

```
DROP DATABASE Test;
```

The above query will drop the database with name **Test** from the system.

`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**.

DML Commands

1. Using INSERT SQL command

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, ...)
```

Example:-

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

s_id	name	age

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

The above command will insert a new record into **student** table.

s_id	Name	age
101	Advik	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, 'Amit');
```

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, 'Amit');
```

Or,

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

The above command will insert only two column values and the other column is set to null.

S_id	S_Name	age
101	Advik	15
102	Amit	

Insert Default value to a column

```
INSERT INTO Student VALUES(103,'Chintu', default)
```

S_id	S_Name	age
101	Advik	15
102	Amit	
103	Chintu	14

Suppose the column `age` in our tabel has a default value of 14.

Also, if you run the below query, it will insert default value into the age column, whatever the default value may be.

```
INSERT INTO Student VALUES(103,'Chintu')
```

2. Using UPDATE SQL command

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.

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

student_id	**name**	**age**
101	Advik	15
102	Amit	
103	Chintu	14

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

S_id	S_Name	age
101	Advik	15
102	Amit	18
103	Chintu	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**.

Updating Multiple Columns

We can also update values of multiple columns using a single `UPDATE` statement.

```
UPDATE student SET name='Abhi', age=17 where s_id=103;
```

The above command will update two columns of the record which has `s_id` 103.

s_id	Name	age
101	Advik	15
102	Amit	18
103	Abhi	17

`UPDATE` Command: Incrementing Integer Value

When we have to update any integer value in a table, then we can fetch and update the value in the table in a single statement.

For example, if we have to update the `age` column of **student** table every year for every student, then we can simply run the following `UPDATE` statement to perform the following operation:

```
UPDATE student SET age = age+1;
```

As you can see, we have used `age = age + 1` to increment the value of age by 1.

NOTE: This style only works for integer values.

3. Using DELETE SQL command

`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	Advik	15
102	Amit	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	Advik	15
102	Amit	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.

NOTE:

Difference between DELETE, DROP and TRUNCATE

1. DELETE :
DELETE is a DML(Data Manipulation Language) command and is used when we specify the row(tuple) that we want to remove or delete from the table or relation. The DELETE command can contain a WHERE clause. If **WHERE** clause is used with DELETE command then it remove or delete only those rows(tuple) that satisfy the condition otherwise by default it removes all the tuples(rows) from the table.

Syntax of DELETE command :

```
DELETE FROM TableName

WHERE condition;
```

Note –
Here we can use the “ROLLBACK” command to restore the tuple.

2. DROP :
DROP is a DDL(Data Definition Language) command and is used to remove table definition and indexes, data, constraints, triggers etc for that table. Performance-wise the DROP command is quick to perform but slower than TRUNCATE because it gives rise to complications. Unlike DELETE we can’t rollback the data after using the DROP command. In the DROP command, table space is freed from memory because it permanently delete table as well as all its contents.
Syntax of DROP command –

```
DROP TABLE table_name;
```

Note –
Here we can’t restore the table by using the “ROLLBACK” command

3. TRUNCATE :
TRUNCATE is a DDL(Data Definition Language) command and is used to delete all the rows or tuples from a table. Unlike the DELETE command, TRUNCATE command does not contain a WHERE clause. In the TRUNCATE command, the transaction log for each deleted data page is recorded. Unlike the DELETE command, the TRUNCATE command is fast and we can’t rollback the data after using the TRUNCATE command.
Syntax of TRUNCATE command:-

```
TRUNCATE TABLE  TableName;
```

Note –
Here we can’t restore the tuples of the table by using the “ROLLBACK” command.

TCL COMMANDS

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 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;
```

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	Aditi
4	Ayush

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	Aditi
4	Ayush
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	Aditi

4	Ayush
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	Aditi
4	Ayush
5	Abhijit

So now you know how the commands `COMMIT`, `ROLLBACK` and `SAVEPOINT` works.

DCL COMMANDS

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 privileges 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/privileges are granted to the user.

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

```
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 table space 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 privileges 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 privileges 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
```

DQL Commands

`SELECT` SQL Query

`SELECT` query is used to retrieve data from a table. It is the most used SQL query. We can retrieve complete table data, or partial by specifying conditions using the `WHERE` clause.

Syntax of `SELECT` query

`SELECT` query is used to retieve records from a table. We can specify the names of the columns which we want in the resultset.

```
SELECT

    column_name1,

    column_name2,

    column_name3,

    ...
    column_nameN
    FROM table_name;
```

Example:

Consider the following **student** table,

s_id	**Name**	**Age**	**address**
101	Advik	15	Chennai
102	Amit	18	Delhi
103	Abhi	17	Banglore
104	Ankit	22	Mumbai

```
SELECT s_id, name, age FROM student;
```

The above query will fetch information of `s_id`, `name` and `age` columns of the **student** table and display them,

s_id	**Name**	**age**
101	Advik	15
102	Amit	18
103	Abhi	17

104	Ankit	22

As you can see the data from `address` column is absent, because we did not specify it in our `SELECT` query.

Select all records from a table

A special character **asterisk** `*` is used to address all the data(belonging to all columns) in a query. `SELECT` statement uses `*` character to retrieve all records from a table, for all the columns.

```
SELECT * FROM student;
```

The above query will show all the records of **student** table, that means it will show complete dataset of the table.

s_id	Name	Age	address
101	Advik	15	Chennai
102	Amit	18	Delhi
103	Abhi	17	Banglore
104	Ankit	22	Mumbai

Select a particular record based on a condition

We can use the `WHERE` clause to set a condition,

```
SELECT * FROM student WHERE name = 'Abhi';
```

The above query will return the following result,

103	Abhi	17	Banglore

Performing Simple Calculations using `SELECT` Query

Consider the following **employee** table.

eid	**Name**	**Age**	**salary**
101	Adnam	26	5000
102	Ricky	42	8000
103	Abhi	25	10000
104	Rohan	22	5000

Here is our `SELECT` query,

```
SELECT eid, name, salary+3000  FROM employee;
```

The above command will display a new column in the result, with **3000** added into existing salaries of the employees.

eid	**Name**	**salary+3000**
101	Adnam	8000
102	Ricky	11000

103	Abhi	13000
104	Rohan	8000

So you can also perform simple mathematical operations on the data too using the `SELECT` query to fetch data from 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	Advik	15	Chennai
102	Amit	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 103.

```
SELECT s_id,

    name,

    age,

    address

    FROM student WHERE s_id = 103;
```

Following will be the result of the above query.

s_id	**Name**	**Age**	**Address**
103	Abhi	17	Banglore

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 are accepted by all.

```
SELECT s_id,

    name,

    age,

    address

    FROM student WHERE name = 'Advik';
```

Following will be the result of the above query.

s_id	**Name**	**Age**	**Address**

101	Advik	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
`>=`	Greater 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	Advik	15
102	Amit	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	Advik	15

102	Amit	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	Advik	15

Using % only

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

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

s_id	**s_Name**	**age**
101	Advik	15

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	Shaan	29	8000
403	Rohan	34	6000
404	Sushant	44	10000
405	Tamanna	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	Shaan	29	8000

405	Tamanna	35	8000
401	Anu	22	9000
404	Sushant	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	Sushant	44	10000
401	Anu	22	9000
405	Tamanna	35	8000
402	Shan	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	Shaan	29	8000
403	Rohan	34	6000
404	Sushant	44	9000
405	Tamanna	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
Shaan	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	Shaan	29	8000
403	Rohan	34	6000
404	Sushant	44	9000
405	Tamanna	35	8000

SQL query will be,

```
SELECT name, salary
```

```
FROM Emp

WHERE age > 25

GROUP BY salary;
```

Result will be.

Name	salary
Rohan	6000
Shaan	8000
Sushant	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	Adnam
13	ord3	2000	Abhi
14	ord4	1000	Adnam
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	Shaan	29	8000
403	Rohan	34	10000
404	Sushant	44	10000
405	Tamanna	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	Shaan	29	8000
403	Rohan	34	12000
404	Sushant	44	10000

405	Tamanna	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**. Here we have used the `AND` operator to specify two conditions with `WHERE` clause.

Eid	**Name**	**age**	**salary**
402	Shaan	29	8000
405	Tamanna	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 behaviour.

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, atleast one condition from the conditions specified must be satisfied by any record to be in the resultset.

Example of `OR` operator

Consider the above **Emp** table.

```
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	Shaan	29	8000

403	Rohan	34	12000
404	Sushant	44	10000
405	Tamanna	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 won’t 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.

Example:-Customer_Detail Table

c_id	**Customer_Name**	**address**
101	Advik	Noida
102	Alex	Delhi
103	Suman	Indore

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)

);
```

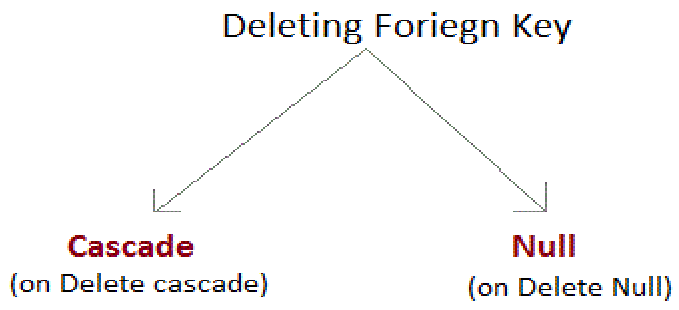
In this query, **c_id** in table **Order_Detail** is made as **foriegn key**, which is a reference of **c_id** column in **Customer_Detail** table.

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 maintain the integrity of data in Child table, when a particular record is deleted in the main table. When two tables are connected with Foreign 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 foreign 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 foreign 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);
```

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
2. Scalar Functions

Aggregate Functions

These functions **return a single value** after performing calculations on a group of values. Following are some of the frequently used Aggregate 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	Shaan	29	8000
403	Rohan	34	6000
404	Sushant	44	10000
405	Tamanna	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 above **Emp** table

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 above **Emp** table

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 above **Emp** table

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

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 above **Emp** table

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,

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 above **Emp** table

SQL query to find sum of salaries will be,

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

Result of above query is,

SUM(salary)
41000

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 above **Emp** table

SQL query for using UCASE is,

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

Result is,

UCASE(name)
ANU
SHAAN

ROHAN
SUSHANT
TAMANNA

`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 above **Emp** table

SQL query for converting string value to Lower case is,

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

Result will be,

LCASE(name)
anu
Shaan
Rohan
Sushant

Tamanna

`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

SQL query will be,

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

Result will come out to be,

MID(name,2,2)
Nu
Ha
Oh
Us
Am

`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	Shaan	29	8000.98
403	Rohan	34	6000.45
404	Sushant	44	10000
405	Tamanna	35	8000.01

SQL query is,

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

Result will be,

ROUND(salary)
9001
8001

6000
10000
8000