

# Full Stack Development Course

## -Foundation with SQL

Prepared By : Prachi Gehani

# SQL

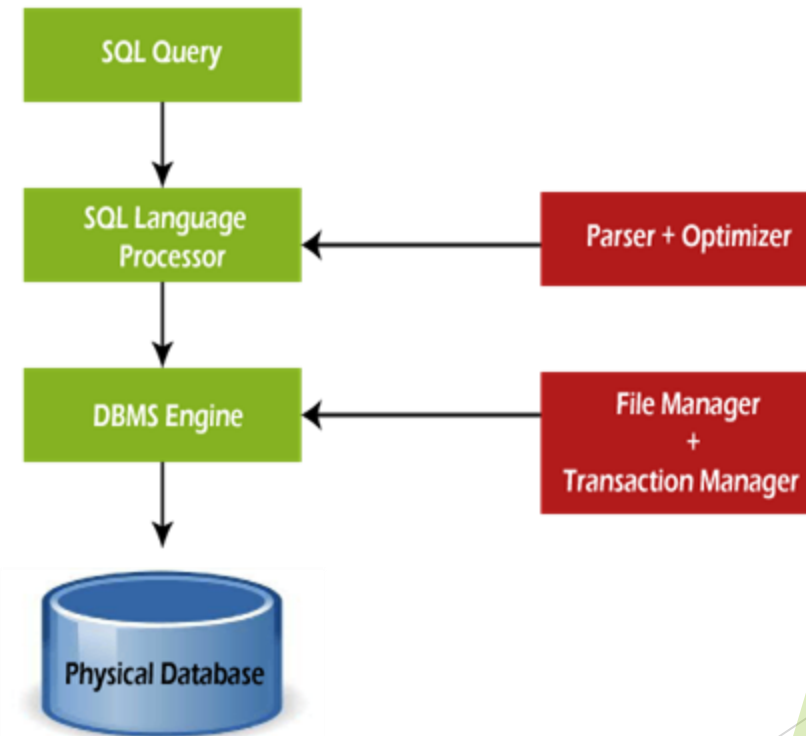
- ▶ SQL (Structured Query Language) is used to perform operations on the records stored in the database, such as updating records, inserting records, deleting records, creating and modifying database tables, views, etc.
- ▶ SQL is not a database system, but it is a query language.
- ▶ You are required to install any database management system in your systems, for example, [Oracle](#), [MySQL](#), [MongoDB](#), [PostgreSQL](#), [SQL Server](#), [DB2](#), etc.
- ▶ You can easily create and manipulate the database, access and modify the table rows and columns, etc.
- ▶ This query language became the standard of ANSI in the year of 1986 and ISO in the year of 1987.

# History of SQL

- ▶ In end of the 1970s, relational software Inc. developed their own first SQL using the concepts of E.F. Codd, Raymond Boyce, and Donald Chamberlin.
- ▶ This SQL was totally based on RDBMS. Relational Software Inc., which is now known as Oracle Corporation, introduced the Oracle V2 in June 1979, which is the first implementation of SQL language.
- ▶ This Oracle V2 version operates on VAX computers.

# Process of SQL

- ▶ When we are executing the command of SQL on any Relational database management system, then the system automatically finds the best routine to carry out our request, and the SQL engine determines how to interpret that particular command.
- ▶ Structured Query Language contains the following four components in its process:
  - Query Dispatcher
  - Optimization Engines
  - Classic Query Engine
  - SQL Query Engine, etc.
- ▶ A classic query engine allows data professionals and users to maintain non-SQL queries. The architecture of SQL is shown in the following diagram:



# Advantages of SQL

It is a perfect query language which allows data professionals and users to communicate with the database.

## **1. No programming needed**

SQL does not require a large number of coding lines for managing the database systems. We can easily access and maintain the database by using simple SQL syntactical rules. These simple rules make the SQL user-friendly.

## **2. High-Speed Query Processing**

A large amount of data is accessed quickly and efficiently from the database by using SQL queries. Insertion, deletion, and updation operations on data are also performed in less time.

## **3. Standardized Language**

SQL follows the long-established standards of ISO and ANSI, which offer a uniform platform across the globe to all its users.

## **4. Portability**

The structured query language can be easily used in desktop computers, laptops, tablets, and even smartphones. It can also be used with other applications according to the user's requirements.

## **5. Interactive language**

We can easily learn and understand the SQL language. We can also use this language for communicating with the database because it is a simple query language. This language is also used for receiving the answers to complex queries in a few seconds.

## **6. More than one Data View**

The SQL language also helps in making the multiple views of the database structure for the different database users.

## What is Relational Database?

Relational database means the data is stored as well as retrieved in the form of relations (tables). Table 1 shows the relational database with only one relation called STUDENT which stores ROLL\_NO, NAME, ADDRESS, PHONE and AGE of students.

### STUDENT

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI	9156768971	18

These are some important terminologies that are used in terms of relation.

**Attribute:** Attributes are the properties that define a relation. e.g.; ROLL\_NO, NAME etc.

**Tuple:** Each row in the relation is known as tuple. The above relation contains 4 tuples, one of which is shown as:

1 RAM DELHI 9455123451 18

**Degree:** The number of attributes in the relation is known as degree of the relation. The STUDENT relation defined above has degree 5.

**Column:** Column represents the set of values for a particular attribute. The column ROLL\_NO is extracted from relation STUDENT.

- ▶ SQL uses certain commands like Create, Drop, Insert, etc. to carry out the required tasks.
- ▶ These SQL commands are mainly categorized into four categories as:
  1. DDL – Data Definition Language
  2. DQL – Data Query Language
  3. DML – Data Manipulation Language
  4. DCL – Data Control Language

# DDL (Data Definition Language)

- ▶ DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database. DDL is a set of SQL commands used to create, modify, and delete database structures but not data. These commands are normally not used by a general user, who should be accessing the database via an application.
- ▶ List of DDL commands:
  - **CREATE**: This command is used to create the database or its objects (like table, index, function, views, store procedure, and triggers).
  - **DROP**: This command is used to delete objects from the database.
  - **ALTER**: This is used to alter the structure of the database.
  - **TRUNCATE**: This is used to remove all records from a table, including all spaces allocated for the records are removed.
  - **COMMENT**: This is used to add comments to the data dictionary.
  - **RENAME**: This is used to rename an object existing in the database.



# DQL (Data Query Language)

- ▶ **DQL** statements are used for performing queries on the data within schema objects. The purpose of the DQL Command is to get some schema relation based on the query passed to it. We can define DQL as follows it is a component of SQL statement that allows getting data from the database and imposing order upon it. It includes the **SELECT** statement. This command allows getting the data out of the database to perform operations with it. When a **SELECT** is fired against a table or tables the result is compiled into a further temporary table, which is displayed or perhaps received by the program i.e. a front-end.
- ▶ List of DQL:
  - **SELECT**: It is used to retrieve data from the database.

# DML(Data Manipulation Language):

- ▶ The SQL commands that deals with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements. It is the component of the SQL statement that controls access to data and to the database. Basically, DCL statements are grouped with DML statements.
- ▶ List of DML commands:
  - **INSERT** : It is used to insert data into a table.
  - **UPDATE**: It is used to update existing data within a table.
  - **DELETE** : It is used to delete records from a database table.
  - **LOCK**: Table control concurrency.
  - **CALL**: Call a PL/SQL or JAVA subprogram.
  - **EXPLAIN PLAN**: It describes the access path to data.

# DCL (Data Control Language)

- ▶ DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.
- ▶ List of DCL commands:
  - **GRANT**: This command gives users access privileges to the database.
  - **REVOKE**: This command withdraws the user's access privileges given by using the GRANT command.
- ▶ Though many resources claim there to be another category of SQL clauses TCL – Transaction Control Language. So we will see in detail about TCL as well. TCL commands deal with the transaction within the database.
- ▶ List of TCL commands:
  - **COMMIT**: Commits a Transaction.
  - **ROLLBACK**: Rollbacks a transaction in case of any error occurs.
  - **SAVEPOINT**: Sets a savepoint within a transaction.
  - **SET TRANSACTION**: Specify characteristics for the transaction.

# SQL Data Types

- ▶ Data types are used to represent the nature of the data that can be stored in the database table. For example, in a particular column of a table, if we want to store a string type of data then we will have to declare a string data type of this column.
- ▶ Data types mainly classified into three categories for every database.
  - String Data types
  - Numeric Data types
  - Date and time Data types

► **MySQL String Data Types :**

<b>CHAR(Size)</b>	<p>It is used to specify a fixed length string that can contain numbers, letters, and special characters. Its size can be 0 to 255 characters. Default is 1.</p> <p>In CHAR, If the length of the string is less than set or fixed-length then it is padded with extra memory space.</p>
<b>VARCHAR(Size)</b>	<p>It is used to specify a variable length string that can contain numbers, letters, and special characters. Its size can be from 0 to 65535 characters.</p> <p>In VARCHAR, If the length of the string is less than the set or fixed-length then it will store as it is without padded with extra memory spaces.</p>
<b>BINARY(Size)</b>	<p>It is equal to CHAR() but stores binary byte strings. Its size parameter specifies the column length in the bytes. Default is 1.</p> <p>Want to store a byte array or other binary data such as a stream or file then use the binary type</p>
<b>VARBINARY(Size)</b>	<p>It is equal to VARCHAR() but stores binary byte strings. Its size parameter specifies the maximum column length in bytes.</p>
<b>TEXT(Size)</b>	<p>It holds a string that can contain a maximum length of 255 characters.</p>

## MySQL Numeric Data Types

<b>BIT(Size)</b>	It is used for a bit-value type. The number of bits per value is specified in size. Its size can be 1 to 64. The default value is 1.
<b>INT(size)</b>	It is used for the integer value. Its signed range varies from -2147483648 to 2147483647 and unsigned range varies from 0 to 4294967295. The size parameter specifies the max display width that is 255.
<b>INTEGER(size)</b>	It is equal to INT(size).
<b>FLOAT(size, d)</b>	It is used to specify a floating point number. Its size parameter specifies the total number of digits. The number of digits after the decimal point is specified by <b>d</b> parameter.
<b>FLOAT(p)</b>	It is used to specify a floating point number. MySQL used p parameter to determine whether to use FLOAT or DOUBLE. If p is between 0 to 24, the data type becomes FLOAT (). If p is from 25 to 53, the data type becomes DOUBLE().
<b>DOUBLE(size, d)</b>	It is a normal size floating point number. Its size parameter specifies the total number of digits. The number of digits after the decimal is specified by d parameter.
<b>DECIMAL(size, d)</b>	It is used to specify a fixed point number. Its size parameter specifies the total number of digits. The number of digits after the decimal parameter is specified by <b>d</b> parameter. The maximum value for the size is 65, and the default value is 10. The maximum value for <b>d</b> is 30, and the default value is 0.
<b>DEC(size, d)</b>	It is equal to DECIMAL(size, d).

## MySQL Date and Time Data Types

<b>DATE</b>	It is used to specify date format YYYY-MM-DD. Its supported range is from '1000-01-01' to '9999-12-31'.
<b>DATETIME(fsp)</b>	It is used to specify date and time combination. Its format is YYYY-MM-DD hh:mm:ss. Its supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.
<b>TIMESTAMP(fsp)</b>	It is used to specify the timestamp. Its value is stored as the number of seconds since the Unix epoch('1970-01-01 00:00:00' UTC). Its format is YYYY-MM-DD hh:mm:ss. Its supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC.
<b>TIME(fsp)</b>	It is used to specify the time format. Its format is hh:mm:ss. Its supported range is from '-838:59:59' to '838:59:59'
<b>YEAR</b>	It is used to specify a year in four-digit format. Values allowed in four digit format from 1901 to 2155, and 0000.

# SQL Operators

- ▶ Every database administrator and user uses SQL queries for manipulating and accessing the data of database tables and views.
- ▶ The manipulation and retrieving of the data are performed with the help of reserved words and characters, which are used to perform arithmetic operations, logical operations, comparison operations, compound operations, etc.



# What is SQL Operator?

- ▶ The SQL reserved words and characters are called operators, which are used with a WHERE clause in a SQL query. In SQL, an operator can either be a unary or binary operator. The unary operator uses only one operand for performing the unary operation, whereas the binary operator uses two operands for performing the binary operation.
- ▶ **Syntax of Unary SQL Operator**  
Operator SQL\_Operand
- ▶ **Syntax of Binary SQL Operator**  
Operand1 SQL\_Operator Operand2

# What is the Precedence of SQL Operator?

- ▶ The precedence of SQL operators is the sequence in which the SQL evaluates the different operators in the same expression. Structured Query Language evaluates those operators first, which have high precedence.
- ▶ In the following table, the operators at the top have high precedence, and the operators that appear at the bottom have low precedence.

SQL Operator Symbols	Operators
**	Exponentiation operator
+, -	Identity operator, Negation operator
*, /	Multiplication operator, Division operator
+, -,	Addition (plus) operator, subtraction (minus) operator, String Concatenation operator
=, !=, <, >, <=, >=, IS NULL, LIKE, BETWEEN, IN	Comparison Operators
NOT	Logical negation operator
&& or AND	Conjunction operator
OR	Inclusion operator

# Types of Operator

- ▶ SQL operators are categorized in the following categories:
  1. SQL Arithmetic Operators
  2. SQL Comparison Operators
  3. SQL Logical Operators

# SQL Arithmetic Operators

- ▶ The **Arithmetic Operators** perform the mathematical operation on the numerical data of the SQL tables. These operators perform addition, subtraction, multiplication, and division operations on the numerical operands.
- ▶ **Following are the various arithmetic operators performed on the SQL data:**
  1. SQL Addition Operator (+)
  2. SQL Subtraction Operator (-)
  3. SQL Multiplication Operator (\*)
  4. SQL Division Operator (/)
  5. SQL Modulus Operator (%)

# SQL Comparison Operators

- ▶ The **Comparison Operators** in SQL compare two different data of SQL table and check whether they are the same, greater, and lesser. The SQL comparison operators are used with the WHERE clause in the SQL queries
- ▶ **Following are the various comparison operators which are performed on the data stored in the SQL database tables:**
  1. SQL Equal Operator (=)
  2. SQL Not Equal Operator (!=)
  3. SQL Greater Than Operator (>)
  4. SQL Greater Than Equals to Operator (>=)
  5. SQL Less Than Operator (<)\
  6. SQL Less Than Equals to Operator (<=)

# SQL Logical Operators

- ▶ The **Logical Operators** in SQL perform the Boolean operations, which give two results **True and False**. These operators provide **True** value if both operands match the logical condition.
- ▶ **Following are the various logical operators which are performed on the data stored in the SQL database tables:**
  1. SQL ALL operator
  2. SQL AND operator
  3. SQL OR operator
  4. SQL BETWEEN operator
  5. SQL IN operator
  6. SQL NOT operator
  7. SQL ANY operator
  8. SQL LIKE operator

# SQL Create Database

- ▶ It creates the database with the name which has been specified in the Create Database statement.

**CREATE DATABASE** Database\_Name;

- ▶ Example:

**CREATE DATABASE** ITVEDANT ;

- ▶ Verify if the database is created or not :

SHOW **DATABASES**;

To replace the existing database :

**CREATE** OR **REPLACE DATABASE** Employee;

# SQL DROP Database

- ▶ The SQL Drop Database statement deletes the existing database permanently from the database system. This statement deletes all the views and tables.

**DROP DATABASE** Database\_Name;

- ▶ **Delete multiple databases**

**DROP DATABASE** Database\_Name1, [ Database\_Name2];

**DROP DATABASE** ITVEDANT;



# Rename the existing Database

- ▶ **Rename Database** statement in SQL is used to change the name of the existing database.

For MySQL

- ▶ **RENAME DATABASE** old\_database\_name **TO** new\_database\_name;  
(not available in current version of MySQL to avoid the security risk)

For Sql

- ▶ **ALTER DATABASE** old\_database\_name **MODIFY NAME** = new\_database\_name;
- ▶ **ALTER DATABASE** ITVEDANT **MODIFY NAME** = ITVTRAINING;

# SQL SELECT Database

- ▶ Select the database on which they want to run the database queries.

`USE database_name;`

`USE ITVEDANT;`

- ▶ To view all databases :

`SHOW DATABASES;`

# SQL Table

- Table is a collection of data, organized in terms of rows and columns. In DBMS term, table is known as relation and row as tuple.

STUDENT		
NAME	ADDRESS	GRADE
Jaya	Vashi	AA
Prachi	Sanpada	AB
Navin	Nerul	BB

# SQL CREATE TABLE

- ▶ SQL CREATE TABLE statement is used to create table in a database.
- ▶ If you want to create a table, you should name the table and define its column and each column's data type.
- ▶ Let's see the simple syntax to create the table.

```
create table "tablename" ("column1" "data type", "column2" "data type", "column3" "data type", "columnN" "data type");
```

- ▶ The data type of the columns may vary from one database to another. For example, NUMBER is supported in Oracle database for integer value whereas INT is supported in MySQL.

```
CREATE TABLE Employee (EmployeeID int, FirstName varchar(255), LastName
```

```
varchar(255), Email varchar(255), AddressLine varchar(255), City varchar(255), birthdate DATE );
```

# SQL KEYS

- ▶ SQL PRIMARY KEY
- ▶ The PRIMARY KEY constraint uniquely identifies each record in a table.
- ▶ Primary keys must contain UNIQUE values, and cannot contain NULL values.
- ▶ A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

```
CREATE TABLE Persons (ID int NOT NULL, LastName  
varchar(255) NOT NULL, FirstName varchar(255), Age int, PRIMARY KEY (ID));
```

```
ALTER TABLE Persons ADD PRIMARY KEY (ID);
```

```
ALTER TABLE Persons DROP PRIMARY KEY;
```

# Primary key on multiple columns

- ▶ **CREATE TABLE** students
- ▶ (
- ▶ S\_Id **int** NOT NULL,
- ▶ LastName **varchar** (255) NOT NULL,
- ▶ FirstName **varchar** (255),
- ▶ Address **varchar** (255),
- ▶ City **varchar** (255),
- ▶ **CONSTRAINT** pk\_StudentID **PRIMARY KEY** (S\_Id, LastName)
- ▶ )

One Primary key made up of 2 columns

# SQL FOREIGN KEY

- ▶ The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.
- ▶ A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.
- ▶ The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

## Persons Table

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

## Orders Table

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

```
CREATE TABLE Orders (  
  OrderID int NOT NULL,  
  OrderNumber  
  int NOT NULL, PersonID  
  int,  
  PRIMARY KEY (OrderID),  
  FOREIGN KEY (PersonID)  
  REFERENCES Persons(PersonID)  
);
```



# SQL CHECK

- ▶ The CHECK constraint is used to limit the value range that can be placed in a column.
- ▶ If you define a CHECK constraint on a column it will allow only certain values for this column.
- ▶ If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

```
CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int  
    CHECK (Age >= 18)  
);
```

- ▶ The DEFAULT constraint is used to set a default value for a column.
- ▶ The default value will be added to all new records, if no other value is specified.

```
CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    City varchar(255) DEFAULT 'Sandnes'  
);
```

# SQL SELECT Statement

- The SELECT statement is the most commonly used command in Structured Query Language. It is used to access the records from one or more database tables and views. It also retrieves the selected data that follow the conditions we want.

**SELECT** Column\_Name\_1, Column\_Name\_2, ....., Column\_Name\_N **FROM** Table\_Name  
;

- To access all the columns :

**SELECT \* FROM** table\_name;

id	firstname	lastname	email	address	city	state
2	PRACHI	GEHANI	prachigehani@gmail.com	vashi	mumbai	mah
1	jaya	kulchandani	jayakul@gmail.com	sanpada	mumbai	mah

# SQL UNIQUE Constraint

- ▶ The UNIQUE constraint ensures that all values in a column are different.
- ▶ Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.
- ▶ A PRIMARY KEY constraint automatically has a UNIQUE constraint.
- ▶ However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

```
CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    UNIQUE (ID)  
);
```

# SQL NOT NULL Constraint

- ▶ By default, a column can hold NULL values.
- ▶ The NOT NULL constraint enforces a column to NOT accept NULL values.
- ▶ This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

```
CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255) NOT NULL,  
    Age int  
);
```

# SQL Auto-Increment

To use the auto increment field, in [MySQL](#), you have to use the AUTO\_INCREMENT keyword. The starting value for AUTO\_INCREMENT is 1 by default, and it will increment by 1 for each new record

```
Column1 DataType  AUTO_INCREMENT,  
Column2 DataType,  
);
```

```
CREATE TABLE Customers (  
CustomerID int AUTO_INCREMENT PRIMARY KEY,  
CustomerName varchar(255),  
Age int,  
PhoneNumber int);
```

Example :

```
INSERT INTO employee( empcode ) VALUES ('xyz'); UPDATE employee SET empcode = concat( empstr, empno ) ;
```

# SQL ALTER TABLE

- ▶ The ALTER TABLE statement in Structured Query Language allows you to add, modify, and delete columns of an existing table. This statement also allows database users to add and remove various SQL constraints on the existing tables.
  - ▶ **ALTER TABLE ADD Column statement in SQL**
- ALTER TABLE table\_name ADD column\_name column-definition;
- ▶ The above syntax only allows you to add a single column to the existing table. If you want to add more than one column to the table in a single SQL statement, then use the following syntax

```
ALTER TABLE table_name  
ADD (column_Name1 column-definition,  
column_Name2 column-definition,  
.....  
column_NameN column-definition);  
example : alter table emp add state varchar(255);
```

To change the existing table column size :

```
ALTER TABLE table_name MODIFY column_name varchar(new_length);
```

```
Example : create table sales(id int, product_name varchar(20), order_date date);  
alter table sales modify product_name varchar(255);
```

To add a column in a table, use the following syntax:

```
ALTER TABLE table_name
```

```
ADD column_name datatype;
```

The following SQL adds an "Email" column to the "Customers" table:

## Example

```
ALTER TABLE Customers
```

```
ADD Email varchar(255);
```

Change name of column :

```
ALTER TABLE table_name CHANGE COLUMN old_column_name new_column_name data_type;
```

```
alter table student CHANGE COLUMN NAME FNAME VARCHAR(20);
```



# SQL INSERT STATEMENT

- ▶ SQL INSERT statement is a SQL query. It is used to insert a single or a multiple records in a table.
- ▶ There are two ways to insert data in a table:
  1. By SQL insert into statement
    1. By specifying column names
    2. Without specifying column names
  2. By SQL insert into select statement
- ▶ **Inserting data directly into a table**
- ▶ You can insert a row in the table by using SQL INSERT INTO command.
- ▶ There are two ways to insert values in a table.
- ▶ **In the first method there is no need to specify the column name where the data will be inserted, you need only their values.**  
**INSERT INTO** table\_name **VALUES** (value1, value2, value3....);
- ▶ **The second method specifies both the column name and values which you want to insert.**  
**INSERT INTO** table\_name (column1, column2, column3....) **VALUES** (value1, value2, value3.....);

example:

: INSERT INTO EMP values(2,'PRACHI','GEHANI','prachigehani@gmail.com','vashi','mumbai','mah');

id	firstname	lastname	email	address	city	state
2	PRACHI	GEHANI	prachigehani@gmail.com	vashi	mumbai	mah
1	jaya	kulchandani	jayakul@gmail.com	sanpada	mumbai	mah

Insert function with column name :\

```
INSERT INTO EMP (id,fname,lname)values(2,'PRACHI','GEHANI');
```

Insert Function with date :

```
INSERT INTO EMP values(2,'PRACHI','GEHANI', '29-JUN-04');
```

# SQL WHERE

- ▶ A **WHERE clause** in SQL is a data manipulation language statement.
- ▶ WHERE clauses are not mandatory clauses of SQL DML statements. But it can be used to limit the number of rows affected by a SQL DML statement or returned by a query.
- ▶ Actually, it filters the records. It returns only those queries which fulfill the specific conditions.
- ▶ WHERE clause is used in SELECT, UPDATE, DELETE statement etc.

**SELECT** column1, **column** 2, ... **column** n

**FROM** table\_name

**WHERE** [conditions]

=	equal
>	greater than
<	less than
>=	greater than or equal
<=	less than or equal
< >	not equal to

► Syntax of SELECT Statement with WHERE clause

**SELECT** \* **FROM** Name\_of\_Table **WHERE** [condition];

Example : **Select \* from emp where id=1;**

id	firstname	lastname	email	address	city	state
1	jaya	kulchandani	jayakul@gmail.com	sanpada	mumbai	mah

- ▶ The **SQL DISTINCT command** is used with SELECT key word to retrieve only distinct or unique data.
- ▶ In a table, there may be a chance to exist a duplicate value and sometimes we want to retrieve only unique values. In such scenarios, SQL SELECT DISTINCT statement is used.

**SELECT DISTINCT** column\_name ,column\_name **FROM** table\_name

e;

id	firstname	lastname	email	address	city	state
2	PRACHI	GEHANI	prachigehani@gmail.com	vashi	mumbai	mah
1	jaya	kulchandani	jayakul@gmail.com	sanpada	mumbai	mah
4	PRACHI	desai	prachdesai@gmail.com	vashi	mumbai	mah

select **DISTINCT** firstname **from** emp;

firstname
PRACHI
jaya

# SQL AGGREGATE FUNCTION

- ▶ An aggregate function in SQL performs a calculation on multiple values and returns a single value. SQL provides many aggregate functions that include avg, count, sum, min, max, etc.
- ▶ An aggregate function ignores NULL values when it performs the calculation, except for the count function.
- ▶ Various types of SQL aggregate functions are:

Count()

Sum()

Avg()

Min()

Max()

- ▶ The **SQL COUNT()** is a function that returns the number of records of the table in the output.
- ▶ This function is used with the SQL SELECT statement.

**SELECT COUNT**(column\_name) **FROM** table\_name;

- ▶ Example:: **select count (firstname) from emp;**
- ▶ In the syntax, we have to specify the column's name after the COUNT keyword and the name of the table on which the Count function is to be executed.
- ▶ The count(\*) function in SQL shows all the Null and Non-Null records present in the table.

Syntax of Count (\*) Function in SQL

**SELECT COUNT**(\*) **FROM** table\_name;

- ▶ Example : **select count(\*) from emp;**

count(*)	
	3

- ▶ The Count Function with WHERE clause in the SELECT statement shows those records that matched the specified criteria.

**SELECT COUNT**(column\_name) **FROM** table\_name **WHERE** [condition];

- ▶ Example : **select count(firstname) from emp where firstname='prachi';**

Count
2

- ▶ The DISTINCT keyword with the COUNT function shows only the numbers of unique rows of a column.

Syntax of Count Function With DISTINCT keyword in SQL

**SELECT COUNT**(**DISTINCT** column\_name) **FROM** table\_name **WHERE** [condition];

Count (distinct firstname)
1



## ► SQL SELECT SUM

It is also known as SQL SUM() function. It is used in a SQL query to return summed value of an expression.

► Let's see the Syntax for the select sum function:

**SELECT SUM** (expression) **FROM** tables **WHERE** conditions;

► Expression may be numeric field or formula.

ID	EMPLOYEE_NAME	SALARY
1	JACK REACHER	32000
2	PADMA MAHESHWARI	22000
3	JOE PETRA	41000
4	AMBUJ AGRAWAL	21000

**SELECT SUM** (salary) **AS** "Total Salary"  
**FROM** employees  
**WHERE** salary > 20000;

SQL MAX function is used to find out the record with maximum value among a record set.

+-----+			
+-----+			
id	name	work_date	daily_typing_pages
+-----+			
1	John	2007-01-24	250
2	Ram	2007-05-27	220
3	Jack	2007-05-06	170
3	Jack	2007-04-06	100
4	Jill	2007-04-06	220
5	Zara	2007-06-06	300
5	Zara	2007-02-06	350
+-----+			

fetch maximum value of daily\_typing\_pages, then you can do so simply using the following command –

```
SQL> SELECT MAX(daily_typing_pages) FROM employee_tbl;
```

+-----+	
+-----+	
MAX(daily_typing_pages)	
+-----+	
350	
+-----+	

SQL MIN function is used to find out the record with minimum value among a record set.

+-----+			
id	name	work_date	daily_typing_pages
+-----+			
1	John	2007-01-24	250
2	Ram	2007-05-27	220
3	Jack	2007-05-06	170
3	Jack	2007-04-06	100
4	Jill	2007-04-06	220
5	Zara	2007-06-06	300
5	Zara	2007-02-06	350
+-----+			

fetch minimum value of daily\_typing\_pages, then you can do so simply using the following command –

```
SQL> SELECT MIN(daily_typing_pages) FROM employee_tbl;
```

+-----+	
MIN(daily_typing_pages)	
+-----+	
100	
+-----+	

SQL AVG function is used to find out the average of a field in various records.

+-----+			
id	name	work_date	daily_typing_pages
+-----+			
1	John	2007-01-24	250
2	Ram	2007-05-27	220
3	Jack	2007-05-06	170
3	Jack	2007-04-06	100
4	Jill	2007-04-06	220
5	Zara	2007-06-06	300
5	Zara	2007-02-06	350
+-----+			

To calculate average of all the dialy\_typing\_pages, then you can do so by using the following command –

```
SQL> SELECT AVG(daily_typing_pages) FROM employee_tbl;
```

+-----+	
AVG(daily_typing_pages)	
+-----+	
230.0000	
+-----+	
1 row in set	(0.03 sec)

SQL SUM function is used to find out the sum of a field in various records.

+-----+			
id	name	work_date	daily_typing_pages
+-----+			
1	John	2007-01-24	250
2	Ram	2007-05-27	220
3	Jack	2007-05-06	170
3	Jack	2007-04-06	100
4	Jill	2007-04-06	220
5	Zara	2007-06-06	300
5	Zara	2007-02-06	350
+-----+			

To calculate total of all the daly\_typing\_pages, then you can do so by using the following command –

```
SQL> SELECT SUM(daily_typing_pages) FROM employee_tbl;
```

+-----+	
SUM(daily_typing_pages)	
+-----+	
1610	
+-----+	

SQL SQRT function is used to find out the square root of any number. You can Use SELECT statement to find out square root of any number as follows –

```
SQL> select SQRT(16);
```

```
+-----+  
| SQRT(16) |  
+-----+  
| 4.000000 |  
+-----+
```

You can use SQRT function to find out square root of various records as well. To understand SQRT function in more detail consider, an employee\_tbl, table which is having the following records –

```
SQL> SELECT * FROM employee_tbl;
```

```
+-----+-----+-----+-----+  
| id | name | work_date | daily_typing_pages |  
+-----+-----+-----+-----+  
| 1 | John | 2007-01-24 | 250 |  
| 2 | Ram | 2007-05-27 | 220 |  
| 3 | Jack | 2007-05-06 | 170 |  
| 3 | Jack | 2007-04-06 | 100 |  
| 4 | Jill | 2007-04-06 | 220 |  
| 5 | Zara | 2007-06-06 | 300 |  
| 5 | Zara | 2007-02-06 | 350 |  
+-----+-----+-----+-----+
```

Now suppose based on the above table you want to calculate square root of all the daily\_typing\_pages, then you can do so by using the following command –

```
SQL> SELECT name, SQRT(daily_typing_pages) FROM employee_tbl;
```

```
+-----+
```

```
| name | SQRT(daily_typing_pages) |
```

```
+-----+
```

```
| John | 15.811388 |
```

```
| Ram | 14.832397 |
```

```
| Jack | 13.038405 |
```

```
| Jack | 10.000000 |
```

```
| Jill | 14.832397 |
```

```
| Zara | 17.320508 |
```

```
| Zara | 18.708287 |
```

# SQL SELECT DATE

- ▶ SQL SELECT DATE is used to retrieve a date from a database. If you want to find a particular date from a database, you can use this statement.
- ▶ MySQL comes with the following data types for storing a date or a date/time value in the database:
  - **DATE** - format YYYY-MM-DD
  - **DATETIME** - format: YYYY-MM-DD HH:MI:SS
  - **TIMESTAMP** - format: YYYY-MM-DD HH:MI:SS
  - **YEAR** - format YYYY or YY
- ▶ **For example:** let's see the query to get all the records after '2013-12-12'.

**SELECT \* FROM table-name WHERE your date-column >= '2013-12-12'**

**Example : SELECT \* FROM student WHERE doj >= '2020-12-12'**

**SELECT \* FROM table\_name WHERE your date-column BETWEEN '2012-12-12' and '2013-12-12'**

**Example : SELECT \* FROM student WHERE doj BETWEEN '2018-12-12' and '2020-12-12'**



In MySQL the default date functions are:

**NOW()**: Returns the current date and time. Example:

```
SELECT NOW();
```

Output:

2017-01-13 08:03:52

**CURDATE()**: Returns the current date. Example:

```
SELECT CURDATE();
```

Output:

2017-01-13

**CURTIME()**: Returns the current time. Example:

```
SELECT CURTIME();
```

Output:

08:05:15

**DATE():** Extracts the date part of a date or date/time expression. Example:

For the below table named 'Test'

Id	Name	BirthTime
4120	Pratik	1996-09-26 16:44:15.581

```
SELECT Name, DATE(BirthTime) AS BirthDate FROM Test;
```

Name	BirthDate
Pratik	1996-09-26

**EXTRACT():** Returns a single part of a date/time. Syntax:

EXTRACT(unit FROM date);

There are several units that can be considered but only some are used such as:

MICROSECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, YEAR, etc.

And 'date' is a valid date expression.

<b>Id</b>	<b>Name</b>	<b>BirthTime</b>
4120	Pratik	1996-09-26 16:44:15.581

### Queries

- `SELECT Name, Extract(DAY FROM BirthTime) AS BirthDay FROM Test;`

Output:

<b>Name</b>	<b>BirthDay</b>
Pratik	26

- `SELECT Name, Extract(YEAR FROM BirthTime) AS BirthYear FROM Test;`

Output:

<b>Name</b>	<b>BirthYear</b>
Pratik	1996

# SQL AND

- The SQL **AND** condition is used in SQL query to create two or more conditions to be met.
- It is used in SQL **SELECT, INSERT, UPDATE** and **DELETE**
- Let's see the syntax for SQL AND:
- SELECT columns FROM tables WHERE condition 1 AND condition 2;
- The SQL AND condition require that both conditions should be met.
- The SQL AND condition also can be used to join multiple tables in a SQL statement
- ▶ **SELECT** columns **FROM** tables **WHERE** condition 1 AND condition 2;

ID	First_Name	Last_Name	Department	Location
1	Harshad	Kuwar	Marketing	Pune
2	Anurag	Rajput	IT	Mumbai
3	Chaitali	Tarle	IT	Chennai
4	Pranjal	Patil	IT	Chennai
5	Suraj	Tripathi	Marketing	Pune
6	Roshni	Jadhav	Finance	Bangalore
7	Sandhya	Jain	Finance	Bangalore

**SELECT** \***FROM** emp **WHERE** Department = "IT" AND Location = "Chennai";

# SQL OR

- ▶ The SQL **OR** condition is used in SQL query to create a SQL statement where records are returned when any one condition met. It can be used in a **SELECT** statement, **INSERT** statement, **UPDATE** statement or **DELETE** statement.
- ▶ Let's see the syntax for the OR condition:  
**SELECT** columns **FROM** tables **WHERE** condition 1 OR condition 2;

## Example

**SELECT** \***FROM** emp **WHERE** Department = "IT" OR Location = "Chennai";

# SQL LIKE Operator

The **LIKE** operator is used in a **WHERE** clause to search for a specified pattern in a column.

There are two wildcards often used in conjunction with the **LIKE** operator:

- The percent sign (%) represents zero, one, or multiple characters
- The underscore sign (\_) represents one, single character

```
SELECT column1,  
       column2, ...  
  
FROM table_name  
  
WHERE columnN LIKE  
       pattern;
```

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

## Example :

selects all customers with a CustomerName starting with "a":

```
SELECT * FROM Customers
```

```
WHERE CustomerName LIKE 'a%';
```

selects all customers with a CustomerName ending with "a"

```
SELECT * FROM Customers
```

```
WHERE CustomerName LIKE '%a';
```

selects all customers with a CustomerName that have "or" in any position:

```
SELECT * FROM Customers
```

```
WHERE CustomerName LIKE '%or%';
```

selects all customers with a CustomerName that have "r" in the second position:

```
SELECT * FROM Customers
```

```
WHERE CustomerName LIKE ' _r%';
```

## Example :

CustomerName that starts with "a" and are at least 3 characters in length:

```
SELECT * FROM Customers
```

```
WHERE CustomerName LIKE 'a__%';
```

customers with a ContactName that starts with "a" and ends with "o":

```
SELECT * FROM Customers
```

```
WHERE ContactName LIKE 'a%o';
```

customers with a CustomerName that does NOT start with "a":

```
SELECT * FROM Customers
```

```
WHERE CustomerName NOT LIKE 'a%';
```



# SQL Wildcard Characters

A wildcard character is used to substitute one or more characters in a string.

Wildcard characters are used with the **LIKE** operator. The **LIKE** operator is used in a **WHERE** clause to

Symbol	Description	Example
*	Represents zero or more characters	bl* finds bl, black, blue, and blob
?	Represents a single character	h?t finds hot, hat, and hit
[]	Represents any single character within the brackets	h[oa]t finds hot and hat, but not hit
!	Represents any character not in the brackets	h[!oa]t finds hit, but not hot and hat
-	Represents any single character within the specified range	c[a-b]t finds cat and cbt
#	Represents any single numeric character	2#5 finds 205, 215, 225, 235, 245, 255, 265, 275, 285, and 295

selects all customers with a City starting with "ber":

```
SELECT * FROM Customers  
WHERE City LIKE 'ber%';
```

selects all customers with a City containing the pattern "es":

```
SELECT * FROM Customers  
WHERE City LIKE '%es%';
```

selects all customers with a City starting with any character, followed by "ondon":

```
SELECT * FROM Customers  
WHERE City LIKE '_ondon';
```

SQL statement selects all customers with a City starting with "L", followed by any character, followed by "n", followed by any character, followed by "on":

```
SELECT * FROM Customers  
WHERE City LIKE 'L_n_on';
```

SQL statement selects all customers with a City starting with "b", "s", or "p":

```
SELECT * FROM Customers  
WHERE City LIKE '[bsp]%';
```

selects all customers with a City starting with "a", "b", or "c":

```
SELECT * FROM Customers  
WHERE City LIKE '[a-c]%';
```

statements select all customers with a City NOT starting with "b", "s", or "p":

```
SELECT * FROM Customers  
WHERE City LIKE '[!bsp]%';
```

# SQL STRING FUNCTIONS

SQL string functions are used primarily for string manipulation. The following table details the important string functions –

## BIN(N)

Returns a string representation of the binary value of N, where N is a longlong (BIGINT) number. This is equivalent to CONV(N,10,2). Returns NULL if N is NULL.

```
SQL> SELECT BIN(12);
```

BIN(12)
1100

## BIT\_LENGTH(str)

Returns the length of the string str in bits.

```
SQL> SELECT BIT_LENGTH('text');
```

BIT_LENGTH('text')
32

## CHAR\_LENGTH(str)

Returns the length of the string str measured in characters. A multi-byte character counts as a single character. This means that for a string containing five two-byte characters, LENGTH() returns 10, whereas CHAR\_LENGTH() returns 5.

```
SQL> SELECT CHAR_LENGTH("text");
```

```
+-----+  
| CHAR_LENGTH("text") |  
+-----+  
| 4 |  
+-----+
```

## CONCAT(str1,str2,...)

Returns the string that results from concatenating the arguments. May have one or more arguments. If all arguments are non-binary strings, the result is a non-binary string. If the arguments include any binary strings, the result is a binary string. A numeric argument is converted to its equivalent binary string form; if you want to avoid that, you can use an explicit type cast, as in this example –

```
SQL> SELECT CONCAT('My', 'S', 'QL');
```

```
+-----+  
| CONCAT('My', 'S', 'QL') |  
+-----+  
| MySQL |  
+-----+
```

## CONCAT\_WS(separator,str1,str2,...)

CONCAT\_WS() stands for Concatenate With Separator and is a special form of CONCAT(). The first argument is the separator for the rest of the arguments. The separator is added between the strings to be concatenated. The separator can be a string, as can the rest of the arguments. If the separator is NULL, the result is NULL.

```
SQL> SELECT CONCAT_WS(',', 'First name', 'Last Name');
```

CONCAT_WS(',', 'First name', 'Last Name')
First name, Last Name

## FORMAT(X,D)

Formats the number X to a format like '#,###,###.##', rounded to D decimal places, and returns the result as a string. If D is 0, the result has no decimal point or fractional part.

```
SQL> SELECT FORMAT(12332.123456, 4);
```

FORMAT(12332.123456, 4)
12,332.1235

## INSERT(str,pos,len,newstr)

Returns the string str, with the substring beginning at position pos and len characters long replaced by the string newstr. Returns the original string if pos is not within the length of the string. Replaces the rest of the string from position pos if len is not within the length of the rest of the string. Returns NULL if any argument is NULL.

```
SQL> SELECT INSERT('Quadratic', 3, 4, 'What');
```

```
+-----+
```

```
| INSERT('Quadratic', 3, 4, 'What') |
```

```
+-----+
```

```
| QuWhattic |
```

```
+-----+
```

## INSTR(str,substr)

Returns the position of the first occurrence of substring substr in string str. This is the same as the two-argument form of LOCATE(), except that the order of the arguments is reversed.

```
SQL> SELECT INSTR('foobarbar', 'bar');
```

```
+-----+
```

```
| INSTR('foobarbar', 'bar') |
```

```
+-----+
```

```
| 4 |
```

```
+-----+
```

## LEFT(str,len)

Returns the leftmost len characters from the string str, or NULL if any argument is NULL.

```
SQL> SELECT LEFT('foobarbar', 5);
```

```
+-----+
| LEFT('foobarbar', 5) |
+-----+
| fooba                |
+-----+
```

## LTRIM(str)

Returns the string str with leading space characters removed.

```
SQL> SELECT LTRIM(' barbar');
```

```
+-----+
| LTRIM(' barbar') |
+-----+
| barbar           |
+-----+
```

## REPEAT(str,count)

Returns a string consisting of the string str repeated count times. If count is less than 1, returns an empty string. Returns NULL if str or count are NULL.

```
SQL> SELECT REPEAT('SQL', 3);
```

```
+-----+  
| REPEAT('SQL', 3) |  
+-----+  
| SQLSQLSQL       |  
+-----+
```

## REPLACE(str,from\_str,to\_str)

Returns the string str with all occurrences of the string from\_str replaced by the string to\_str. REPLACE() performs a case-sensitive match when searching for from\_str.

```
SQL> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
```

```
+-----+  
| REPLACE('www.mysql.com', 'w', 'Ww') |  
+-----+  
| WwWwWw.mysql.com                   |  
+-----+
```



# SQL DROP TABLE

- ▶ A SQL DROP TABLE statement is used to delete a table definition and all data from a table.
- ▶ This is very important to know that once a table is deleted all the information available in the table is lost forever, so we have to be very careful when using this command.
- ▶ Let's see the syntax to drop the table from the database.

**DROP TABLE** "table\_name";

**DROP TABLE** employee;

Check if table exists or not

**DESC** employee;

# SQL DELETE TABLE

- ▶ The DELETE statement is used to delete rows from a table. If you want to remove a specific row from a table you should use WHERE condition.

**DELETE FROM** table\_name [**WHERE** condition];

- ▶ The **DELETE statement** only deletes the rows from the table based on the condition defined by WHERE clause or delete all the rows from the table when condition is not specified.  
But it does not free the space containing by the table.

Example : Insert values

**insert into employee values**(1,'jaya','kulchandani','jayakul@gmail.com','vashi','navi mumbai');

Delete values

**delete from employee where id=1;**

# TRUNCATE statement

- ▶ The **TRUNCATE statement**: it is used to delete all the rows from the table **and free the containing space**.

**TRUNCATE TABLE** employee;

# Difference b/w DROP and TRUNCATE statements

- ▶ When you use the drop statement it deletes the table's row together with the table's definition so all the relationships of that table with other tables will no longer be valid.
- ▶ **When you drop a table:**
  - Table structure will be dropped
  - Relationship will be dropped
  - Integrity constraints will be dropped
  - Access privileges will also be dropped
- ▶ On the other hand when we **TRUNCATE** a table, the table structure remains the same, so you will not face any of the above problems.

# SQL UPDATE

- ▶ The SQL commands (*UPDATE* and *DELETE*) are used to modify the data that is already in the database. The SQL DELETE command uses a WHERE clause.
- ▶ **SQL UPDATE** statement is used to change the data of the records held by tables. Which rows is to be update, it is decided by a condition. To specify condition, we use WHERE clause.
- ▶ The UPDATE statement can be written in following form:

**UPDATE** table\_name **SET** [column\_name1= value1,... column\_nameN = valueN] [**WHERE** condition]

**UPDATE** table\_name **SET** column\_name = expression **WHERE** conditions

**UPDATE** table\_name **SET** field1 = new-value1, field2 = new-value2, [**WHERE** CLAUSE]

Student_Id	FirstName	LastName	User_Name
1	Ada	Sharma	sharmili
2	Rahul	Maurya	sofamous
3	James	Walker	jonny

**UPDATE** students  
**SET** User\_Name = 'beinghuman'  
**WHERE** Student\_Id = '3'

Student_Id	FirstName	LastName	User_Name
1	Ada	Sharma	sharmili
2	Rahul	Maurya	sofamous
3	James	Walker	<b>beinghuman</b>

# SQL RENAME TABLE

- ▶ Database administrators and users want to change the name of the table in the SQL database because they want to give a more relevant name to the table.
- ▶ Any database user can easily change the name by using the RENAME TABLE and ALTER TABLE statement in Structured Query Language.

```
RENAME TABLE old_table_name To new_table_name;
```

```
RENAME TABLE employee To emp;
```

- ▶ ALTER TABLE old\_table\_name RENAME TO new\_table\_name;
- ▶ ALTER TABLE ITVEDANT RENAME TO DATASCIENCE\_ITVEDANT;

# SQL COPY TABLE

- ▶ Copy the data of one SQL table into another SQL table in the same SQL server, then it is possible by using the SELECT INTO statement in SQL.
- ▶ The SELECT INTO statement in Structured Query Language copies the content from one existing table into the new table. SQL creates the new table by using the structure of the existing table.

```
SELECT * INTO New_table_name FROM old_table_name;
```

```
SELECT * INTO employeedetails FROM emp;
```

In mysql, use CREATE TABLE SELECT command to insert the data

Example : CREATE TABLE empdetail SELECT \* FROM emp;



# SQL ORDER BY Clause

- Whenever we want to sort the records based on the columns stored in the tables of the SQL database, then we consider using the ORDER BY clause in SQL.
- The ORDER BY clause in SQL will help us to sort the records based on the specific column of a table. This means that all the values stored in the column on which we are applying ORDER BY clause will be sorted, and the corresponding column values will be displayed in the sequence in which we have obtained the values in the earlier step.
- Using the ORDER BY clause, we can sort the records in ascending or descending order as per our requirement. The records will be sorted in ascending order whenever the **ASC keyword** is used with ORDER by clause. **DESC keyword** will sort the records in descending order.
- *If no keyword is specified after the column based on which we have to sort the records, in that case, the sorting will be done by default in the ascending order.*

► Syntax to sort the records in ascending order:

```
SELECT ColumnName1,...,ColumnNameN FROM TableName ORDER BY ColumnName ASC;
```

► Syntax to sort the records in descending order:

```
SELECT ColumnName1,...,ColumnNameN FROM TableName ORDER BY ColumnName  
DESC;
```

# SQL ORDER BY RANDOM

- ▶ If you want the resulting record to be **ordered randomly**, you should use the following codes according to several databases.
- ▶ Here is a question: what is the need to fetch a random record or a row from a database?
- ▶ Sometimes you may want to display random information like *articles, links, pages*, etc., to your user.
- ▶ If you want to fetch random rows from any of the databases, you have to use some altered queries according to the databases.
- **Select a random row with MySQL:**
- ▶ If you want to return a random row with MY SQL, use the following syntax:

**SELECT column FROM table ORDER BY RAND () LIMIT 1;**

ID	NAME	AGE	ADDRESS	SALARY
2	Shiva Tiwari	22	Bhopal	21000
3	Ajeet Bhargav	45	Meerut	65000
4	Ritesh Yadav	36	Azamgarh	26000
5	Balwant Singh	45	Varanasi	36000
6	Mahesh Sharma	26	Mathura	22000
7	Rohit Shrivastav	19	Ahemdabad	38000
8	Neeru Sharma	29	Pune	40000
9	Aakash Yadav	32	Mumbai	43500
10	Sahil Sheikh	35	Aurangabad	68800

**SELECT \*FROM** customers **ORDER BY Name ASC;**

ID	NAME	AGE	ADDRESS	SALARY
9	Aakash Yadav	32	Mumbai	43500
3	Ajeet Bhargav	45	Meerut	65000
5	Balwant Singh	45	Varanasi	36000
1	Himani Gupta	21	Modinagar	22000
6	Mahesh Sharma	26	Mathura	22000
8	Neeru Sharma	29	Pune	40000
4	Ritesh Yadav	36	Azamgarh	26000
7	Rohit Shrivastav	19	Ahemdabad	38000
10	Sahil Sheikh	35	Aurangabad	68800
2	Shiva Tiwari	22	Bhopal	21000

# The SQL GROUP BY Statement :

The **GROUP BY** statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

The **GROUP BY** statement is often used with aggregate functions (**COUNT()**, **MAX()**, **MIN()**, **SUM()**, **AVG()**) to group the result-set by one or more columns.

## GROUP BY Syntax

```
SELECT column_name(s)
```

```
FROM table_name
```

```
WHERE condition
```

```
GROUP BY column_name(s)
```

```
ORDER BY column_name(s);
```

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

statement lists the number of customers in each country:

```
SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country;
```

Statement lists the number of customers in each country, sorted high to low:

```
SELECT COUNT(CustomerID), Country FROM Customers GROUP BY Country ORDER BY  
COUNT(CustomerID) DESC;
```

# SQL HAVING Clause

The HAVING clause places the condition in the groups defined by the GROUP BY clause in the SELECT statement.

This SQL clause is implemented after the 'GROUP BY' clause in the 'SELECT' statement.

This clause is used in SQL because we cannot use the WHERE clause with the SQL aggregate functions. Both WHERE and HAVING clauses are used for filtering the records in SQL queries.

## Difference between HAVING and WHERE Clause

### HAVING Syntax

```
SELECT column_name(s) FROM table_name WHERE condition GROUP BY column_name(s)  
HAVING condition ORDER BY column_name(s);
```



CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

## SQL HAVING Examples

The following SQL statement lists the number of customers in each country. Only include countries with more than 5 customers:

```
SELECT COUNT(CustomerID), Country FROM Customers GROUP BY  
Country HAVING COUNT(CustomerID) > 5;
```

example :

```
SELECT COUNT(CustomerID), Country  
  
FROM Customers  
  
GROUP BY Country  
  
HAVING COUNT(CustomerID) > 5  
  
ORDER BY COUNT(CustomerID) DESC;
```

Below is a selection from the "Orders" table in the Northwind sample database:

And a selection from the "Employees" table:

EmployeeID	LastName	FirstName	BirthDate	Photo	Notes
1	Davolio	Nancy	1968-12-08	EmpID1.pic	Education includes a BA....
2	Fuller	Andrew	1952-02-19	EmpID2.pic	Andrew received his BTS....
3	Leverling	Janet	1963-08-30	EmpID3.pic	Janet has a BS degree....

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10248	90	5	1996-07-04	3
10249	81	6	1996-07-05	1
10250	34	4	1996-07-08	2

# SQL queries on Multiple Tables


From the following tables, write a SQL query to find the salespeople and customers who live in the same city. Return customer name, salesperson name and salesperson city.

*Sample table: salesman*

salesman_id	name	city	commission
5001	James Hoog	New York	0.15
5002	Nail Knite	Paris	0.13
5005	Pit Alex	London	0.11
5006	Mc Lyon	Paris	0.14
5007	Paul Adam	Rome	0.13
5003	Lauson Hen	San Jose	0.12

*Sample table: customer*

customer_id	cust_name	city	grade	salesman_id
3002	Nick Rimando	New York	100	5001
3007	Brad Davis	New York	200	5001
3005	Graham Zusi	California	200	5002
3008	Julian Green	London	300	5002
3004	Fabian Johnson	Paris	300	5006
3009	Geoff Cameron	Berlin	100	5003
3003	Jozv Altidor	Moscow	200	5007



```
SELECT customer.cust_name,  
salesman.name, salesman.city  
FROM salesman, customer  
WHERE salesman.city = customer.city;
```

or

```
SELECT c.cust_name,  
s.name, s.city  
FROM salesman s, customer c  
WHERE s.city = c.city;
```

write a SQL query to find those customers who placed orders on October 5, 2012. Return customer\_id, cust\_name, city, grade, salesman\_id, ord\_no, purch\_amt, ord\_date, customer\_id and salesman\_id.

*Sample table: salesman*

salesman_id	name	city	commission
5001	James Hoog	New York	0.15
5002	Nail Knite	Paris	0.13
5005	Pit Alex	London	0.11
5006	Mc Lyon	Paris	0.14
5007	Paul Adam	Rome	0.13
5003	Lauson Hen	San Jose	0.12

*Sample table: customer*

customer_id	cust_name	city	grade	salesman_id
3002	Nick Rimando	New York	100	5001
3007	Brad Davis	New York	200	5001
3005	Graham Zusi	California	200	5002
3008	Julian Green	London	300	5002
3004	Fabian Johnson	Paris	300	5006
3009	Geoff Cameron	Berlin	100	5003
3003	Jozy Altidor	Moscow	200	5007

*Sample table: orders*

ord_no	purch_amt	ord_date	customer_id	salesman_id
70001	150.5	2012-10-05	3005	5002
70009	270.65	2012-09-10	3001	5005
70002	65.26	2012-10-05	3002	5001
70004	110.5	2012-08-17	3009	5003
70007	948.5	2012-09-10	3005	5002
70005	2400.6	2012-07-27	3007	5001
70008	5760	2012-08-10	3003	5001

The background features abstract, overlapping green geometric shapes in various shades, creating a modern, layered effect on the right side of the slide.

SELECT \*

FROM customer a,orders b

WHERE a.customer\_id=b.customer\_id

AND b.ord\_date='2012-10-05';

# SQL Join

- ▶ JOIN means "**to combine two or more tables**"
- ▶ If you want to combine two or more table then SQL JOIN statement is used .it combines rows of that tables in one table and one can retrieve the information by a SELECT statement.
- ▶ The joining of two or more tables is based on common field between them.

JOINS are used to return data that is related in a relational database. Data can be related in 3 ways

- One to Many relationship (A *Person* can have many *Transactions*)
- Many to Many relationship (A *Doctor* can have many *Patients*, but a *Patient* can have more than one *Doctor*)
- One to One relationship (One *Person* can have exactly one *Passport* number)



# Types of Joins

- Inner Join
  - Natural Join
- Left (Outer) Join
- Right (Outer) Join
- (Full) Outer Join
- Cross Join
- Equi-Join

# Sample Tables

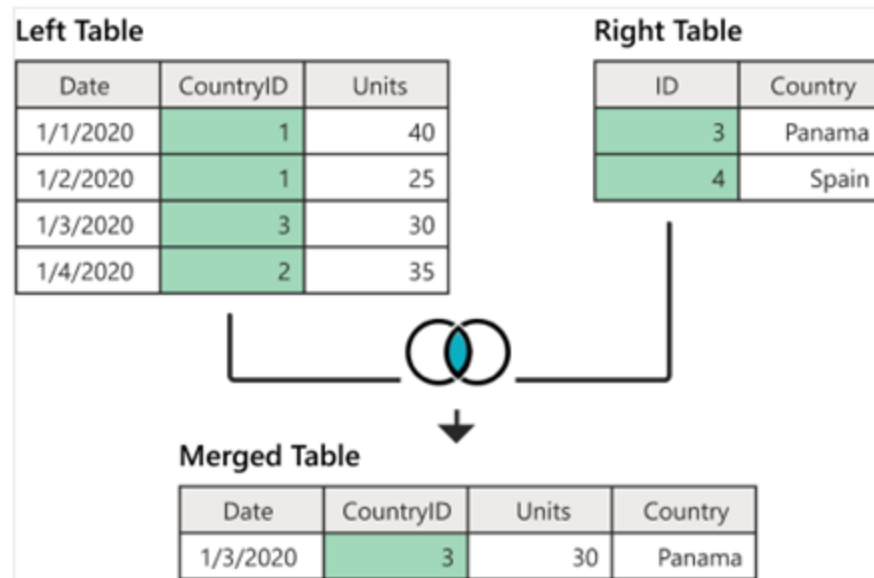
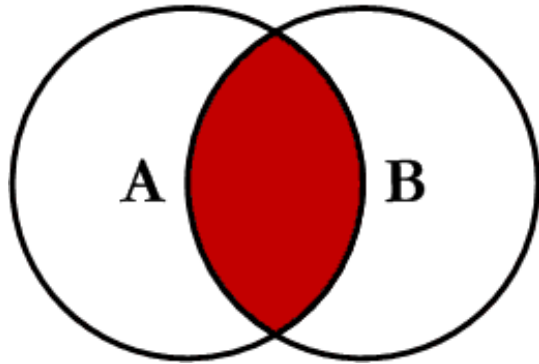
Table

A PK	Value
1	FOX
2	COP
3	TAXI
6	WASHINGTON
7	DELL
5	ARIZONA
4	LINCOLN
10	LUCENT

Table

B PK	Value
1	TROT
2	CAR
3	CAB
6	MONUMENT
7	PC
8	MICROSOFT
9	APPLE
11	SCOTCH

# Inner Join



- **Inner join** produces only the set of records that match in both Table A and Table B
- Most commonly used, best understood join

# Inner Join

Table A Value	PK	Table B PK	Value
FOX	1	1	TROT
COP	2	2	CAR
TAXI	3	3	CAB
WASHINGTON	6	6	MONUMENT
DELL	7	7	PC

SELECT \* FROM TableA **INNER JOIN** TableB ON  
TableA.PK = TableB.PK

- This is the same as doing  
SELECT \* FROM TableA, TableB **WHERE** TableA.PK =  
TableB.PK

## Inner Join (continued)

- Inner Joins do not have to use equality to join the fields
- Can use <, >, <>

## Inner Join (continued)

```
SELECT * FROM  
TableA INNER  
JOIN TableB  
ON TableA.PK  
> TableB.PK
```

Table A PK	Value	Table B PK	Value
2	COP	1	TROT
3	TAXI	1	TROT
3	TAXI	2	CAR
4	LINCOLN	1	TROT
4	LINCOLN	2	CAR
4	LINCOLN	3	CAB
5	ARIZONA	1	TROT
5	ARIZONA	2	CAR
5	ARIZONA	3	CAB
...	More...	Rows...	

# Natural Join

A natural join is a type of join operation that creates an implicit join by combining tables based on columns with the same name and data type. It is similar to the **INNER**, but we cannot use the ON or USING clause with natural join as we used in them.

## Points to remember:

- There is no need to specify the column names to join.
- The resultant table always contains unique columns.
- It is possible to perform a natural join on more than two tables.
- Do not use the ON clause.

## Syntax:

The following is a basic syntax to illustrate the natural join:

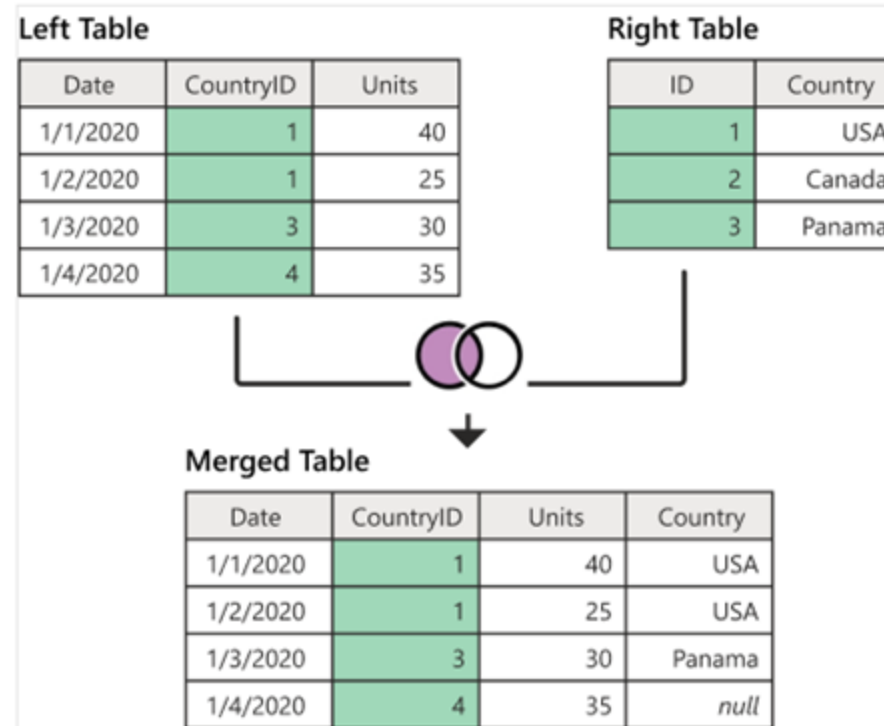
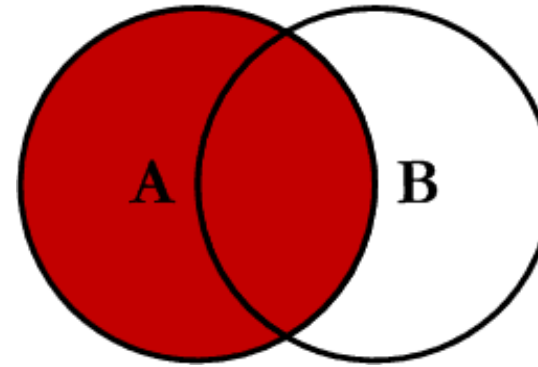
```
SELECT [column_names | *]
```

```
FROM table_name1
```

```
NATURAL JOIN table_name2;
```

# Left Outer Join

- Left outer join produces a complete set of records from Table A, with the matching records (where available) in Table B. If there is no match, the right side will contain null.





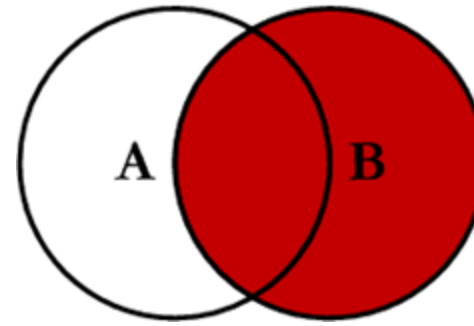
# Left Outer Join

Table A Value	PK	Table B PK	Value
FOX	1	1	TROT
COP	2	2	CAR
TAXI	3	3	CAB
LINCOLN	4	NULL	NULL
ARIZONA	5	NULL	NULL
WASHINGTON	6	6	MONUMENT
DELL	7	7	PC
LUCENT	10	NULL	NULL

- `SELECT * FROM TableA LEFT OUTER JOIN TableB ON TableA.PK = TableB.PK`

# Right Outer Join

- Right outer join produces a complete set of records from Table B, with the matching records (where available) in Table A. If there is no match, the left side will contain null.



Left Table

Date	CountryID	Units
1/1/2020	1	40
1/2/2020	1	25
1/3/2020	3	30
1/4/2020	4	35

Right Table

ID	Country
3	Panama



Merged Table

Date	CountryID	Units	Country
1/3/2020	3	30	Panama

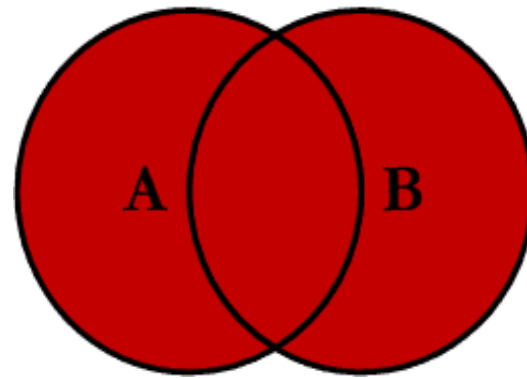
# Right Outer Join

Table A Value	PK	Table B PK	Value
FOX	1	1	TROT
COP	2	2	CAR
TAXI	3	3	CAB
WASHINGTON	6	6	MONUMENT
DELL	7	7	PC
NULL	NULL	8	MICROSOFT
NULL	NULL	9	APPLE
NULL	NULL	11	SCOTCH

- `SELECT * FROM TableA RIGHT OUTER JOIN TableB ON TableA.PK = TableB.PK`

# Full Outer Join

- Full outer join produces the set of all records in Table A and Table B, with matching records from both sides where available. If there is no match, the missing side will contain null.

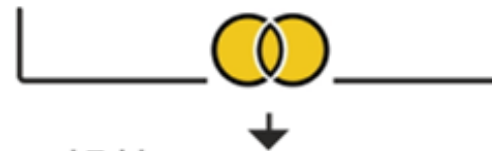


Left Table

Date	CountryID	Units
1/1/2020	1	40
1/2/2020	1	25
1/3/2020	3	30
1/4/2020	2	35

Right Table

ID	Country
1	USA
2	Canada
3	Panama
4	Spain



Merged Table

Date	CountryID	Units	Country
1/1/2020	1	40	USA
1/2/2020	1	25	USA
1/4/2020	2	35	Canada
1/3/2020	3	30	Panama
null	null	null	Spain

# Full Outer Join

TableA Value	PK	TableB PK	Value
FOX	1	1	TROT
COP	2	2	CAR
TAXI	3	3	CAB
LINCOLN	4	NULL	NULL
ARIZONA	5	NULL	NULL
WASHINGTON	6	6	MONUMENT
DELL	7	7	PC
LUCENT	10	NULL	NULL
NULL	NULL	8	MICROSOFT
NULL	NULL	9	APPLE
NULL	NULL	11	SCOTCH

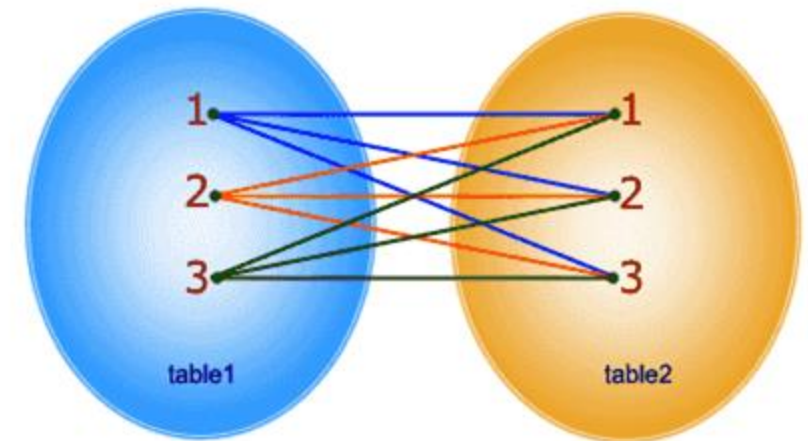
- SELECT \* FROM TableA FULL OUTER JOIN TableB ON  
TableA.PK

TableB.PK

# Cross Join

- If we use the cross join to combine two different tables, then we will get the Cartesian product of the sets of rows from the joined table.
- When each row of the first table is combined with each row from the second table, it is known as Cartesian join or cross join.
- After performing the cross join operation, the total number of rows present in the final table will be equal to the product of the number of rows present in table 1 and the number of rows present in table 2.
- **For example:**  
If there are two records in table 1 and three records in table 2, then after performing cross join operation, we will get six records in the final table.

```
SELECT * FROM table1 CROSS JOIN table2;
```



In CROSS JOIN, each row from 1st table joins with all the rows of another table.  
If 1st table contain x rows and y rows in 2nd one the result set will be  $x * y$  rows.

### Sample table: foods

ITEM_ID	ITEM_NAME	ITEM_UNIT	COMPANY_ID
1	Chex Mix	Pcs	16
6	Cheez-It	Pcs	15
2	BN Biscuit	Pcs	15
3	Mighty Munch	Pcs	17
4	Pot Rice	Pcs	15
5	Jaffa Cakes	Pcs	18
7	Salt n Shake	Pcs	

### Sample table: company

COMPANY_ID	COMPANY_NAME	COMPANY_CITY
18	Order All	Boston
15	Jack Hill Ltd	London
16	Akas Foods	Delhi
17	Foodies.	London
19	sip-n-Bite.	New York

## SQL Code:

```
SELECT foods.item_name,foods.item_unit, company.company_name,company.company_city FROM  
foods CROSS JOIN company;
```

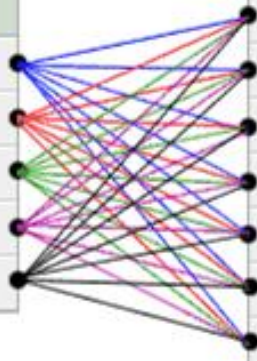
```
SELECT foods.item_name,foods.item_unit,  
company.company_name,company.company_city  
FROM foods  
CROSS JOIN company;
```

Co.ID	Co.Name	Co.City
18	Order All	Boston
15	Jack Hill Ltd	London
16	Akas Foods	Delhi
17	Foodies.	London
19	sip-n-Bite.	New York

Company

I.ID	ITEM_NAME	I.Unit	Co.ID
1	Chex Mix	Pcs	16
6	Cheez-It	Pcs	15
2	BN Biscuit	Pcs	15
3	Mighty Munch	Pcs	17
4	Pot Rice	Pcs	15
5	Jaffa Cakes	Pcs	18
7	Salt n Shake	Pcs	-

Foods





ITEM_NAME	ITEM_UNIT	COMPANY_NAME	COMPANY_CITY
Chex Mix	Pcs	Order All	Boston
Cheez-It	Pcs	Order All	Boston
BN Biscuit	Pcs	Order All	Boston
Mighty Munch	Pcs	Order All	Boston
Pot Rice	Pcs	Order All	Boston
Jaffa Cakes	Pcs	Order All	Boston
Salt n Shake	Pcs	Order All	Boston
Chex Mix	Pcs	Jack Hill Ltd	London
Cheez-It	Pcs	Jack Hill Ltd	London
BN Biscuit	Pcs	Jack Hill Ltd	London
Mighty Munch	Pcs	Jack Hill Ltd	London
Pot Rice	Pcs	Jack Hill Ltd	London
Jaffa Cakes	Pcs	Jack Hill Ltd	London
Salt n Shake	Pcs	Jack Hill Ltd	London
Chex Mix	Pcs	Akas Foods	Delhi
Cheez-It	Pcs	Akas Foods	Delhi
BN Biscuit	Pcs	Akas Foods	Delhi
Mighty Munch	Pcs	Akas Foods	Delhi
Pot Rice	Pcs	Akas Foods	Delhi
Jaffa Cakes	Pcs	Akas Foods	Delhi
Salt n Shake	Pcs	Akas Foods	Delhi
Chex Mix	Pcs	Foodies.	London
.....			

# SQL Subquery

- ▶ The Subquery or Inner query is an SQL query placed inside another SQL query. It is embedded in the HAVING or WHERE clause of the SQL statements.
- ▶ **Following are the important rules which must be followed by the SQL Subquery:**
- ▶ 1. The SQL subqueries can be used with the following statements along with the SQL expression operators:
  - ▶ SELECT statement,
  - ▶ UPDATE statement,
  - ▶ INSERT statement, and
  - ▶ DELETE statement.
- ▶ 2. The subqueries in SQL are always enclosed in the parenthesis and placed on the right side of the SQL operators
- ▶ 3. We cannot use the ORDER BY clause in the subquery. But, we can use the GROUP BY clause, which performs the same function as the ORDER BY clause.
- ▶ 4. If the subquery returns more than one record, we have to use the multiple value operators before the Subquery.
- ▶ 5. We can use the BETWEEN operator within the subquery but not with the subquery

# Subquery with SELECT statement

- ▶ In SQL, inner queries or nested queries are used most frequently with the SELECT statement. The syntax of Subquery with the SELECT statement is described in the following block:
  - ▶ `SELECT Column_Name1, Column_Name2, ..., Column_NameN FROM Table_Name WHERE Column_Name Comparison_Operator`
  - ▶ `( SELECT Column_Name1, Column_Name2, ..., Column_NameN FROM Table_Name WHERE condition;`

**Example 1:** This example uses the Greater than comparison operator with the Subquery.

Let's take the following table named Student\_Details, which contains Student\_RollNo., Stu\_Name, Stu\_Marks, and Stu\_City column.

The following SQL query returns the record of those students whose marks are greater than the average of total marks:

Student_RollNo.	Stu_Name	Stu_Marks	Stu_City
1001	Akhil	85	Agra
1002	Balram	78	Delhi
1003	Bheem	87	Gurgaon
1004	Chetan	95	Noida
1005	Diksha	99	Agra
1006	Raman	90	Ghaziabad
1007	Sheetal	68	Delhi

- ▶ `SELECT * FROM Student_Details WHERE Stu_Marks > ( SELECT AVG(Stu_Marks ) FROM Student_Details);`

Student_RollNo.	Stu_Name	Stu_Marks	Stu_City
1003	Bheem	87	Gurgaon
1004	Chetan	95	Noida
1005	Diksha	99	Agra
1006	Raman	90	Ghaziabad

**Example 2:** This example uses the IN operator with the subquery.

Let's take the following two tables named **Faculty\_Details** and **Department** tables. The **Faculty\_Details** table contains ID, Name, Dept\_ID, and address of faculties. And, the Department table contains the Dept\_ID, Faculty\_ID, and Dept\_Name.

Faculty_ID	Name	Dept_ID	Address
101	Bheem	1	Gurgaon
102	Chetan	2	Noida
103	Diksha	NULL	Agra
104	Raman	4	Ghaziabad
105	Yatin	3	Noida
106	Anuj	NULL	Agra
107	Rakes	5	Gurgaon

```
SELECT * FROM Department WHERE Faculty_ID IN ( SELECT Faculty_ID FROM Faculty  
WHERE City = 'Noida' OR City = 'Gurgaon' ) ;
```

Dept_ID	Faculty_ID	Dept_Name
1	101	BCA
2	102	B.Tech
3	105	BBA
4	104	MBA
5	107	MCA

**Output:**

Dept_ID	Faculty_ID	Dept_Name
1	101	BCA
2	102	B.Tech
3	105	BBA
5	107	MCA



# Subquery with the INSERT statement

- `INSERT INTO Table_Name SELECT * FROM Table_Name WHERE Column_Name Operator (Subquery);`

Emp_ID	Emp_Name	Emp_Salary	Address
1001	Akhil	50000	Agra
1002	Balram	25000	Delhi
1003	Bheem	45000	Gurgaon
1004	Chetan	60000	Noida
1005	Diksha	30000	Agra
1006	Raman	50000	Ghaziabad
1007	Sheetal	35000	Delhi

► Table: Old\_Employee

Emp_ID	Emp_Name	Emp_Salary	Address
1008	Sumit	50000	Agra
1009	Akash	55000	Delhi
1010	Devansh	65000	Gurgaon

```
INSERT INTO New_Employee SELECT * FROM Old_Employee WHERE Emp_Salary > 40000;
```

# Subquery with the UPDATE statement

- ▶ The subqueries and nested queries can be used with the UPDATE statement in Structured Query Language for updating the columns of the existing table.
- ▶ **Syntax of Subquery with the UPDATE statement**

```
UPDATE Table_Name SET Column_Name = New_value WHERE Value OPERATOR  
(SELECT COLUMN_NAME FROM TABLE_NAME WHERE Condition);
```

Emp_ID	Emp_Name	Emp_Salary	Dept_ID
1001	Akhil	50000	404
1002	Balram	25000	403
1003	Bheem	45000	405
1004	Chetan	60000	402
1005	Ram	65000	407
1006	Shyam	55500	NULL
1007	Shobhit	60000	NULL

**Table:**  
**Employee\_Details**

Dept_ID	Dept_Name	Emp_ID	Dept_Grade
401	Administration	1008	B
402	HR	1004	A
403	Testing	1002	A
404	Coding	1001	B
405	Sales	1003	A
406	Marketing	NULL	C
407	Accounting	1005	A

The data of Department table is shown in the below table:  
The following updates the salary of those employees whose Department Grade is A:

**UPDATE Employee\_Details SET**

**Emp\_Salary = Emp\_Salary + 5000 WHERE Emp\_ID IN ( SELECT Emp\_ID FROM Department WHERE Dept\_Grade = 'A' );**

# Subquery with the DELETE statement

- ▶ Delete one or more records from the SQL table using Subquery with the DELETE statement in Structured Query Language.
- ▶ `DELETE FROM Table_Name WHERE Value OPERATOR (SELECT COLUMN_NAME FROM TABLE_NAME WHERE Condition);`

Emp_ID	Emp_Name	Emp_Salary	Dept_ID
1001	Akhil	50000	404
1002	Balram	25000	403
1003	Bheem	45000	405
1004	Chetan	60000	402
1005	Ram	65000	407
1006	Shyam	55500	NULL
1007	Shobhit	60000	NULL
1008	Ankit	48000	401

Dept_ID	Dept_Name	Emp_ID	Dept_Grade
401	Administration	1008	C
402	HR	1004	A
403	Testing	1002	C
404	Coding	1001	B
405	Sales	1003	A
406	Marketing	NULL	C
407	Accounting	1005	C

DELETE FROM Employee\_Details WHERE Emp\_ID IN ( SELECT Emp\_ID FROM Department WHERE Dept\_Grade = 'C' );



## SQL · Using IN operator with a Multiple Row Subquery

IN operator is used to checking a value within a set of values. The list of values may come from the results returned by a subquery.

Sample table: agents

AGENT_CODE	AGENT_NAME	WORKING_AREA	COMMISSION	PHONE_NO	COUNTRY
A007	Ramasundar	Bangalore	0.15	077-25814763	
A003	Alex	London	0.13	075-12458969	
A008	Alford	New York	0.12	044-25874365	
A011	Ravi Kumar	Bangalore	0.15	077-45625874	
A010	Santakumar	Chennai	0.14	007-22388644	
A012	Lucida	San Jose	0.12	044-52981425	
A005	Anderson	Brisban	0.13	045-21447739	
A001	Subbarao	Bangalore	0.14	077-12346674	
A002	Mukesh	Mumbai	0.11	029-12358964	
A006	McDen	London	0.15	078-22255588	
A004	Ivan	Toronto	0.15	008-22544166	
A009	Benjamin	Hampshair	0.11	008-22536178	

Sample table: orders

ORD_NUM	ORD_AMOUNT	ADVANCE_AMOUNT	ORD_DATE	CUST_CODE	AGENT_CODE	ORD_DESCRIPTION
200114	3500	2000	15-AUG-08	C00002	A008	
200122	2500	400	16-SEP-08	C00003	A004	
200118	500	100	20-JUL-08	C00023	A006	
200119	4000	700	16-SEP-08	C00007	A010	
200121	1500	600	23-SEP-08	C00008	A004	
200130	2500	400	30-JUL-08	C00025	A011	
200134	4200	1800	25-SEP-08	C00004	A005	
200108	4000	600	15-FEB-08	C00008	A004	
200103	1500	700	15-MAY-08	C00021	A005	
200105	2500	500	18-JUL-08	C00025	A011	
200109	3500	800	30-JUL-08	C00011	A010	
200101	3000	1000	15-JUL-08	C00001	A008	
200111	1000	300	10-JUL-08	C00020	A008	
200104	1500	500	13-MAR-08	C00006	A004	
200106	2500	700	20-APR-08	C00005	A002	
200125	2000	600	10-OCT-08	C00018	A005	
200117	800	200	20-OCT-08	C00014	A001	
200123	500	100	16-SEP-08	C00022	A002	
200120	500	100	20-JUL-08	C00009	A002	
200116	500	100	13-JUL-08	C00010	A009	
200124	500	100	20-JUN-08	C00017	A007	
200126	500	100	24-JUN-08	C00022	A002	
200129	2500	500	20-JUL-08	C00024	A006	
200127	2500	400	20-JUL-08	C00015	A003	
200128	3500	1500	20-JUL-08	C00009	A002	
200135	2000	800	16-SEP-08	C00007	A010	
200131	900	150	26-AUG-08	C00012	A012	
200133	1200	400	29-JUN-08	C00009	A002	
200100	1000	600	08-JAN-08	C00015	A003	
200110	3000	500	15-APR-08	C00019	A010	
200107	4500	900	30-AUG-08	C00007	A010	
200112	2000	400	30-MAY-08	C00016	A007	

**in the outer query:**

'agent\_code' of 'orders' table must be in the list within IN operator in inner query :

**in inner query:**

'working\_area' of 'agents' table must be 'Bangalore',

Here is the complete SQL statement :

**SQL Code:**

```
SELECT ord_num,ord_amount,ord_date, cust_code, agent_code FROM orders WHERE agent_code IN(  
SELECT agent_code FROM agents WHERE working_area='Bangalore');
```

```
SELECT ord_num,ord_amount,ord_date,
cust_code, agent_code
FROM orders
```

```
WHERE agent_code IN
```

```
( SELECT agent_code
FROM agents
WHERE working_area='Bangalore' );
```

AGENT_CODE	WORKING_AREA
A003	London
A001	Bangalore
A009	Hampshire
A007	Bangalore
A008	New York
A011	Bangalore
A010	Chennai
A012	San Jose

agents

AGENT_CODE
A001
A007
A011

results of  
inner query

```
SELECT ord_num,ord_amount,ord_date,
cust_code, agent_code FROM orders
```

```
WHERE agent_code IN(A001,A007,A011);
```

ORD_NUM	ORD_AMOUNT	C	AGENT_CODE
200130	2500	07	A011
200105	2500	07	A011
200117	800	10	A001
200124	500	06	A007
200112	2000	05	A007

ORD_NUM	ORD_AM	AGENT_CODE
200114	3500	A008
200122	2500	A004
200121	1500	A004
200130	2500	A011
200134	4200	A005
200105	2500	A011
200109	3500	A010
200101	3000	A008
200111	1000	A008
200117	800	A001
200123	500	A002
200120	500	A002
200116	500	A009
200124	500	A007
200126	500	A002

orders

To get 'ord\_num', 'ord\_amount', 'ord\_date', 'cust\_code' and 'agent\_code' from the table 'orders' with following conditions :

**in outer query:**

'agent\_code' of 'orders' table must be other than the list within IN operator.

**in inner query :**

'working\_area' of 'agents' table must be 'Mumbai'

```
SELECT ord_num,ord_amount,ord_date,
```

```
cust_code, agent_code
```

```
FROM orders
```

```
WHERE agent_code NOT IN(
```

```
SELECT agent_code FROM agents
```

```
WHERE working_area='Bangalore');
```



A011	Bangalore
A010	Chennai
A012	San Jose

agents

```
SELECT ord_num,ord_amount,ord_date,
cust_code, agent_code FROM orders
```

```
WHERE agent_code NOT IN(A001,A007,A011);
```

ORD_NUM	ORD_AMOUNT	AGENT_CODE
200129	2500	A006
200118	500	A006
200111	1000	A008
200101	3000	A008
200114	3500	A008
200100	1000	A003
200127	2500	A003

ORD_NUM	ORD_AM	AGENT_CODE
200114	3500	A008
200122	2500	A004
200121	1500	A004
200130	2500	A011
200134	4200	A005
200105	2500	A011
200109	3500	A010
200101	3000	A008
200111	1000	A008
200117	800	A001
200123	500	A002
200120	500	A002
200116	500	A009
200124	500	A007
200126	500	A002

orders

## SQL: Using ANY with a Multiple Row Subquery

You can use the ANY operator to compare a value with any value in a list. You must place an =, <>, >, <, <= or >= operator before ANY in your query. The following example uses ANY to check if any of the agent who belongs to the country 'UK'.

To get 'agent\_code', 'agent\_name', 'working\_area', 'commission' from 'agents' table with following conditions -

**in outer query:**

'agent\_code' should be any 'agent\_code' from 'customer' table

**in inner query:**

) 'cust\_country' in the 'customer' table must be 'UK',

Here is the complete SQL statement :

**SQL Code:**

```
SELECT agent_code,agent_name,working_area,commission FROM agents WHERE agent_code=ANY( SELECT  
agent_code FROM customer WHERE cust_country='UK');
```

```
SELECT agent_code,agent_name,working_area,
commission
FROM agents
```

```
WHERE agent_code = ANY
```

```
( SELECT agent_code
FROM customer
WHERE cust_country='UK' );
```

AGENT_CODE	CUST_COUNTRY
A007	India
A003	UK
A008	USA
A008	USA
A011	India
A006	UK
A003	UK
A008	USA
A005	Australia

customer  
customer

AGENT_CODE
A009
A003
A006
A003
A006

results of  
inner query

```
SELECT agent_code,agent_name,working_area,
commission FROM agents
```

```
WHERE agent_code=ANY( any row from inner  
query );
```

AGENT_CODE	AGENT_NAME	COMMISSION
A003	Alex	.13
A001	Subbarao	.14
A009	Benjamin	.11
A007	Ramasunda	.15
A008	Alford	.12
A011	Ravi Kumar	.15
A010	Santakumar	.14
A012	Lucida	.12

agents

AGENT_CODE	AGENT_NAME	COMMISSION
A009	Benjamin	.11
A003	Alex	.13
A006	McDen	.15



# SQL Views

- ▶ In SQL, a view is a virtual table based on the result-set of an SQL statement.
- ▶ A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.
- ▶ You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.
- ▶ Hides the complexity of the data and restricts unnecessary access to the database. It permits the users to access only a particular column rather than the whole data of the table.
- ▶ Like the SQL tables, Views also store data in rows and columns, but the rows do not have any physical existence in the database.

A view is created with the CREATE VIEW statement.

```
CREATE VIEW view_name AS  
SELECT column1, column2, ...  
FROM table_name  
WHERE condition;
```

```
CREATE VIEW [Brazil Customers] AS  
SELECT CustomerName, ContactName  
FROM Customers  
WHERE Country = 'Brazil';
```

### ► SQL SELECT AS

- SQL '**AS**' is used to assign a new name temporarily to a table column or even a table.
- It makes an easy presentation of query results and allows the developer to label results more accurately without permanently renaming table columns or even the table itself.
- Let's see the syntax of select as:

```
SELECT Column_Name1 AS New_Column_Name, Column_Name2 As New_Column_Nam  
e FROM Table_Name;
```

Here, the Column\_Name is the name of a column in the original table, and the New\_Column\_Name is the name assigned to a particular column only for that specific query. This means that New\_Column\_Name is a temporary name that will be assigned to a query.

# Index

- ▶ Indexes are special lookup tables that the database search engine can use to speed up data retrieval. Simply put, an index is a pointer to data in a table. An index in a database is very similar to an index in the back of a book.
- ▶ For example, if you want to reference all pages in a book that discusses a certain topic, you first refer to the index, which lists all the topics alphabetically and are then referred to one or more specific page numbers.
- ▶ An index helps to speed up SELECT queries and WHERE clauses, but it slows down data input, with the UPDATE and the INSERT statements. Indexes can be created or dropped with no effect on the data.
- ▶ Creating an index involves the CREATE INDEX statement, which allows you to name the index, to specify the table and which column or columns to index, and to indicate whether the index is in an ascending or descending order.
- ▶ Indexes can also be unique, like the UNIQUE constraint, in that the index prevents duplicate entries in the column or combination of columns on which there is an index.

## The CREATE INDEX Command

The basic syntax of a CREATE INDEX is as follows.

```
CREATE INDEX index_name ON table_name;
```

### Single-Column Indexes

A single-column index is created based on only one table column. The basic syntax is as follows.

```
CREATE INDEX index_name ON table_name (column_name);
```

### Unique Indexes

Unique indexes are used not only for performance, but also for data integrity. A unique index does not allow any duplicate values to be inserted into the table. The basic syntax is as follows.

```
CREATE UNIQUE INDEX index_name on table_name (column_name);
```

### Composite Indexes

A composite index is an index on two or more columns of a table. Its basic syntax is as follows.

```
CREATE INDEX index_name on table_name (column1, column2);
```

Whether to create a single-column index or a composite index, take into consideration the column(s) that you may use very frequently in a query's WHERE clause as filter conditions.

Should there be only one column used, a single-column index should be the choice. Should there be two or more columns that are frequently used in the WHERE clause as filters, the composite index would be the best choice.

Thank You....