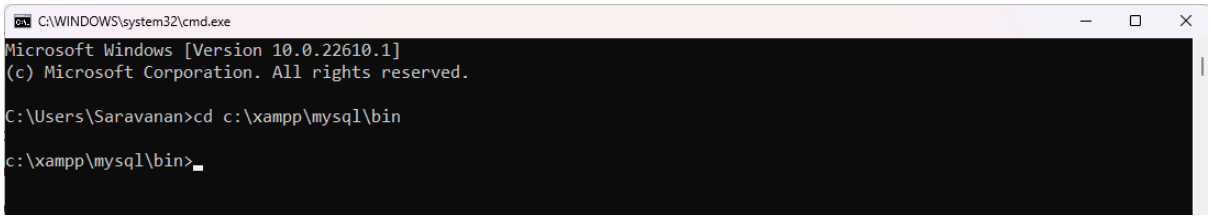
https://www.apachefriends.org/xampp-files/8.1.6/xampp-windows-x64-8.1.6-0-VS16-installer.exe

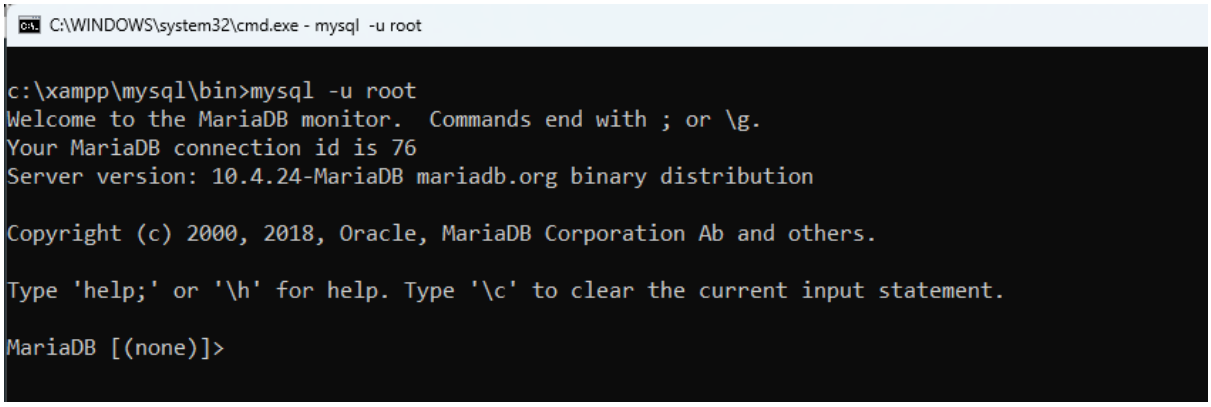
## Initial Steps to Login in the Command Prompt

## Under Run Command type CMD (Windows Key + R)


## Type CD C:\xampp\mysql\bin




## Type mysql – u root



Clear the screen:

```
MariaDB [customerdb]> \! cls
```

**Type Show Databases;**

```
MariaDB [(none)]> show databases;
+--------------------+
| Database           |
+--------------------+
| game               |
| information_schema |
| mysql              |
| performance_schema |
| phpmyadmin         |
| test               |
+--------------------+
6 rows in set (0.010 sec)
```

**Type use game;**

```
MariaDB [(none)]> use game
Database changed
MariaDB [game]>
```

**Type Show tables;**

```
MariaDB [game]> show tables;
+----------------+
| Tables_in_game |
+----------------+
| players        |
+----------------+
1 row in set (0.001 sec)
```

**Drop Database Game;**

```
MariaDB [game]> drop database game;
Query OK, 1 row affected (0.019 sec)
```

**Create Database Game;**

```
MariaDB [(none)]> create database game;
Query OK, 1 row affected (0.002 sec)
```

**Use Game:**

```
MariaDB [(none)]> use game;
Database changed
MariaDB [game]>
```

**Create Table: Create Table TableName (Column1 DataType Constraints, Column2 DataType Constraints,……ColumnN DataType Constraints);**

```
MariaDB [game]> CREATE TABLE Players(
    ->     id int NOT NULL AUTO_INCREMENT Primary Key,
    ->     PlayerName varchar(45) NOT NULL,
    ->     Matches int not null,
    ->     Runs int NOT NULL
    -> );
Query OK, 0 rows affected (0.019 sec)
```

**Show create table Players;**

**Below is Insert Command;**

```
MariaDB [game]> insert into players(PlayerName,Matches,Runs)
    -> values ('Allen',10,401),
    -> ('Peter',59=0,1802),
    -> ('Denis',45,1308),
    -> ('Tom',36,896),
    -> ('Michael',45,2856);
Query OK, 5 rows affected (0.042 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

**Select * from Players;**

```
MariaDB [game]> select * from players;
+----+------------+---------+------+
| id | PlayerName | Matches | Runs |
+----+------------+---------+------+
|  1 | Allen      |      10 |  401 |
|  2 | Peter      |       0 | 1802 |
|  3 | Denis      |      45 | 1308 |
|  4 | Tom        |      36 |  896 |
|  5 | Michael    |      45 | 2856 |
+----+------------+---------+------+
5 rows in set (0.003 sec)
```

Alter Syntax:

ALTER TABLE table_name

ADD column_name datatype;

ALTER TABLE Players

ADD Email varchar(255);

ALTER TABLE Players
DROP COLUMN Email;

ALTER TABLE Players
MODIFY Runs bigint;

ALTER TABLE table_name

CHANGE COLUMN old_column_name new_column_name Data Type;

```
MariaDB [game]> alter table players
    -> change column runs run int not null;
Query OK, 0 rows affected (0.411 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

---

DML is short name of Data Manipulation Language which deals with data manipulation and includes most common SQL statements such INSERT, UPDATE, DELETE and it is used to store, modify, retrieve, delete and update data in a database.

- **MySQL INSERT statement:** The INSERT statement allows you to insert one or more rows into a table. There are multiple ways to insert the data into the database:
    - **Insert a Single row in Table**: Following is the syntax to insert single record in the table:

      INSERT INTO table_name(c1,c2,...) VALUES (v1,v2,...);

```
MariaDB [mytestdb]> CREATE TABLE STUDENT (StudID int, Name VARCHAR(20));
Query OK, 0 rows affected (0.42 sec)
```

```
MariaDB [myTestDB]> INSERT INTO student (StudID, Name) VALUES (1, 'Sunil');
Query OK, 1 row affected (0.11 sec)
```

**Inserting data into enum:** We are creating a table to record hardware and software issues that will have priority bit as Low, Medium and High.

```
MariaDB [myTestDB]> CREATE TABLE tickets (
    ->     id INT PRIMARY KEY AUTO_INCREMENT,
    ->     title VARCHAR(255) NOT NULL,
    ->     priority ENUM('Low', 'Medium', 'High') NOT NULL
    -> );
Query OK, 0 rows affected (1.84 sec)
```

Here the records are entered using the exact keywords of the Enum.

```
MariaDB [myTestDB]> INSERT INTO tickets(title, priority)
    -> VALUES('Scan virus for computer A', 'High');
Query OK, 1 row affected (0.18 sec)
```

Here the records are getting inserted based on the order number of the enum, which means when the user enter 1 = Low, 2 = Medium and 3 = High.

```
MariaDB [myTestDB]> INSERT INTO tickets(title, priority)
    -> VALUES('Upgrade Windows OS for all computers', 1);
Query OK, 1 row affected (0.20 sec)
```

Now if you enter some other value in the ENUM column the value entered will be blank.

```
MariaDB [myTestDB]> INSERT INTO tickets(title, priority)
    -> VALUES('Install Google Chrome for Mr. John', 'Top');
Query OK, 1 row affected, 1 warning (0.09 sec)
```

```
MariaDB [myTestDB]> select * from tickets;
+----+----------------------------------------+----------+
| id | title                                  | priority |
+----+----------------------------------------+----------+
|  1 | Scan virus for computer A              | High     |
|  2 | Upgrade Windows OS for all computers   | Low      |
|  3 | Install Google Chrome for Mr. John     |          |
+----+----------------------------------------+----------+
3 rows in set (0.02 sec)
```

**Inserting data into set:** We are creating a table to record Language known by faculty members. And the same will be stored by using a set.

- **Insert in Multiple rows into a table:** To add multiple rows into a table using a single INSERT statement, you use the following syntax:

```
INSERT INTO table_name(c1,c2,...)
VALUES (v11,v12,...), (v21,v22,...),
  ... (vnn,vn2,...);
```

```
MariaDB [myTestDB]> INSERT INTO student (StudID, Name)
    -> VALUES (2, 'Vidya'),
    -> (3, 'Sreeja'),
    -> (4, 'Arpan')
    -> ;
Query OK, 3 rows affected (0.08 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

- **Insert in Multiple columns, not all columns:**

```
INSERT INTO table_name(column_names)
VALUES (values as per column_names)
```

```
MariaDB [myTestDB]> INSERT INTO people (ID, FirstName, City) VALUES
    -> (1, 'Sneha', 'Mumbai');
Query OK, 1 row affected, 1 warning (0.25 sec)
```

- **Insert using select statement:** We will observe the syntax in the DQL section.
- **Creating table and inserting data using select statement:** We will observe the syntax in the DQL section.

## Update Comment

- **MySQL Update Statement:** We use the UPDATE statement to update existing data in a table. We can use the UPDATE statement to change column values of a single row, a group of rows, or all rows in a table. Following is syntax to **update all rows in a table**:

```
UPDATE table_name
SET
  column_name1 = expr1,
  column_name2 = expr2,
  ...
```

**Single Update:**

```
MariaDB [myTestDB]> UPDATE student SET city='Mumbai';
Query OK, 4 rows affected (0.13 sec)
Rows matched: 4  Changed: 4  Warnings: 0
```

Following the syntax to **update a particular record/row**:

```
UPDATE table_name
SET
  column_name1 = expr1,
  column_name2 = expr2,
  ...
WHERE
  Condition;
```

```
MariaDB [myTestDB]> UPDATE student SET city='Thane' where StudID = 1;
Query OK, 1 row affected (0.14 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

## Multiple Update:

```
MariaDB [game]> update players set playername='John', matches=25 where id=1;
Query OK, 1 row affected (0.005 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

## Delete Statement

- **MySQL Delete Statement:** To delete data from a table, you use the MySQL DELETE statement. Following is syntax to **delete a particular record/row from table:**

  DELETE FROM table_name
  WHERE condition;

```
MariaDB [myTestDB]> DELETE FROM student
    -> where StudID = 2;
Query OK, 1 row affected (0.09 sec)
```

  Following the syntax to **delete all records from table**:

```
MariaDB [myTestDB]> DELETE FROM student;
Query OK, 3 rows affected (0.09 sec)
```

**Difference between DROP, DELETE and TRUNCATE:**
**DROP**: Command removes a table from the database. All the tables' rows, indexes and privileges will also be removed. No DML triggers will be fired. The operation cannot be rolled back.
**TRUNCATE:** Removes all rows from a table. The operation cannot be rolled back and no triggers will be fired. As such, TRUNCATE is faster and doesn't use as much undo space as a DELETE.
**DELETE:** Command is used to remove rows from a table. A WHERE clause can be used to only remove some rows. If no WHERE condition is specified, all rows will be removed. After performing a DELETE operation, you need to COMMIT or ROLLBACK the transaction to make the change permanent or to undo it.

Synatax:

TRUNCATE [TABLE] table_name;

Example:

TRUNCATE TABLE books;

## Comment Statement

### Single Line

Example

-- Select all:

```
SELECT * FROM Customers;
SELECT * FROM Customers -- WHERE City='Berlin';
```

**Multi-line Comments**

Example

```
/*Select all the columns
of all the records
in the Customers table:*/
SELECT * FROM Customers;
SELECT CustomerName, /*City,*/ Country FROM Customers;
```

Rename Syntax

Following is the basic syntax of the RENAME TABLE statement −

```
RENAME TABLE table_name TO new_name;
```

Example:

```
RENAME TABLE Demo to Players;
```

Drop Table

```
DROP TABLE  table_name;
```

## Constraints

- **NOT NULL Constraint**: A column with a NOT NULL constraint, cannot have NULL values.

```
MariaDB [mytestdb]> CREATE TABLE People(Id INTEGER, LastName TEXT NOT NULL,
    -> FirstName TEXT NOT NULL,
    -> City VARCHAR(55));
Query OK, 0 rows affected (0.39 sec)
```

- **UNIQUE:** The UNIQUE constraint ensures that all data are unique in a column.

```
MariaDB [mytestdb]> CREATE TABLE Brands(Id INTEGER, BrandName VARCHAR(30) UNIQUE);
Query OK, 0 rows affected (0.40 sec)
```

- **PRIMARY KEY:** The PRIMARY KEY constraint uniquely identifies each record in a database table. It is a special case of unique keys. Primary keys cannot be NULL, unique keys can be. There can be more UNIQUE columns, but only one primary key in a table. Primary keys are important when designing the database tables. Primary keys are unique ids. We use them to refer to table rows. Primary keys become foreign keys in other tables, when creating relations among tables.

```
MariaDB [mytestdb]> CREATE TABLE Brands(Id INTEGER PRIMARY KEY, BrandName VARCHAR(30) UNIQUE);
Query OK, 0 rows affected (0.50 sec)
```

- **FOREIGN KEY:** A FOREIGN KEY in one table points to a PRIMARY KEY in another table. It is a referential constraint between two tables. The foreign key identifies a column or a set of columns in one (referencing) table that refers to a column or set of columns in another (referenced) table.

```
MariaDB [mytestdb]> CREATE TABLE Authors(AuthorId INTEGER PRIMARY KEY,
    -> Name VARCHAR(70));
Query OK, 0 rows affected (0.35 sec)

MariaDB [mytestdb]> CREATE TABLE Books(BookId INTEGER PRIMARY KEY,
    -> Title VARCHAR(50), AuthorId INTEGER,
    -> FOREIGN KEY(AuthorId) REFERENCES Authors(AuthorId));
Query OK, 0 rows affected (0.36 sec)
```

- **ENUM:** An ENUM is a string object with a value chosen from a list of permitted values. They are enumerated explicitly in the column specification at table creation time.

```
MariaDB [mytestdb]> CREATE TABLE Shops(Id INTEGER, Name VARCHAR(55),
    -> Quality ENUM('High', 'Average', 'Low'));
Query OK, 0 rows affected (0.41 sec)
```

- **SET:** A SET can have zero or more values. Each of the values must be chosen from a list of permitted values.

```
MariaDB [mytestdb]> CREATE TABLE Students(Id INTEGER, Name VARCHAR(55),
    -> Certificates SET('A1', 'A2', 'B1', 'C1'));
Query OK, 0 rows affected (0.39 sec)
```

**Checking all the existing tables in the database**

```
MariaDB [myTestDB]> SHOW TABLES;
+--------------------+
| Tables_in_mytestdb |
+--------------------+
| authors            |
| books              |
| brands             |
| people             |
| shops              |
| student            |
| students           |
| tasks              |
| test               |
| test2              |
| work_items         |
+--------------------+
11 rows in set (0.00 sec)
```

# Importing Table thru Command Prompt

***Create Database:***

```
MariaDB [(none)]> create database customerdb;
Query OK, 1 row affected (0.044 sec)
```

***Use Table:***

```
MariaDB [(none)]> use customerdb;
```

***Create Table:***

*create table Customer(CustomerID int not null primary key,*

*CustomerName varchar(50), ContactName varchar(50),*

*Address varchar(255), City varchar(50),*

*PostalCode varchar(20), Country varchar(50));*

```
MariaDB [customerdb]> Create table Customer(CustomerID int primary key, Custome
rName varchar(50),
    -> ContactName varchar(50), Address varchar(255), City varchar(50), PostalC
ode varchar(20),
    -> Country varchar(50));
Query OK, 0 rows affected (0.066 sec)
```

***show columns from customer;***

***Load Table:***

*LOAD DATA INFILE 'C:/temp/Customers.csv'*

*INTO TABLE Customer*

*FIELDS*

*TERMINATED BY ','*

*ENCLOSED BY '"'*

*LINES TERMINATED BY '\n'*

*IGNORE 1 ROWS;*

```
MariaDB [customerdb]> LOAD DATA INFILE 'C:/temp/Customers.csv'
    -> INTO TABLE Customer
    -> FIELDS
    ->   TERMINATED BY ','
    ->   ENCLOSED BY '"'
    ->   LINES TERMINATED BY '\n'
    -> IGNORE 1 ROWS;
Query OK, 91 rows affected (0.054 sec)
Records: 91  Deleted: 0  Skipped: 0  Warnings: 0
```

**LOAD DATA INFILE:** It is a statement to load the file from a specified path. As we are loading the CSV file from the C:/country.csv.

**TERMINATED BY:** It is the field where we specify that our data in the file are separated by a specific delimiter like CSV file data is separated with comma(,). So we have mentioned the comma with a single quotation mark in the above code.

**ENCLOSED BY:** It is also the field where we specify that our data is enclosed with some symbol.

**LINES TERMINATED BY:** It is the field to specify that when the line end in the file or it is was for MariaDB to identify the end of the line in the file.

**IGNORE 1 ROWS:** Ignoring 1st Header Row.

***Display Customers:*** *Select * from Customer;*

```
MariaDB [customerdb]> select * from customer;
+------------+----------------------------------------+----------------------+---
-------------------------------------------+-----------------+------------+--
------------+
| CustomerID | CustomerName                           | ContactName          | Ad
dress                                    | City            | PostalCode | C
ountry      |
+------------+----------------------------------------+----------------------+---
-------------------------------------------+-----------------+------------+--
------------+
|          1 | Alfreds Futterkiste                    | Maria Anders         | Ob
ere Str. 57                              | Berlin          | 12209      | G
ermany
|          2 | Ana Trujillo Emparedados y helados     | Ana Trujillo         | Av
```

***Create Table Products***

1. ***Create database product db;***
2. ***Use productdb;***
3. ***create table Products(ProductID int not null primary key,***
   ***ProductName varchar(255), SupplierID int,***
   ***CategoryID int, Unit varchar(255),***
   ***Price int);***

*LOAD DATA INFILE 'C:/temp/Products.csv'*

*INTO TABLE Products*

*FIELDS*

*TERMINATED BY ','*

*ENCLOSED BY '"'*

*LINES TERMINATED BY '\n'*

# Normalization

**Data Normalisation**:
Relational database theory, and the principles of normalisation, were first constructed by people with a strong mathematical background. They wrote about databases using terminology which was not easily understood outside those mathematical circles. Below is an attempt to provide understandable explanations. Data normalisation is a set of rules and techniques concerned with:

- Identifying relationships among attributes.
- Combining attributes to form relations.
- Combining relations to form a database.

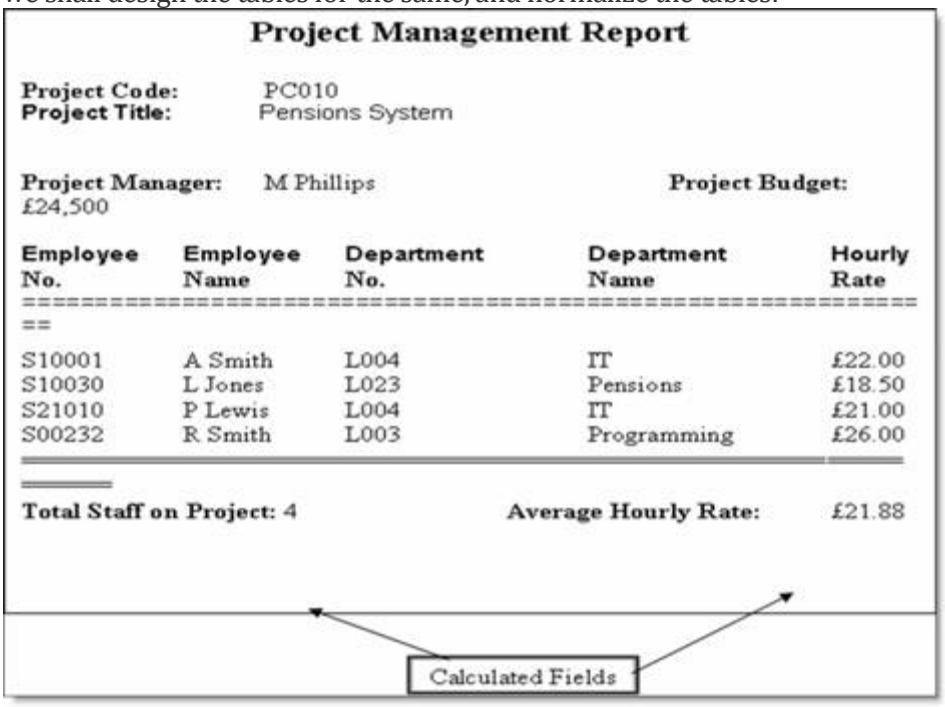
It follows a set of rules worked out by E F Codd in 1970. A normalised relational database provides several benefits:

- Elimination of redundant data storage.
- Close modeling of real world entities, processes, and their relationships. Structuring of data so that the model is flexible.
- Decompose relations such that each non key attribute is dependent on all the attributes in the key.
- Combine relations with identical primary keys (2nd normal form).
- Identify all transitive dependencies.
    - Check relations for dependencies of one non key attribute with another non key attribute.
    - Check for dependencies within each primary key (i.e. dependencies of one attribute in the key on other attributes within the key).
- The guidelines for developing relations in 3rd Normal Form can be summarized as follows:
    - Define the attributes.
    - Group logically related attributes into relations.
    - Identify candidate keys for each relation.
    - Select a primary key for each relation.
    - Identify and remove repeating groups.
    - Combine relations with identical keys (1st normal form).
    - Identify all functional dependencies.
    - Decompose relations such that there are no transitive dependencies.
    - Combine relations with identical primary keys (3rd normal form) if there are no transitive dependencies.

**Levels in normalisation**:

- **1st Normal Form** : A table is in first normal form if all the key attributes have been defined and it contains no repeating groups.
- **2nd Normal Form**: A table is in second normal form (2NF) if and only if it is in 1NF and every non key attribute is fully functionally dependent on the whole of the primary key (i.e. there are no partial dependencies). Anomalies can occur when attributes are dependent on only part of a multi-attribute (composite) key. A relation is in second normal form when all non-key attributes are dependent on the whole key. That is, no attribute is dependent on only a part of the key. Any relation having a key with a single attribute is in second normal form.
- **3rd Normal Form**: A table is in third normal form (3NF) if and only if it is in 2NF and every non key attribute is non-transitively dependent on the primary key (i.e. there are no transitive dependencies). Anomalies can occur when a relation contains one or more transitive dependencies. A relation is in 3NF when it is in 2NF and has no transitive dependencies. A relation is in 3NF when "All non-key attributes are dependent on the key, the whole key and nothing but the key".

**Example of Normalization:** The below image is a report that is supposed to be generated from the system, so we shall design the tables for the same, and normalize the tables:



**Project Management Report**

**Project Code:** PC010
**Project Title:** Pensions System

**Project Manager:** M Phillips **Project Budget:** £24,500

| Employee No. | Employee Name | Department No. | Department Name | Hourly Rate |
|---|---|---|---|---|
| S10001 | A Smith | L004 | IT | £22.00 |
| S10030 | L Jones | L023 | Pensions | £18.50 |
| S21010 | P Lewis | L004 | IT | £21.00 |
| S00232 | R Smith | L003 | Programming | £26.00 |

**Total Staff on Project:** 4 **Average Hourly Rate:** £21.88

Calculated Fields

**Step1**:

- Convert the report into a table
- Ignore the calculated fields
- Identify the primary key (underlined column header)

| Project Code | Project Title | Project Manager | Project Budget | Employee No. | Employee Name | Department No. | Department Name | Hourly Rate |
|---|---|---|---|---|---|---|---|---|
| PC010 | Pensions System | M Phillips | 24500 | S10001 | A Smith | L004 | IT | 22.00 |
| PC010 | Pensions System | M Phillips | 24500 | S10030 | L Jones | L023 | Pensions | 18.50 |
| PC010 | Pensions System | M Phillips | 24500 | S21010 | P Lewis | L004 | IT | 21.00 |
| PC045 | Salaries System | H Martin | 17400 | S10010 | B Jones | L004 | IT | 21.75 |
| PC045 | Salaries System | H Martin | 17400 | S10001 | A Smith | L004 | IT | 18.00 |
| PC045 | Salaries System | H Martin | 17400 | S31002 | T Gilbert | L028 | Database | 25.50 |
| PC045 | Salaries System | H Martin | 17400 | S13210 | W Richards | L008 | Salary | 17.00 |
| PC064 | HR System | K Lewis | 12250 | S31002 | T Gilbert | L028 | Database | 23.25 |
| PC064 | HR System | K Lewis | 12250 | S21010 | P Lewis | L004 | IT | 17.50 |
| PC064 | HR System | K Lewis | 12250 | S10034 | B James | L009 | HR | 16.50 |

**Step 2:**

Starting with **First Normal Form (1NF):**

- Remove repeating (multiple) groups within the primary entities (tables) so that each record (row) within the entity is the same length.
- Repeating groups then become new entities, linked together by a one-to-many relationship.
- Relationships are created by including a primary key from one entity as a foreign key in another entity

| Project Code | Project Title | Project Manager | Project Budget |
|---|---|---|---|
| PC010 | Pensions System | M Phillips | 24500 |
| PC045 | Salaries System | H Martin | 17400 |
| PC064 | HR System | K Lewis | 12250 |

| Project Code | Employee No. | Employee Name | Department No. | Department Name | Hourly Rate |
|---|---|---|---|---|---|
| PC010 | S10001 | A Smith | L004 | IT | 22.00 |
| PC010 | S10030 | L Jones | L023 | Pensions | 18.50 |
| PC010 | S21010 | P Lewis | L004 | IT | 21.00 |
| PC045 | S10010 | B Jones | L004 | IT | 21.75 |
| PC045 | S10001 | A Smith | L004 | IT | 18.00 |
| PC045 | S31002 | T Gilbert | L028 | Database | 25.50 |
| PC045 | S13210 | W Richards | L008 | Salary | 17.00 |
| PC064 | S31002 | T Gilbert | L028 | Database | 23.25 |
| PC064 | S21010 | P Lewis | L004 | IT | 17.50 |
| PC064 | S10034 | B James | L009 | HR | 16.50 |

**Step 3:**

Starting with **Second Normal Form (2NF):**

- Test for dependency by testing each particular attribute in turn to check that it can be uniquely identified by making use of all the primary key. This test need not be completed unless you have at least one table which requires a concatenated Primary Key
- Remove all partially dependent attributes to a new entity.
- Note – A concatenated key occurs when you need two fields together in order to uniquely identify a record.

| Project Code | Project Title | Project Manager | Project Budget |
|---|---|---|---|
| PC010 | Pensions System | M Phillips | 24500 |
| PC045 | Salaries System | H Martin | 17400 |
| PC064 | HR System | K Lewis | 12250 |

| Project Code | Employee No. | Hourly Rate |
|---|---|---|
| PC010 | S10001 | 22.00 |
| PC010 | S10030 | 18.50 |
| PC010 | S21010 | 21.00 |
| PC045 | S10010 | 21.75 |
| PC045 | S10001 | 18.00 |
| PC045 | S31002 | 25.50 |
| PC045 | S13210 | 17.00 |
| PC064 | S31002 | 23.25 |
| PC064 | S21010 | 17.50 |
| PC064 | S10034 | 16.50 |

| Employee No. | Employee Name | Department No. | Department Name |
|---|---|---|---|
| S10001 | A Smith | L004 | IT |
| S10030 | L Jones | L023 | Pensions |
| S21010 | P Lewis | L004 | IT |
| S10010 | B Jones | L004 | IT |
| S31002 | T Gilbert | L028 | Database |
| S13210 | W Richards | L008 | Salary |
| S10034 | B James | L009 | HR |

**Step 4:**
Starting with **Third Normal Form (3NF):**

- Test each attribute in turn to check for dependency on the primary key.
- Remove all transitive dependencies to a new entity.
    - A transitive dependency is where an attribute is dependent on another attribute (or attributes) that is (are) NOT the primary key

| Project Code | Project Title | Project Manager | Project Budget |
|---|---|---|---|
| PC010 | Pensions System | M Phillips | 24500 |
| PC045 | Salaries System | H Martin | 17400 |
| PC064 | HR System | K Lewis | 12250 |

| Project Code | Employee No. | Hourly Rate |
|---|---|---|
| PC010 | S10001 | 22.00 |
| PC010 | S10030 | 18.50 |
| PC010 | S21010 | 21.00 |
| PC045 | S10010 | 21.75 |
| PC045 | S10001 | 18.00 |
| PC045 | S31002 | 25.50 |
| PC045 | S13210 | 17.00 |
| 064 | S31002 | 23.25 |
| PC064 | S21010 | 17.50 |
| PC064 | S10034 | 16.50 |

| Employee No. | Employee Name | Department No. * |
|---|---|---|
| S10001 | A Smith | L004 |
| S10030 | L Jones | L023 |
| S21010 | P Lewis | L004 |
| S10010 | B Jones | L004 |
| S31002 | T Gilbert | L023 |
| S13210 | W Richards | L008 |
| S10034 | B James | L0009 |

| Department No. | Department Name |
|---|---|
| L004 | IT |
| L023 | Pensions |
| L028 | Database |
| L008 | Salary |
| L009 | HR |

```
CREATE TABLE Players(
  ID INT,
  First_Name VARCHAR(255),
  Last_Name VARCHAR(255),
  Date_Of_Birth date,
  Place_Of_Birth VARCHAR(255),
  Country VARCHAR(255),
  PRIMARY KEY (ID)
);
```

```
insert into Players values(1, 'Shikhar', 'Dhawan', DATE('1981-12-05'), 'Delhi', 'India');
insert into Players values(2, 'Jonathan', 'Trott', DATE('1981-04-22'), 'CapeTown',
'SouthAfrica');
insert into Players values(3, 'Kumara', 'Sangakkara', DATE('1977-10-27'), 'Matale',
'Srilanka');
```

```
COMMIT;
```

```
SET autocommit = 0;
```

```
insert into Players values(4, 'Virat', 'Kohli', DATE('1988-11-05'), 'Delhi', 'India');
insert into Players values(5, 'Rohit', 'Sharma', DATE('1987-04-30'), 'Nagpur', 'India');
insert into Players values(6, 'Ravindra', 'Jadeja', DATE('1988-12-06'), 'Nagpur', 'India');
insert into Players values(7, 'James', 'Anderson', DATE('1982-06-30'), 'Burnley', 'England');
```

```
SELECT * FROM Players;
```

```
ROLLBACK;
```

```
SELECT * FROM Players;
```

# Data Query Language
# SELECT Statement

**SELECT:** The SELECT statement allows you to fetch data from tables or views. The result of the SELECT statement is called a result set that is a list of rows, each consisting of the same number of columns. With the help of SELECT statement we can perform multiple tasks like:

- **Querying Data**: This is a simple mechanism of retrieving information from tables, following are different ways to retrieve information:
    - **Select all attributes and tuples from a table**:
      SELECT * FROM table_name;

```
MariaDB [myTestDB]> select * from employees;
+---------------+----------+-----------+-----------+-------------------------------+------------+-----------+----------------------+
| employeeNumber | lastName | firstName | extension | email                         | officeCode | reportsTo | jobTitle             |
+---------------+----------+-----------+-----------+-------------------------------+------------+-----------+----------------------+
|          1002 | Murphy   | Diane     | x5800     | dmurphy@classicmodelcars.com  | 1          |      NULL | President            |
|          1056 | Patterson | Mary     | x4611     | mpatterso@classicmodelcars.com | 1          |      1002 | VP Sales             |
|          1076 | Firrelli | Jeff      | x9273     | jfirrelli@classicmodelcars.com | 1          |      1002 | VP Marketing         |
|          1088 | Patterson | William  | x4871     | wpatterson@classicmodelcars.com | 6          |      1056 | Sales Manager (APAC) |
|          1102 | Bondur   | Gerard    | x5408     | gbondur@classicmodelcars.com  | 4          |      1056 | Sale Manager (EMEA)  |
|          1143 | Bow      | Anthony   | x5428     | abow@classicmodelcars.com     | 1          |      1056 | Sales Manager (NA)   |
|          1165 | Jennings | Leslie    | x3291     | ljennings@classicmodelcars.com | 1          |      1143 | Sales Rep            |
|          1166 | Thompson | Leslie    | x4065     | lthompson@classicmodelcars.com | 1          |      1143 | Sales Rep            |
|          1188 | Firrelli | Julie     | x2173     | jfirrelli@classicmodelcars.com | 2          |      1143 | Sales Rep            |
|          1216 | Patterson | Steve    | x4334     | spatterson@classicmodelcars.com | 2          |      1143 | Sales Rep            |
|          1286 | Tseng    | Foon Yue  | x2248     | ftseng@classicmodelcars.com   | 3          |      1143 | Sales Rep            |
|          1323 | Vanauf   | George    | x4102     | gvanauf@classicmodelcars.com  | 3          |      1143 | Sales Rep            |
|          1337 | Bondur   | Loui      | x6493     | lbondur@classicmodelcars.com  | 4          |      1102 | Sales Rep            |
|          1370 | Hernandez | Gerard   | x2028     | ghernande@classicmodelcars.com | 4          |      1102 | Sales Rep            |
|          1401 | Castillo | Pamela    | x2759     | pcastillo@classicmodelcars.com | 4          |      1102 | Sales Rep            |
|          1501 | Bott     | Larry     | x2311     | lbott@classicmodelcars.com    | 7          |      1102 | Sales Rep            |
```

- **Select few attributes and all tuples from a table:**

  SELECT attribute_list(comma separated) FROM table_name;

```
MariaDB [myTestDB]> select employeeNumber, jobtitle from employees;
+----------------+----------------------+
| employeeNumber | jobtitle             |
+----------------+----------------------+
|           1002 | President            |
|           1056 | VP Sales             |
|           1076 | VP Marketing         |
|           1088 | Sales Manager (APAC) |
|           1102 | Sale Manager (EMEA)  |
|           1143 | Sales Manager (NA)   |
|           1165 | Sales Rep            |
|           1166 | Sales Rep            |
```

select * from *table_name*;

select * from customers;

```
select customername, city, country from customers;

select country from customers;

select distinct country from customers;

select count(distinct country) from customers;
```

# WHERE Clause Search Condition

- **Filtering Data:** Here we combine the select statement with conditions to filter data as per requirement. We use WHERE clause to write the condition. Below is the general syntax of WHERE clause:

  SELECT field1,field2,...FROM tablename WHERE condition

```
select * from customers
where country = 'USA';
```

```
select * from customers
where customernumber = 125;
```

# Arithmetic, Comparison and Logical Operator

- **WHERE clause with Relational/ Comparison operators:** Following are the relational or comparison operators: greater than (>), less that (<), greater than equal to (>=), less than equal to (<=), equal to (=), not equal to (<> or !=)

```
MariaDB [myTestDB]> select customerNumber, customerName
    -> from customers
    -> where creditLimit >= 70000;
+----------------+----------------------------------+
| customerNumber | customerName                     |
+----------------+----------------------------------+
|            112 | Signal Gift Stores               |
|            114 | Australian Collectors, Co.       |
|            119 | La Rochelle Gifts                |
|            121 | Baane Mini Imports               |
|            124 | Mini Gifts Distributors Ltd.     |
|            131 | Land of Toys Inc.                |
|            141 | Euro+ Shopping Channel           |
|            145 | Danish Wholesale Imports         |
|            146 | Saveley & Henriot, Co.           |
|            148 | Dragon Souveniers, Ltd.          |
|            151 | Muscle Machine Inc               |
|            157 | Diecast Classics Inc.            |
|            161 | Technics Stores Inc.             |
```

```
MariaDB [myTestDB]> select customerNumber, customerName
    -> from customers
    -> where city = 'NYC';
+----------------+----------------------+
| customerNumber | customerName         |
+----------------+----------------------+
|            131 | Land of Toys Inc.    |
|            151 | Muscle Machine Inc   |
|            181 | Vitachrome Inc.      |
|            424 | Classic Legends Inc. |
|            456 | Microscale Inc.      |
+----------------+----------------------+
5 rows in set (0.00 sec)
```

- **WHERE clause with Logical operators:** Following are the logical operators: **AND**: all conditions must evaluate to true **OR**: any one condition can evaluate to true **NOT**: negation.

```
MariaDB [myTestDB]> select customerNumber, customerName, creditLimit
    -> from customers
    -> where creditLimit > 25000 and creditlimit < 50000;
+----------------+--------------------------------+-------------+
| customerNumber | customerName                   | creditLimit |
+----------------+--------------------------------+-------------+
|            173 | Cambridge Collectables Co.     |    43400.00 |
|            233 | Québec Home Shopping Network   |    48700.00 |
|            328 | Tekni Collectables Inc.        |    43000.00 |
|            362 | Gifts4AllAges.com              |    41900.00 |
|            447 | Gift Ideas Corp.               |    49700.00 |
|            452 | Mini Auto Werke                |    45300.00 |
|            456 | Microscale Inc.                |    39800.00 |
|            473 | Frau da Collezione             |    34800.00 |
|            489 | Double Decker Gift Stores, Ltd |    43300.00 |
+----------------+--------------------------------+-------------+
9 rows in set (0.00 sec)
```

```
MariaDB [myTestDB]> select employeeNumber, firstName, jobtitle, officeCode
    -> from employees
    -> where jobtitle = 'Sales Rep'  or officeCode = 6;
+----------------+-----------+----------------------+------------+
| employeeNumber | firstName | jobtitle             | officeCode |
+----------------+-----------+----------------------+------------+
|           1088 | William   | Sales Manager (APAC) | 6          |
|           1165 | Leslie    | Sales Rep            | 1          |
|           1166 | Leslie    | Sales Rep            | 1          |
|           1188 | Julie     | Sales Rep            | 2          |
|           1216 | Steve     | Sales Rep            | 2          |
|           1286 | Foon Yue  | Sales Rep            | 3          |
|           1323 | George    | Sales Rep            | 3          |
|           1337 | Loui      | Sales Rep            | 4          |
|           1370 | Gerard    | Sales Rep            | 4          |
|           1401 | Pamela    | Sales Rep            | 4          |
|           1501 | Larry     | Sales Rep            | 7          |
|           1504 | Barry     | Sales Rep            | 7          |
|           1611 | Andy      | Sales Rep            | 6          |
|           1612 | Peter     | Sales Rep            | 6          |
|           1619 | Tom       | Sales Rep            | 6          |
|           1621 | Mami      | Sales Rep            | 5          |
|           1625 | Yoshimi   | Sales Rep            | 5          |
|           1702 | Martin    | Sales Rep            | 4          |
+----------------+-----------+----------------------+------------+
18 rows in set (0.00 sec)
```

```
MariaDB [myTestDB]> select employeeNumber, firstName, jobtitle, officeCode
    -> from employees
    -> where not jobtitle = 'Sales Rep';
+----------------+-----------+----------------------+------------+
| employeeNumber | firstName | jobtitle             | officeCode |
+----------------+-----------+----------------------+------------+
|           1002 | Diane     | President            | 1          |
|           1056 | Mary      | VP Sales             | 1          |
|           1076 | Jeff      | VP Marketing         | 1          |
|           1088 | William   | Sales Manager (APAC) | 6          |
|           1102 | Gerard    | Sale Manager (EMEA)  | 4          |
|           1143 | Anthony   | Sales Manager (NA)   | 1          |
+----------------+-----------+----------------------+------------+
6 rows in set (0.00 sec)
```

- **Selection with arithmetic operators:** Arithmetic operators (+, -, *, /) can be used for deriving new values in the result set.

```
MariaDB [myTestDB]> select customerNumber, customerName, creditLimit,
    -> creditLimit + (creditLimit * 0.05) as "Cr. Limit with 5% bonus"
    -> from customers;
+----------------+---------------------------------+-------------+--------------------------+
| customerNumber | customerName                    | creditLimit | Cr. Limit with 5% bonus  |
+----------------+---------------------------------+-------------+--------------------------+
|            103 | Atelier graphique               |    21000.00 |               22050.0000 |
|            112 | Signal Gift Stores              |    71800.00 |               75390.0000 |
|            114 | Australian Collectors, Co.      |   117300.00 |              123165.0000 |
|            119 | La Rochelle Gifts               |   118200.00 |              124110.0000 |
|            121 | Baane Mini Imports              |    81700.00 |               85785.0000 |
|            124 | Mini Gifts Distributors Ltd.    |   210500.00 |              221025.0000 |
|            125 | Havel & Zbyszek Co              |        0.00 |                   0.0000 |
|            128 | Blauer See Auto, Co.            |    59700.00 |               62685.0000 |
|            129 | Mini Wheels Co.                 |    64600.00 |               67830.0000 |
|            131 | Land of Toys Inc.               |   114900.00 |              120645.0000 |
|            141 | Euro+ Shopping Channel          |   227600.00 |              238980.0000 |
|            144 | Volvo Model Replicas, Co        |    53100.00 |               55755.0000 |
```

```
select * from customers
where country = 'germany' and city = 'berlin';
```

```
select * from customers
where country = 'germany' or country = 'spain';
```

```
select * from customers
where not country = 'germany';
```

```
select * from customers
where country = 'germany' and (city = 'berlin' or city = 'stuttgart');


select * from customers
where not country = 'germany' and not country = 'usa';
```

# Range Operator

- **WHERE clause with range operator (BETWEEN):** It can be used for range of dates, numeric values. Similar to AND condition but for single field.

```
MariaDB [mytestdb]> select customerNumber, customerName, creditlimit
    -> from customers
    -> where creditLimit between 25000 and 50000;
+----------------+--------------------------------+-------------+
| customerNumber | customerName                   | creditlimit |
+----------------+--------------------------------+-------------+
|            173 | Cambridge Collectables Co.     |    43400.00 |
|            233 | Québec Home Shopping Network   |    48700.00 |
|            328 | Tekni Collectables Inc.        |    43000.00 |
|            362 | Gifts4AllAges.com              |    41900.00 |
|            447 | Gift Ideas Corp.               |    49700.00 |
|            452 | Mini Auto Werke                |    45300.00 |
|            456 | Microscale Inc.                |    39800.00 |
|            473 | Frau da Collezione             |    34800.00 |
|            489 | Double Decker Gift Stores, Ltd |    43300.00 |
+----------------+--------------------------------+-------------+
9 rows in set (0.06 sec)
```

select * from products
where price between 10 and 20;

# List Operator

**SELECT with List operator (IN):** It can be used instead of OR condition for a single field and can be used with character or date values.

select *column_name(s)*
from *table_name*
where *column_name* in (*value1*, *value2*, ...);

select * from customers
where country in ('germany', 'france', 'uk');

select * from customers
where country not in ('germany', 'france', 'uk');

# Like Operator

- **WHERE clause with LIKE operator:** It is used when we want to select rows to display that are similar to another field or constant. It is used with % (percentage) or _ (underscore) for character data types. % represent many, _ represent single character.

```
MariaDB [myTestDB]> select customerName from customers where customerName like 'Auto%';
+-------------------------+
| customerName            |
+-------------------------+
| Auto-Moto Classics Inc. |
| Auto Associés & Cie.    |
| Auto Canal+ Petit       |
+-------------------------+
3 rows in set (0.00 sec)
```

```
MariaDB [myTestDB]> select firstname, lastname from employees where firstname like '%y';
+-----------+-----------+
| firstname | lastname  |
+-----------+-----------+
| Mary      | Patterson |
| Anthony   | Bow       |
| Larry     | Bott      |
| Barry     | Jones     |
| Andy      | Fixter    |
+-----------+-----------+
5 rows in set (0.00 sec)
```

```
MariaDB [myTestDB]> select firstname, lastname from employees where firstname like 'Le__ie';
+-----------+----------+
| firstname | lastname |
+-----------+----------+
| Leslie    | Jennings |
| Leslie    | Thompson |
+-----------+----------+
2 rows in set (0.00 sec)
```

```
SELECT * FROM Customers
WHERE CustomerName LIKE 'a%';
```

```
-- CustomerName that have "or" in any position

SELECT * FROM Customers
WHERE CustomerName LIKE '%or%';
```

```
--Finding the second position

SELECT * FROM Customer
WHERE CustomerName LIKE '_r%';
```

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

```
SELECT * FROM Customer
WHERE CustomerName LIKE 'a__%';
```

-- ContactName that starts with "a" and ends with "n"

```
SELECT * FROM Customer

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

--CustomerName that does NOT start with "a"

```
SELECT * FROM Customer
WHERE CustomerName NOT LIKE 'a%';
```

--CustomerName that does NOT end with "a"

```
SELECT * FROM Customer
WHERE CustomerName NOT LIKE '%a';
```

# Using ORDER BY, DISTINCT and TOP

- **Remove all duplicate values from the records:**

  SELECT DISTINCT attribute_name FROM table_name;

```
MariaDB [myTestDB]> SELECT DISTINCT jobtitle from employees;
+----------------------+
| jobtitle             |
+----------------------+
| President            |
| VP Sales             |
| VP Marketing         |
| Sales Manager (APAC) |
| Sale Manager (EMEA)  |
| Sales Manager (NA)   |
| Sales Rep            |
+----------------------+
7 rows in set (0.00 sec)
```

- **Renaming the attribute name:**

  SELECT attribute_name1 AS alias_name, attribute_name2 AS alias_name ….. FROM table_name;

  SELECT attribute_name1 alias_name, attribute_name2 alias_name ….. FROM table_name;

```
MariaDB [myTestDB]> select employeeNumber 'Emp Code', officeCode as 'Deptartment Number'
    -> FROM employees;
+----------+--------------------+
| Emp Code | Deptartment Number |
+----------+--------------------+
|     1002 | 1                  |
|     1056 | 1                  |
|     1076 | 1                  |
|     1143 | 1                  |
|     1165 | 1                  |
|     1166 | 1                  |
|     1188 | 2                  |
|     1216 | 2                  |
|     1286 | 3                  |
|     1323 | 3                  |
|     1102 | 4                  |
|     1337 | 4                  |
|     1370 | 4                  |
```

  Please note if you are using a space in the alias name, you need to give the alias in single quote (‘) or double quote (“) else the following error will be displayed.

```
MariaDB [myTestDB]> select employeeNumber Emp Code, officeCode as 'Deptartment Number'
    -> FROM employees;
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corres
ponds to your MariaDB server version for the right syntax to use near 'Code, offic
eCode as 'Deptartment Number'
FROM employees' at line 1
```

- **Sorting Data:** Here we understand how to sort the result set using ORDER BY clause. The custom sort order with the FIELD function.

```
MariaDB [mytestdb]> select firstname, lastname
    -> from employees
    -> order by firstname;
+-----------+-----------+
| firstname | lastname  |
+-----------+-----------+
| Andy      | Fixter    |
| Anthony   | Bow       |
| Barry     | Jones     |
| Diane     | Murphy    |
| Foon Yue  | Tseng     |
| George    | Vanauf    |
| Gerard    | Bondur    |
| Gerard    | Hernandez |
```

```
MariaDB [mytestdb]> select firstname, lastname
    -> from employees
    -> order by firstname desc;
+-----------+-----------+
| firstname | lastname  |
+-----------+-----------+
| Yoshimi   | Kato      |
| William   | Patterson |
| Tom       | King      |
| Steve     | Patterson |
| Peter     | Marsh     |
| Pamela    | Castillo  |
| Mary      | Patterson |
```

```
MariaDB [mytestdb]> SELECT ordernumber, orderlinenumber, quantityOrdered * priceEach AS subtotal
    -> FROM orderdetails
    -> ORDER BY ordernumber, orderLineNumber, subtotal
    -> LIMIT 20;
+-------------+-----------------+----------+
| ordernumber | orderlinenumber | subtotal |
+-------------+-----------------+----------+
|       10100 |               1 |  1729.21 |
|       10100 |               2 |  2754.50 |
|       10100 |               3 |  4080.00 |
|       10100 |               4 |  1660.12 |
|       10101 |               1 |  4343.56 |
|       10101 |               2 |  2040.10 |
|       10101 |               3 |  1463.85 |
|       10101 |               4 |  2701.50 |
```

- **SELECT with List operator (IN):** It can be used instead of OR condition for single field and can be used with character or date values.

- **SELECT with LIMIT:** This is command is to restrict the number of records in the result set based on the requirement. This is similar to the TOP command in SQL.

```
MariaDB [(none)]> use mytestdb
Database changed
MariaDB [mytestdb]> select customerNumber, customerName
    -> from customers
    -> limit 10;
+----------------+-----------------------------+
| customerNumber | customerName                |
+----------------+-----------------------------+
|            103 | Atelier graphique           |
|            112 | Signal Gift Stores          |
|            114 | Australian Collectors, Co.  |
|            119 | La Rochelle Gifts           |
|            121 | Baane Mini Imports          |
|            124 | Mini Gifts Distributors Ltd.|
|            125 | Havel & Zbyszek Co          |
|            128 | Blauer See Auto, Co.        |
|            129 | Mini Wheels Co.             |
|            131 | Land of Toys Inc.           |
+----------------+-----------------------------+
10 rows in set (0.23 sec)
```

**ORDER BY Syntax**

SELECT *column1*, *column2, ...*
FROM *table_name*
ORDER BY *column1, column2, ...* ASC|DESC;

SELECT * FROM Customers ORDER BY Country;

SELECT * FROM Customers
ORDER BY Country DESC;

SELECT * FROM Customers
ORDER BY Country, CustomerName;

SELECT * FROM Customers
ORDER BY Country ASC, CustomerName DESC;

**Distinct Syntax:**

SELECT DISTINCT *column1*, *column2, ...*
FROM *table_name*;

SELECT DISTINCT Country FROM Customer;

```
SELECT COUNT(DISTINCT Country) FROM Customer;
```

**TOP Syntax:**

SELECT *column_name(s)* FROM *table_name* WHERE *condition* LIMIT *number*;

```
SELECT * FROM Customers LIMIT 3;
```

```
SELECT * FROM Customers
WHERE Country='Germany'
LIMIT 3;
```

# Using IS NULL and IS NOT NULL

- **WHERE clause with IS NULL operator:** Null is not equal to zero or a blank space.

```
MariaDB [myTestDB]> select customerNumber, customerName, city
    -> from customers
    -> where salesRepEmployeeNumber is null;
+----------------+--------------------------------+------------------+
| customerNumber | customerName                   | city             |
+----------------+--------------------------------+------------------+
|            125 | Havel & Zbyszek Co             | Warszawa         |
|            169 | Porto Imports Co.              | Lisboa           |
|            206 | Asian Shopping Network, Co     | Singapore        |
|            223 | Natürlich Autos                | Cunewalde        |
|            237 | ANG Resellers                  | Madrid           |
|            247 | Messner Shopping Network       | Frankfurt        |
|            273 | Franken Gifts, Co              | München          |
|            293 | BG&E Collectables              | Fribourg         |
|            303 | Schuyler Imports               | Amsterdam        |
|            307 | Der Hund Imports               | Berlin           |
|            335 | Cramer Spezialitäten, Ltd      | Brandenburg      |
|            348 | Asian Treasures, Inc.          | Cork             |
|            356 | SAR Distributors, Co           | Hatfield         |
|            361 | Kommission Auto                | Münster          |
```

```
MariaDB [myTestDB]> select customerNumber, customerName, city
    -> from customers
    -> where salesRepEmployeeNumber is not null;
+----------------+-----------------------------------+--------------------+
| customerNumber | customerName                      | city               |
+----------------+-----------------------------------+--------------------+
|            103 | Atelier graphique                 | Nantes             |
|            112 | Signal Gift Stores                | Las Vegas          |
|            114 | Australian Collectors, Co.        | Melbourne          |
|            119 | La Rochelle Gifts                 | Nantes             |
|            121 | Baane Mini Imports                | Stavern            |
|            124 | Mini Gifts Distributors Ltd.      | San Rafael         |
|            128 | Blauer See Auto, Co.              | Frankfurt          |
```

```
MariaDB [myTestDB]> select customerNumber, customerName, city
    -> from customers
    -> where salesRepEmployeeNumber is not null and city = 'Singapore';
+----------------+-------------------------+-----------+
| customerNumber | customerName            | city      |
+----------------+-------------------------+-----------+
|            148 | Dragon Souveniers, Ltd. | Singapore |
|            166 | Handji Gifts& Co        | Singapore |
+----------------+-------------------------+-----------+
2 rows in set (0.00 sec)
```

```
SELECT ISNULL(NULL);

SELECT ISNULL("gfg");

select customername, city from customers where postalcode is null;

select customername, city from customers where postalcode is not null;
```

# Various Other Clauses

The `CASE` statement goes through conditions and returns a value when the first condition is met (like an if-then-else statement). So, once a condition is true, it will stop reading and return the result. If no conditions are true, it returns the value in the `ELSE` clause.

If there is no `ELSE` part and no conditions are true, it returns NULL.

CASE
  WHEN *condition1* THEN *result1*
  WHEN *condition2* THEN *result2*
  WHEN *conditionN* THEN *resultN*
  ELSE *result*
END;

- **SELECT with CASE expression:** When you wish to check multiple conditions on the database you can use CASE expression. Following the general syntax for the same:

  SELECT fieldname 1, CASE fieldname

  WHEN expr THEN action

  WHEN expr THEN action

  ELSE expr

  END, fieldname 2, ... FROM table_name;

```
MariaDB [mytestdb]> SELECT distinct productLine,
    -> CASE productLine
    -> WHEN 'Motorcycles' then 'Cost increased by $100'
    -> WHEN 'Trucks and Buses' then 'Cost increased by $50'
    -> ELSE 'Cost increased by $10'
    -> END as "Remarks" from products;
+-----------------+-----------------------+
| productLine     | Remarks               |
+-----------------+-----------------------+
| Classic Cars    | Cost increased by $10 |
| Motorcycles     | Cost increased by $100|
| Planes          | Cost increased by $10 |
| Ships           | Cost increased by $10 |
| Trains          | Cost increased by $10 |
| Trucks and Buses| Cost increased by $50 |
| Vintage Cars    | Cost increased by $10 |
+-----------------+-----------------------+
7 rows in set (0.03 sec)
```

```
SELECT OrderNumber, QuantityOrdered,
CASE
    WHEN QuantityOrdered > 30 THEN 'The quantity is greater than 30'
    WHEN QuantityOrdered = 30 THEN 'The quantity is 30'
    ELSE 'The quantity is under 30'
END AS QuantityText
FROM OrderDetails limit 10;


SELECT OrderID, Quantity,

CASE

    WHEN Quantity > 30 THEN 'The quantity is greater than 30'

    WHEN Quantity = 30 THEN 'The quantity is 30'

    ELSE 'The quantity is under 30'

END AS QuantityText

FROM OrderDetails limit 10;


SELECT CustomerName, City, Country FROM Customers

ORDER BY (CASE

WHEN City IS NULL THEN Country

ELSE City

END);
```

**Questions to solve in the class:**

1. List all the orders placed in month of February, 2003.
   ```
   select * from orders
   where orderdate between '2003-02-01' and '2003-02-28';
   ```
1. Calculate the number of days required in the month of August 2004 to ship the products.
   ```
   select orderNumber, orderDate, shippedDate,
   shippedDate - orderDate as "Days Required"
   from orders
   where orderDate between '2004-08-01' and '2004-08-31';
   ```
1. List product details which has stock more that 6500.
   ```
   select productCode, productName from products
   where quantityInStock > 6500;
   ```
1. Display all the product names that don't end with S.
   ```
   select productName from Products
   where productName not like '%s' or productName not like '%S';
   ```
1. List names of the employees in descending order of their office numbers
   ```
   select firstname, lastname, officeCode from employees
   order by officeCode desc;
   ```
1. List the names, job title and office no of everyone whose name falls in the alphabetical range 'C' to 'L'
   ```
   Select firstname from employees
   where firstname between 'c%' and 'l%';
   select firstname, lastname from employees
   where firstname REGEXP '^[C-L].*$'
   ```
1. Display all the office cities which have NULL values in State.
   ```
   select city from offices where state is null;
   ```
1. List employee details working in Office Code 2, 3 or 5.
   ```
   select firstname, lastname from employees
   where officeCode in (2, 3, 5);
   ```
1. Display all the customers with no sales representative and they belong to either Singapore or Frankfurt cities.
   ```
   select customerName,city from customers
   where SalesRepEmployeeNumber is null and
   city = 'Singapore' or city = 'Frankfurt';
   ```
1. Select the name of all employees who are Sales Representatives.
   ```
   select firstname, lastname, jobtitle from employees
   where jobtitle = 'Sales Rep';
   ```

# Built-in SQL functions

# String Function

- **String Functions**: Return string values. Return values with varchar or char. If the length of the return values exceeds the limit, the return value is truncated without any error message.
    - **CONCAT(exp1, exp2, … , expn):** Concatenate two or more strings into one.

```
MariaDB [myTestDB]> select concat('Good','morning');
+--------------------------+
| concat('Good','morning') |
+--------------------------+
| Goodmorning              |
+--------------------------+
1 row in set (0.00 sec)

MariaDB [myTestDB]> select concat('Good',' ','Morning');
+-------------------------------+
| concat('Good',' ','Morning') |
+-------------------------------+
| Good Morning                  |
+-------------------------------+
1 row in set (0.02 sec)
```

- **LOWER(expr):** converts all the characters to Lowercase

```
MariaDB [myTestDB]> select lower('DATABASE MANAGEMENT SYSTEM');
+-------------------------------------+
| lower('DATABASE MANAGEMENT SYSTEM') |
+-------------------------------------+
| database management system          |
+-------------------------------------+
1 row in set (0.00 sec)
```

- **UPPER(expr):** converts all the characters to Uppercase

```
MariaDB [myTestDB]> select UPPER('database management system');
+-------------------------------------+
| UPPER('database management system') |
+-------------------------------------+
| DATABASE MANAGEMENT SYSTEM          |
+-------------------------------------+
1 row in set (0.01 sec)
```

- **REPLACE(expr1,char to be replaced, with what):** Search and replace a substring in a string.

```
MariaDB [myTestDB]> SELECT REPLACE('Let us Learn','Learn','Earn');
+----------------------------------------+
| REPLACE('Let us Learn','Learn','Earn') |
+----------------------------------------+
| Let us Earn                            |
+----------------------------------------+
1 row in set (0.03 sec)
```

- **SUBSTR(expr1,m,n):** Extracts 'n' number of characters from 'mth' position. m can be positive or negative. If m is positive, extracting of characters starts from left hand side. If m is negative, extracting of characters starts from right hand side. If m is a floating point, it is converted to integer.

```
MariaDB [myTestDB]> SELECT SUBSTR('Extracting characters',2,5);
+-------------------------------------+
| SUBSTR('Extracting characters',2,5) |
+-------------------------------------+
| xtrac                               |
+-------------------------------------+
1 row in set (0.00 sec)
MariaDB [myTestDB]> SELECT SUBSTR('Extracting characters',12,5);
+-------------------------------------+
| SUBSTR('Extracting characters',12,5) |
+-------------------------------------+
| chara                               |
+-------------------------------------+
1 row in set (0.00 sec)
```

```
MariaDB [myTestDB]> SELECT SUBSTR('Extracting characters',-2.5,5);
+---------------------------------------+
| SUBSTR('Extracting characters',-2.5,5) |
+---------------------------------------+
| ers                                   |
+---------------------------------------+
1 row in set (0.11 sec)
MariaDB [myTestDB]> SELECT SUBSTR('Extracting characters',2.5,5);
+---------------------------------------+
| SUBSTR('Extracting characters',2.5,5) |
+---------------------------------------+
| tract                                 |
+---------------------------------------+
1 row in set (0.02 sec)
```

- **LENGTH(expr) and CHAR_LENGTH(exp):** Returns the length of the string

```
MariaDB [myTestDB]> SELECT LENGTH('MySQL is robust');
+---------------------------+
| LENGTH('MySQL is robust') |
+---------------------------+
|                        15 |
+---------------------------+
1 row in set (0.00 sec)
```

```
SELECT CustomerName, ASCII(CustomerName) AS NumCodeOfFirstChar
FROM Customers;
```

```
SELECT CHAR_LENGTH("ItVedant") AS LengthOfString;
```

```
SELECT CHAR_LENGTH(CustomerName) AS LengthOfName
FROM Customers;
```

```
SELECT CHARACTER_LENGTH("ITVedant") AS LengthOfString;
```

```
SELECT CHARACTER_LENGTH(CustomerName) AS LengthOfName
FROM Customers;
```

```
SELECT CONCAT("Learning ", "SQL ", "is ", "fun!") AS ConcatenatedString;
```

```
SELECT CONCAT(Addressline1, "  ",Addressline2," ", PostalCode, "  ", City) AS Address
FROM Customers;
```

```
SELECT CONCAT(Addressline1, " ", PostalCode, "  ", City) AS Address
FROM Customers;
```

```
SELECT CONCAT_WS("-", "SQL", "is", "fun!") AS ConcatenatedString;
```

```
SELECT CONCAT_WS("  ", Addressline1, PostalCode, City) AS Address
FROM Customers;
```

Return the index position of "Q" in the string list:

```
SELECT FIELD("Q", "s", "q", "l");
```

```
SELECT FIELD(5, 0, 1, 2, 3, 4, 5);
```

```
SELECT FIND_IN_SET("q", "s,q,l");

SELECT FORMAT(250500.5634, 0);

SELECT INSTR("ITvedant.com", "3") AS MatchPosition;
```

| MySQL String Functions | |
|---|---|
| Function | Description |
| ASCII | Returns the ASCII value for the specific character |
| CHAR_LENGTH | Returns the length of a string (in characters) |
| CHARACTER_LENGTH | Returns the length of a string (in characters) |
| CONCAT | Adds two or more expressions together |
| CONCAT_WS | Adds two or more expressions together with a separator |
| FIELD | Returns the index position of a value in a list of values |
| FIND_IN_SET | Returns the position of a string within a list of strings |
| FORMAT | Formats a number to a format like "#,###,###.##", rounded to a specified number of decimal places |
| INSERT | Inserts a string within a string at the specified position and for a certain number of characters |
| INSTR | Returns the position of the first occurrence of a string in another string |
| LCASE | Converts a string to lower-case |
| LEFT | Extracts a number of characters from a string (starting from left) |
| LENGTH | Returns the length of a string (in bytes) |
| LOCATE | Returns the position of the first occurrence of a substring in a string |
| LOWER | Converts a string to lower-case |
| LPAD | Left-pads a string with another string, to a certain length |
| LTRIM | Removes leading spaces from a string |
| MID | Extracts a substring from a string (starting at any position) |
| POSITION | Returns the position of the first occurrence of a substring in a string |
| REPEAT | Repeats a string as many times as specified |
| REPLACE | Replaces all occurrences of a substring within a string, with a new substring |
| REVERSE | Reverses a string and returns the result |
| RIGHT | Extracts a number of characters from a string (starting from right) |
| RPAD | Right-pads a string with another string, to a certain length |
| RTRIM | Removes trailing spaces from a string |
| SPACE | Returns a string of the specified number of space characters |
| STRCMP | Compares two strings |
| SUBSTR | Extracts a substring from a string (starting at any position) |
| SUBSTRING | Extracts a substring from a string (starting at any position) |
| SUBSTRING_INDEX | Returns a substring of a string before a specified number of delimiter occurs |

| | |
|---|---|
| TRIM | Removes leading and trailing spaces from a string |
| UCASE | Converts a string to upper-case |
| UPPER | Converts a string to upper-case |

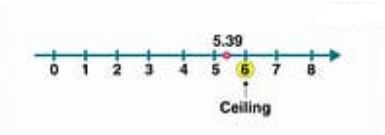
# Math Function

- **Math Functions:** Take numeric values and return numeric values. The values returned are accurate up to 38 decimal digits.
    - **ABS(X):** Returns the absolute value of X.

```
MariaDB [myTestDB]> SELECT ABS(2);
+--------+
| ABS(2) |
+--------+
|      2 |
+--------+
1 row in set (0.00 sec)
MariaDB [myTestDB]> SELECT ABS(-32);
+----------+
| ABS(-32) |
+----------+
|       32 |
+----------+
1 row in set (0.07 sec)
```

- **CEIL(X):** CEIL() is a synonym for CEILING()

The ceiling function of a real number is the **least integer number greater than or equal to the given number**. In the case of 4.5, the integers greater than 4.5 are 5, 6, 7, 8, ..... The smallest of all is 5. Therefore, [4.5] = 5. In the case of -4.5, the integers that are greater than − 4.5 are − 4, − 3, − 2,...




```
MariaDB [myTestDB]> select CEIL (17.3);
+-------------+
| CEIL (17.3) |
+-------------+
|          18 |
+-------------+
1 row in set (0.05 sec)

MariaDB [myTestDB]> select CEIL (17.7);
+-------------+
| CEIL (17.7) |
+-------------+
|          18 |
+-------------+
1 row in set (0.00 sec)
```

- **FLOOR(X):** Returns the largest integer value not greater than X.

```
MariaDB [myTestDB]> select FLOOR(17.3);
+-------------+
| FLOOR(17.3) |
+-------------+
|          17 |
+-------------+
1 row in set (0.00 sec)

MariaDB [myTestDB]> select FLOOR(17.7);
+-------------+
| FLOOR(17.7) |
+-------------+
|          17 |
+-------------+
1 row in set (0.00 sec)
```

- **MOD(N,M), N % M, N MOD M:** Modulo operation. Returns the remainder of N divided by M.

```
MariaDB [myTestDB]>  SELECT MOD(29,9);
+-----------+
| MOD(29,9) |
+-----------+
|         2 |
+-----------+
1 row in set (0.00 sec)

MariaDB [myTestDB]> SELECT 29 MOD 9;
+----------+
| 29 MOD 9 |
+----------+
|        2 |
+----------+
1 row in set (0.00 sec)
MariaDB [myTestDB]> SELECT MOD(234, 10);
+--------------+
| MOD(234, 10) |
+--------------+
|            4 |
+--------------+
1 row in set (0.00 sec)

MariaDB [myTestDB]> SELECT 253 % 7;
+---------+
| 253 % 7 |
+---------+
|       1 |
+---------+
1 row in set (0.00 sec)
```

- **ROUND(X), ROUND(X,D):** Rounds the argument X to D decimal places. The rounding algorithm depends on the data type of X. D defaults to 0 if not specified. D can be negative to cause D digits left of the decimal point of the value X to become zero.

```
MariaDB [myTestDB]> SELECT ROUND(-1.23);
+--------------+
| ROUND(-1.23) |
+--------------+
|           -1 |
+--------------+
1 row in set (0.03 sec)

MariaDB [myTestDB]> SELECT ROUND(-1.58);
+--------------+
| ROUND(-1.58) |
+--------------+
|           -2 |
+--------------+
1 row in set (0.00 sec)
MariaDB [myTestDB]> SELECT ROUND(1.298, 1);
+-----------------+
| ROUND(1.298, 1) |
+-----------------+
|             1.3 |
+-----------------+
1 row in set (0.00 sec)

MariaDB [myTestDB]> SELECT ROUND(1.298, 0);
+-----------------+
| ROUND(1.298, 0) |
+-----------------+
|               1 |
+-----------------+
1 row in set (0.00 sec)

MariaDB [myTestDB]> SELECT ROUND(23.298, -1);
+------------------+
| ROUND(23.298, -1) |
+------------------+
|                20 |
+------------------+
1 row in set (0.00 sec)
```

- **TRUNCATE(X,D):** Returns the number X, truncated to D decimal places. If D is 0, the result has no decimal point or fractional part. D can be negative to cause D digits left of the decimal point of the value X to become zero.

```
MariaDB [myTestDB]> SELECT TRUNCATE(1.223,1);
+------------------+
| TRUNCATE(1.223,1) |
+------------------+
|              1.2 |
+------------------+
1 row in set (0.00 sec)

MariaDB [myTestDB]> SELECT TRUNCATE(1.999,1);
+------------------+
| TRUNCATE(1.999,1) |
+------------------+
|              1.9 |
+------------------+
1 row in set (0.00 sec)

MariaDB [myTestDB]>  SELECT TRUNCATE(1.999,0);
+------------------+
| TRUNCATE(1.999,0) |
+------------------+
|                1 |
+------------------+
1 row in set (0.00 sec)
MariaDB [myTestDB]> SELECT TRUNCATE(122,-2);
+-----------------+
| TRUNCATE(122,-2) |
+-----------------+
|             100 |
+-----------------+
1 row in set (0.00 sec)

MariaDB [myTestDB]> SELECT TRUNCATE(10.28*100,0);
+----------------------+
| TRUNCATE(10.28*100,0) |
+----------------------+
|                 1028 |
+----------------------+
1 row in set (0.03 sec)
```

- **EXP(X):** Returns the value of e (the base of natural logarithms) raised to the power of X. The inverse of this function is LOG() (using a single argument only) or LN().

```
MariaDB [myTestDB]> SELECT EXP(2);
+------------------+
| EXP(2)           |
+------------------+
| 7.38905609893065 |
+------------------+
1 row in set (0.06 sec)

MariaDB [myTestDB]>  SELECT EXP(-2);
+--------------------+
| EXP(-2)            |
+--------------------+
| 0.1353352832366127 |
+--------------------+
1 row in set (0.00 sec)

MariaDB [myTestDB]> SELECT EXP(0);
+--------+
| EXP(0) |
+--------+
|      1 |
+--------+
1 row in set (0.00 sec)
```

- **POW(X,Y) / POWER(X,Y):** Returns the value of X raised to the power of Y.

```
MariaDB [myTestDB]> SELECT POW(2,2);
+----------+
| POW(2,2) |
+----------+
|        4 |
+----------+
1 row in set (0.05 sec)

MariaDB [myTestDB]>  SELECT POW(2,-2);
+-----------+
| POW(2,-2) |
+-----------+
|      0.25 |
+-----------+
1 row in set (0.00 sec)
```

- **SQRT(X):** Returns the square root of a nonnegative number X.

```
MariaDB [myTestDB]> SELECT SQRT(4);
+---------+
| SQRT(4) |
+---------+
|       2 |
+---------+
1 row in set (0.00 sec)

MariaDB [myTestDB]> SELECT SQRT(20);
+------------------+
| SQRT(20)         |
+------------------+
| 4.47213595499958 |
+------------------+
1 row in set (0.00 sec)

MariaDB [myTestDB]> SELECT SQRT(-16);
+-----------+
| SQRT(-16) |
+-----------+
|      NULL |
+-----------+
1 row in set (0.00 sec)
```

# Date Function

- **Date Functions:**
  - **CURDATE()**: Returns the current date.

```
MariaDB [myTestDB]> select curdate();
+------------+
| curdate()  |
+------------+
| 2019-02-09 |
+------------+
1 row in set (0.00 sec)
```

  - **NOW()**: Returns the current date and time at which the statement executed.

```
MariaDB [myTestDB]> select now();
+---------------------+
| now()               |
+---------------------+
| 2019-02-09 15:01:01 |
+---------------------+
1 row in set (0.00 sec)
```

  - **SYSDATE()**: Returns the current date.

```
MariaDB [myTestDB]> select sysdate();
+---------------------+
| sysdate()           |
+---------------------+
| 2019-02-09 15:02:10 |
+---------------------+
1 row in set (0.00 sec)
```

  - **Last_day(date):** returns the last day of the corresponding month for a date or datetime value.

```
MariaDB [myTestDB]> select last_day('2012-12-01');
+------------------------+
| last_day('2012-12-01') |
+------------------------+
| 2012-12-31             |
+------------------------+
1 row in set (0.01 sec)
```

```
MariaDB [myTestDB]> select last_day(curdate());
+---------------------+
| last_day(curdate()) |
+---------------------+
| 2019-02-28          |
+---------------------+
1 row in set (0.05 sec)
```

- Date_format(date, format): To format a date value to a specific format, you use the DATE_FORMAT function.

| Format | Description |
|---|---|
| %a | Abbreviated weekday name (Sun to Sat) |
| %b | Abbreviated month name (Jan to Dec) |
| %c | Numeric month name (0 to 12) |
| %D | Day of the month as a numeric value, followed by suffix (1st, 2nd, 3rd, ...) |
| %d | Day of the month as a numeric value (01 to 31) |
| %e | Day of the month as a numeric value (0 to 31) |
| %f | Microseconds (000000 to 999999) |
| %H | Hour (00 to 23) |
| %h | Hour (00 to 12) |
| %I | Hour (00 to 12) |
| %i | Minutes (00 to 59) |
| %j | Day of the year (001 to 366) |

| | |
|---|---|
| %k | Hour (0 to 23) |
| %l | Hour (1 to 12) |
| %M | Month name in full (January to December) |
| %m | Month name as a numeric value (00 to 12) |
| %p | AM or PM |
| %r | Time in 12 hour AM or PM format (hh:mm:ss AM/PM) |
| %S | Seconds (00 to 59) |
| %s | Seconds (00 to 59) |
| %T | Time in 24 hour format (hh:mm:ss) |
| %U | Week where Sunday is the first day of the week (00 to 53) |
| %u | Week where Monday is the first day of the week (00 to 53) |
| %V | Week where Sunday is the first day of the week (01 to 53). Used with %X |
| %v | Week where Monday is the first day of the week (01 to 53). Used with %X |
| %W | Weekday name in full (Sunday to Saturday) |
| %w | Day of the week where Sunday=0 and Saturday=6 |
| %X | Year for the week where Sunday is the first day of the week. Used with %V |
| %x | Year for the week where Monday is the first day of the week. Used with %V |

| | |
|---|---|
| %Y | Year as a numeric, 4-digit value |
| %y | Year as a numeric, 2-digit value |

```
MariaDB [myTestDB]> SELECT DATE_FORMAT("2017-06-15", "%Y");
+----------------------------------+
| DATE_FORMAT("2017-06-15", "%Y") |
+----------------------------------+
| 2017                             |
+----------------------------------+
1 row in set (0.02 sec)
MariaDB [myTestDB]> SELECT DATE_FORMAT("2017-06-15", "%M %d %Y");
+---------------------------------------+
| DATE_FORMAT("2017-06-15", "%M %d %Y") |
+---------------------------------------+
| June 15 2017                          |
+---------------------------------------+
1 row in set (0.06 sec)
```

- **DATEDIFF(expr1,expr2):** returns expr1 - expr2 expressed as a value in days from one date to the other. expr1 and expr2 are date or date-and-time expressions. Only the date parts of the values are used in the calculation.

```
MariaDB [myTestDB]> SELECT DATEDIFF('2007-12-31 23:59:59','2007-12-30');
+-----------------------------------------------+
| DATEDIFF('2007-12-31 23:59:59','2007-12-30') |
+-----------------------------------------------+
|                                             1 |
+-----------------------------------------------+
1 row in set (0.04 sec)
MariaDB [myTestDB]> SELECT DATEDIFF('2010-11-30 23:59:59','2010-12-31');
+-----------------------------------------------+
| DATEDIFF('2010-11-30 23:59:59','2010-12-31') |
+-----------------------------------------------+
|                                           -31 |
+-----------------------------------------------+
1 row in set (0.00 sec)
```

- **MONTH(date):** Returns the month for date, in the range 1 to 12 for January to December, or 0 for dates such as '0000-00-00' or '2008-00-00' that have a zero month part.

```
MariaDB [myTestDB]> select month(sysdate())
    -> ;
+------------------+
| month(sysdate()) |
+------------------+
|                2 |
+------------------+
1 row in set (0.04 sec)
```

- **YEAR(date):** Returns the year for date, in the range 1000 to 9999, or 0 for the "zero" date.

```
MariaDB [myTestDB]> select year(now());
+-------------+
| year(now()) |
+-------------+
|        2019 |
+-------------+
1 row in set (0.00 sec)
```

# Aggregate Function

- **Comparison Functions:**
  - **COALESCE(value,...):** Returns the first non-NULL value in the list, or NULL if there are no non-NULL values.

```
MariaDB [myTestDB]> SELECT COALESCE(NULL,1);
+------------------+
| COALESCE(NULL,1) |
+------------------+
|                1 |
+------------------+
1 row in set (0.05 sec)

MariaDB [myTestDB]>  SELECT COALESCE(NULL,NULL,NULL);
+--------------------------+
| COALESCE(NULL,NULL,NULL) |
+--------------------------+
|                     NULL |
+--------------------------+
1 row in set (0.00 sec)
```

  - **ISNULL(expr):** If expr is NULL, ISNULL() returns 1, otherwise it returns 0.

```
MariaDB [myTestDB]> SELECT ISNULL(1+1);
+-------------+
| ISNULL(1+1) |
+-------------+
|           0 |
+-------------+
1 row in set (0.09 sec)

MariaDB [myTestDB]> SELECT ISNULL(1/0);
+-------------+
| ISNULL(1/0) |
+-------------+
|           1 |
+-------------+
1 row in set (0.04 sec)
```

  - **GREATEST(value1,value2,...) / LEAST(value1, value2,...):** With two or more arguments, returns the largest (maximum-valued) argument. The arguments are compared using the same rules as for LEAST().

```
MariaDB [myTestDB]> SELECT GREATEST(2,0,10, 41);
+---------------------+
| GREATEST(2,0,10, 41) |
+---------------------+
|                  41 |
+---------------------+
1 row in set (0.01 sec)
MariaDB [myTestDB]> SELECT LEAST(2,0,10, 41);
+--------------------+
| LEAST(2,0,10, 41) |
+--------------------+
|                  0 |
+--------------------+
1 row in set (0.00 sec)
```

- **IF(expr1,expr2,expr3):** If expr1 is TRUE (expr1 <> 0 and expr1 <> NULL) then IF() returns expr2; otherwise it returns expr3. IF() returns a numeric or string value, depending on the context in which it is used.

```
MariaDB [myTestDB]> SELECT IF(1<2,'yes','no');
+--------------------+
| IF(1<2,'yes','no') |
+--------------------+
| yes                |
+--------------------+
1 row in set (0.05 sec)
```

# Load Sample Database


Use the source command to load data into the MySQL Server:

```
MariaDB [(none)]>  source c:\temp\mysqlsampledatabase.sql
```

```
MariaDB [(none)]> > show databases;
```

```
MariaDB [(none)]> \! Cls
```

# The MySQL Aggregate Functions

MySQL supports the following aggregate functions

| Function | Description |
|---|---|
| AVG() | Returns the average of the values in the selected column |
| COUNT() | Returns the number of rows returned for a selection |
| MAX() | Returns the maximum value for a column |
| MIN() | Returns the minimum value of a column |
| SUM() | Returns the sum of the values in a specified column |

- **Aggregation functions:** The aggregation functions act on set of records or the entire table. Many aggregation functions accept the DISTINCT clause. Aggregation functions are also known as group functions.
    - **AVG([DISTINCT] expr):** Returns the average value of expr. The DISTINCT option can be used to return the average of the distinct values of expr**.**

```
MariaDB [myTestDB]> select AVG(amount) from payments
    -> where paymentDate between '2004-01-01' and '2004-03-31';
+--------------+
| AVG(amount)  |
+--------------+
| 32409.015217 |
+--------------+
1 row in set (0.00 sec)
```

- **COUNT(expr):** Returns a count of the number of non-NULL values of expr in the rows retrieved by a SELECT statement. The result is a BIGINT value.

```
MariaDB [myTestDB]> select count(*) from orders;
+----------+
| count(*) |
+----------+
|      326 |
+----------+
1 row in set (0.14 sec)
```

- **MAX([DISTINCT] expr):** Returns the maximum value of expr.

```
MariaDB [myTestDB]> select max(amount) from payments;
+-------------+
| max(amount) |
+-------------+
|   120166.58 |
+-------------+
1 row in set (0.05 sec)
```

- **MIN([DISTINCT] expr):** Returns the minimum value of expr.

```
MariaDB [myTestDB]> select min(amount) from payments;
+-------------+
| min(amount) |
+-------------+
|      615.45 |
+-------------+
1 row in set (0.00 sec)
```

- **SUM([DISTINCT] expr):** Returns the sum of expr. If the return set has no rows, SUM() returns NULL. The DISTINCT keyword can be used to sum only the distinct values of expr.

```
MariaDB [myTestDB]> select sum(amount) from payments;
+-------------+
| sum(amount) |
+-------------+
|  8853839.23 |
+-------------+
1 row in set (0.00 sec)
```

```
MariaDB [ordersdb]> select avg(quantity) from orderdetails;
+---------------+
| avg(quantity) |
+---------------+
|       23.8130 |
+---------------+
```

```
MariaDB [ordersdb]> select count(*) from orderdetails;
+----------+
| count(*) |
+----------+
|     2155 |
+----------+
```

```
MariaDB [ordersdb]> select max(quantity) from orderdetails;
+---------------+
| max(quantity) |
+---------------+
|           130 |
+---------------+
1 row in set (0.004 sec)
```

```
MariaDB [ordersdb]> select min(quantity) from orderdetails;
+---------------+
| min(quantity) |
+---------------+
|             1 |
+---------------+
1 row in set (0.003 sec)
```

```
MariaDB [ordersdb]> select sum(quantity) from orderdetails;
+---------------+
| sum(quantity) |
+---------------+
|         51317 |
+---------------+
1 row in set (0.040 sec)
```

# GROUP BY Clause with Having

- **Group by clause:** GROUP BY clause is used to group selected rows and return a single row of summary information based on that group. The group by clause can have following types of expressions:
  - Constants
  - Group functions
  - The function SYSDATE

  The group by clause can contain not more than 255 expressions.

```
SELECT fieldname on which you group by,
group function(fieldname),….
FROM tablename WHERE condition
GROUP BY fieldname HAVING condition;
```

```
MariaDB [myTestDB]> select city, count(customerNumber)
    -> from customers
    -> group by city;
+-------------------+-----------------------+
| city              | count(customerNumber) |
+-------------------+-----------------------+
| Aachen            |                     1 |
| Allentown         |                     1 |
| Amsterdam         |                     1 |
| Auckland          |                     3 |
| Barcelona         |                     1 |
| Bergamo           |                     1 |
```

**Select city, count(customernumber) from customers group by city limit 10;**

- **When to use Having clause:**
  - When you need to use a aggregate function in a condition.
  - Eg : Display those departments having minimum salary is less than 5000
  - SELECT deptno, MIN(sal) FROM emp WHERE MIN(sal) < 5000 GROUP BY deptno;
  - Will give an error because where clause cannot have a Group function with GROUP BY clause.
  - SELECT deptno, MIN(salary) FROM emp GROUP BY deptno HAVING MIN(sal)<5000;

    **Example:**
    **SELECT CustomerNumber, MIN(amount) FROM Payments GROUP BY paymentdate HAVING min(amount) < 10000 limit 10;**

```
MariaDB [myTestDB]> select city, count(customerNumber)
    -> from customers
    -> group by city
    -> having count(customerNumber) >= 3;
+------------+----------------------+
| city       | count(customerNumber) |
+------------+----------------------+
| Auckland   |                    3 |
| Brickhaven |                    3 |
| Madrid     |                    5 |
| NYC        |                    5 |
| Paris      |                    3 |
| Singapore  |                    3 |
+------------+----------------------+
6 rows in set (0.03 sec)
```

```
select city, count(customernumber)
from customers
group by city
having count(customernumber) >=3;
```

```
SELECT
   ordernumber,
   SUM(quantityOrdered) AS itemsCount,
   SUM(priceeach*quantityOrdered) AS total
FROM
   orderdetails
GROUP BY ordernumber limit 10;
```

```
SELECT
   ordernumber,
   SUM(quantityOrdered) AS itemsCount,
   SUM(priceeach*quantityOrdered) AS total
FROM
   orderdetails
GROUP BY
  ordernumber
HAVING
  total > 1000;
```

```
SELECT
   ordernumber,
   SUM(quantityOrdered) AS itemsCount,
   SUM(priceeach*quantityOrdered) AS total
FROM
   orderdetails
GROUP BY ordernumber
HAVING
   total > 1000 AND
   itemsCount > 600;
```

# Export Data

```
SELECT
   orderNumber, status, orderDate, requiredDate, comments
FROM
   orders
WHERE
   status = 'Cancelled'
INTO OUTFILE 'C:/temp/cancelled_orders.csv'
FIELDS ENCLOSED BY '"'
TERMINATED BY ';'
ESCAPED BY '"'
LINES TERMINATED BY '\r\n';
```

# Introduction to Types of Sub-Queries

**Subqueries** - A subquery is a SELECT statement that is embedded in a Clause of another SELECT statement.

**When is Subquery Used?**

- Whenever you need a result of a particular query to get the necessary result set.
- Subquery is also known as nested queries.

```
SELECT field list FROM table
WHERE fieldname operator ( SELECT statement);
```

**Types of Subqueries:**

- **Single row subqueries:** They are queries that return only one row from the inner select statement.
  Example: When we want to find out the employees of a office in which GEORGE is working.

  ```
  SELECT officeCode FROM employees
  WHERE firstname = 'George';
  ```

  Once it returns the office code (let us say 3) you would then give

  ```
  SELECT firstName, lastName FROM employees
  WHERE officeCode = 3;
  ```

  This can be done using the subquery as follows:

  ```
  SELECT firstName, lastName FROM employees
  WHERE officeCode = (SELECT officeCode FROM employees
  WHERE firstname = 'George');
  ```

- **Multiple row subqueries:** They are queries that return more than one row from the inner select statement.
  Multiple row operator:

  - **IN**: equal to any member in the list
    Example: Display all the employees who are in same office as 'Tom' or 'Martin'.

    ```
    SELECT firstName, lastName FROM employees
    WHERE officeCode IN (SELECT officeCode FROM employees
    WHERE firstName IN ('Tom', 'Martin'));
    ```

  - **ANY**: compare to each value returned by the subquery
    Any Operator:
    - **< ANY** means less than the maximum.
    - **> ANY** means more than the minimum
    - **= ANY** is equal to IN

    Example: Display all the products whose MSRP is less than any Motorcycles and are not Motorcycles.

    ```
    SELECT productCode, productName, MSRP, productLine
    FROM products
    WHERE MSRP > ANY (SELECT MSRP FROM products
    WHERE productLine = 'MotorCycles')
    AND productLine <> 'MotorCycles';
    ```

```
SELECT productCode, productName, MSRP, productLine
FROM products
WHERE MSRP < ANY (SELECT MSRP FROM products
WHERE productLine = 'MotorCycles')
AND productLine <> 'MotorCycles';
```

- **ALL**: compare value to every value returned by the subquery.
- ALL operator:
  - **< ALL** means less than the minimum.
  - **> ALL** means more than maximum.

Example: Display all the productNames whose buyPrice is greater than the average buyPrice of all the categories.

```
SELECT productCode, productName, productLine, buyPrice

FROM products
WHERE buyPrice > ALL (SELECT AVG(buyPrice) FROM
products GROUP BY productLine);
```

Example: Display all the productNames whose buyPrice is less than the average buyPrice of all the categories.

```
SELECT productCode, productName, productLine, buyPrice

FROM products
WHERE buyPrice < ALL (SELECT AVG(buyPrice) FROM
products GROUP BY productLine);
```

**Guidelines for using a subquery:**

- Enclose subqueries in ().
- Place subqueries on the right side of the comparison operator.
- Do not add an ORDER BY clause to a subquery.
- Use single-row operators with single-row subqueries. (<,>,<=,>=,<>)
- Use multiple-row operators with multiple-row subqueries (IN, ANY, ALL).
- We can write a sub query with in a subquery.

**Multiple Column Subqueries:**

Main query compares to values from a multiple-row and multiple-column subquery.

```
SELECT field1, field2,….FROM table

WHERE (field1, field2,..) IN

(SELECT field, field,…FROM table WHERE condition);
```

Example: Display all the employees with same job and office as Pamela.

```
SELECT  firstName, LastName FROm employees

WHERE jobtitle IN (SELECT jobTitle FROM employees

WHERE firstName = 'Pamela')

AND officeCode = (SELECT officeCode FROM employees
WHERE firstName = 'Pamela')
AND firstName <> 'Pamela';
```

OR

```
SELECT  firstName, LastName FROm employees
WHERE (jobTitle, officeCode) IN
(SELECT jobTitle, officeCode FROM employees
WHERE firstName = 'Pamela') AND firstName <> 'Pamela';
```

**Subqueries in DDL and DML**

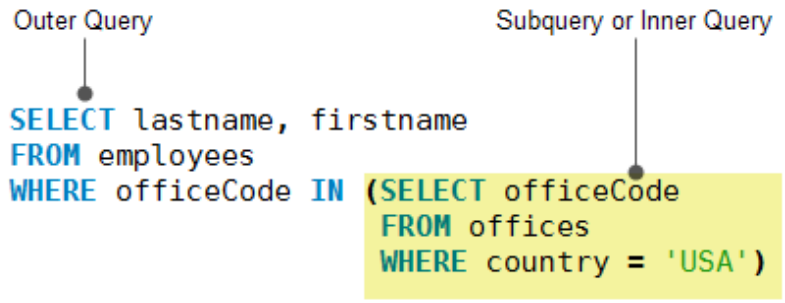
Example of **DML with subquery**: Increase the credit limit for the customers by 5% whose Sales Representative is Larry.

```
UPDATE customers
SET creditLimit = creditLimit + (creditLimit * 0.05)
WHERE salesRepEmployeeNumber =
(SELECT employeeNumber from employees
WHERE firstName = 'Larry');
```

Example of **DDL with subquery**: Create a duplicate employee table named NemEmp.

```
CREATE TABLE NewEmp AS SELECT employeeNumber,
CONCAT(firstName, ' ' , lastName) as fullname,
officeCode FROM employees;
```

Example: The following query uses a subquery to return the employees who work in the offices located in the USA.




```
SELECT  lastName, firstName
FROM employees
WHERE
   officeCode IN (SELECT
         officeCode
      FROM
         offices
      WHERE
         country = 'USA');
```

**Example:** The following query returns the customer who has the highest payment.

```
SELECT
   customerNumber,
   checkNumber,
   amount
FROM
   payments
WHERE
   amount = (SELECT MAX(amount) FROM payments);
```

# Search Condition

## Where Clause:

1) The **Where** clause allows you to specify a search condition for the rows returned by a query.

2) Where Clause is use for to filter the record.

3)The WHERE clause is used to extract only those records that fulfill a specified condition.

Syntax:

select column_list from tableName where search_condition;

```
create database employeedb

use employeedb

create table emp(ID int not null primary key,
Name varchar(20), Salary int, Contact bigint);

insert into emp(ID, Name, Salary, Contact)
values (1, 'Sagar', 10000, 9966558822),
(2, 'Ram', 15000, 8855224411),
(3, 'Rajesh', 8000, 9966554411),
(4, 'sameer', 35000, 7744115566),
(5, 'Kala', 22000, 6622554433),
(6, 'Avinash', 40000, 9569852514),
(7, 'Roshan', 25000, 8855886688);
```

## EMP - Table

| ID | Name | Salary | Contact |
|---|---|---|---|
| 1 | Sagar | 10000 | 9966558822 |
| 2 | Ram | 15000 | 8855224411 |
| 3 | Rajesh | 8000 | 9966554411 |
| 4 | Sameer | 35000 | 7744115566 |
| 5 | Kala | 22000 | 6622554433 |
| 6 | Avinash | 40000 | 9569852514 |
| 7 | Roshan | 25000 | 8855886688 |

**Filter Example:**

**1) Select * from emp where id=3;**

**2) Select id,name,salary from emp where id=5 and name='Kala';**

**3)select * from emp where id=5 or id=6 or id=4;**

**4) Select id,name,contact,salary from emp where id between 1 and 5;**

**5) select * from emp where salary >20000;**

**6) select * from emp where salary >20000 and salary <40000;**

Select * from emp where id=3;

Select id,name,salary from emp where id=5 and name='Kala';

select * from emp where id=5 or id=6 or id=4;

select id,name,contact,salary from emp where id between 1 and 5;

select * from emp where salary >20000;

select * from emp where salary >20000 and salary <40000;

## Comparison operators :

are used in the WHERE clause to determine which records to select. Here is a list of the comparison operators that you can use in MySQL.

| Name | Description |
|---|---|
| > | Greater than operator |
| >= | Greater than or equal operator |
| < | Less than operator |
| <>, != | Not equal operator |
| <= | Less than or equal operator |
| <=> | NULL-safe equal to operator |
| = | Equal operator |

**EMP - Table**

| ID | Name | Salary | Contact |
|---|---|---|---|
| 1 | Sagar | 10000 | 9966558822 |
| 2 | Ram | 15000 | 8855224411 |
| 3 | Rajesh | 8000 | 9966554411 |
| 4 | Sameer | 35000 | 7744115566 |
| 5 | Kala | 22000 | 6622554433 |
| 6 | Avinash | 40000 | 9569852514 |
| 7 | Roshan | 25000 | 8855886688 |

1) select * from emp where id=1 or id=2;

2) select * from emp where id>1 and id<5;

3)select * from emp where salary>=15000 and salary<=40000;

## In Clause:

1) The function returns 1 if expr is equal to any of the values in the IN list, otherwise, returns 0.

2) It is used to help reduce the need for multiple Or condition in a SELECT, INSERT, UPDATE, or DELETE statement.

**EMP - Table**

| ID | Name | Salary | Contact |
|---|---|---|---|
| 1 | Sagar | 10000 | 9966558822 |
| 2 | Ram | 15000 | 8855224411 |
| 3 | Rajesh | 8000 | 9966554411 |
| 4 | Sameer | 35000 | 7744115566 |

1) Select 10 in (12,10,45);

2)select * from emp where id in (1,2,3,6);

3)select * from emp where name in("Sameer",'Raj','Rajesh','Kala');

4)select * from emp where name not in("Sameer",'Raj','Rajesh','Kala');

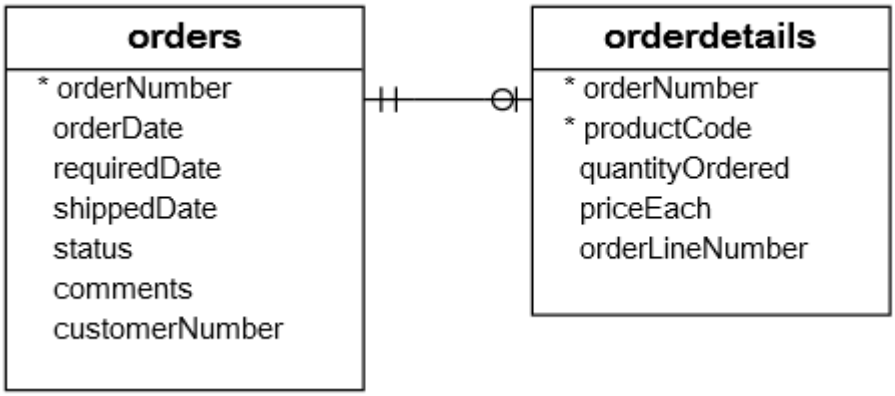
5)select * from emp where salary in(10000,8000,40000,22000,35000);

6) select * from emp where id in( select id from emp where salary > 15000);

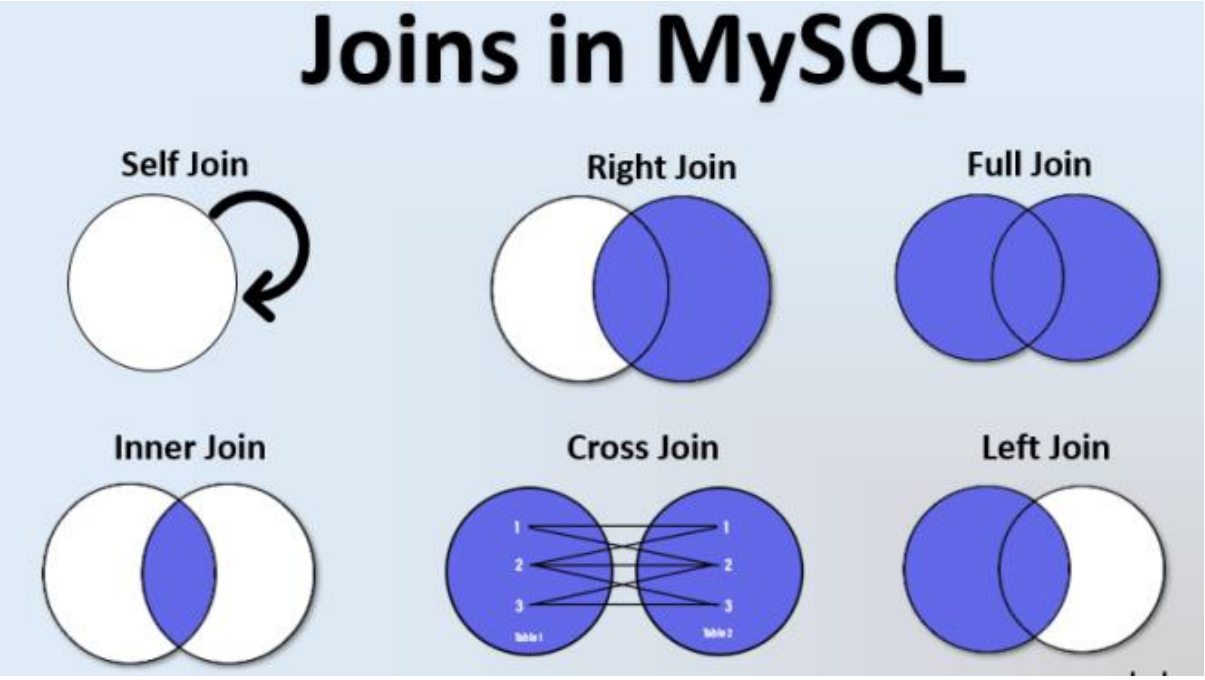
# Introduction to Joins

A relational database consists of multiple related tables linking together using common columns, which are known as foreign key columns. Because of this, data in each table is incomplete from the business perspective.

For example, in the sample database, we have the orders and orderdetails tables that are linked using the orderNumber column:

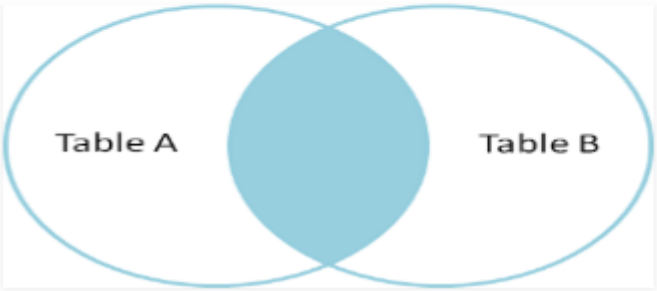


**Types of Joins:**



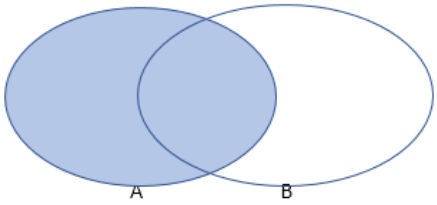

## 1. Inner Join

Inner join returns the value which is matching in both the tables.



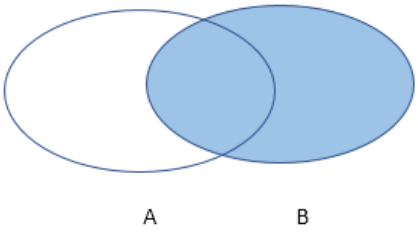
## 2. Left Join

This join returns all the records from the left table and the matched records from the right table.



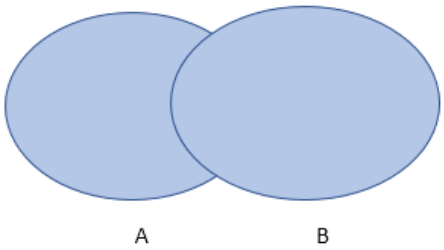

## 3. Right Join

This join returns all the records from the right table and the matching ones from the left table.




## 4. Full Outer Join

The full outer join returns all the records from both the tables if a common field is shared.

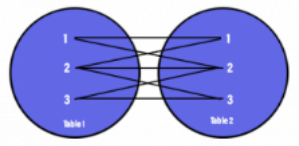


## 5. Self-Join

Self-join is a regular join, and here the table joins with itself only.

## 6. Cross Join

This join produces a result where the number of rows in the first table gets multiplied with the rows in the second table. This kind of result is called the Cartesian Product. If we use the WHERE clause with this join, then this will work as an inner join.




- **Equai Join:** An equijoin is a join which contains an equality operator.
  SELECT tab1.fieldname, tab1.fieldname,
  tab2.fieldname, tab2.fieldname,
  FROM tab1, tab2 WHERE tab1.fieldname = tab2.fieldname;
  Alias can be used in place of table name

Example: SELECT employeeNumber,
CONCAT(firstName, ' ', lastName) AS "Full Name",
o.officeCode, city FROM employees e, offices o
WHERE e.officeCode = o.officeCode;

**Self Join:** It is a join which relates the table to itself.
Example: Display the names of all employees who has manager.

```
SELECT Concat(e1.firstName, ' ', e1.lastName, ' works for ', e2.firstName, ' ', e2.lastName)
AS "Employee works for Manager"
FROM employees e1, employees e2
WHERE e1.reportsTo = e2.employeeNumber;
```

- **Non-equal join:** A join condition using any other operator than '=' is a non-equal join.
  Example: Display customer details along with order details between 12-Jan-2003 to 31-Mar-2003.

```
SELECT c.customerName, o.orderNumber, o.orderDate,
od.quantityOrdered * od.priceEach as "Total Cost"
FROM customers c, orders o, orderdetails od
WHERE c.customerNumber = o.customerNumber and
o.orderNumber = od.orderNumber and
o.orderDate between '2003-01-12' and '2003-03-31';
```

**On Clause:** The ON clause is used to join tables where the column names don't match in both tables. The join conditions are removed from the filter conditions in the WHERE clause. In other words, the ON clause is similar to the WHERE clause as you can put multiple conditions in the ON clause.

```
SELECT Concat(e.firstName, ' ', e.lastname) as "EmployeeName", c.customerName FROM employees
e JOIN customers c
ON (e.employeeNumber = c.SalesRepEmployeeNumber);
```

- **Using Clause:** The columns listed in the USING clause must be present in both of the two tables being joined. The USING clause will be transformed to an ON clause that checks for equality between the named columns in the two tables.
  ```
  SELECT concat(e.firstname, ' ', e.lastname), o.city
  FROM employees e JOIN offices o USING (officeCode);
  ```

- **Cross Join:** This join produces a Cartesian product. CROSS JOIN returns the Cartesian product of the sets of rows from the joined tables. Cartesian product is a join of every row of one table to every row of another table. This normally happens when no matching join columns are specified.
  ```
  SELECT CONCAT(e.firstName, ' ', e.lastname) AS "EmployeeName", c.customerName
  FROM employees e CROSS JOIN customers c;
  ```

- **Outer Join**: An outer join returns all rows that satisfy the join condition and also returns some or all of those rows from one table for which no rows from the other table satisfy the join condition. Such rows are not satisfied by a simple join.
  - **Left Outer Join**:
    ```
    SELECT e.firstName, c.customerName
    FROM employees e LEFT OUTER JOIN customers c
    ON (e.employeeNumber=c.SalesRepEmployeeNumber);
    ```

  - **Right Outer Join**:
    ```
    SELECT e.firstName, c.customerName
    FROM employees e RIGHT OUTER JOIN customers c
    ON (e.employeeNumber=c.SalesRepEmployeeNumber);
    ```

# Views

Views in SQL are kind of virtual tables.

A view also has rows and columns as they are in a real table in the database.

We can create a view by selecting fields from one or more tables present in the database.

A View can either have all the rows of a table or specific rows based on certain condition.

**Why Use Views?**

- To restrict data access
- To make complex queries easy
- To provide data independence
- To present different views of the same data

**Types of Views:**

There are two types of views:

- Simple View
- Complex Views

**Features of Views:**

| **Features** | **Simple Views** | **Complex Views** |
|---|---|---|
| Numbers of tables | One | One or More |
| Contains PL/SQL (Function) | No | Yes |
| Contains group of data | No | Yes |
| DML Operations through a view | Yes | Not Always |

**Creating or Modifying a View:**

You embed a subquery within the CREATE VIEW statement.

```
CREATE [OR REPLACE] VIEW view_name
AS subquery
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY [CONSTRAINT constraint]];
```

The subquery can contain complex SELECT syntax.

```
mysql> CREATE VIEW cust AS
    -> SELECT customerNumber, customerName, creditLimit
    -> FROM customers WHERE creditLimit >= 65000;
Query OK, 0 rows affected (0.26 sec)
```

```
mysql> SELECT * from cust;
+----------------+-----------------------------------+-------------+
| customerNumber | customerName                      | creditLimit |
+----------------+-----------------------------------+-------------+
|            112 | Signal Gift Stores                |    71800.00 |
|            114 | Australian Collectors, Co.        |   117300.00 |
|            119 | La Rochelle Gifts                 |   118200.00 |
|            121 | Baane Mini Imports                |    81700.00 |
|            124 | Mini Gifts Distributors Ltd.      |   210500.00 |
|            131 | Land of Toys Inc.                 |   114900.00 |
|            141 | Euro+ Shopping Channel            |   227600.00 |
|            145 | Danish Wholesale Imports          |    83400.00 |
|            146 | Saveley & Henriot, Co.            |   123900.00 |
```

Create a view by using column aliases in the subquery:

```
mysql> CREATE VIEW emp_OC235 AS
    -> SELECT employeeNumber, CONCAT(firstName, ' ', lastName) as "Full Name"
    -> FROM employees WHERE officeCode IN (3, 5, 2);
Query OK, 0 rows affected (0.14 sec)
```

```
mysql> SELECT * FROM emp_OC235;
+----------------+-----------------+
| employeeNumber | Full Name       |
+----------------+-----------------+
|           1188 | Julie Firrelli  |
|           1216 | Steve Patterson |
|           1286 | Foon Yue Tseng  |
|           1323 | George Vanauf   |
|           1621 | Mami Nishi      |
|           1625 | Yoshimi Kato    |
+----------------+-----------------+
6 rows in set (0.00 sec)
```

Modifying the emp_OC235 view by using ALTER VIEW clause or CREATE OR REPLACE VIEW clause. Add an alias for each column name.

```
mysql> ALTER VIEW emp_OC235 AS
    -> SELECT employeeNumber EmpNO,
    -> CONCAT(firstName, ' ', lastName) "Full Name",
    -> officeCode oCode
    -> FROM employees WHERE officeCode IN (2, 3, 5);
Query OK, 0 rows affected (0.17 sec)
```

```
mysql> CREATE OR REPLACE VIEW emp_OC235 AS
    -> SELECT employeeNumber EmpNO,
    -> CONCAT(firstName, ' ', lastName) "Full Name",
    -> officeCode oCode
    -> FROM employees WHERE officeCode IN (2, 3, 5);
Query OK, 0 rows affected (0.21 sec)
```

**Creating a Complex View:** Create a complex view that contains group functions to display values from two tables.

```
mysql> CREATE OR REPLACE VIEW SalesRepWiseCL
    -> AS
    -> SELECT e.firstName,
    -> min(c.creditLimit) minCL, max(c.creditLimit) maxCL,
    -> avg(c.creditLimit) avgCL
    -> FROM employees e, customers c
    -> WHERE e.employeeNumber = c.salesRepEmployeeNumber
    -> GROUP BY e.firstName;
Query OK, 0 rows affected (0.04 sec)
```

DROP VIEW IF EXISTS SalesRepwiseCL;

## *Importing Table thru Command Prompt*

Create Database:

```
MariaDB [(none)]> create database discounts;
Query OK, 1 row affected (0.001 sec)
```

Use Discounts:

```
MariaDB [(none)]> use discounts;
Database changed
MariaDB [discounts]>
```

Create Table:

```
MariaDB [discounts]> CREATE TABLE discounts (
    ->     id INT NOT NULL AUTO_INCREMENT,
    ->     title VARCHAR(255) NOT NULL,
    ->     expired_date DATE NOT NULL,
    ->     amount DECIMAL(10 , 2 ) NULL,
    ->     PRIMARY KEY (id)
    -> );
Query OK, 0 rows affected (0.074 sec)
```

discounts.csv

```
1  id,title,expired date,amount
2  1,"Spring Break 2014",20140401,20
3  2,"Back to School 2014",20140901,25
4  3,"Summer 2014",20140825,10
```

## Load Data:

```
LOAD DATA INFILE 'c:/temp/discounts.csv'
INTO TABLE discounts
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;

SELECT * FROM discounts;
```

---

*Importing Database thru SQL*

---

```
MariaDB [(none)]> source c:\temp\mysqlsampledatabase.sql
```

```
MariaDB [classicmodels]> show tables;
+-------------------------+
| Tables_in_classicmodels |
+-------------------------+
| customers               |
| employees               |
| offices                 |
| orderdetails            |
| orders                  |
| payments                |
| productlines            |
| products                |
+-------------------------+
8 rows in set (0.001 sec)
```

## *Importing Table thru PhpMyAdmin*

## *Exporting Table thru Command Prompt*

```
Use classicmodels;
SELECT
   orderNumber, status, orderDate, requiredDate, comments
FROM
   orders
WHERE
   status = 'Cancelled';
```

```
SELECT
    orderNumber, status, orderDate, requiredDate, comments
FROM
    orders
WHERE
    status = 'Cancelled'
INTO OUTFILE 'C:/temp/cancelled_orders.csv'
FIELDS ENCLOSED BY '"'
TERMINATED BY ';'
ESCAPED BY '"'
LINES TERMINATED BY '\r\n';
```

```
SELECT
    orderNumber, status, orderDate, requiredDate, comments
FROM
    orders
INTO OUTFILE 'C:/temp/orders.csv'
FIELDS ENCLOSED BY '"'
TERMINATED BY ';'
ESCAPED BY '"'
LINES TERMINATED BY '\r\n';
```

## *Exporting Table thru PhpMyAdmin*

# What is a Key..

- A Key is a data item that exclusively identifies a record.
- Keys are also used to generate relationship among different database tables.
- Key can be single attribute of a group of attribute.

| StudentID | Name | Class | DOB | Email |
|---|---|---|---|---|
| 1001 | John | 1st Year | 01-08-2008 | john1@mycollege.com |
| 1002 | Edward | 3rd Year | 04-12-2005 | edward@mycollege.com |
| 1002 | Peter | 4th Year | 03-09-2003 | peter@mycollege.com |
| 1004 | Smith | 2nd Year | 09-30-2006 | smith@mycollege.com |
| 1005 | John | 1st Year | 01-17-2007 | john2@mycollege.com |

# Primary Key

- Primary key is a set of one or more fields ( columns) of a table that uniquely identify a record in database table.
- A table can have only one primary key and one candidate key can select as a primary key.
- The primary key should be chosen such that its attributes are never or rarely changed.
- Cannot contain "**NULL**".
- Primary key field contain a clustered index.

| StudentID | Name | Class | DOB | Email |
|---|---|---|---|---|
| 1001 | John | 1st Year | 01-08-2008 | john1@mycollege.com |
| 1002 | Edward | 3rd Year | 04-12-2005 | edward@mycollege.com |
| 1003 | Peter | 4th Year | 03-09-2003 | peter@mycollege.com |
| 1004 | Smith | 2nd Year | 09-30-2006 | smith@mycollege.com |
| 1005 | John | 1st Year | 01-17-2007 | john2@mycollege.com |

***studentInfo Table***

# Unique Key:

- A unique key is a set of one or more attribute that can be used to uniquely identify the records in table.
- Unique key is similar to primary key but unique key field can contain a "**Null**" value but primary key doesn't allow "**Null**" value.
- Unique field contain a non-clustered index.

| StudentID | Name | Class | DOB | Email |
|---|---|---|---|---|
| 1001 | John | 1st Year | 01-08-2008 | john1@mycollege.com |
| 1002 | Edward | 3rd Year | 04-12-2005 | edward@mycollege.com |
| 1002 | Peter | 4th Year | 03-09-2003 | peter@mycollege.com |
| 1004 | Smith | 2nd Year | 09-30-2006 | smith@mycollege.com |
| 1005 | John | 1st Year | 01-17-2007 | john2@mycollege.com |

***studentInfo Table***

# Foreign Key:

- Foreign Key is a field in database table that is Primary key in another table.
- Foreign key is used to generate the relationship between the tables.
- A foreign key can accept null and duplicate value.

*studentInfo Table*

| StudentID | Name | Class | DOB | Email |
|---|---|---|---|---|
| 1001 | John | 1st Year | 01-08-2008 | john1@mycollege.com |
| 1002 | Edward | 3rd Year | 04-12-2005 | edward@mycollege.com |
| 1003 | Peter | 4th Year | 03-09-2003 | peter@mycollege.com |
| 1004 | Smith | 2nd Year | 09-30-2006 | smith@mycollege.com |
| 1005 | John | 1st Year | 01-17-2007 | john2@mycollege.com |

*invoiceInfo Table*

| StudentID | InvoiceNo | Amount | InvoiceDate |
|---|---|---|---|
| 1001 | A-5789 | $1500.00 | 04-11-2020 |
| 1002 | A-8854 | $1000.00 | 05-17-2019 |
| 1002 | A-9845 | $250.00 | 07-25-2020 |
| 1004 | A-2323 | $750.00 | 01-18-2021 |
| 1011 | A-6841 | $70.00 | 12-13-2021 |

**SELECT** *expressions*

**From** *table_name*

[**WHERE** *clause*]

[**GROUP BY** *clause*]

[**HAVING** *clause*]

[**ORDER BY** *clause*]

**code in brackets is optional*

**words in bold letters are key words or reserved words*

# Expressions

- An expression is a combination of one or more values, operators and SQL functions that evaluate to a value.
- These SQL EXPRESSIONs are like formulae and they are written in query language.
- You can also use them to query the database for a specific set of data.
- SELECT column1, column2, columnN FROM table_name WHERE [EXPRESSION | CONDITION];

# Conditions

- A condition specifies a combination of one or more expressions and logical (Boolean) operators and returns a value of TRUE, FALSE, or unknown.
- SELECT *

  FROM suppliers

  WHERE (state = 'California' AND supplier_id <> 900)

  OR (supplier_id = 100);

**Primary Key**

```
CREATE TABLE table_name(
   primary_key_column datatype PRIMARY KEY,
   ...
);
```

```
CREATE TABLE table_name(
   primary_key_column1 datatype,
   primary_key_column2 datatype,
   ...,
   PRIMARY KEY(column_list)
);
```

```
CREATE TABLE users(
  user_id INT AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(40),
  password VARCHAR(255),
  email VARCHAR(255)
);
```

**Foreign Key Constraint**

```
[CONSTRAINT constraint_name]
FOREIGN KEY [foreign_key_name] (column_name, ...)
REFERENCES parent_table(colunm_name,...)
[ON DELETE reference_option]
[ON UPDATE reference_option]
```

```
CREATE TABLE categories(
   categoryId INT AUTO_INCREMENT PRIMARY KEY,
   categoryName VARCHAR(100) NOT NULL
```

```
) ENGINE=INNODB;

CREATE TABLE products(
    productId INT AUTO_INCREMENT PRIMARY KEY,
    productName varchar(100) not null,
    categoryId INT,
    CONSTRAINT fk_category
    FOREIGN KEY (categoryId)
        REFERENCES categories(categoryId)
) ENGINE=INNODB;
```

**Unique Key**

```
CREATE TABLE table_name(
   ...
   column_name1 column_definition,
   column_name2 column_definition,
   ...,
   UNIQUE(column_name1,column_name2)
);

CREATE TABLE suppliers (
    supplier_id INT AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    phone VARCHAR(15) NOT NULL UNIQUE,
    address VARCHAR(255) NOT NULL,
    PRIMARY KEY (supplier_id),
    CONSTRAINT uc_name_address UNIQUE (name , address)
);
```

**Check Constrain**

**CREATE TABLE parts (**

```
   part_no VARCHAR(18) PRIMARY KEY,
   description VARCHAR(40),
   cost DECIMAL(10,2 ) NOT NULL CHECK (cost >= 0),
   price DECIMAL(10,2) NOT NULL CHECK (price >= 0)
);

INSERT INTO parts(part_no, description,cost,price)
VALUES('A-001','Cooler',0,-100);

Error Code: 3819. Check constraint 'parts_chk_2' is violated.

DROP TABLE IF EXISTS parts;

CREATE TABLE parts (
   part_no VARCHAR(18) PRIMARY KEY,
   description VARCHAR(40),
   cost DECIMAL(10,2 ) NOT NULL CHECK (cost >= 0),
   price DECIMAL(10,2) NOT NULL CHECK (price >= 0),
   CONSTRAINT parts_chk_price_gt_cost
      CHECK(price >= cost)
);

SHOW CREATE TABLE parts;

INSERT INTO parts(part_no, description,cost,price)
VALUES('A-001','Cooler',200,100);

Error Code: 3819. Check constraint 'parts_chk_price_gt_cost' is violated.
```