

# Structured Query Language

## SQL - Day 1

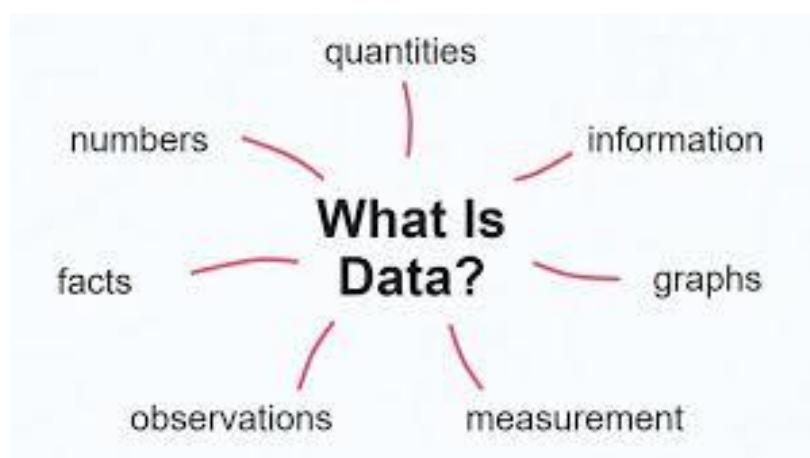
### Agenda

- Data
- Data Base.
- Data Base Management System (DBMS).
- Relational Database RDBMS.
- Structured Query Language.



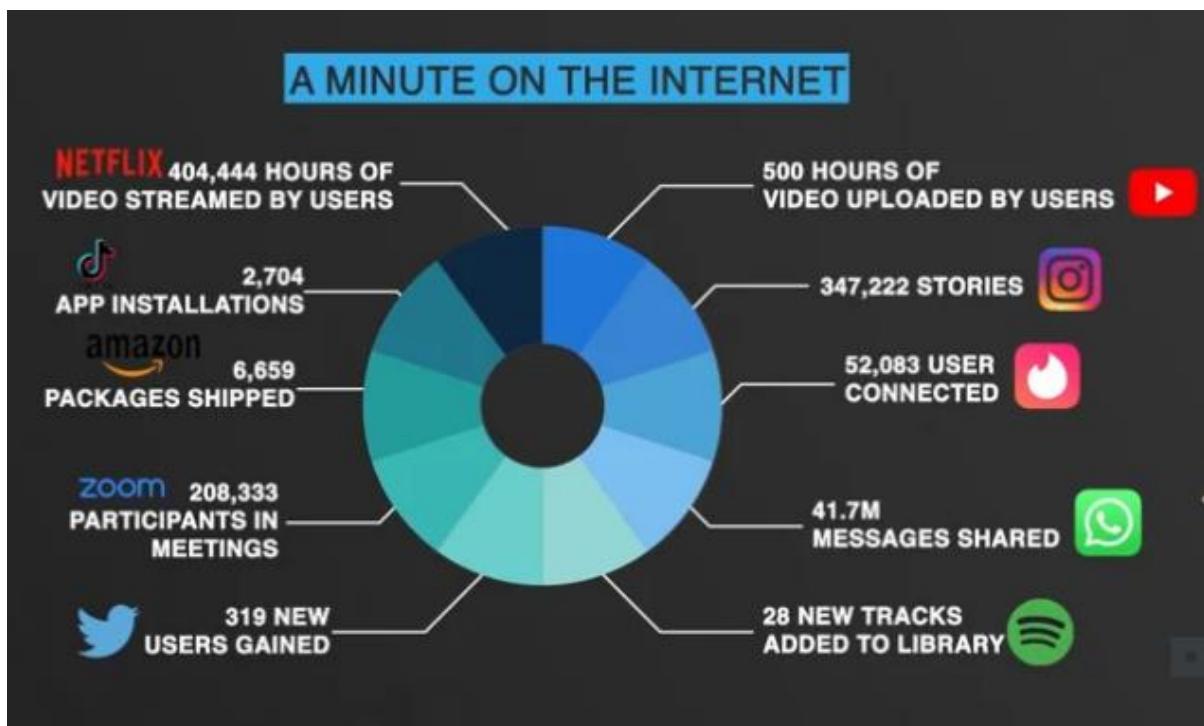
### What is Data?

Data is a collection of facts, such as numbers, words, measurements, observations or just descriptions of things.



# Date Is Huge Collection of Information.

There are huge collections of data generated in a minute from various different sources.



## Where all this data is stored?

Well! All this data is stored in an area called as "Database".



# **What is DATA BASE?**

- A database is an organized collection of structured information, or data, typically stored electronically in a computer system.
- A Database is a systematic collection of information.
- A Database is an arranged format of the data.
- A Database is an organized collection of information.



## **How these huge data is managed?**

### **What is data managing?**

Data can be Created

Data can be Read

Data can be Updated

Data can be Deleted



Whichever data is generated can be managed or can be performed with 4 operations i.e. CRUD(Create Read Update Delete) operation. Performing CRUD operations on data is only data managing.

**These huge data is managed by one of the system known as DBMS or Data Base Management System.**

# How this huge data is organized?

Database maintains "tabular format" to maintain the data in organized form.

Data generated in the source

	Name: Arun Username: arun_aru
	E-mail ID: arunaru@gmail.com Password: *****
	Name: Azhar Username: azhar_cheeku
	E-mail ID: azhar75@gmail.com Password: *****
	Name: Kranthi Username: knox_clicks
	E-mail ID: kranthi75@gmail.com Password: *****



Name	Username	E-mail ID	Password
Arun	arun_aru	arunaru@gmail.com	*****
Azhar	azhar_cheeku	azhar75@gmail.com	*****
Kranthi	knox_clicks	kranthi75@gmail.com	*****

Data stored in Database

Data is stored in table format inside the database. Tabular format or table format contains rows and columns to store data in organized or structured manner. The database which stores or manages the data in the form of rows and columns or tabular form is known as "Relational Database Management".

## Advantage of storing data in tabular form.

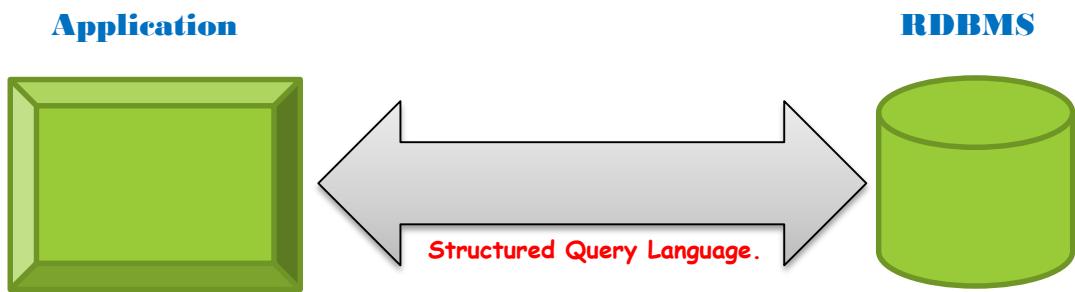
1. The large mass of confusing data is easily reduced to reasonable form that is understandable.
2. A table facilitates representation of even large amounts of data in an attractive, easy to read, easy to access form.

Data present inside the Relational Database Management is managed using the system known as "Relational Database Management System or RDBMS."

## What is SQL?



SQL stands for Structured Query Language. SQL query language is a kind of programming language that's designed to facilitate retrieving specific information from relational databases.



SQL is a query language which is used to communicate between the application and the Relational Database Management System or RDBMS.

**SQL is called as Structured Query Language because:**

**Structured:** SQL deals with RDBMS, where RDBMS is a database System which stores the data in **structured form**.

**Query Language:** SQL language uses queries to communicate with Database.

## How this Query Language Looks Like?



TO BE CONTINUED.....

# **Structured Query Language**

## **SQL - Day 2**

### **Agenda**

- Entity
- Different types of attributes.
  - Multi valued attributes.
  - Simple/ Atomic attributes.
  - Compound/Composite attributes.
  - Derived attributes.
  - Stored attributes
  - Complex attributes
  - Key attributes
  - Non-Key attributes
  - Required attributes
  - Optional / null value attributes.

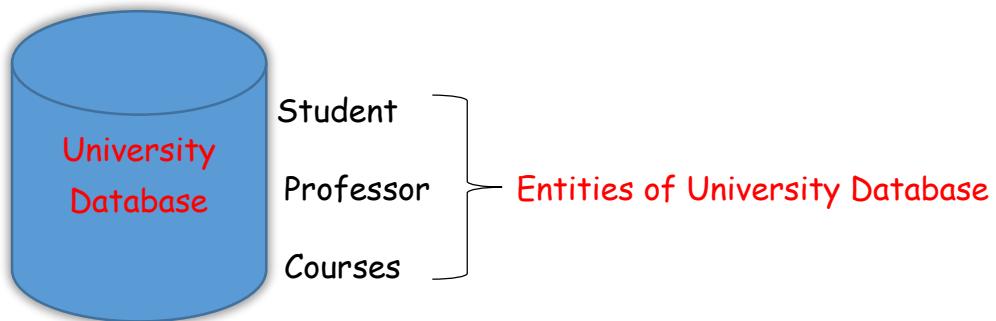


# What is an Entity?

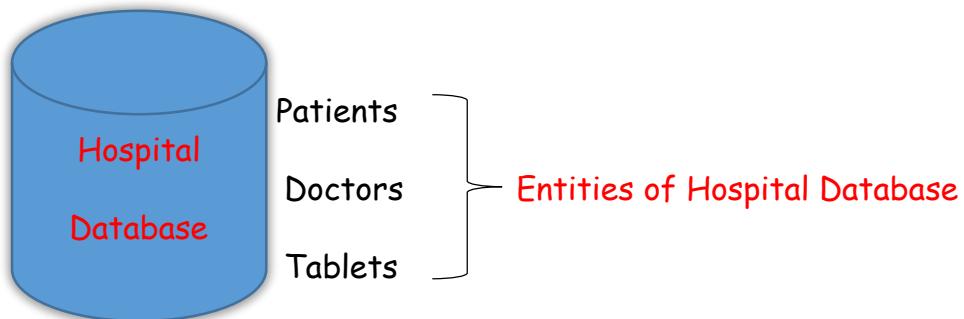
Objects whose information can be stored inside a database are referred as "Entity".

Let's look into few examples of entities in database.

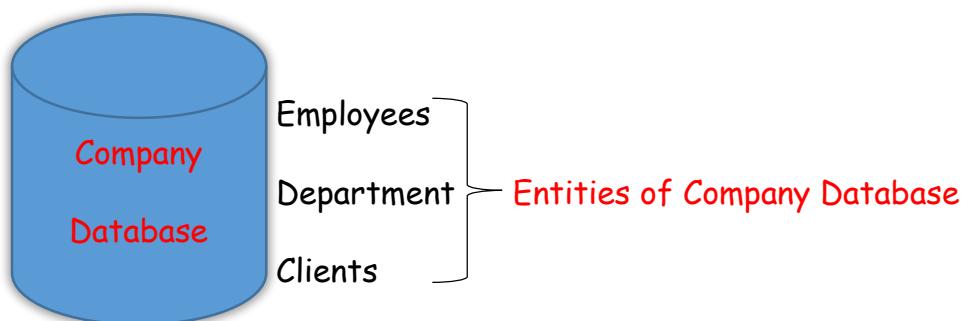
Example 1:



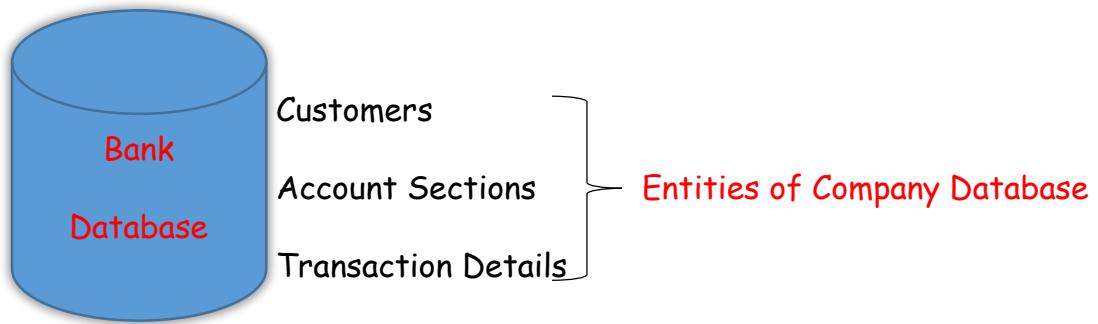
Example 2:



Example 3:



Example 4:



## Types of Entities:

- 1. Tangible Entity**
- 2. Intangible Entity**

### Tangible Entity:

Tangible Entities are those entities which exist in the real world physically.

Example: student, professor, doctors, employees, clients, customers etc.

### Intangible Entity:

Intangible Entity: Intangible Entities are those entities which exist only logically and have no physical existence.

Example: Bank Account, courses, tablets, department, transaction details, account sections.

# Let Us understand what is



**Table**

**Rows**

**Columns**

**Cells**

## Table:

A table is a data structure that organizes information into rows and columns. It can be used to both store and display data in a structured format.

USN	NAME	MIDDLE NAME	DOB	AGE	GENDER	CONTACT NUMBER	E-MAIL ID	ADDRESS	YEAR OF JOINING
15EC11	Wil Williams	John	26-03-2003	18	M	+44 99991 99999	will@gmail.com	1 N. Gainsway Ave.Apt I Oxnard, CA 93035	2015
11MBA1	Dave Davidson	Kate	28-06-1999	21	M	+67 99993 12999	dave@gmail.com	721 Plymouth St. Glendale, AZ 85302	2011
19ME12	Mercy Janis	Mike	21-04-2001	19	F	+55 79923 12999	mercy@yahoo.com	20 Berkshire Drive West Fargo, ND 58078	2019
20BSC1	Don Demarco	Jan	18-02-2001	19	M	+34 98193 12999	don@yahoo.com	8306 S. Central Rd. Worcester, MA 01604	2020
17CS42	Wil Williams	Andy	29-02-2003	17	M	+58 98193 12999	will17@gmail.com	9798 Talbot St. New Rochelle, NY 10801	2017

For example, databases store data in tables so that information can be quickly accessed.

## Rows:

A row is a series of data placed out horizontally in a table. It is a horizontal arrangement of the objects, words, numbers, and data. In Row, data objects are arranged face-to-face with lying next to each other on the straight line.

Rows are also called as tuples or record.

USN	NAME	MIDDLE NAME	DOB	AGE	GENDER	CONTACT NUMBER	E-MAIL ID	ADDRESS	YEAR OF JOINING
15EC11	Wil Williams	John	26-03-2003	18	M	+44 99991 99999	will@gmail.com	1 N. Gainsway Ave.Apt 1 Oxnard, CA 93035	2015
11MBA1	Dave Davidson	Kate	28-06-1999	21	M	+67 99993 12999	dave@gmail.com	721 Plymouth St. Glendale, AZ 85302	2011
19ME12	Mercy Janis	Mike	21-04-2001	19	F	+55 79923 12999	mercy@yahoo.com	20 Berkshire Drive West Fargo, ND 58078	2019
20BSC1	Don Demarco	Jan	18-02-2001	19	M	+34 98193 12999	don@yahoo.com	8306 S. Central Rd. Worcester, MA 01604	2020
17CS42	Wil Williams	Andy	29-02-2003	17	M	+58 98193 12999	will17@gmail.com	9798 Talbot St. New Rochelle, NY 10801	2017

# Columns

A column is a vertical series of cells in a table. It is an arrangement of figures, facts, words, etc.

Columns are mostly placed one after another in the continuous sequence. In a table, columns are mostly separated from each other by lines, which help to enhance readability and attractiveness.

Columns are also called as attributes.

USN	NAME	MIDDLE NAME	DOB	AGE	GENDER	CONTACT NUMBER	E-MAIL ID	ADDRESS	YEAR OF JOINING
15EC11	Wil Williams	John	26-03-2003	18	M	+44 99991 99999	will@gmail.com	1 N. Gainsway Ave.Apt I Oxnard, CA 93035	2015
11MBA1	Dave Davidson	Kate	28-06-1999	21	M	+67 99993 12999	dave@gmail.com	721 Plymouth St. Glendale, AZ 85302	2011
19ME12	Mercy Janis	Mike	21-04-2001	19	F	+55 79923 12999	mercy@yahoo.com	20 Berkshire Drive West Fargo, ND 58078	2019
20BSC1	Don Demarco	Jan	18-02-2001	19	M	+34 98193 12999	don@yahoo.com	8306 S. Central Rd. Worcester, MA 01604	2020
17CS42	Wil Williams	Andy	29-02-2003	17	M	+58 98193 12999	will17@gmail.com	9798 Talbot St. New Rochelle, NY 10801	2017

## Cells:

The intersections of both rows and columns is known as cells.

Cells are also known as Fields.

USN	NAME	MIDDLE NAME	DOB	AGE	GENDER	CONTACT NUMBER	E-MAIL ID	ADDRESS	YEAR OF JOINING
15EC11	Wil Williams	John	26-03-2003	18	M	+44 99991 99999	will@gmail.com	1 N. Gainsway Ave.Apt I Oxnard, CA 93035	2015
11MBA1	Dave Davidson	Kate	28-06-1999	21	M	+67 99993 12999	dave@gmail.com	721 Plymouth St. Glendale, AZ 85302	2011
19ME12	Mercy Janis	Mike	21-04-2001	19	F	+55 79923 12999	mercy@yahoo.com	20 Berkshire Drive West Fargo, ND 58078	2019
20BSC1	Don Demarco	Jan	18-02-2001	19	M	+34 98193 12999	don@yahoo.com	8306 S. Central Rd. Worcester, MA 01604	2020
17CS42	Wil Williams	Andy	29-02-2003	17	M	+58 98193 12999	will17@gmail.com	9798 Talbot St. New Rochelle, NY 10801	2017

## Let's Understand Different types of Attributes or Columns.

Different types of attributes/columns available are:

- Single valued attributes.
- Multi valued attributes.
- Simple/ Atomic attributes.
- Compound/Composite attributes.
- Derived attributes.
- Stored attributes
- Complex attributes
- Key attributes
- Non-Key attributes
- Required attributes
- Optional / null value attributes.



**Let's understand each type of attributes one by one...**



## **Single valued attributes.**

Single-valued attribute is an attribute that can have only a single value.

For example: A person can have only one age, only one gender, and a manufactured part can have only one serial number.

## **Multi valued attributes.**

An attribute that can hold multiple values is known as multivalued attribute. It is represented with double ovals in an ER Diagram.

For example - A person can have more than one phone numbers or email ids so the phone number or email id attributes are multivalued attribute.

## **Composite attribute**

Composite attribute is an attribute where the values of that attribute can be further subdivided into meaningful sub-parts."

For examples - Persons 'Name' is a composite attribute which can be stored as first name, last name, middle initial.

'Address' is a composite attribute which can be divided into street number, city, state, country etc.

## **Simple/ Atomic attributes.**

Simple/ Atomic attributes is an attribute where the values of that attribute cannot be further subdivided into sub-parts.

For examples - Date of birth of a person cannot be further divided into sub parts.

## **Derived attribute**

Derived attributes are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database.

For example:

- Age of a person can be derived from the date of birth attribute.
- Duration of the course can be derived from start date of joining and end date of course.

## **Stored attribute.**

Stored attributes are the attributes whose values cannot be derived using other attributes.

For Example - student\_id, name, roll\_no, course\_Id, mobile number, email.

## **Complex attribute.**

Complex Attribute is a type of attribute in database. It is formed by nesting composite attributes and multi-valued attributes in arbitrary way. We can say this as the both are in the attribute.

For Example- Address attribute, address can be a combination of all the other attributes like street Number, Locality Number, city, state, country.

## **Key attribute.**

A key attribute is the unique characteristic of the entity.

For Example- USN of Student, Bank Account Number, Employee Id.

## **Non-Key Attribute.**

Attributes that are not unique (attributes other than Key Attribute).

For Example- Name, Address, DOB, AGE.

## **Required Attribute.**

A required attribute is an attribute that must have a value in it.

For Example- Name, Address, DOB, AGE.

## **Optional/null value Attribute.**

Optional attribute may or may not have a value in it or it can be left blank.

For Example-Middle name of person.

**Thank You** ☺☺

# **Structured Query Language**

## **SQL - Day 3**

### **Agenda**

- **ER Diagram**
- **Relationship**
  - **One : One**
  - **One : Many**
  - **Many : One**
  - **Many : Many**
- **Cardinality Ratio.**
- **Strong Entity**
- **Weak Entity**
- **Weak Relationship**
- **Identifying Relationship**
- **Participation**
  - **Total Participation**
  - **Partial Participation**
- **Diagrams/Notations Used in ER Diagram.**
- **Case Study**



## **What is an ER Diagram?**

An Entity relationship diagram (ERD) shows the relationships of entity sets stored in a database. By defining the entities, their attributes, and showing the relationships between them, an ER diagram illustrates the logical structure of databases. ER diagrams are used to sketch out the design of a database.

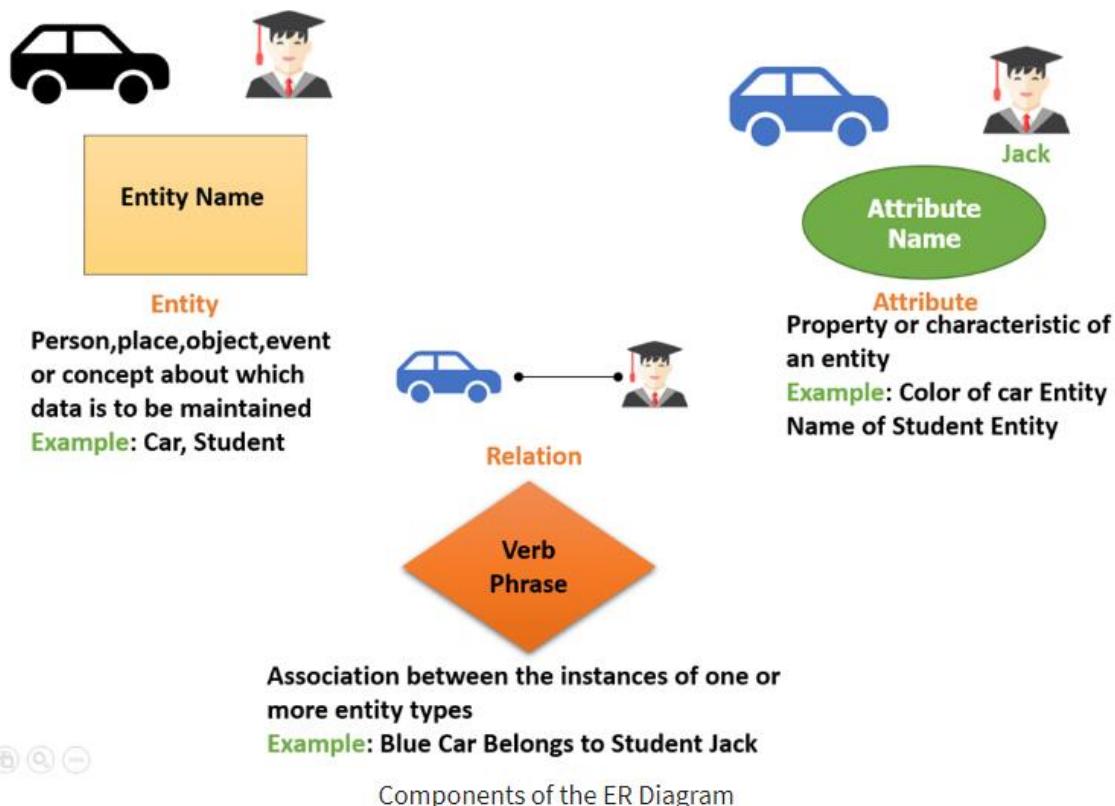
## **Why ER Diagram?**

ER diagram helps to analyse data requirements systematically to produce a well-designed database. So, it is considered a best practice to complete ER modelling before implementing your database.



# Example of ER Diagram?

For example, in a University database, we might have entities for Students, Courses, and Lecturers. Student's entity can have attributes like Rollno, Name, and DeptID. They might have relationships with Courses and Lecturers.



# Relationship/ Cardinality Ratio & Types of Relationship

## What is Relationship/ Cardinality Ratio?

Association between two entities is considered as relationship.

Cardinality tells how many times the entity of an entity set participates in a relationship

## Types of Relationship/ Cardinality Ratio:

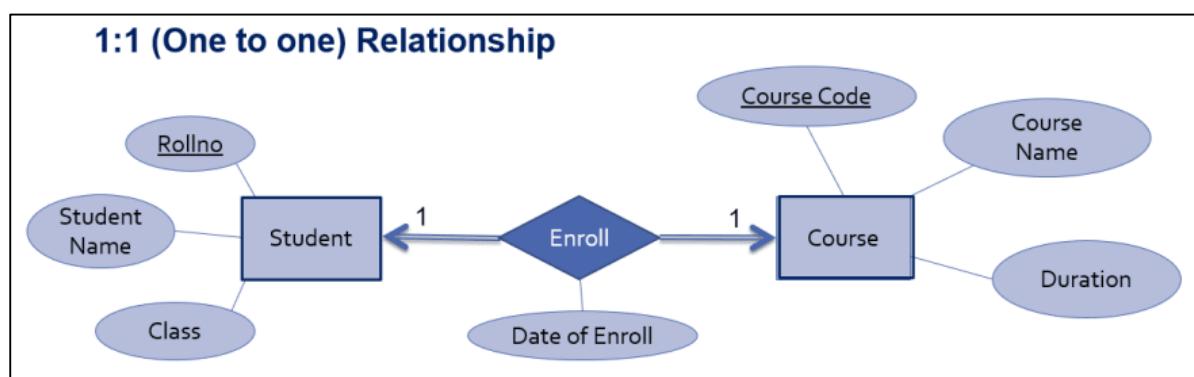
- One : one
- One : Many
- Many : Many
- Many : One

### One : One Relationship

In One-to-one relationship, one record in a table is associated with one and only one record in another table.

For Example:

Consider same relationship set enroll exist between entity sets student and course , which means one student can enroll in only one courses

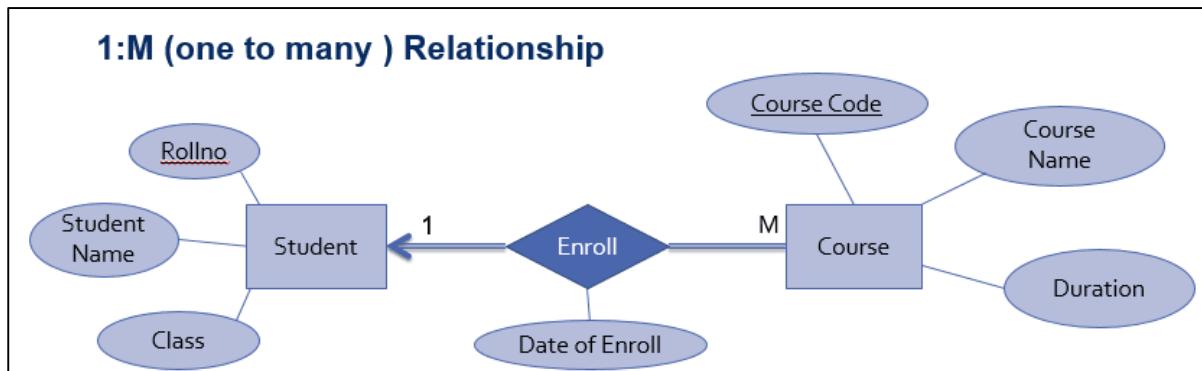


## One : Many Relationship

A one-to-many relationship occurs when one record in a table are associated with multiple records in another table.

For Example:

Consider 1:M relationship set enrolled exist between entity sets student and course as follow.

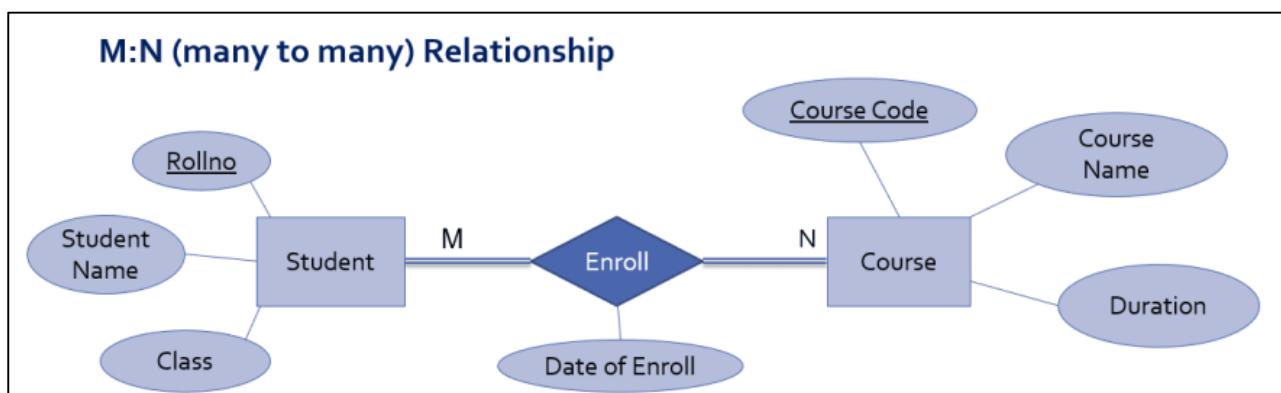


## Many : Many Relationship.

A many-to-many relationship occurs when multiple records in a table are associated with multiple records in another table.

For Example:

Consider same relationship set enrolled exist between entity sets student and course ,which means multiple student can enroll in multiple courses.

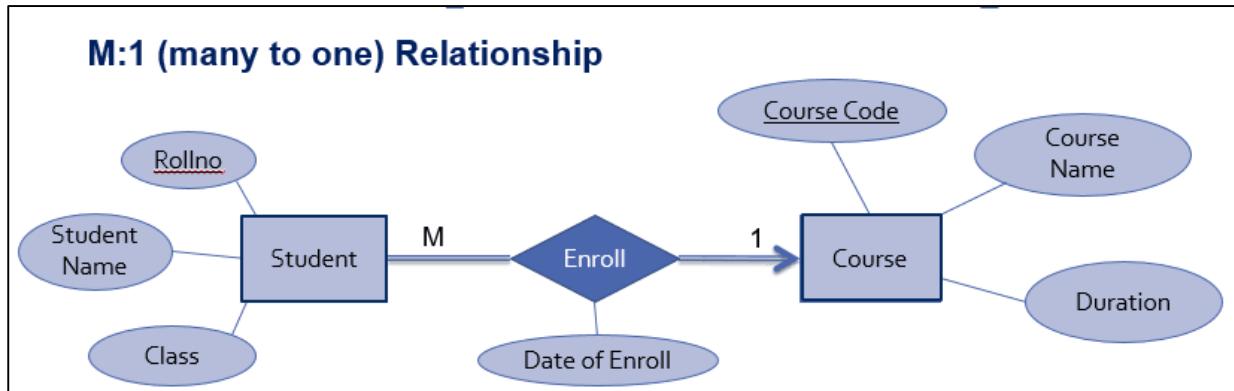


## Many : One Relationship.

In Many-to-one relationship many side will keep reference of the one side.

For Example:

Consider same relationship set enroll exist between entity sets student and course . but here student is many side entity set while course is one side entity set. Which means many student can enroll in one course.



## What is Weak Entity.?

A weak entity is an entity set that does not have sufficient attributes for Unique Identification of its records. Simply a weak entity is nothing but an entity which does not have a primary key attribute.

Representation:

A double rectangle is used for representing a weak entity set

The double diamond symbol is used for representing the relationship between a strong entity and weak entity which is known as identifying relationship

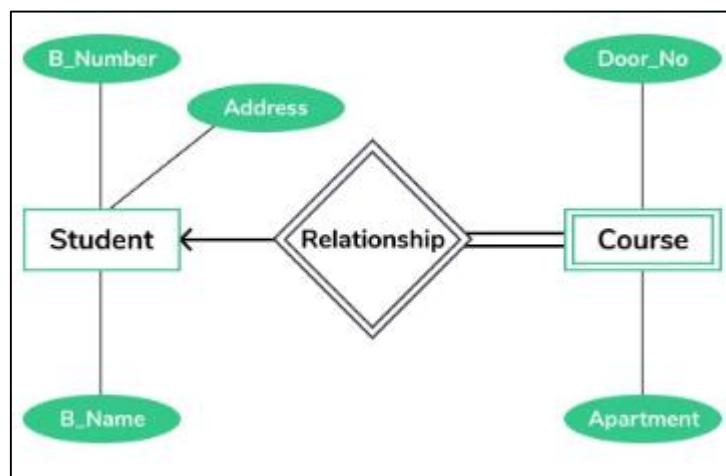
Double lines are used for presenting the connection with a weak entity set with relationship.

## Example for weak entity

In the ER diagram, we have two entities building and apartment.

Building is a strong entity because it has a primary key attribute called building number which is capable of uniquely identifying all the flats present in the apartment.

Unlike building, apartment is weak entity because it does not have any primary key and door number here acts only as a discriminator because door number cannot be used as a primary key, there might be multiple flats in the building with the same door number or on different floors.



## What is Strong entity?

A strong entity set is an entity that contains sufficient attributes to uniquely identify all its entities. Simply strong entity is nothing but an entity set having a primary key attribute or a table which consists of a primary key column.

The primary key of the strong entity is represented by underlining it.

Representation

The strong entity is represented by a single rectangle.

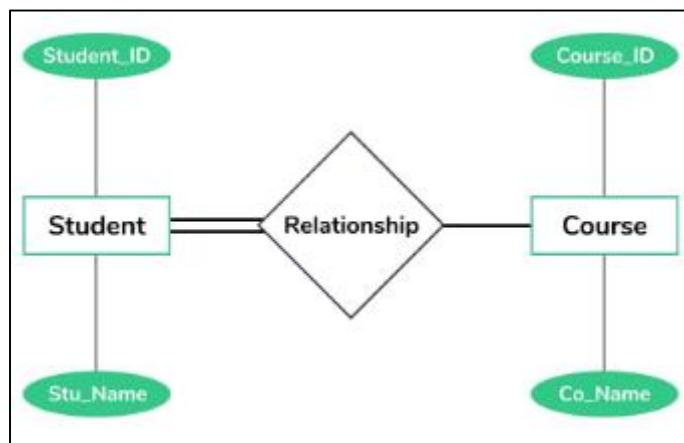
Relationship between two strong entities is represented by a single diamond.

## Examples for the strong entity

Consider the ER diagram which consists of two entities student and course

Student entity is a strong entity because it consists of a primary key called student id which is enough for accessing each record uniquely

The same way, course entity contains of course ID attribute which is capable of uniquely accessing each row it is each course details



## Difference between Strong Entity and Week Entity.

Strong Entity	Week Entity
<ul style="list-style-type: none"><li>1) Strong entity always has a primary key.</li><li>2) It is not dependent on any other entity.</li><li>3) Represented by a single rectangle.</li><li>4) Relationship between two strong entities is represented by a single diamond.</li><li>5) A strong entity has <b>may or may not have total participation</b></li></ul>	<ul style="list-style-type: none"><li>1) Will not have a primary key but it has partial discriminator key</li><li>2) Which entity is dependent on the strong entity</li><li>3) Represented by double rectangle</li><li>4) Relationship between a strong entity and the weak entity is represented by double Diamond.</li><li>5) It has always total participation.</li></ul>

## What is identifying relationship.?

An identifying relationship is a relationship between two entities in which an instance of a child entity is identified through its association with a parent entity, which means the child entity is dependent on the parent entity for its identity and cannot exist without it.

## What is Total Participation.?

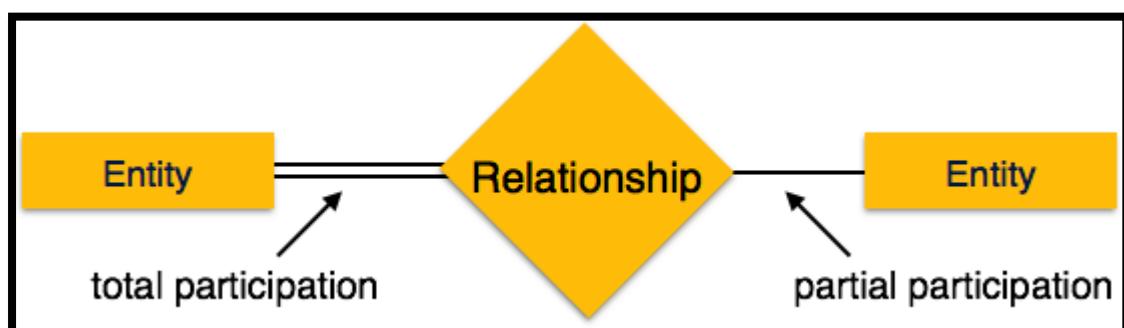
Total Participation is when each entity in the entity set occurs in at least one relationship in that relationship set."

For example, Consider three entities Users, Buyers and Sellers, if a company policy states that every User must be a Seller or Buyer, then a User entity cannot be existing without being a Buyer or Seller. So, this participation is called total participation meaning that every entity in the total set of User entities must be related to a Buyer or Seller entity.

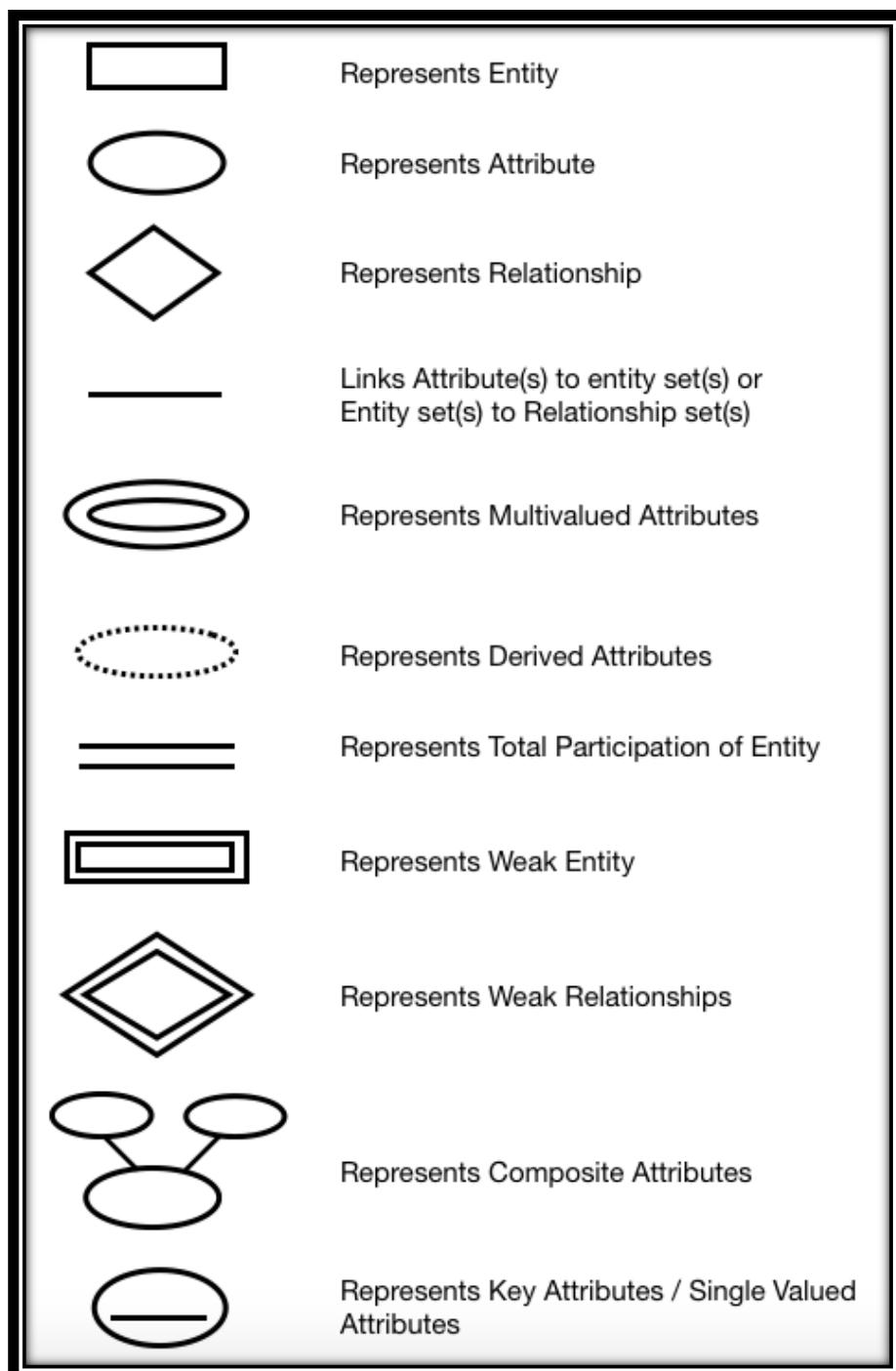
## What is Partial Participation.?

Partial Participation is when each entity in the entity set may not occur in at least one relationship in that relationship set"

For example, consider two entities Person and Student related with the help of relationship Status. Here, not all Persons are expected to be a Student (also there may be employees, retiree etc.), so the participation is called partial participation.



# Different Symbols of ER Model



### ◀1:1 RELATIONSHIP



### ◀1:M RELATIONSHIP



### ◀M:1 RELATIONSHIP



### ◀M:M RELATIONSHIP



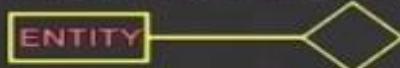
### ◀1:1 OPTIONAL RELATIONSHIP



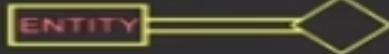
### ◀1:1 MADATORY RELATIONSHIP



### ◀PARTIAL PARTICIPATION



### ◀TOTAL PARTICIPATION



## Case Study:

### University ER Diagram

In university, a student enrols in courses. A student can enrol to one or more courses. Each course is taught by a single professor. To maintain instruction quality, a professor can deliver only one course.

#### Steps to create ER Diagram.

Step 1: Identify the number of entities.

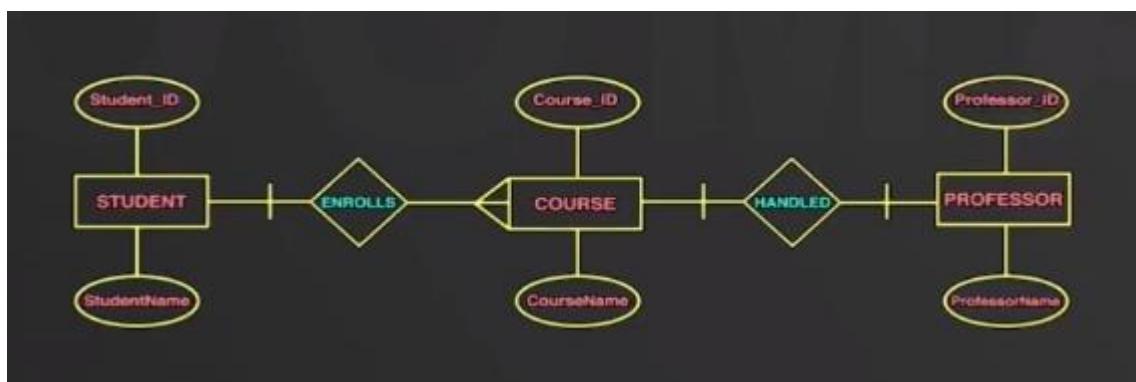
Step 2: Establish the relationship between the entities.

Step 3: Identify the cardinal ratio between the entities.

Step 4: Identify the number of attributes associated to each entity.

Step 5: Identity the type of each attribute.

#### ER Diagram.



**Thank You** ☺☺

# **Structured Query Language**

## **SQL - Day 4**

### **Agenda**

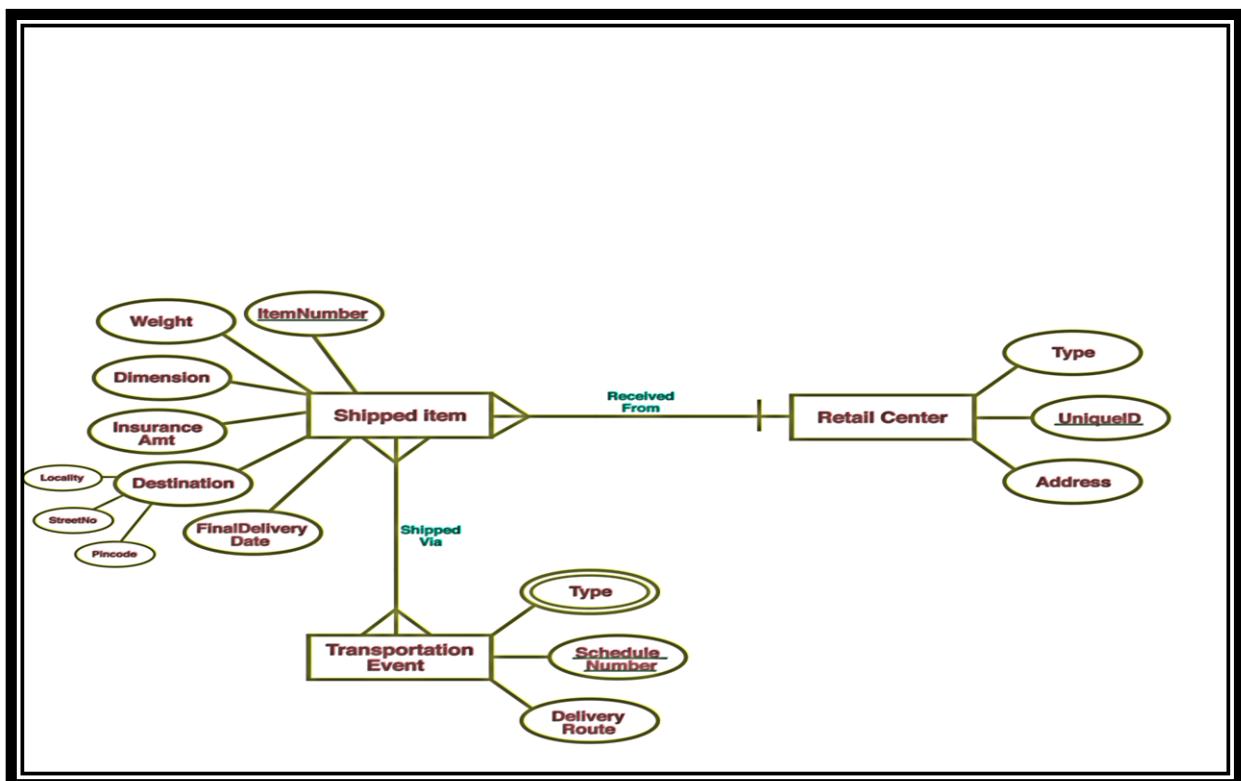
- ER Diagram Case Study



## Case Study 1:

UPS prides itself on having up-to-date information on the processing and current location of each shipped item. To do this, UPS relies on a company-wide information system. Shipped items are the heart of the UPS product tracking information system. Shipped items can be characterized by item number (unique), weight, dimensions, insurance amount, destination, and final delivery date. Shipped items are received into the UPS system at a single retail centre. Retail centres are characterized by their type, uniqueID, and address. Shipped items make their way to their destination via one or more standard UPS transportation events (i.e., flights, truck deliveries). These transportation events are characterized by a unique scheduleNumber, a type (e.g, flight, truck), and a deliveryRoute. Create an Entity Relationship diagram that captures this information about the UPS system. Be certain to indicate identifiers and cardinality constraints.

ER Diagram for the above database.



## Case Study 2:

In a Hospital database.

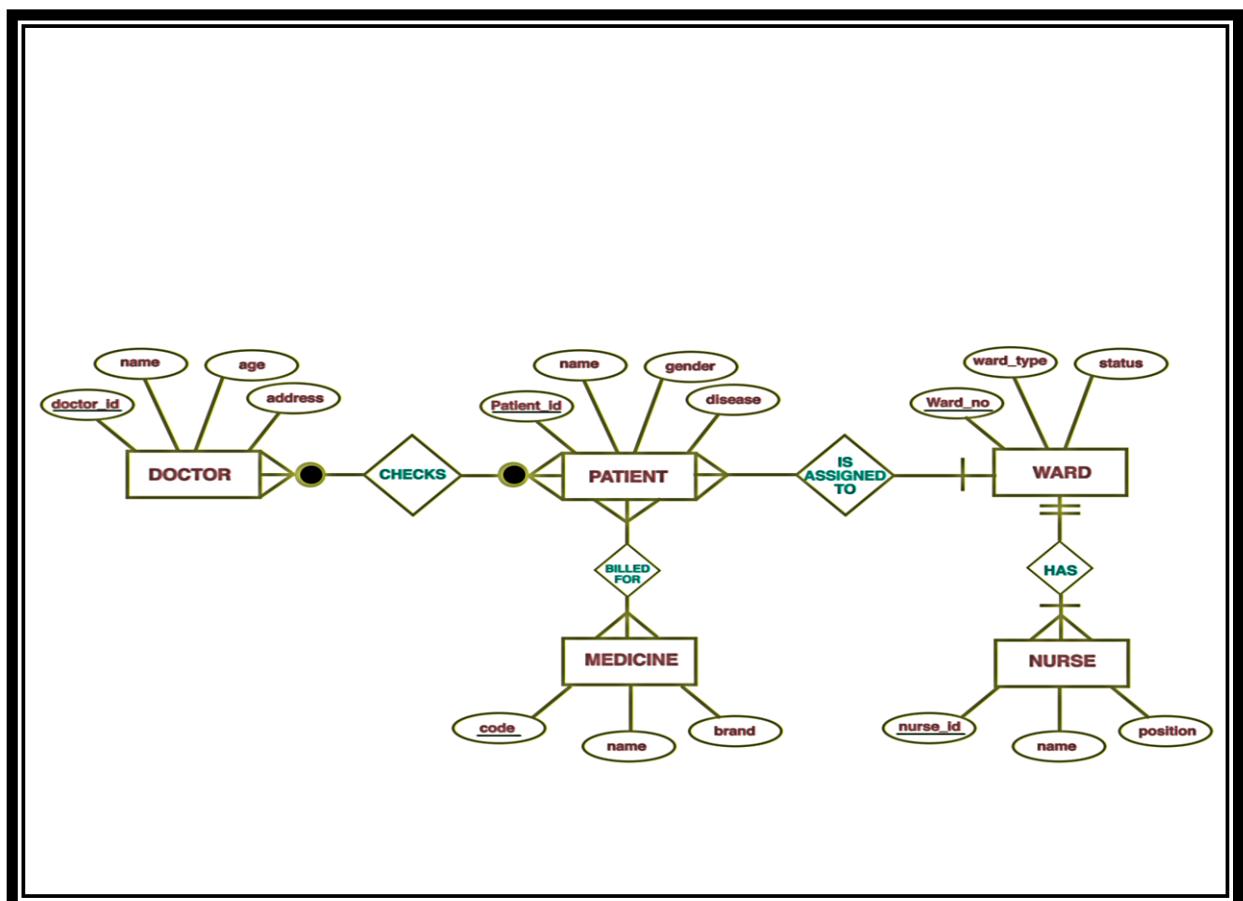
A patient is assigned a particular ward. A single ward is assigned to many patients during a given period.

A ward has or more nurses. On the other hand each nurse must be assigned a single ward only and each ward must have a nurse.

A doctor may check zero or many patients during a given period. A patients may be checked by Zero or many doctors.

A patients is billed for one or many medicines. On the other hand, a particular medicine may appear in zero or more patients.

**ER Diagram for the above database.**



**Thank You** ☺☺

# Structured Query Language

## SQL - Day 5

### Agenda

- Difference between SQL and MySQL.
- What is Query
- Different Commands used in SQL queries
- Different Languages used in SQL
- Usage of SQL languages and SQL commands.
- Establishing connection with MySql Server
- Finding the database present within MYSQL server.
- How to use a particular database present within MYSQL server.
- How to know which database is currently in Use.
- How to know which are the tables present inside particular database.
- How to fetch all the data present inside particular table.



### Case Study

- Creation of University database and its manipulation.

# Difference between SQL and MySQL.

SQL is a Structured Query Language. It is useful to manage relational databases.

MySQL is an RDBMS to store, retrieve, modify and administrate a database using SQL.

## What is Query?

A query is a question, often expressed in a formal way. A database query can be either a select query or an action query. A select query is a data retrieval query, while an action query asks for additional operations on the data, such as insertion, updating or deletion.

## Let Us understand what is query.

Examples for queries.

From the given STUDENT table write a **Query** to get the all students details whose **grade is greater than 70.**



STUDENT TABLE			
ID	NAME	AGE	GRADE
1	JOHN	21	72
2	MIKE	20	68
3	KATE	22	86
4	ANDY	21	94

**Query:**

**SELECT \* FROM STUDENT WHERE THE GRADE > 70;**

**OUTPUT:**

ID	NAME	AGE	GRADE
1	JOHN	21	72
2	KATE	22	86
3	ANDY	21	94

**NOTE:** Query which we have written had fetched all the student details whose grade is greater than 70% from STUDENT table and given a separate table as shown (output) above.

Unlike SELECT command there are many different types of command available in SQL.

## Different Commands used in SQL queries.

Here is the list of different SQL commands:

SQL  
Commands



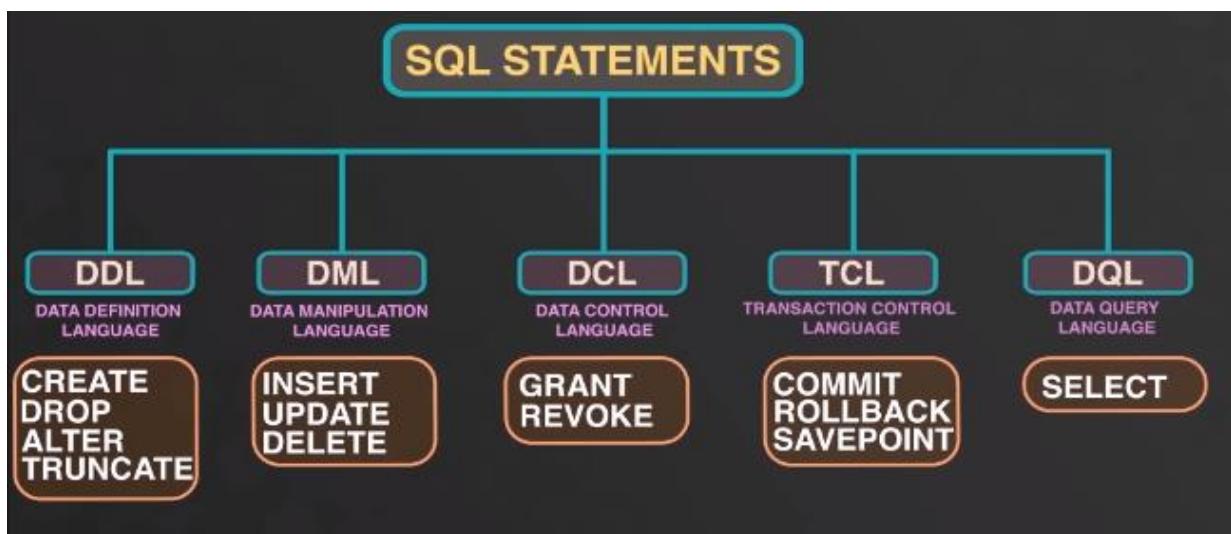
These commands are classified into different categories of sql languages.

## Let us know the Different Languages and its commands used in SQL.

There are 5 different languages in sql, they are:

1. DDL (DATA Definition Language)
2. DML (DATA Manipulation Language)
3. DCL (DATA control Language)
4. TCL (Transaction Control Language)
5. DQL (DATA Query Language)

**Note:** SQL statements are categorised into different languages and all the sql commands are categorised into these different languages as shown below.



## What is the usage of these SQL languages and SQL Commands?



**Usage of SQL Languages and its Commands.**

## **DDL (DATA Definition Language)**



**DDL(Data Definition Language) :** DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database and is used to create and modify the structure of database objects in the database.

**DDL commands:**

**CREATE** - is used to create the database or its objects (like table, index, function, views, store procedure and triggers).

**DROP** - is used to delete objects from the database.

**ALTER**-is used to alter the structure of the database.

**TRUNCATE**-is used to remove all records from a table, including all spaces allocated for the records are removed.

## **DQL (DATA QUERY LANGUAGE):**



DQL statements are used for performing queries on the data within database table

**DQL Commands:**

**SELECT** - is used to retrieve data from the database.

## **DML (Data Manipulation Language):**



**DML (Data Manipulation Language):** The SQL commands that deals with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements.

### **DML Commands:**

**INSERT** - is used to insert data into a table.

**UPDATE** - is used to update existing data within a table.

**DELETE** - is used to delete records from a database table.

## **DCL (Data Control Language):**



**DCL (Data Control Language):** DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions and other controls of the database system.

### **DCL commands:**

**GRANT** - gives user's access privileges to the database.

**REVOKE** - withdraw user's access privileges given by using the GRANT command.

## TCL (transaction Control Language):



**TCL (transaction Control Language):** TCL commands deal with the transaction within the database.

### TCL commands:

**COMMIT** - commits a Transaction.

**ROLLBACK** - rollbacks a transaction in case of any error occurs.

**SAVEPOINT** -sets a save point within a transaction.

## How to establish a Connection with MYSQL server.



Before establishing the connection first install the MYSQL in your system.

**MySQL command:**  
Invokes the MYSQL sever

**-U:** Specifies USER NAME

**-P:** Specifies Password.

```
COMMAND PROMPT
C:\Users\Studio>mysql -u rooman -p
Enter password: ****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 14
Server version: 8.0.22 MySQL Community Server - GPL

Copyright (c) 2000, 2020, Oracle and/or its affiliates. All ri

Oracle is a registered trademark of Oracle Corporation and/or
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current

mysql>
```

# How to find the database present within MYSQL server.



Show database command:

Used to retrieve all the database present inside MYSQL server.

These are the 6 default database present inside MYSQL server

```
COMMAND PROMPT
owners.

Type 'help;' or '\h' for help. Type '\c' to

mysql> show databases;
+-----+
| Database
+-----+
| information_schema
| mysql
| performance_schema
| sakila
| sys
| world
+-----+
6 rows in set (0.00 sec)
```

# How to use a particular database present within MYSQL server.



To make use of SAKILA database present inside MYSQL server, make use of **USE command** followed with database name.

**USE command:** use command is used to make use of any database present inside the MYSQL server

```
COMMAND PROMPT
mysql
performance_schema
sakila
sys
world
+-----+
6 rows in set (0.00 sec)

mysql> use sakila;
Database changed
mysql>
```

## How to know which database is currently in Use.

Select database () Command is used to know which is the database id currently in use.



```
mysql> use sakila;
Database changed
mysql> select database();
+-----+
| database() |
+-----+
| sakila    |
+-----+
1 row in set (0.00 sec)
```

## How to get to know which are the tables present inside particular database?

Select tables () Command is used to know which are the tables present inside the database.



```
mysql> show tables;
+-----+
| Tables_in_sakila |
+-----+
| actor
| actor_info
| address
| category
| city
| country
| customer
| customer_list
| film
| film_actor
| film_category
| film_list
| film_text
| inventory
+-----+
```

# How to fetch all the data present inside particular table?



Select \* FROM TABLE\_NAME

Note: Using the above SQL query we can fetch all the data present inside the database.

```
mysql> SELECT * FROM STAFF_LIST;
+----+-----+-----+-----+
| ID | name           | address          | zip code |
+----+-----+-----+-----+
| 1  | Mike Hillyer   | 23 Workhaven Lane |          |
| 2  | Jon Stephens    | 1411 Lillydale Drive |          |
+----+-----+-----+-----+
```

Hey!! I understood about establishing a database connection, How to find the database present within MYSQL server, How to know which database is currently in use, How to get to know which are the tables present inside particular database, How to fetch all the data present inside particular table.

But!! How to create new database and add data into it??



# Let's understand how to create database and how to add data into it.

## Creating University database.

**CREATE DATABASE** is the command used to create new database.

```
mysql> CREATE DATABASE UNIVERSITY;
Query OK, 1 row affected (0.01 sec)
```

Syntax:

**CREATE database\_name**

## Making University Database as current database.

**USE University** is the command used to make university database as the current database.

```
mysql> USE UNIVERSITY;
Database changed
```

Syntax: **USE database\_name**

## Finding the data present inside University Database.

**SHOW TABLES** command is used to find the data present inside current database.

```
mysql> SHOW TABLES;
Empty set (0.00 sec)
```

**Note:** We have created university database but no data has been added to it, so it is displaying as EMPTY SET. Whenever data is not added into the database then it displays **EMPTY SET**.

## **Deleting the university database.**

Drop DATABASE UNIVERSITY is the command used to delete a database from the server.

```
mysql> DROP DATABASE UNIVERSITY;  
Query OK, 0 rows affected (0.01 sec)
```

Syntax:

Drop DATABASE database\_name

Note: Once the database is deleted/dropped from the server, then it cannot be used. If we try to use the database which is not present inside the database server then it will generate Error message.

```
mysql> USE UNIVERSITY;  
ERROR 1049 (42000): Unknown database 'university'
```

## **How to create table in database and add data inside table.?**



To be continued...

# Structured Query Language

## SQL - Day 6

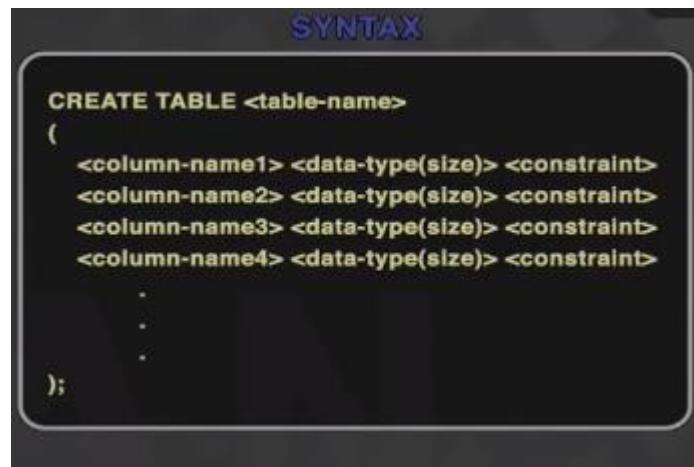
### Agenda

- Different DATA TYPES in SQL
- Different Constraints in SQL



# How to create table in database and add data inside table.?

Here is the **SYNTAX** for creating table .



Before creating table Lets first understand what are Data types and Constraints.

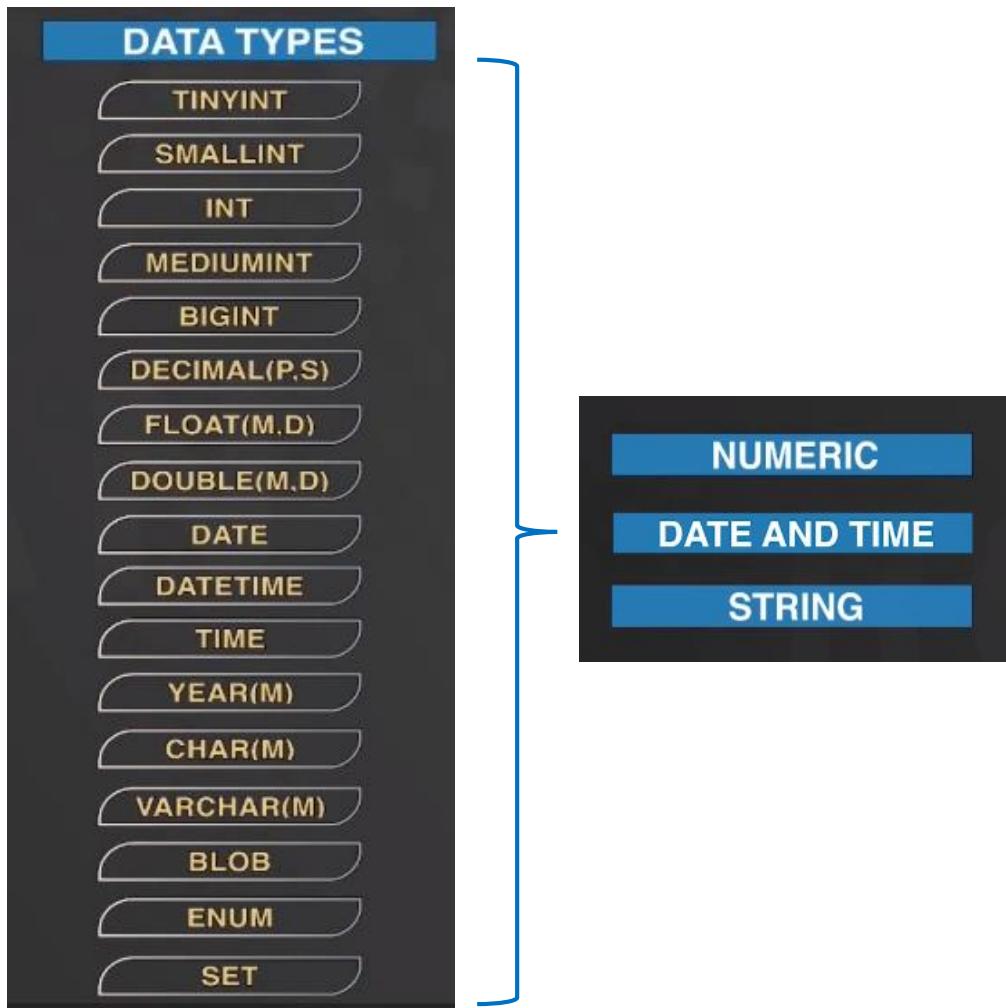
## DATA TYPES in SQL:

Each column in a database table is required to have a name and a data type.

An SQL developer must decide what type of data that will be stored inside each column when creating a table. The data type is a guideline for SQL to understand what type of data is expected inside of each column, and it also identifies how SQL will interact with the stored data.

**Note:** Data types might have different names in different database. And even if the name is the same, the size and other details may be different! Always check the documentation!

MySQL provides different data types and each data types is categorised into three main data types such as string, numeric, and date and time as shown below:



**Let's know see which data type fall under which category and what is the purpose of using it to store data in SQL database.**

## **NUMERIC DATA TYPE.**

Numeric data type is used to store numeric data/numbers.

**Note:** There are two types of number **whole number** and **decimal number**. There are different data types available for different types of number.

## Whole Number.

Whole numbers are numbers without any decimal point.

Example: Age, mobile number, pin code, order number etc.

Data types which is used to store the whole numbers is listed below.

NUMERIC			
DATATYPE	RANGE(signed)	RANGE(unsigned)	Width
TINYINT	-128 to 127	0 to 255	4
SMALLINT	-32768 to 32767	0 to 65535	5
MEDIUMINT	-8388608 to 8388607	0 to 16777215	9
INT	-2147483648 to 2147483647	0 to 4294967295	11
BIGINT	-9223372036854775808 to 9223372036854775807	0 to 18446744073709551615	20

### TINYINT:

TINYINT stores very small integer. Signed range is from -128 to 127. Unsigned range is from 0 to 255. The size parameter specifies the maximum display width 4.

### SMALLINT:

SMALLINT stores small integer. Signed range is from -32768 to 32767. Unsigned range is from 0 to 65535. The size parameter specifies the maximum display width 5.

### MEDIUMINT:

MEDIUMINT stores medium integer. Signed range is from -8388608 to 8388607. Unsigned range is from 0 to 16777215. The size parameter specifies the maximum display width 9.

## **INT:**

INT stores medium integer. Signed range is from -2147483648 to 2147483647. Unsigned range is from 0 to 4294967295. The size parameter specifies the maximum display width 11.

## **BIGINT:**

BIGINT stores large integer. Signed range is from -9223372036854775808 to 9223372036854775807. Unsigned range is from 0 to 18446744073709551615. The size parameter specifies the maximum display width 20.

## **Decimal/Real Number.**

Decimal numbers are number with precision or decimal values.

**Example:** Distance, price, height, weight, salary.

Data types which is used to store the decimal numbers is listed below.

### **DECIMAL(P,S) P-> precision S->scale**

*ex: price DECIMAL(5,2). COLUMN can store any value with 5 digits and 2 decimal*

**RANGE: -999.99 TO 999.99**

### **FLOAT(M,D) M->length D->decimal. Default values of(M,D)->(10,2)**

**Decimal precision can go to 24 places for a FLOAT**

*ex: price FLOAT (6,3). COLUMN can store any value with 6 digits and 3 decimal*

**RANGE: -999.999 TO 999.999**

**If you insert 999.0009 into a FLOAT(6,3) column, the approximate result is 999.001.**

### **DOUBLE(M,D) M->length D->decimal. Default values of(M,D)->(16,4)**

**Decimal precision can go to 53 places for a DOUBLE**

### **Decimal (P,S):**

Decimal (P,S) data type stores decimal numbers, where P is precision and S is scale. Decimal (P,S) can store any values with 5 digits and 2 decimals.

### **Float (M,D):**

Float (M,D) stores decimal numbers, where M is length and D is decimal. By default M can store 10 digits and D can store 2 decimal but it can go up to 24 places.

### **Double (M,D):**

Double (M,D) stores decimal numbers, M is length and D is decimal. By default M can store 16 digits and D can store 4 decimal but it can go up to 53 places.

## **DATE & TIME DATA TYPE.**

Date & Time data type is used to store date and time data in the database. For example: Date of Birth, Hiring Date and Salary date etc.

There are different data types available to store Date & Time data. They are listed below:

DATE AND TIME		
	FORMAT	RANGE
DATE	YYYY-MM-DD	1000-01-01 TO 9999-12-31
TIME	HH:MM:SS	-838:59:59 TO 838:59:59
DATETIME	YYYY-MM-DD HH:MM:SS	1000-01-01 00:00:00 TO 9999-12-31 23:59:59
TIMESTAMP	YYYY-MM-DD HH:MM:SS	1970-01-01 00:00:01 UTC TO 2038-01-19 03:14:07 UTC
YEAR	YYYY	1901 TO 2155

## Date data type

The date data type is used to store date. Date data type uses YYYY-MM-DD format. The supported range is from '1000-01-01' to '9999-12-31'.

## Time data type

The time data type is used to store TIME. Time data type uses hh:mm:ss format. The supported range is from '-838:59:59' to '838:59:59'.

## DATETIME data type

The datetime data type is used to store date and time combination. Time data type uses YYYY-MM-DD hh:mm:ss format. The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.

## TIMESTAMP data type

TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). TIMESTAMP data type uses YYYY-MM-DD hh:mm:ss. format. The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC.

## YEAR data type

Year data type stores year in four-digit format. Values allowed in four-digit format: 1901 to 2155, and 0000.

# String Data Type.

STRING		
CHAR(M)	FIXED LENGTH STRINGS	RANGE 255 CHARACTERS
VARCHAR(M)	VARIABLE LENGTH STRINGS	65,535 CHARACTERS
BLOB (BINARY LARGE OBJECT)	LARGE BINARY DATA	65,535 BYTES
ENUM	LIST OF POSSIBLE VALUES	65,535 DISTINCT VALUES
SET	LIST OF POSSIBLE VALUES	65,535 VALUES

## Char data type

Char(M) data type stores fixed length strings. M is the number of characters that can be stored. It can store up to 255 characters.

## Varchar data type

Varchar data type stores variable length strings. M is the number of characters that can be stored. It can store up to 65535 characters.

## BLOB data type

A BLOB is a binary large object that can hold a variable amount of data/binary data such as audio, video and image.

## **ENUM data type:**

ENUM stands for Enumeration. It allows us to limit the value chosen from a list of permitted values in the column specification at the time of table creation. It can store up to **65535 DISTINCT VALUES.**

**SYNTAX:** COLUMN\_NAME ENUM ('value-1','value-2',.....,'value-n')

For Example: if the user gives the list of values as

COLUMN\_NAME ENUM ('ONE','TWO','THREE')

Then user can enter only one of these values ('ONE','TWO','THREE') inside that column.

## **SET data type:**

A SET is a string object that can have zero or more values, each of which must be chosen from a list of permitted values specified when the table is created. SET column values that consist of multiple set members are specified with members separated by commas (.). A consequence of this is that SET member values should not themselves contain commas.

**SYNTAX:** COLUMN\_NAME SET ('value-1','value-2',.....,'value-n')

For Example: if the user gives the list of values as

COLUMN\_NAME SET ('ONE','TWO','THREE')

Then user can choose set of these values ('ONE','TWO','THREE') inside that column separated with comma like : 'one','two'.

**NOTE:** There are other string data type such as **CLOB (Character Large Object)** which is used to store large text like description, text books etc. Anyways using Char and Varchar data type we can store the large text so throughout the course we are using these **Char and Varchar data type instead of CLOB.**

# Constraints in SQL.

SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data entered , then the data is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

Below are different types of constraints available in SQL,



## NOT NULL CONSTRAINTS:

By default, a column can hold NULL values, but the NOT NULL constraint enforces a column to NOT accept NULL values.

This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

## **UNIQUE CONSTRAINTS:**

The **UNIQUE** constraint ensures that all values in a column are different.

Both the **UNIQUE** and **PRIMARY KEY** constraints provide a guarantee for uniqueness for a column or set of columns.

A **PRIMARY KEY** constraint automatically has a **UNIQUE** constraint.

However, you can have many **UNIQUE** constraints per table, but only one **PRIMARY KEY** constraint per table.

## **PRIMARY KEY CONSTRAINTS:**

The **PRIMARY KEY** constraint uniquely identifies each record in a table.

Primary keys must contain **UNIQUE** values, and cannot contain **NUL** values.

A table can have **only ONE primary key**; and in the table, this primary key can consist of single or multiple columns (fields).

## **FOREIGN KEY CONSTRAINTS:**

The **FOREIGN KEY** constraint is used to prevent actions that would destroy links between tables.

A **FOREIGN KEY** is a field (or collection of fields) in one table, that refers to the **PRIMARY KEY** in another table.

The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

### **CHECK CONSTARINTS:**

The CHECK constraint is used to limit the value range that can be placed in a column.

If you define a CHECK constraint on a column it will allow only certain values for this column.

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

### **DEFAULT CONSTARINTS:**

The DEFAULT constraint is used to set a default value for a column.

The default value will be added to all new records, if no other value is specified.

### **AUTO INCREMENT CONSTARINTS:**

Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.

Often this is the primary key field that we would like to be created automatically every time a new record is inserted.

# Structured Query Language

## SQL - Day 7

### Agenda

- Creating Database, table and manipulating data.
- Different ways of creating PRIMARY KEY.
- ALTER, ADD, DROP, MODIFY Command.



## How to create a Database, table and manipulate data.

Let's create 'ROOMAN' database and create 'STUDENT' table and try to insert values/data inside it as shown below.

STUDENT TABLE			
S_ID	NAME	AGE	GRADE
1	JOHN	21	72
2	MIKE	20	68
3	KATE	22	86
4	ANDY	21	94

First Lets Create 'ROOMAN' database.

Using this command,  
we have created  
'ROOMAN' database



```
mysql> CREATE DATABASE rooman;  
Query OK, 1 row affected (0.01 sec)
```

**Now we have created ‘ROOMAN’ database, inside database how can we create ‘STUDENT’ table.**

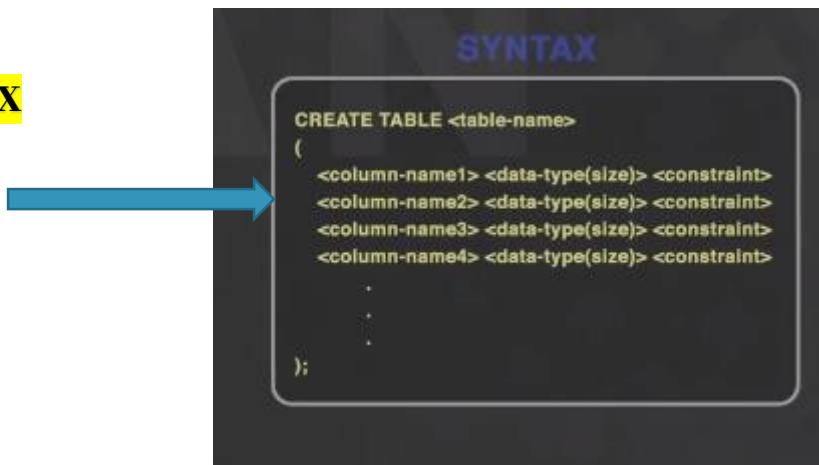
**Here is the SYNTAX**

**to create a table,**

**Lets create**

**‘STUDENT’ table**

**using this syntax.**



**Before creating database  
we have to make use of  
‘ROOMAN’ database.**

```
mysql> USE rooman;  
Database changed
```

**Creating ‘STUDENT’  
database.**

```
mysql> CREATE TABLE student(
-> s_id tinyint PRIMARY KEY,
-> name varchar(10) NOT NULL,
-> age tinyint CHECK(age>18),
-> grade decimal(2,2) DEFAULT 0.0
-> );
Query OK, 0 rows affected (0.05 sec)
```

## HOW to get the description of ‘STUDENT’ TABLE.

```
mysql> DESCRIBE student;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key  | Default | Extra |
+-----+-----+-----+-----+-----+
| s_id  | tinyint | NO   | PRI  | NULL    |        |
| name  | varchar(10) | NO  |        | NULL    |        |
| age   | tinyint | YES  |        | NULL    |        |
| grade | decimal(2,2) | YES |        | 0.00    |        |
+-----+-----+-----+-----+-----+
4 rows in set (0.01 sec)
```

Note:

In this table description,

- In **s\_id**/ Student ID field, type is mentioned as **tinyint**, Because ID is a numeric value, and **Null** column is mentioned as **NO** because **s\_id** should not be Null/ **s\_id** column should not be empty. **s\_id** is made as **PRIMARY KEY** hence **s\_id** always remains unique. Default values as **NULL**.
- In **name** field, type is mentioned as **varchar(10)**, because name is a string value, and **Null** column is mentioned as **NO** because name should not be Null/ **name** column should not be empty. Default values as **NULL**
- In **age** field, type is mentioned as **tinyint**, Because age is a numeric value, and **Null** column is mentioned as **YES** which means in **age** field **Null** column can be empty. Default values as **NULL**.
- In **grade** field, type is mentioned as **decimal(2,2)**, which means grade value can have 2 digits before the decimal and 2 digits after the decimal, and **Null** column is mentioned as **YES** which means in **age** field **Null** column can be empty. Default values as **0.0**.

While creating 'STUDENT' table or any table PRIMARY KEY can be created in two different ways as shown below:

### Method 1:

```
mysql> CREATE TABLE student(
    -> s_id tinyint PRIMARY KEY,
    -> name varchar(10) NOT NULL,
    -> age tinyint CHECK(age>18),
    -> grade decimal(2,2) DEFAULT 0.0
    -> );
Query OK, 0 rows affected (0.05 sec)
```

### Method 2:

```
mysql> CREATE TABLE student(
    -> s_id tinyint,
    -> name varchar(10) NOT NULL,
    -> age tinyint CHECK(age>18),
    -> grade decimal(2,2) DEFAULT 0.0,
    -> PRIMARY KEY(s_id)
    -> );
```

WE have seen how to create TABLE, now let's understand how to alter the table / modify the existing table.

Here is the SYNTAX to  
ALTER any TABLE

```
ALTER TABLE <table-name>
ADD <column-name><data-type><constraint>
DROP <column-name>
MODIFY <column-name><data-type><constraint>
```

Note:

- To add a column to the existing table

SYNTAX:

```
ALTER TABLE <table-name>ADD<column-name><data-type><constraint>
```

- To delete the column in the existing table

SYNTAX:

```
ALTER TABLE <table-name>DROP<column-name>
```

- To modify the column in the existing table

SYNTAX:

```
ALTER TABLE <table-name>MODIFY<column-name>< data-type><constraint>
```

**Now, Lets try to add new Column to the Student table as shown below. For that lets start writtig a query to ALTER ‘STUDENT’ table.**

STUDENT TABLE				
S_ID	NAME	AGE	GRADE	CONTACT_NUM
1	JOHN	21	72	
2	MIKE	20	68	
3	KATE	22	86	
4	ANDY	21	94	



## **WRITE A QUERY TO ADD CONTACT\_NUM COLUMN TO STUDENT TABLE.**

**Query:**

```
mysql> ALTER TABLE student ADD contact_num INT NOT NULL;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

**Description after adding CONTACT\_NUM column:**

```
mysql> DESCRIBE student;
+-----+-----+-----+-----+-----+
| Field | Type   | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| s_id  | tinyint | NO   | PRI | NULL    |       |
| name  | varchar(10) | NO  |     | NULL    |       |
| age   | tinyint | YES  |     | NULL    |       |
| grade | decimal(2,2) | YES |     | 0.00    |       |
| contact_num | int | NO  |     | NULL    |
+-----+-----+-----+-----+-----+
```

**Output:**

STUDENT TABLE				
S_ID	NAME	AGE	GRADE	CONTACT_NUM
1	JOHN	21	72	
2	MIKE	20	68	
3	KATE	22	86	
4	ANDY	21	94	

**WRITE A QUERY TO DELETE OR DROP CONTACT\_NUM COULMN FROM STUDENT TABLE.**

**Query:**

```
mysql> ALTER TABLE student DROP contact_num;
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

**Description after DELETING CONTACT\_NUM column:**

```
mysql> DESCRIBE student;
+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| s_id  | tinyint   | NO   | PRI | NULL    |       |
| name  | varchar(10) | NO   |     | NULL    |       |
| age   | tinyint   | YES  |     | NULL    |       |
| grade | decimal(2,2) | YES  |     | 0.00    |       |
+-----+-----+-----+-----+-----+
```

**Output:**

STUDENT TABLE			
S_ID	NAME	AGE	GRADE
1	JOHN	21	72
2	MIKE	20	68
3	KATE	22	86
4	ANDY	21	94

**WRITE A QUERY TO MODIFY TYPE OF AGE FIELD IN THE 'STUDENT' TABLE TO SMALLINT AND THE CONSTRAINTS AS NOT NULL.**

**Query:**

```
mysql> ALTER TABLE student MODIFY age smallint NOT NULL;
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

**Description after modifying the table :**

```
mysql> DESCRIBE student;
+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| s_id  | tinyint   | NO   | PRI | NULL    |       |
| name  | varchar(10) | NO   |     | NULL    |       |
| age   | smallint  | NO   |     | NULL    |       |
| grade | decimal(2,2)| YES  |     | 0.00    |       |
+-----+-----+-----+-----+-----+
```



# Structured Query Language

## SQL - Day 8

### Agenda

- INSERT, MODIFY.
- AUTO\_INCREMENT
- Types of Primary Key
- Case studies for CONSTRAINTS Violation.



We have understood how to create table and alter the table.

Now, Let's understand how to INSERT values into the TABLE.

### Syntax : to Insert

values into the  **INSERT INTO table-name VALUES (value1,value2,value3,value4);**  
table.

**WRITE A QUERY TO INSERT THE BELOW DATA INTO STUDENT TABLE.**

```
s_id : 1,  
name : JHON  
age : 21  
grade : 72.
```

### Query to insert data into table.

```
mysql> INSERT INTO student VALUES (1,'JOHN',21,72);  
Query OK, 1 row affected (0.00 sec)
```

**WRITE A QUERY TO GET PERTICULAR COLUMN DATA PRESENT IN STUDENT TABLE.**

**Syntax:**

```
SELECT <column1><column2><column3><column4> FROM <table-name>;
```

**Query to get s\_id, name, age, and grade data from student.**

```
mysql> SELECT s_id,name,age,grade FROM student;
+-----+-----+-----+
| s_id | name | age  | grade |
+-----+-----+-----+
|    1 | JOHN |   21 | 72.00 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

**WRITE A QUERY TO GET ALL COLUMN DATA PRESENT IN STUDENT TABLE.**

**Syntax:**

```
SELECT * FROM <table-name>;
```

**Query to get all column data from table:**

```
mysql> SELECT * FROM student;
+-----+-----+-----+
| s_id | name | age  | grade |
+-----+-----+-----+
|    1 | JOHN |   21 | 72.00 |
+-----+-----+-----+
1 row in set (0.00 sec)
```

**Note:** There is another way to insert data inside the table as shown below:

Instead of directly specifying the values, we can first specify the column name and then specify the values.

```
mysql> INSERT INTO student (s_id,name,age,grade) VALUES (2,'MIKE',20,68);
Query OK, 1 row affected (0.00 sec)
```

But column name should be in the same order in which the columns are created in table or you will get an 'ERROR' as shown below:

```
mysql> INSERT INTO student VALUES ('KATE',3,22,86);
ERROR 1366 (HY000): Incorrect integer value: 'KATE' for column 's_id'
mysql> INSERT INTO student (name,s_id,age,grade) VALUES ('KATE',3,22,86);
```

Now, Let's try to enter new row into the table.

```
mysql> INSERT INTO student VALUES (4,'ANDY',21,94);
Query OK, 1 row affected (0.00 sec)
```

Successfully we have entered new row, but if we try to enter the same value into the table we will get **ERROR**.

```
mysql> INSERT INTO student VALUES (4,'ANDY',21,94);
ERROR 1062 (23000): Duplicate entry '4' for key 'student.PRIMARY'
```

**Note:** In STUDENT table, CONSTRAINT of **s\_id column** is PRIMARY KEY. So whenever the constraints of any field in a table is mentioned as PRIMARY KEY, then duplicate entries will not be allowed and that column can't be empty. **PRIMARY KEY** is a combination of **UNIQUE** and **NOT NULL**.

Now let's enter the proper value into the column, and see all the rows and column present in STUDENT table.

```
mysql> INSERT INTO student VALUES (6,'ANDY',21,94);
Query OK, 1 row affected (0.01 sec)
```

```
mysql> SELECT * FROM student;
+-----+-----+-----+-----+
| s_id | name | age  | grade |
+-----+-----+-----+-----+
|    1 | JOHN |   21 | 72.00 |
|    2 | MIKE |   20 | 68.00 |
|    3 | KATE |   22 | 86.00 |
|    4 | ANDY |   21 | 94.00 |
|    6 | ANDY |   21 | 94.00 |
+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

Let's now violet few **CONSTRAINTS** and check how exactly the Query works.



## NOT NULL CONSTRAINT VIOLATION:

Inside the STUDENT table we have given CONSTRAINT of NAME filed as NOT NULL, Now Let's try to create new row without entering anything in NAME field.

```
mysql> INSERT INTO student (s_id,age,grade) VALUES (5,21,88.2);
ERROR 1364 (HY000): Field 'name' doesn't have a default value
```

Note: We got ERROR message because, when a column CONSTRAINT is mentioned as NOT NULL in the table, then that column should be specified with some values, it cannot be empty.

## PRIMARY KEY CONSTRAINT VIOLATION:

Now, let's try to create new row without giving s\_id .

```
mysql> INSERT INTO student (name,age,grade) VALUES ('TOM',23,89.2);
ERROR 1364 (HY000): Field 's_id' doesn't have a default value
```

Note: We got ERROR message because, when a column CONSTRAINT is mentioned as PRIMARY KEY, if the column CONSTRAINT is mentioned as PRIMARY KEY then the COLUMN cannot be left empty or duplicate values will not be allowed.

## AGE CONSTRAINTS VIOLATION:

```
mysql> INSERT INTO student VALUES (7,'TOM',16,76.4);
ERROR 3819 (HY000): Check constraint 'student_chk_1' is violated.
```

Note: Here, we got ERROR because, while creating the AGE column we given the CONSTRAINT as greater than 18. So when we tried to enter value less than 18 i.e. 16, it generated error.

Let's now try to insert new row without specifying any value into grade column.

```
mysql> INSERT INTO student (s_id,name,age) VALUES (7,'BOB',22);
Query OK, 1 row affected (0.01 sec)
```

Note: Here we have not got any ERROR, because we have set any CONSTRAINT or for **grade** column **DEFAULT** value is given as 0.0.

For **s\_id**, **name** and **age** column **DEFAULT** value was not mentioned or certain **CONSTRAINT** was given, so we got ERROR.

AFTER ALL THESE CHANGES LET'S SEE HOW STUDENT TABLE LOOK LIKE.

```
mysql> SELECT * FROM student;
+-----+-----+-----+-----+
| s_id | name | age  | grade |
+-----+-----+-----+-----+
|    1 | JOHN |   21 | 72.00 |
|    2 | MIKE |   20 | 68.00 |
|    3 | KATE |   22 | 86.00 |
|    4 | ANDY |   21 | 94.00 |
|    5 | TOM  |   21 | 76.40 |
|    6 | ANDY |   21 | 94.00 |
|    7 | BOB  |   22 | 0.00  |
+-----+-----+-----+-----+
7 rows in set (0.00 sec)
```

Note: while inserting last row, we have not given any value but default value to grade column have been added as 0.0. Because while creating grade column we have given the default value as 0.0.

**Note:** If we try to insert a row without specifying the any value to age column then by default **NULL** will be inserted to age column, because while creating age column we have not given any **PRIMARY key** or **NOT NULL constraint**.

```
mysql> SELECT * FROM student;
+----+----+----+----+
| s_id | name | age | grade |
+----+----+----+----+
| 1   | JOHN | 21  | 72.00 |
| 2   | MIKE | 20  | 68.00 |
| 3   | KATE | 22  | 86.00 |
| 4   | ANDY | 21  | 94.00 |
| 5   | TOM  | 21  | 76.40 |
| 6   | ANDY | 21  | 94.00 |
| 7   | BOB  | 22  | 0.00  |
| 8   | ALEX  | NULL | 98.90 |
+----+----+----+----+
8 rows in set (0.00 sec)
```

**Note:** If you notice, in STUDENT table data is ordered in ascending order, that is because of **PRIMARY KEY CONSTRAINT**. If we mention **PRIMARY KEY CONSTRAINTS** to any column then automatically physical ordering of data will be done in ascending order.

```
mysql> SELECT * FROM student;
+----+----+----+----+
| s_id | name | age | grade |
+----+----+----+----+
| 1   | JOHN | 21  | 72.00 |
| 2   | MIKE | 20  | 68.00 |
| 3   | KATE | 22  | 86.00 |
| 4   | ANDY | 21  | 94.00 |
| 5   | TOM  | 21  | 76.40 |
| 6   | ANDY | 21  | 94.00 |
| 7   | BOB  | 22  | 0.00  |
| 8   | ALEX  | NULL | 98.90 |
+----+----+----+----+
8 rows in set (0.00 sec)
```

## AUTO-INCREMENT CONSTRAINT.

Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.

Often this is the primary key field that we would like to be created automatically every time a new record is inserted.

Let's modify or alter STUDENT table s\_id constraints to AUTO\_INCREMENT.

```
mysql> ALTER TABLE student MODIFY s_id tinyint AUTO_INCREMENT;
Query OK, 8 rows affected (0.12 sec)
Records: 8  Duplicates: 0  Warnings: 0
```

**Note 1:** Already we have set Constraint for s\_id as PRIMARY\_KEY, again we modified the CONSTRATINTS to AUTO\_INCREMENT, but still PRIMARY KEY constraint will not be changed or in other word AUTO\_INCREMENT is set as EXTRA CONSTRAINT.

```
mysql> DESCRIBE student;
+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra
+-----+-----+-----+-----+
| s_id  | tinyint   | NO   | PRI | NULL    | auto_increment
| name  | varchar(10) | NO   |     | NULL    |
| age   | tinyint   | YES  |     | NULL    |
| grade | decimal(4,2)| YES  |     | 0.00    |
+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

**Note 2:** AUTO\_INCREMENT CONSTRAINT can be set only for the field for which PRIMARY KEY CONSTRAINTS as set.

Here, we are trying to set AUTO\_INCREMENT CONSTRAINT to age column,

```
mysql> ALTER TABLE student MODIFY age tinyint AUTO_INCREMENT;
ERROR 1075 (42000): Incorrect table definition; there can be only one auto column and it must be defined as a key
```

**Note 3:** In the previous STUDENT table we have inserted 8 rows, now if we try to insert row without specifying the s\_id , then automatically the value will be set to 9 as shown below.

```
mysql> INSERT INTO student (name,age,grade) VALUES ('RAM',19,88);
Query OK, 1 row affected (0.00 sec)
```

```
mysql> SELECT * FROM student;
+----+----+----+----+
| s_id | name | age | grade |
+----+----+----+----+
| 1   | JOHN | 21  | 72.00 |
| 2   | MIKE | 20  | 68.00 |
| 3   | KATE | 22  | 86.00 |
| 4   | ANDY | 21  | 94.00 |
| 5   | TOM  | 21  | 76.40 |
| 6   | ANDY | 21  | 94.00 |
| 7   | BOB  | 22  | 0.00  |
| 8   | ALEX | NULL | 98.90 |
| 9   | RAM  | 19  | 88.00 |
+----+----+----+----+
9 rows in set (0.00 sec)
```

Here, we have not specified the s\_id value but still it has been incremented and given the value as 9, same way if we try to insert one more row then AUTO\_INCREMENT will give the value as 10.

**Note 4: AUTO\_INCREMENT can be started with any sequence of letters,**

If we start with certain sequence then the next column will be automatically set the value based on the previous value

```
mysql> INSERT INTO student VALUES (101,'MERCY',19,79.2);
Query OK, 1 row affected (0.01 sec)
```

Here, we have inserted the row which is having the **s\_id** as 101. If we try to insert one more row then automatically that row **s\_id** will set as 102, as shown below.

```
mysql> INSERT INTO student (name,age,grade) VALUES ('ADAM',21,83.2);
```

s_id   name   age   grade			
1	JOHN	21	72.00
2	MIKE	20	68.00
3	KATE	22	86.00
4	ANDY	21	94.00
5	TOM	21	76.40
6	ANDY	21	94.00
7	BOB	22	0.00
8	ALEX	NULL	98.90
9	RAM	19	88.00
101	MERCY	19	79.20
102	ADAM	21	83.20

11 rows in set (0.00 sec)

**PRIMARY KEYS** are classified into two types, as shown below;



**Natural key:** Natural keys are such keys which is cannot be generated programmaticaly or automaticcally using AUTO\_INCREMENT CONSTRAINT.

Example: Email, SSN.

**Surrogate Key:** Surrogate keys are such keys which is generated programmaticaly or automaticcally using AUTO\_INCREMENT CONSTRAINT.

Example: Student Id.

## TYPES OF KEYS, FOREIGN KEY

### **Candidate Key:**

A candidate key is a column or a set of columns that can qualify as a primary key in the database. There can be multiple SQL candidate keys in a database relation and each candidate can work as a primary key for the table.

**student\_table**

s_id	name	age	grade	email id	ssn
1	JOHN	21	72	john@gmail.com	123478
2	MIKE	20	68	mike@yahoo.com	124563
3	KATE	22	86	kate@gmail.com	145678
4	ANDY	21	94	andy@gmail.com	167982

If you consider the table above here **s\_Id, email id, ssn(social security number)** all are unique identifiers for the table so all these columns represent **candidate keys**.

Candidate key is classified into

- **Primary key**
- **Alternate key or Secondary key**

Here s\_id is represented as the primary key so email id and ssn are considered as Alternate Key or secondary key.

Now primary key is further classified as

- Simple primary key
- Composite primary key

**Simple primary key:** A simple primary key is made up of a single field only, where only one column is a unique identifier for the table. If you considered the table below here s\_id is the unique identifier and is represented as simple primary key

**Student table**

s_id	name	age	grade
1	JOHN	21	72
2	MIKE	20	68
3	KATE	22	86
4	ANDY	21	94

**Composite primary key:**

A composite primary key combines more than one field to make a unique value.

**Student table**

s_id	name	age	grade
1	JOHN	21	72
2	MIKE	20	68
3	KATE	22	86
4	ANDY	21	94

4	KATE	23	65
---	------	----	----

If you observe the table above here there are two students with id 4 in this table. In this situation it is difficult to identify the row uniquely with only student id but with the combination of student id and name it will be easy to identify the row uniquely this is only called as composite key

You can create a MySQL composite Primary Key in the following two ways:

- During table creation using **CREATE TABLE statements**.
- After creating tables using the **ALTER TABLE statement**.

#### - **Creating MySQL Composite Primary Key while Table Creation**

To create a MySQL composite primary key during table creation you can follow the steps.

Suppose you want to create a **student** with the fields (s\_id, name, age, grade) table with a MySQL Composite primary key (s\_id, name).

```
CREATE TABLE student (
    s_id INT,
    name VARCHAR(45),
    age INT,
    grade INT,
    PRIMARY KEY (s_id, name)
);
```

In the above, a MySQL Composite primary is created with the column names **s\_id** and **name**.

You can verify the same using the command as below:

```
DESCRIBE student;
```

## Output:

Field	Type	Null	Key
s_id	int	NO	PRI
name	varchar(45)	NO	PRI
age	int	YES	
grade	int	YES	

After the successful execution of the above query, you can see that the Key column has two **PRI**. It means you have successfully added the MySQL Composite Primary key on **s\_id** and **name** columns.

**Next, you need to insert the values into this table as given below:**

```
INSERT INTO student (s_id, name, age, grade) VALUES (1,
'JOHN', 21, 72);
INSERT INTO student (s_id, name, age, grade) VALUES (2,
'MIKE', 20, 68);
INSERT INTO student (s_id, name, age, grade) VALUES (3,
'KATE', 22, 86);
INSERT INTO student (s_id, name, age, grade) VALUES (4,
'ANDY', 21, 94);
INSERT INTO student (s_id, name, age, grade) VALUES (4,
'KATE', 23, 65);
```

Next, you can execute the following command to display the data in the table:

```
SELECT * FROM student;
```

**Output:**

**Student table**

s_id	name	age	grade
1	JOHN	21	72
2	MIKE	20	68
3	KATE	22	86
4	ANDY	21	94
4	KATE	23	65

To understand Composite key with better clarity

If you try to execute below query

```
INSERT INTO student (s_id, name, age, grade) VALUES (1,
'JOHN', 21, 72);
```

**Output:**

```
Error Code: 1062. Duplicate entry '1-JOHN' for key
'student.PRIMARY'
```

You will get the above error because you are trying to add the same student id and name into the student table.

If you try to execute below query

```
INSERT INTO student (s_id, name, age, grade) VALUES (1,  
'ANDY', 21, 72);
```

The data will be inserted successfully as the combination of name and id is different.

## - Adding MySQL Composite Primary Key in Existing Table

Now to do this let's delete the constraint primary key from the above table. This can be achieved using the query below

```
ALTER TABLE student DROP primary key;
```

To just check whether primary key constraint is deleted let's describe the table

```
DESCRIBE student;
```

**Output:**

Field	Type	Null	key
s_id	int	NO	
name	varchar(45)	NO	
age	int	YES	
grade	int	YES	

As you can clearly see from the above table, s\_id and name are not the primary key in the key column.

Now, you can execute the **ALTER TABLE** statement to add a **MySQL Composite Primary key** as follows:

```
ALTER TABLE sql_notes.student ADD PRIMARY KEY(s_id, name);
```

You can verify the MySQL Composite Primary key is added into a table or not using the following command:

You can use the following command to verify if the MySQL Composite Primary key has been added to the table.

```
DESCRIBE sql_notes.student;
```

**Output:**

Field	Type	Null	key
s_id	int	NO	PRI
name	varchar(45)	NO	PRI
age	int	YES	
grade	int	YES	

In the output, we can see that the key column has **2 PRI**, which means we have successfully added the composite primary key to **s\_id** and **name** columns.

## Foreign Key:

Foreign Key helps establish database relationships and maintain referential integrity. They help link one or more columns in one table to another table. Here's how to add foreign key in MySQL.

If an attribute is a **primary key** in one table but was not used as a primary key in another table then the attribute which is not a primary key in the other table is called a foreign key.

You can add foreign key to the table in two ways:

- During Table Creation using **CREATE Command**
- After Table creation using **ALTER Command**

- **Adding foreign key during the creation of table**

If you consider the table student and course below, here c\_id is the fireign key which is referring to the primary key c\_id in the courses table.

The **course table is the parent table** which contains primary key which is acting like foriegn key is student table

The **student table is the child table** which contains foriegn key

### Student

s_id	name	age	grade	c_id
1	JOHN	21	72	1J
2	MIKE	20	68	1J
3	KATE	22	86	2P
4	ANDY	21	94	4PFS

## Course

c_id	name	rating
1J	Java	4.6
2P	Python	4.8
3JFS	Java Full Stack	4.8
4PFS	Python Full Stack	4.9

- First child table that is course table need to be created as shown below

```
CREATE TABLE sql_notes.courses(
    c_id VARCHAR(5) PRIMARY KEY,
    name VARCHAR(10),
    rating decimal
);
```

- Create parent table with foreign key

```
CREATE TABLE sql_notes.student(
    s_id tinyint PRIMARY KEY,
    name VARCHAR(10),
    age tinyint,
    grade decimal(4,2),
    c_id VARCHAR(5),
    FOREIGN KEY(c_id) REFERENCES sql_notes.courses(c_id)
);
```

You can verify the MySQL Composite Primary key is added into a table or not using the following command:

You can use the following command to verify if the MySQL foreign key has been added to the table.

```
DESCRIBE sql_notes.student;
```

### **Output:**

Field	Type	Null	Key
s_id	tinyint	NO	PRI
name	varchar(10)	YES	
age	tinyint	YES	
grade	decimal(4,2)	YES	
c_id	varchar(5)	YES	MUL

Here If you observe the key of c\_id is MUL. MUL is basically an index that is neither a primary key nor a unique key. The name comes from “multiple” because multiple occurrences of the same value are allowed. IT indicates that c\_id is foreign key.

Note: If you try to add any c\_id in the student table which is not there in the course table that time you will get an error.



Now let's insert the values in the table

### Course table

```
INSERT INTO courses (c_id, name, rating) VALUES ('1J', 'Java', 4.6);
INSERT INTO courses (c_id, name, rating) VALUES ('2P', 'Python', 4.8);
INSERT INTO courses (c_id, name, rating) VALUES ('3JFS', 'Java Full Stack', 4.8);
INSERT INTO courses(c_id, name, rating) VALUES ('4PFS', 'Python Full Stack', 4.9);
```

### Student table

```
INSERT INTO student (s_id, name, age, grade, c_id) VALUES (1, 'JOHN', 21, 72, '1J');
INSERT INTO student (s_id, name, age, grade, c_id) VALUES (2, 'MIKE', 20, 68, '1J');
INSERT INTO student (s_id, name, age, grade, c_id) VALUES (3, 'KATE', 22, 86, '2P');
INSERT INTO student (s_id, name, age, grade, c_id) VALUES (4, 'ANDY', 21, 94, '4PFS');
```

- After Table creation using **ALTER Command**

To understand this first let's drop the foreign key constraint available in the student table using the query below

```
ALTER TABLE student DROP FOREIGN KEY c_id;
```

### Output:

<pre>0 3 18:53:4 ALTER TABLE       5 7           sql_notes.student DROP                       FOREIGN KEY c_id</pre>	<b>Error Code: 1091. Can't</b> <b>DROP 'c_id'; check that</b> <b>column/key exists</b>	<b>0.000</b> <b>sec</b>
--	--	----------------------------

You will get the above error because you have not defined a constraint for foreign key.

Now let's try to add constraint for the foreign key using alter command

```
ALTER TABLE sql_notes.student  
ADD CONSTRAINT cIdFk FOREIGN KEY (c_id)  
REFERENCES sql_notes.courses(c_id);
```

Now by executing the below query you can drop a foreign key constraint

```
ALTER TABLE sql_notes.student DROP FOREIGN KEY cIdFk ;
```

Now there is no foreign key constraint in the table let's add foreign key using alter command

To create a **FOREIGN KEY** constraint on the "c\_id" column when the "student" table is already created, use the following MySQL:

```
ALTER TABLE student ADD FOREIGN KEY (c_id) REFERENCES courses(c_id);
```

## UPDATE, DELETE, ALTER COMMAND

### Update Query:

The UPDATE statement updates data in the existing table. It allows you to change the values in one or more columns of a single row or multiple rows.

Syntax of the UPDATE statement is shown below:

**UPDATE table\_name SET column\_name = new\_value [WHERE condition];**

HERE

- UPDATE table\_name is the command that tells MySQL to update the data in a table .
- SET column\_name = new\_value are the names and values of the fields to be affected by the update query.
- [WHERE condition] is optional and can be used to put a filter that restricts the number of rows affected by the UPDATE MySQL query.

**Let's considered the student table below to understand the update query**

**student**

s_id	name	age	grade
1	John	21	72
2	Mike	20	68
3	Kate	22	86
4	Andy	21	94

**Scenario 1:** Update all the students grade to 90 with the students grade greater than 80

```
UPDATE student SET grade = 90 WHERE grade > 80;
```

You can verify weather grade updated for all the students with grade > 80 by executing the query below

```
SELECT * FROM student;
```

**Output:****student**

s_id	name	age	grade
1	John	21	72
2	Mike	20	68
3	Kate	22	90
4	Andy	21	90

If you observe the table kate and andy grade is updated with grade = 90

**Scenario 2:** Update all the students grade to 70 with the students grade equal to 68

```
UPDATE student SET grade = 70 WHERE (grade = 68);
```

You can verify weather grade updated for all the students with grade = 68 by executing the query below

```
SELECT * FROM student;
```

**Output:****student**

s_id	name	age	grade

1	John	21	72
2	Mike	20	68
3	Kate	22	90
4	Andy	21	90

If you observe the table kate grade is updated with grade = 68

**Scenario 3:** Update all the students grade to 90 with the student id equal to 1

```
UPDATE student SET grade = 90 WHERE (s_id = 1);
```

You can verify weather grade updated for all the students with student id is 1 by executing the query below

```
SELECT * FROM student;
```

**Output:**

**student**

s_id	name	age	grade
1	John	21	90
2	Mike	20	68
3	Kate	22	90
4	Andy	21	90

If you observe the table John grade is updated with student id is 1



Now let's try to see how to update multiple columns and understand how it works

**Scenario 4:** Update the student name = 'Alex' and age = 23 with the student id equal to 3

```
UPDATE student SET name = 'Alex', age = 23 WHERE (s_id = 3);
```

You can verify whether name and grade updated for all the students with student id is 3 by executing the query below

```
SELECT * FROM student;
```

**Output:**

**student**

s_id	name	age	grade
1	John	21	90
2	Mike	20	68
3	Alex	23	90
4	Andy	21	90

If you observe the table name and age is updated with student id 3.

## **DELETE Query**

The **DELETE** statement is used to delete existing records in a table.

Syntax of the **DELETE** statement is shown below:

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

Consider the table given below:

**student**

s_id	name	age	grade
1	John	21	90
2	Mike	20	68
3	Alex	23	90
4	Andy	21	90

**Scenario 1:** DELETE the student with student id 4

```
DELETE FROM student WHERE s_id = 4;
```

You can verify weather student with student id is 3 deleted by executing the query below

```
SELECT * FROM student;
```

**Output:**

**student**

s_id	name	age	grade
1	John	21	90

2	Mike	20	68
3	Alex	23	90

If you observe the table there is no row with student id 4

**Scenario 2:** DELETE the student with student id 3 and age 22

```
DELETE FROM student WHERE s_id = 3 AND age = 22;
```

You can verify weather student with student id is 3 and age 22 is deleted by executing the query below

```
SELECT * FROM student;
```

**Output:**

**student**

s_id	name	age	grade
1	John	21	90
2	Mike	20	68
3	Alex	23	90

If you observe there was no record which match both student id and age so no row affected

## ALTER QUERY

The **ALTER TABLE** statement is used to add, delete, or modify columns in an existing table.

Syntax of the **ALTER** statement is shown below:

**ALTER TABLE** table\_name **ADD** column\_name datatype;

Consider the table given below:

**student**

s_id	name	age	grade
1	John	21	90
2	Mike	20	68
3	Alex	23	90

**Scenario 1:** Alter the student table by deleting the column grade

```
ALTER TABLE student DROP grade;
```

You can verify weather grade column is deleted by executing the query below

```
SELECT * FROM student;
```

**Output:**

**student**

s_id	name	age
1	John	21
2	Mike	20

3	Alex	23
---	------	----

If you observe the table there is no column grade

**Scenario 1:** Write a query to delete a particular value present in the table

```
UPDATE student SET age = NULL WHERE s_id = 2;
```

You can verify weather age is updated with null by executing the query below

```
SELECT * FROM student;
```

**Output:**

**student**

s_id	name	age
1	John	21
2	Mike	Null
3	Alex	23

If you observe the table age of mike with s\_id is 2 is updated

Let us understand the difference between **DELETE**, **DROP** and **TRUNCATE**

### **DELETE Table:**

**DELETE** is a DML (Data Manipulation Language) command. This command removes records from a table. It is used only for deleting data from a table, not to remove the table from the database.

You can delete **all records** with the syntax:

```
DELETE FROM table_name;
```

Or you can delete a **group of records** using the WHERE clause:

```
DELETE FROM table_name WHERE col=value;
```

### **TRUNCATE Table:**

**TRUNCATE TABLE** is similar to **DELETE**, but this operation is a DDL (Data Definition Language) command. It also deletes records from a table without removing table structure, but it doesn't use the WHERE clause.

Here's the syntax:

```
TRUNCATE TABLE table_name;
```

**TRUNCATE** is faster than **DELETE**, as it doesn't scan every record before removing it. TRUNCATE TABLE locks the whole table to remove data from a table; thus, this command also uses less transaction space than DELETE.

### **DROP Table:**

The **DROP TABLE** is another DDL (Data Definition Language) operation. But it is not used for simply removing data from a table; it deletes the table structure from the database, along with any data stored in the table.

Here is the syntax of this command:

```
DROP TABLE table_name;
```

## OPERATORS

### Relational / Comparison Operator:

Let us now understand relational/comparison operator by considering the table below:

**student**

s_id	name	age	grade
1	John	21	72
2	Mike	20	68
3	Kate	22	86
4	Andy	21	94

**Scenario 1:** Get all the students with grade greater than 80

```
SELECT * FROM student WHERE grade > 80;
```

### Output:

**student**

s_id	name	age	grade
3	Kate	22	86
4	Andy	21	94

If you observe the result above, you got the table with student greater than 80

**Scenario 2:** Get all the students with grade less than 80

```
SELECT * FROM student WHERE grade < 80;
```

**Output:**

**student**

s_id	name	age	grade
1	John	21	72
2	Mike	20	68

**Scenario 3:** Get all the students with grade equal to 72

```
SELECT * FROM student WHERE grade = 72;
```

**Output:**

**student**

s_id	name	age	grade
1	John	21	72

**Scenario 4:** Get all the students name and age with grade equal to 86

```
SELECT name, age FROM student WHERE grade = 86;
```

**Output:**

**student**

name	age
Kate	22

**Scenario 5:** Get all the students with age not equal to 21

```
SELECT * FROM sql_notes.student WHERE age != 21;
```

**Output:**

**student**

s_id	name	age	grade
2	Mike	20	68
3	Kate	22	86

Whenever you are working with Not equal you can use **<>** or **!=** both will perform same operations

**Scenario 6:** Get all the students with name greater than John

```
SELECT * FROM sql_notes.student WHERE name > 'John';
```

**Output:**

**student**

s_id	name	age	grade

2	Mike	20	68
3	Kate	22	86

Here as you are passing string and trying to fetch the data where student name is greater than John, now it will fetch the data of the student where alphabetically(ASCII Value) the student name starts with greater than J

## Let's add one more column DOJ to the student table

```
ALTER TABLE student ADD doj DATE;
```

To cross verify weather DOJ column is added in the student you can execute the below query

```
DESCRIBE student;
```

**Student**

Field	Type	Null	Key
s_id	tinyint	NO	PRI
name	varchar(10)	YES	
age	tinyint	YES	
grade	decimal(10,0)	YES	
doj	date	YES	

If you observe from the above output the DOJ field is successfully added.

Let's try to insert few more entry into the student table by executing the following query

```
INSERT INTO student(s_id, name, age, grade, DOJ) VALUES (5,
'Alex', 23, 98, '2021-01-26');
INSERT INTO student (s_id, name, age, grade, DOJ) VALUES (6,
'Bob', 23, 98, '2020-12-26');
INSERT INTO student (s_id, name, age, grade, DOJ) VALUES (7,
'Tom', 23, 98, '2021-12-26');
```

**Output:**

**student**

s_id	name	age	grade	doj
1	John	21	72	NULL
2	Mike	20	68	NULL
3	Kate	22	86	NULL
4	Andy	21	94	NULL
5	Alex	23	98	2021-01-26
6	Bob	23	98	2020-12-26
7	Tom	23	98	2021-12-26

**Scenario 6:** Write a query to fetch the students of all the data from the above table where date of joining(dojo) is greater than '2020-12-26'

```
SELECT * FROM sql_notes.student WHERE DOJ > '2020-12-26';
```

**Output:**

s_id	name	age	grade	doj
5	Alex	23	98	2021-01-26
7	Tom	23	98	2021-12-26

The output gets the data to match DOJ greater than '**2020-12-26**'

## Logical Operator:

Syntax Of AND

```
SELECT column1, column2 FROM table_name WHERE condition1  
AND condition2 ....;
```

Syntax of OR

```
SELECT column1, column2 FROM table_name WHERE condition1  
OR condition2 ....;
```

Syntax of NOT

```
SELECT column1, column2 FROM table_name WHERE NOT  
condition;
```

**Scenario 7:** Write a query to fetch the data from the student table where age is greater than 20 and grade is greater 70

```
SELECT * FROM sql_notes.student WHERE age > 20 AND grade  
> 70;
```

Output:

s_id	name	age	grade	doj
1	John	21	72	NULL
3	Kate	22	86	NULL
4	Andy	21	94	NULL
5	Alex	23	98	2021-01-26

6	Bob	23	98	2020-12-26
7	Tom	23	98	2021-12-26

**Scenario 8:** Write a query to fetch the data from the student table where age is greater than 20 or grade is greater 70

```
SELECT * FROM sql_notes.student WHERE age > 20 OR grade > 70;
```

**Output:**

s_id	name	age	grade	doj
1	John	21	72	NULL
3	Kate	22	86	NULL
4	Andy	21	94	NULL
5	Alex	23	98	2021-01-26
6	Bob	23	98	2020-12-26
7	Tom	23	98	2021-12-26

**Scenario 9:** Write a query to fetch the data from the student table where age is Not equal to 22

```
SELECT * FROM student WHERE NOT age > 22;
```

**Output:**

s_id	name	age	grade	doj
1	John	21	72	NULL
2	Mike	20	68	NULL
3	Kate	22	86	NULL
4	Andy	21	94	NULL

**Scenario 10:** write a query to fetch the data of the students when student id is not less than 4 and either age should be greater than 22 or grade should be greater than 80

```
SELECT * FROM sql_notes.student WHERE NOT s_id < 4 AND
(age < 22 OR grade > 80);
```

Output:

**student**

s_id	name	age	grade	doj
4	Andy	21	94	NULL
5	Alex	23	98	2021-01-26
6	Bob	23	98	2020-12-26
7	Tom	23	98	2021-12-26

## Arithmetic Operator

**Arithmetic operators are:**

- + [Addition]
- [Subtraction]
- / [Division]
- \* [Multiplication]
- % [Modulus]

**Scenario 1:** write a query to add two numbers using addition Operator

```
SELECT 1100 + 1900;
```

**Output:**

1100 + 1900

3000

**Scenario 2:** write a query to subtract two numbers using subtraction Operator

```
SELECT 1933 - 1288;
```

**Output:**

1933 - 1288

645

**Scenario 3:** write a query to multiply two numbers using multiplication Operator

```
SELECT 1200 * 1200;
```

**Output:**

1200 * 1200
1440000

**Scenario 4:** write a query to divide two numbers using Division Operator

```
SELECT 1200/500;
```

**Output:**

1200 / 500
2.4000

**Scenario 5:** write a query to get remainder of two numbers using modulus Operator

```
SELECT 11%2;
```

**Output:**

11%2
------

1
---

To change the column name you can make use of aliasing as shown below

```
SELECT 11%2 as result;
```

**Output:**

result
1

**Scenario 6:** write a query for the expression **100/10\*2+5-2%3**

```
SELECT 100/10*2+5-2%3 as result;
```

**Here whenever you are working on the expression it will look for precedent and precedence looks as shown below**

Precedence
------------

%
*, /
+, -

**Output:**

result
23.0000



**Now let us consider the department and employee table to understand the arithmetic operator**

### Department table

dept_id	dept_name	mrg_id
20	sales	8
30	executive	4
40	shipping	6
50	marketing	3
70	accounting	2
80	IT	1

### Employee table:

emp_id	first_name	last_name	email	hire_date	salary	dept_id
1	kelly	davis	davis@gmail.com	2021-01-22	78000	80
2	tom	taylor	tom@gmail.com	2020-09-22	84200	70
3	mike	whalen	mike@gmail.com	2021-06-30	98200	50
4	andy	lumb	andy@gmail.com	2021-02-27	42200	80
5	anjel	nair	anj@gmail.com	2019-09-26	42200	40
6	ram	kumar	ram@gmail.com	2018-12-26	64200	40

7	rohan	sharma	ro@gmail.com	2018-11-24	84200	20
8	john	king	j0@gmail.com	2021-02-09	124200	20

**Scenario 1:** Display first name, last name and salary of all the employees by adding 1000 to the salary of all the employees

```
SELECT
first_name as fn, last_name as ln, salary+1000 as salary
FROM
employee;
```

**Output:**

fn	ln	salary
kelly	davis	79000
tom	taylor	85200
mike	whalen	99200
andy	lumb	43200
anjel	nair	43200
ram	kumar	65200
rohan	sharma	85200
john	king	125200

**Scenario 2:** Display first name, last name and salary of all the employees by adding dept\_id to the salary of all the employees

```
SELECT  
first_name as fn, last_name as ln, salary+dept_id as salary  
FROM  
employee;
```

**Output:**

fn	ln	salary
kelly	davis	78080
tom	taylor	84270
mike	whalen	98250
andy	lumb	42280
anjel	nair	42240

ram	kumar	64240
rohan	sharma	84220
john	king	124220

**Scenario 3:** Display first name, last name and salary of all the employees by subtracting 1000 to the salary of all the employees

```
SELECT
first_name as fn, last_name as ln, salary-1000 as salary
FROM
employee;
```

**Output:**

fn	ln	salary
kelly	davis	77000
tom	taylor	83200
mike	whalen	97200
andy	lumb	41200
anjel	nair	41200
ram	kumar	63200
rohan	sharma	83200
john	king	123200

**Scenario 4:** Display first name, last name and annual salary of all the employees.

$$\text{annual salary} = \text{salary} * 12$$

```
SELECT
first_name as fn, last_name as ln, salary*12 as annula_salary
FROM
employee;
```

**Output:**

fn	ln	salary
kelly	davis	936000
tom	taylor	1010400
mike	whalen	1178400
andy	lumb	506400
anjel	nair	506400
ram	kumar	770400
rohan	sharma	1010400
john	king	1490400

**Scenario 5:** Display first name, last name and annual salary of all the employees with dept\_id as 20

```
SELECT
first_name as fn, last_name as ln, salary*12 as annual_salary
FROM
employee
WHERE
dept_id =20;
```

**Output:**

fn	ln	annual_salary
rohan	sharma	1010400
john	king	1490400

**Scenario 6:** Display first name, last name and half year salary of all the employees

```
SELECT
first_name as fn, last_name as ln, salary*12/2 as annual_salary
FROM
employee;
```

**Output:**

fn	ln	salary
kelly	davis	468000.0000
tom	taylor	505200.0000
mike	whalen	589200.0000
andy	lumb	253200.0000
anjel	nair	253200.0000
ram	kumar	385200.0000
rohan	sharma	505200.0000
john	king	745200.0000

**Scenario 7:** Display first name, last name and annual, half year, quarterly salary of all the employees

```

SELECT
first_name as fn, last_name as ln, (salary * 12) as
annual_salary, (salary*12/2) as half_yearly_salary,
(salary*12/4) as quarterly_salary
FROM
employee;
    
```

Output:

first_name	last_name	annual_salary	half_yearly_salary	quarterly_salary
kelly	davis	936000	468000.0000	234000.0000
tom	taylor	1010400	505200.0000	252600.0000
mike	whalen	1178400	589200.0000	294600.0000
andy	lumb	506400	253200.0000	126600.0000
anjel	nair	506400	253200.0000	126600.0000
ram	kumar	770400	385200.0000	192600.0000
rohan	sharma	1010400	505200.0000	252600.0000
john	king	1490400	745200.0000	372600.0000

**Scenario 8:** Display first name, last name and salary of all the employees where salary is not divisible(reminder is 0) by 3

```
SELECT
first_name as fn, last_name as ln, salary
FROM
employee
WHERE
salary%3 != 0;
```

Output:

fn	ln	salary
tom	taylor	84200
mike	whalen	98200
andy	lumb	42200
anjel	nair	42200
rohan	sharma	84200

**Scenario 9:** Display first name, last name and salary of all the employees with salary increment by 10%

```
SELECT first_name as fn, last_name as ln,
salary+(salary*10/100) FROM sql_notes.employee;
```

Output:

fn	ln	salary
kelly	davis	85800.0000
tom	taylor	92620.0000

mike	whalen	108020.0000
andy	lumb	46420.0000
anjel	nair	46420.0000
ram	kumar	70620.0000
rohan	sharma	92620.0000
john	king	136620.0000

## Keyword Operator

**Keyword operator are:**

LIKE

DISTINCT

IS NULL

IN

BETWEEN AND

Let us consider the table below to understand the keyword operator

**Employee table:**

emp_id	first_name	last_name	email	hire_date	salary	dept_id
1	kelly	davis	davis@gmail.com	2021-01-22	78000	80
2	tom	taylor	tom@gmail.com	2020-09-22	84200	70
3	mike	whalen	mike@gmail.com	2021-06-30	98200	50
4	andy	lumb	andy@gmail.com	2021-02-27	42200	80
5	anjel	nair	anj@gmail.com	2019-09-26	42200	40
6	ram	kumar	ram@gmail.com	2018-12-26	64200	40
7	rohan	sharma	ro@gmail.com	2018-11-24	84200	20
8	john	king	j0@gmail.com	2021-02-09	124200	20

**DISTINCT:**

The **SELECT DISTINCT** statement is used to return only distinct (different) values.

**SELECT DISTINCT Syntax**

**SELECT DISTINCT column1, column2, ... FROM table\_name;**

Scenario 1: write a query to display the distinct salary from employee

```
SELECT DISTINCT salary FROM sql_notes.employee;
```

Output:

salary
78000
84200
98200
42200
64200
124200

Scenario 2: write a query to update the hire date of the employee to '2021-02-09' where employee id is 7.

```
UPDATE sql_notes.employee SET hire_date = '2021-02-09' WHERE emp_id = 7;
```

To verify weather it is updated or not you can execute the following query

```
SELECT * FROM sql_notes.employee;
```

**Output:**

**Employee table:**

emp_id	first_name	last_name	email	hire_date	salary	dept_id
1	kelly	davis	davis@gmail.com	2021-01-22	78000	80
2	tom	taylor	tom@gmail.com	2020-09-22	84200	70
3	mike	whalen	mike@gmail.com	2021-06-30	98200	50
4	andy	lumb	andy@gmail.com	2021-02-27	42200	80
5	anjel	nair	anj@gmail.com	2019-09-26	42200	40
6	ram	kumar	ram@gmail.com	2018-12-26	64200	40
7	rohan	sharma	ro@gmail.com	2021-02-09	84200	20
8	john	king	j0@gmail.com	2021-02-09	124200	20

## BETWEEN AND Operator:

The **BETWEEN** command is used to select values within a given range. The values can be numbers, text, or dates.

The **BETWEEN** command is inclusive: begin and end values are included.

**BETWEEN AND** operator is also called as **Range Test Operator**

**Syntax:**

```
SELECT column1, column2.. FROM table_name WHERE column_name
BETWEEN value 1 AND value 2;
```

**Scenario 1:** write a query to display the first\_name, last\_name of all the employees where salary is in the range of 40000 - 70000

```
SELECT
    first_name, last_name
FROM
    employee
WHERE
    salary >= 40000 AND salary <= 70000;
```

**Output:**

first_name	last_name
andy	lumb
anjel	nair
ram	kumar

Now let's try to write the same query using BETWEEN operator

```
SELECT
    first_name, last_name
FROM
    employee
WHERE
    salary
BETWEEN 40000 AND 70000;
```

**Output:**

first_name	last_name
andy	lumb
anjel	nair
ram	kumar

**Scenario 2:** write a query to display the first\_name, last\_name of all the employees where hire date is in the range of **2018-01-01 to 2020-01-01**

```
SELECT
    first_name, last_name
FROM
    employee
WHERE
    hire_date
BETWEEN
    '2018-01-01' AND '2020-01-01';
```

**Output:**

first_name	last_name
anjel	nair
ram	kumar

**Scenario 3:** write a query to display the first\_name, last\_name of all the employees where first letter of first name is in between A and L

```

SELECT
    first_name, last_name
FROM
    employee
WHERE
    first_name
BETWEEN
    'A' AND 'L';

```

**Output:**

first_name	last_name
kelly	davis
andy	lumb
anjel	nair
john	king

**Scenario 4:** write a query to display the first\_name, salary of all the employees

where salary is not in the range of 40000 to 70000

```
SELECT
    first_name, salary
FROM
    employee
WHERE
    salary
NOT BETWEEN 40000 AND 70000;
```

**Output:**

first_name	salary
kelly	78000
tom	84200
mike	98200
rohan	84200
john	124200

**IN Operator:**

The IN operator allows you to specify multiple values in a WHERE clause.

The IN operator is a shorthand for multiple OR conditions.

**Syntax:**

```
SELECT column_name(s)
FROM table_name
```

```
WHERE column_name IN (value1, value2, ...);
```

**Scenario 4:** write a query to display the first\_name, salary of all the employees where salary is equal to 78000 or 42200 or 64200

```
SELECT
    first_name, salary
FROM
    employee
WHERE
    salary
IN (78000, 42200, 64200);
```

**Output:**

first_name	salary
kelly	78000
andy	42200
anjel	42200
ram	64200

**Scenario 5:** write a query to display the first\_name, salary of all the employees where salary is not equal to 78000 or 42200 or 64200

```

SELECT
    first_name, salary
FROM
    employee
WHERE
    salary
NOT IN (78000, 42200, 64200);

```

**Output:**

first_name	salary
tom	84200
mike	98200
rohan	84200
john	124200

**Scenario 6:** write a query to display the first\_name, salary of all the employees where salary is not equal to 78000 or 42200 or 64200

```

SELECT
    first_name, salary
FROM
    employee
WHERE
    salary
NOT IN (78000, 42200, 64200);

```

### Output:

first_name	salary
tom	84200
mike	98200
rohan	84200
john	124200

### IS NULL Operator:

To understand the IS NULL operator let us set the dept\_id of the employee where emp\_id is greater than 6 by executing the following statement

```
UPDATE
    employee
SET
    dept_id = NULL
WHERE
    emp_id > 6;
```

To verify weather the values have been updated with NULL by executing the below query

```
SELECT * FROM employee;
```

### Output:

emp_id	first_name	last_name	email	hire_date	salary	dept_id

1	kelly	davis	davis@gmail.com	2021-01-22	78000	80
2	tom	taylor	tom@gmail.com	2020-09-22	84200	70
3	mike	whalen	mike@gmail.com	2021-06-30	98200	50
4	andy	lumb	andy@gmail.com	2021-02-27	42200	80
5	anjel	nair	anj@gmail.com	2019-09-26	42200	40
6	ram	kumar	ram@gmail.com	2018-12-26	64200	40
7	rohan	sharma	ro@gmail.com	2021-02-09	84200	NULL
8	john	king	j0@gmail.com	2021-02-09	124200	NULL

**Scenario 1:** Write a query to display all the employee with dept\_id is NULL

```
SELECT * FROM
    employee
WHERE
    dept_id
IS NULL;
```

**Output:**

emp_id	first_name	last_name	email	hire_date	salary	dept_id
7	rohan	sharma	ro@gmail.com	2021-02-09	84200	NULL
8	john	king	j0@gmail.com	2021-02-09	124200	NULL

**Scenario 2:** Write a query to display first name and salary of all the employee with dept\_id is NOT NULL

```
SELECT
    first_name,salary
FROM
    employee
WHERE
    dept_id
IS NOT NULL;
```

**Output:**

first_name	salary
anjel	42200
ram	64200
mike	98200
tom	84200
kelly	78000
andy	42200

## LIKE Operator

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

You can use two wildcards with LIKE:

- % - Represents zero, one, or multiple characters
- \_ - Represents a single character (MS Access uses a question mark (?) instead)

**Scenario 1:** Write a query to display first name and last name of the employee whose first name starts with ‘AN’

Here we need to match first two characters as an after that any number of characters means ‘an%’**is the pattern matching**

```
SELECT
    first_name, last_name
FROM
    employee
WHERE
    first_name
    LIKE 'an%';
```

**Output:**

first_name	last_name
andy	lumb
anjel	nair

**Scenario 2:** Write a query to display the first name and last name of the employee whose first name has the second character ‘o’.

Here we are matching for second character as o, so there should be any single character before o(\_o) and after o any number of characters so '\_o%' is the pattern matching

```
SELECT
    first_name, last_name
FROM
    employee
WHERE
    first_name
LIKE '_o%';
```

**Output:**

first_name	last_name
tom	taylor
rohan	sharma
john	king

**Scenario 3:** Write a query to display first name and last name of the employee whose last name ends with ‘r’

```
SELECT first_name, last_name FROM sql_notes.employee WHERE
last_name LIKE '%r';
```

**Output:**

first_name	last_name
tom	taylor
anjel	nair
ram	kumar

**Scenario 4:** Write a query to display first name and last name of the employee whose last name contain substring ‘MA’

```
SELECT first_name, last_name FROM sql_notes.employee WHERE
last_name LIKE '%ma%';
```

**Output:**

first_name	last_name
ram	kumar
rohan	sharma

**Scenario 5:** Write a query to display first name and last name of the employee whose first name begins with ‘r’ and at least 3 characters in length

```
SELECT first_name, last_name FROM sql_notes.employee WHERE
first_name LIKE 'r__%';
```

**Output**

first_name	last_name
ram	kumar

rohan	sharma
-------	--------

**Scenario 6:** Write a query to display first name and last name of the employee whose first name begins contains ‘A’

```

SELECT
    first_name, last_name
FROM
    employee
WHERE
    first_name
LIKE '%a%';

```

**Output:**

first_name	last_name
andy	lumb
anjel	nair
ram	kumar
rohan	sharma

**Scenario 7:** Write a query to display first name and last name of the employee whose last name has last third character ‘L’

```

SELECT
    first_name, last_name

```

```

FROM
    employee
WHERE
    last_name LIKE '%l__';

```

**Output:**

first_name	last_name
tom	taylor
mike	whalen

**Scenario 8:** Write a query to display first name and last name of the employee whose last name has third character and last character ‘a’

```

SELECT
    first_name, last_name
FROM
    employee
WHERE
    last_name LIKE '__a%a';

```

**Output:**

first_name	last_name
rohan	sharma

**Scenario 9:** Write a query to display first name and last name of the employee whose first name has two consecutive ‘ll’

```

SELECT
    first_name, last_name

```

```

FROM
    employee
WHERE
    first_name LIKE '%ll%';

```

**Output:**

first_name	last_name
kelly	davis

**Scenario 10:** Write a query to display first name and last name of the employee whose first name is 5 characters long and has third character ‘h’

```

SELECT
    first_name, last_name
FROM
    employee
WHERE
    first_name LIKE '__h__';

```

**Output:**

first_name	last_name
rohan	sharma

**Scenario 11:** Write a query to display first name and last name of the employee whose last name consists of character ‘a’ twice in it

```

SELECT

```

```

        first_name, last_name
FROM
    employee
WHERE
    last_name LIKE '%a%a%' ;

```

**Output:**

first_name	last_name
rohan	sharma

**Scenario 12:** Write a query to display first name, last name and salary of the employee who are earning 6 digit salary

```

SELECT
        first_name, last_name, salary
FROM
    employee
WHERE
    salary LIKE '_____';

```

**Output:**

first_name	last_name	salary
john	king	124200

**Scenario 13:** Write a query to display first name, last name and salary of the employee whose salary starts with 4

```
SELECT  
    first_name, last_name, salary  
FROM  
    employee  
WHERE  
    salary LIKE '4%';
```

**Output:**

first_name	last_name	salary
andy	lumb	42200
anjel	nair	42200

**Scenario 14:** Write a query to display first name, last name and hire date of the employee who are hired in the year 2018

```
SELECT first_name, last_name, hire_date
FROM employee
WHERE hire_date LIKE '2018%';
```

**Output:**

first_name	last_name	hire_date
ram	kumar	2018-12-26

**Scenario 15:** Write a query to display first name, last name and hire date of the employee who are hired in the month of february

```
SELECT first_name, last_name, hire_date
FROM employee
WHERE hire_date LIKE '____-02%';
```

**Output:**

first_name	last_name	hire_date
andy	lumb	2021-02-27
rohan	sharma	2021-02-09

john	king	2021-02-09
------	------	------------

**Scenario 16:** Write a query to display first name, last name and hire date of the employee whose salary has last 3 digit as 0's

```
SELECT
    first_name, last_name, salary
FROM
    employee
WHERE
    salary LIKE '%000';
```

Output:

first_name	last_name	salary
kelly	davis	78000

## Like Operator with escape keyword

1. Query the first name and last name of employees whose first name is not 5 characters long

NOT LIKE will fetch all the data that will not match the pattern

```
SELECT
    first_name, last_name
FROM
    employee
WHERE
    First_name NOT LIKE '_____';
```

### Output:

first_name	last_name
tom	taylor
mike	whalen
andy	lumb
ram	kumar
john	king

2. Query the first name and last name of employees whose first name does not begin with 'A'

```
SELECT
    first_name, last_name
FROM
    employee
WHERE
    first_name NOT LIKE 'a%';
```

**Output:**

<b>first_name</b>	<b>last_name</b>
kelly	davis
tom	taylor
mike	whalen
ram	kumar
rohan	sharma
john	king

3. Query the first name and last name of employees whose first name ends with ‘n’ but does not begin with ‘A’

```

SELECT
    first_name, last_name
FROM
    employee
WHERE
    first_name
        LIKE '%n' AND first_name NOT LIKE 'a%';
    
```

**Output:**

<b>first_name</b>	<b>last_name</b>
rohan	sharma
john	king

## ESCAPE KEYWORD WITH LIKE OPERATOR:

The ESCAPE keyword is used to **escape pattern matching characters** such as the (%) percentage and underscore (\_) if they form part of the data.

Before we understand escape keyword with an example let us include one record with the special character using UPDATE query

4. Query the first name and email id of employees with email id's containing \_ in it

```

SELECT
    first_name, email
FROM
    employee
WHERE
    email LIKE '%#_%' ESCAPE '#';

```

In this query, the ESCAPE clause specified that the character# is the escape character. It instructs the LIKE operator to treat the \_ character as a literal string instead of a wildcard. Note that without the ESCAPE clause, the query would return an empty result set.

### Output:

first_name	email
john	jo__% %\$@gmail.com

5. Display the first name and email id of employees with email ids containing % in it

```
SELECT
    first_name, email
FROM
    employee
WHERE
    email
LIKE '%1%' ESCAPE '1';
```

In this query, the ESCAPE clause specified that the character 1 is the escape character. It instructs the LIKE operator to treat the % character as a literal string instead of a wildcard. Note that without the ESCAPE clause, the query would return an empty result set.

#### **Output:**

first_name	email
me	
john	jo__%%\$@gmail.com

6. Display the first name, email id of employees with email ids containing two consecutive \_\_ in it

```
SELECT
    first_name, email
FROM
    employee
WHERE
    email
LIKE '%#_#_%' ESCAPE '#';
```

#### **Output:**

first_name	email

john	jo__% %\$@gmail.com
------	---------------------

7. Display the first name, email id of employees with email ids containing two \_\_ in it

```
SELECT
    first_name, email
FROM
    employee
WHERE
    email
LIKE '%#_%#_%' ESCAPE '#';
```

**Output:**

first_name	email
john	jo__% %\$@gmail.com

## Hackerrank Queries

### 1. Weather Observation station 3

<https://www.hackerrank.com/challenges/weather-observation-station-3/problem?isFullScreen=true>

Query a list of CITY names from STATION for cities that have an even ID number. Print the results in any order, but exclude duplicates from the answer.

The STATION table is described as follows:

STATION	
Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT\_N is the northern latitude and LONG\_W is the western longitude.

#### Solution

```
SELECT DISTINCT
    CITY
FROM
    STATION
WHERE ID%2 =0;
```

Here DISTINCT Keyword is used as they are asking for unique values in the result. We are trying to fetch the values where the ID is an even number so we are checking when the number divided by 2 remainder will be zero or not.

## 2. Weather Observation Station 6

<https://www.hackerrank.com/challenges/weather-observation-station-6/problem?isFullScreen=true>

Query the list of CITY names starting with vowels (i.e., a, e, i, o, or u) from STATION. Your result cannot contain duplicates.

### Input Format

The STATION table is described as follows:

**STATION**

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT\_N is the northern latitude and LONG\_W is the western longitude.

### Solution:

```
SELECT DISTINCT
    CITY
FROM
    STATION
WHERE
    (CITY LIKE 'A%' OR CITY LIKE 'E%' OR CITY LIKE 'I%' OR CITY LIKE
    'O%' OR CITY LIKE 'U%');
```

### 3. Weather Observation Station 9

<https://www.hackerrank.com/challenges/weather-observation-station-9/problem?isFullScreen=true>

Query the list of CITY names from STATION that do not start with vowels.

Your result cannot contain duplicates.

#### Input Format

The STATION table is described as follows:

STATION	
Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT\_N is the northern latitude and LONG\_W is the western longitude

#### Solution:

```
SELECT DISTINCT
    CITY
FROM
    STATION
WHERE NOT
    (CITY LIKE 'A%' OR CITY LIKE 'E%' OR CITY LIKE 'I%' OR CITY LIKE
    'O%' OR CITY LIKE 'U%');
```

#### 4. Weather Observation Station 7

<https://www.hackerrank.com/challenges/weather-observation-station-7/problem?isFullScreen=true>

Query the list of CITY names ending with vowels (a, e, i, o, u) from STATION.

Your result cannot contain duplicates.

#### Input Format

The STATION table is described as follows:

STATION	
Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT\_N is the northern latitude and LONG\_W is the western longitude.

#### Solution:

```
SELECT DISTINCT
    CITY
FROM
    STATION
WHERE
    (CITY LIKE '%A' OR CITY LIKE '%E' OR CITY LIKE '%I' OR CITY LIKE
    '%O' OR CITY LIKE '%U');
```

## 5. Weather Observation Station 10

<https://www.hackerrank.com/challenges/weather-observation-station-10/problem?isFullScreen=true>

Query the list of CITY names from STATION that do not end with vowels.

Your result cannot contain duplicates.

### Input Format

The STATION table is described as follows:

**STATION**

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT\_N is the northern latitude and LONG\_W is the western longitude.

### Solution:

```
SELECT DISTINCT
    CITY
FROM
    STATION
WHERE NOT
    (CITY LIKE '%A' OR CITY LIKE '%E' OR CITY LIKE '%I' OR CITY LIKE
    '%O' OR CITY LIKE '%U');
```

## CONCATENATION OPERATORS

concatenation operator (//), which joins two distinct strings into one string value MySQL supports CONCAT as a synonym for the SQL concatenation operator and uses the || operator for logical OR.

If you want to use (||) operator sometimes it will not support always so you need to run the below commands

```
SET @old_sql_mode = @@sql_mode;
SET @@sql_mode = PIPES_AS_CONCAT;
```

1. Write a query to display first\_name, last\_name of all the employees as full\_name

```
SELECT
    first_name || last_name
FROM
    employee;
```

**Output:**

first_name    last_name
kellydavis
tomtaylor
mikewhalen
andylumb
anjelnair
ramkumar

rohansharma

johnking

Here above we have used `||` which is known as the **Concatenation operator** which is used to link 2 or as **many** columns as you want in your select query and it is **independent of the datatype** of the column.

Here above we have linked 2 columns i.e, `first_name+last_name` as well.

2. Write a query to display the following output:

KELLY GETS 78000 AS SALARY

TOM GETS 84200 AS SALARY

Like this for all employees

```
SELECT
    first_name || ' ' || 'GETS' || ' ' || salary || ' as ' ||
'salary' as details
FROM
    employee;
```

Output:

details
kelly GETS 78000 as salary
tom GETS 84200 as salary
mike GETS 98200 as salary
andy GETS 42200 as salary
anjel GETS 42200 as salary
ram GETS 64200 as salary

```
rohan GETS 84200 as salary
```

```
john GETS 124200 as salary
```

Here whenever you want to get the value that you have written as it is on the console you need to write that inside single quotes so that it will get printed.

3. Write a query to display the following output:

```
EMPLOYEE KELLY IS HIRED ON 2021-01-22
```

```
EMPLOYEE TOM IS HIRED ON 2020-09-22
```

```
.
```

```
.
```

Like this for all employees

```
SELECT
    'EMPLOYEE ' || first_name || ' IS ' || ' HIRED '
|| ' ON ' || hire_date as details
FROM
    employee;
```

Output:

details
EMPLOYEE kelly IS HIRED ON 2021-01-22
EMPLOYEE tom IS HIRED ON 2020-09-22
EMPLOYEE mike IS HIRED ON 2021-06-30
EMPLOYEE andy IS HIRED ON 2021-02-27
EMPLOYEE anjel IS HIRED ON 2019-09-26
EMPLOYEE ram IS HIRED ON 2018-12-26

EMPLOYEE rohan IS HIRED ON 2021-02-09

EMPLOYEE john IS HIRED ON 2021-02-09

4. Write a query to concatenate ram and rohan

```
SELECT  
    'RAM' || 'ROHAN'  
FROM  
    employee;
```

**Output:**

'RAM'    'ROHAN'
RAMROHAN

Here you are getting the concatenated result but you are getting the values as many numbers of employees are there in the record.

If you want the concatenated result one time you need to execute the below query

```
SELECT 'RAM' || 'ROHAN';
```

**Output:**

```
'RAM'||'ROHAN'
```

```
RAMROHAN
```

5. Write a query to display first name, last name, email id of first two employees separated by comma

```
SELECT
    first_name || ',' || last_name || ',' || email
FROM
    employee
WHERE
    emp_id < 3;
```

**Output:**

```
first_name || last_name || email
```

```
kelly,davis,davis@gmail.com
```

```
tom,taylor,tom@gmail.com
```

**LIMIT Keyword:**

The limit keyword is used to limit the number of rows returned in a query result.

6. Write a query to display first name, last name, email id of any two employees separated by comma.

Here whenever the restriction is to number of rows that need to be displayed in result you can make use of **LIMIT Keyword**

```
SELECT
```

```
first_name || ',' || last_name || ',' || email as  
details  
FROM  
    employee  
LIMIT 2;
```

**Output:**

details
kelly,davis,davis@gmail.com
tom,taylor,tom@gmail.com

**OFFSET Keyword:**

The **OFFSET** value is also most often used together with the **LIMIT** keyword.  
The OFF SET value allows us to specify which row to start from retrieving data

7. Write a query to display first name, last name, email id of any the employees separated by comma from 3rd row

```
SELECT
    first_name || ',' || last_name || ',' || email as
details
FROM
    employee
LIMIT 2 OFFSET 2;
```

**Output:**

details
mike,whalen,mike@gmail.com
andy,lumb,andy@gmail.com

Now you are getting the data from 3 rows as the offset is 2, it will start fetching the data from the 3rd row.

## Queries On Operators

1. Query first name, last name, salary of employees whose salary is greater than 60000 and their first name should not end with ‘n’ or ends with ‘m’

```

SELECT
    first_name, last_name, salary
FROM
    employee
WHERE
    salary > 60000
AND
    (first_name NOT LIKE '%n' OR first_name LIKE '%m');

```

### Output:

first_name	last_name	salary
kelly	davis	78000
tom	taylor	84200
mike	whalen	98200
ram	kumar	64200

2. Query first name, dept id, salary of employees whose dept id is either 20 or 40 or 80 and half yearly salary is greater than 500000

```

SELECT
    first_name, dept_id, salary
FROM
    employee
WHERE
    dept_id IN (20, 40, 80)
AND
    salary > 500000;

```

```
(salary *6) > 50000;
```

**Output:**

first_name	dept_id	salary
anjel	40	42200
ram	40	64200
kelly	80	78000
andy	80	42200

Whenever you are trying to fetch the values that matches either of the values you can make use of **IN/OR operator**

3. Query first name, last name, dept id, hire date of employees whose first name do not end with n and is hired in the year 2021 and has dept id in the range 20 AND 70

```
SELECT
    first_name, last_name, dept_id ,hire_date
FROM
    employee
WHERE
    first_name NOT LIKE '%n'
AND
    hire_date LIKE '2021%'
AND
    dept_id between 20 AND 70;
```

**Output:**

firs_name	last_name	dept_id	hire_date
mike	whalen	50	2021-06-30

4. Query the details of employees who earns in the range of 50000 AND 80000 OR has dept id 20 or 40

```

SELECT
    first_name, salary, dept_id
FROM
    employee
WHERE
    salary BETWEEN 50000 AND 80000
OR
    dept_id IN (20,40);

```

#### Output:

first_name	salary	dept_id
kelly	78000	80
anjel	42200	40
ram	64200	40
rohan	84200	20
john	124200	20

5. Query the first name AND last name as full name of all employees whose salary is greater than 40000 but less than 90000 and first name starting with r and ending with n with mail id not null

```

SELECT
    first_name||last_name as fullname
FROM
    employee
WHERE

```

```
    salary  
BETWEEN 40000 AND 90000  
AND  
    first_name LIKE 'r%n'  
AND  
    email IS NOT null;
```

**Output:**

<b>fullname</b>
rohansharma

## Built In Functions

**MySQL can do much more than just store and retrieve data.** We can also **perform manipulations on the data** before retrieving or saving it. That's where MySQL Functions come in. Functions are simply pieces of code that perform some operations and then return a result. Some functions accept parameters while other functions do not accept parameters.

### Types Of Functions:

- String Functions
- Numeric Functions
- Date Functions
- Control Flow Function
- Conversion Function
- Aggregate Function

### String Functions:

1. Query the first name of all the employees in uppercase

Now to convert the string to uppercase you can use of upper() function as shown below

#### **UPPER(str)**

The function changes all characters of the specified str string to uppercase and outputs the result.

```
SELECT
    upper(first_name) as firstname
FROM
    employee ;
```

**Output:**

firstname
KELLY
TOM
MIKE
ANDY
ANJEL
RAM
ROHAN
JOHN

2. Query the last name of all the employees in lowercase

Now to convert the string to lowercase you can use of lower() function as shown below

**LOWER(str)**

The function changes all characters of the specified str string to lowercase and outputs the result.

```
SELECT
    LOWER(last_name) as lastname
FROM
    employee ;
```

**Output:**

lastname
davis
taylor
whalen
lumb
nair
kumar
sharma
king

3. Display all the letters of ‘john’ in upper case

```
SELECT UPPER('john');
```

**Output:**

UPPER('john')
JOHN

4. Query the length of the string ‘Ronaldo’

To retrieve the length of the string you can make use of length()

**Syntax:**

```
SELECT LENGTH(string) as alias;
```

```
SELECT LENGTH('Ronaldo');
```

Output:

<b>LENGTH('Ronaldo')</b>
7

5. Query the first name of all the employees along with their size

```
SELECT  
    first_name, length(first_name) as len  
FROM  
    employee;
```

Output:

first_name	len
kelly	5
tom	3
mike	4
andy	4
anjel	5
ram	3
rohan	5

john	4
------	---

6. Query firstname, salary of all employees who are earning 5 digits salary without using like operator

```

SELECT
    first_name, salary
FROM
    employee
WHERE
    length(salary) =5;

```

#### Output:

first_name	salary
kelly	78000
tom	84200
mike	98200
andy	42200
anjel	42200
ram	64200
rohan	84200

7. Write a query to concatenate first name and last name of all employee without using concatenation operator

For this you can make use of **CONCAT()**. This function is beneficial when we need to concatenate or merge two or more strings or words.

```
SELECT  
CONCAT(first_name, last_name) as details  
FROM employee;
```

**Output:**

details
kellydavis
tomtaylor
mikewhalen
andylumb
anjelnair
ramkumar
rohansharma
johnking

8. Query first name, last name, email id of all the employees separated by spaces

```
SELECT  
CONCAT(first_name, ' ', last_name, ' ', email) as details  
FROM employee;
```

**Output:**

details
kelly davis davis@gmail.com
tom taylor tom@gmail.com
mike whalen mike@gmail.com
andy lumb andy@gmail.com
anjel nair anj@gmail.com
ram kumar ram@gmail.com
rohan sharma ro@gmail.com
john king jo__%\$@gmail.com

You can achieve the same using **CONCAT\_WS()**. The syntax is as shown below

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

```
SELECT
CONCAT_WS('_',first_name, last_name, email) as details
FROM employee;
```

**Output:**

details
kelly_davis_davis@gmail.com
tom_taylor_tom@gmail.com

mike\_whalen\_mike@gmail.com

andy\_lumb\_andy@gmail.com

anjel\_nair\_anj@gmail.com

ram\_kumar\_ram@gmail.com

rohan\_sharma\_ro@gmail.com

john\_king\_jo\_\_%%%\$@gmail.com

9. Query the substring from the string ‘RONALDO’ FROM 2nd position and extract 5 characters

This can be achieved using **SUBSTRING()**. The syntax of substring() is shown below

**SELECT SUBSTRING("STRING", starting, length) AS ExtractString;**

**OR**

**SUBSTRING(string FROM start FOR length)**

This function returns the specified number of characters from a particular position of a given string.

```
SELECT SUBSTRING('RONALDO', 2, 5) as sub;
```

**Output:**

<b>sub</b>
ONALD

10.Query the first name of all the employees along with first character of first name

```
SELECT  
    first_name, SUBSTRING(first_name, 1,1) as sub  
FROM  
    employee ;
```

**Output:**

first_name	sub
kelly	k
tom	t
mike	m
andy	a
anjel	a
ram	r
rohan	r
john	j

11.Query the first name along with last character of first name

```
SELECT  
    first_name, SUBSTRING(first_name, length(first_name)) as sub  
FROM  
    employee ;
```

**OR**

```
SELECT  
    first_name, SUBSTRING(first_name, -1) as sub  
FROM  
    employee ;
```

**Output:**

first_name	sub
kelly	y
tom	m
mike	e
andy	y
anjel	l
ram	m
rohan	n
john	n



12.Query the first name of all the employees whose first character begins with ‘R’ without using like operator

```
SELECT
    first_name
FROM
    employee
WHERE
    SUBSTRING(first_name, 1,1) = 'r';
```

**Output:**

first_name
ram
rohan

13.Query the first name of all the employees whose first character is a vowel

```
SELECT
    first_name
FROM
    employee
WHERE SUBSTRING(first_name, 1,1) IN
    ('a','e', 'i', 'o', 'u');
```

**Output:**

first_name
andy
anjel

14.Query the last name of all the employees whose last but 1 character is not a vowel

```
SELECT
    last_name
FROM
    employee
WHERE
    SUBSTRING(last_name, -2,1)
NOT IN ('a','e', 'i', 'o', 'u');
```

**Output:**

last_name
lumb
sharma
king

15.Query the first name of all the employees with first character in uppercase

```
SELECT
    UPPER(SUBSTRING(first_name, 1,1)) ||
    SUBSTRING(first_name, 2)
as firstname
FROM employee;
```

**Output:**

first_name
------------

Kelly

Tom

Mike

Andy

Anjel

Ram

Rohan

John

Here first we are extracting the first character using substring() then for that function we are calling upper() so that the first character will get converted to uppercase. Next all other characters extracted using substring and concatenate with a concatenation operator we can make use of CONCAT() as well.

16.Query the first name of all the employees with first character in lowercase and remaining characters in uppercase

```
SELECT
LOWER(SUBSTRING(first_name, 1,1)) ||
UPPER(SUBSTRING(first_name, 2))
as firstname
FROM employee;
```

**Output:**

<b>first_name</b>
-------------------

kELLY
tOM
mIKE
aNDY
aNJEL
rAM
rOHAN
jOHN

17.Query the first half characters of first name in uppercase and rest in lowercase

```
SELECT
    UPPER(SUBSTRING(first_name, 1,length(first_name)/2)) ||
    LOWER(SUBSTRING(first_name, length(first_name)/2+1))
as FN
FROM employee;
```

**Output:**

<b>FN</b>
KELly
TOm

MIke

ANdy

ANJel

RAm

ROHan

JOhn

18.Query the characters from position 5 to 9 after concatenating first name and last name

```
SELECT
SUBSTRING(first_name || last_name, 5, 5) as F_N
FROM employee;
```

**Output:**

F_N
ydavi
aylor
whale
lumb
lnair
umar
nshar
king

## TRIM() and REPLACE()

### TRIM() Function:

The SQL TRIM() removes leading and trailing characters(or both) from a character string.

### Syntax:

```
TRIM( [ LEADING | TRAILING | BOTH] [removed_str] FROM  
str);
```

1. Write a query to trim leading ‘D’ in the string ‘DAVID’

```
SELECT  
TRIM(LEADING 'D' FROM 'DAVID') as TR;
```

### Output:

TR
AVID

2. Write a query to trim trailing ‘D’ in the string ‘DAVID’

```
SELECT  
TRIM(TRAILING 'D' FROM 'DAVID') as TR;
```

### Output:

TR
DAVI

3. Write a query to trim both 'D' in the string 'DAVID'

```
SELECT  
TRIM(BOTH 'D' FROM 'DAVID') as TR;
```

Output:

TR
AVI

4. Write a query to remove redundant white spaces from 'DAVID'

```
SELECT  
TRIM(BOTH ' ' FROM ' DAVID ') as TR;
```

OR

```
SELECT TRIM(' DAVID ') as TR;
```

If you don't specify leading, trailing, both that time by default it will take leading and trailing both and will remove space

Output:

TR
DAVID

5. Write a query to remove leading white spaces from ' DAVID '

```
SELECT TRIM(LEADING ' ' FROM ' DAVID ') as TR;
```

Output:

TR
----

DAVID

6. Write a query to remove trailing white spaces from ' DAVID '

```
SELECT TRIM(TRAILING ' ' FROM ' DAVID ') as TR;
```

**Output:**

TR

DAVID

7. Write a query to remove redundant white space from first name

```
SELECT TRIM(first_name) FROM employee;
```

**OR**

```
SELECT LTRIM(first_name) FROM employee;
```

**Output:**

TRIM(first\_name)

kelly

tom

mike

andy

anjel

ram

rohan
john

8. Update the first name of all employees by removing unwanted white spaces

```
UPDATE employee SET first_name = TRIM(first_name);
```

To verify whether it updated or not you can execute the query below

```
SELECT first_name FROM sql_notes.employee;
```

### Output:

first_name
kelly
tom
mike
andy
anjel
ram
rohan
john



## **REPLACE() Function:**

To replace all occurrences of a substring within a string with a new substring, you use the REPLACE() function as follows:

```
REPLACE(input_string, substring, new_substring);
```

1. Write a query to replace character ‘A’ from all first name to ‘@’

```
SELECT  
REPLACE(first_name, 'a', '@')  
as RL  
FROM employee;
```

## **Output:**

RL
kelly
tom
mike
@ndy
@njel
r@m
roh@n
john

If you observe from the above output here all the character a is replaced with character @ using substring.

2. Write a query to change ‘SOON’ to ‘MOON’

```
SELECT REPLACE('SOON', 'S', 'M') as RP;
```

**Output:**

RP
MOON

3. Write a query to change ‘BOAT’ to ‘FLOAT’

```
SELECT REPLACE('BOAT', 'B', 'FL') as RP;
```

**Output:**

RP
FLOAT

4. Write a query to change ‘JACK AND JILL’ to ‘HACK AND HILL’

```
SELECT REPLACE('JACK AND JILL', 'J', 'H') as RP;
```

**Output:**

RP
HACK AND HILL

5. Write a query to change ‘JACK AND JUE’ to ‘BLACK AND BLUE’

```
SELECT REPLACE('JACK AND JUE', 'J', 'BL') as RP;
```

**Output:**

RP

HACK AND HILL

6. Write a query to correct the spelling of gmail in email column

```
UPDATE  
    employee  
SET  
    email = replace(email, 'gamil', 'gmail');
```

To verify you can execute the query below

```
SELECT email FROM sql_notes.employee ;
```

**Output:**

email
davis@gmail.com
tom@gmail.com
mike@gmail.com
andy@gmail.com
anj@gmail.com
ram@gmail.com
ro@gmail.com
jo__%%%\$@gmail.com

7. Write a query to format the number (987)6783457 to 9876783457

```
SELECT REPLACE(REPLACE('(987)6783457', '(', ''), ')', '') as  
RP;
```

**Output:**

RP
9876783457

## TRIM() and REPLACE()

### TRIM() Function:

The SQL TRIM() removes leading and trailing characters(or both) from a character string.

### Syntax:

```
TRIM( [ LEADING | TRAILING | BOTH] [removed_str] FROM  
str);
```

1. Write a query to trim leading ‘D’ in the string ‘DAVID’

```
SELECT  
TRIM(LEADING 'D' FROM 'DAVID') as TR;
```

### Output:

TR
AVID

2. Write a query to trim trailing ‘D’ in the string ‘DAVID’

```
SELECT  
TRIM(TRAILING 'D' FROM 'DAVID') as TR;
```

### Output:

TR
DAVI

3. Write a query to trim both 'D' in the string 'DAVID'

```
SELECT  
TRIM(BOTH 'D' FROM 'DAVID') as TR;
```

Output:

TR
AVI

4. Write a query to remove redundant white spaces from 'DAVID'

```
SELECT  
TRIM(BOTH ' ' FROM ' DAVID ') as TR;
```

OR

```
SELECT TRIM(' DAVID ') as TR;
```

If you don't specify leading, trailing, both that time by default it will take leading and trailing both and will remove space

Output:

TR
DAVID

5. Write a query to remove leading white spaces from ' DAVID '

```
SELECT TRIM(LEADING ' ' FROM ' DAVID ') as TR;
```

Output:

TR
----

DAVID

6. Write a query to remove trailing white spaces from ' DAVID '

```
SELECT TRIM(TRAILING ' ' FROM ' DAVID ') as TR;
```

**Output:**

TR

DAVID

7. Write a query to remove redundant white space from first name

```
SELECT TRIM(first_name) FROM employee;
```

**OR**

```
SELECT LTRIM(first_name) FROM employee;
```

**Output:**

TRIM(first\_name)

kelly

tom

mike

andy

anjel

ram

rohan
john

8. Update the first name of all employees by removing unwanted white spaces

```
UPDATE employee SET first_name = TRIM(first_name);
```

To verify whether it updated or not you can execute the query below

```
SELECT first_name FROM sql_notes.employee;
```

### Output:

first_name
kelly
tom
mike
andy
anjel
ram
rohan
john



## **REPLACE() Function:**

To replace all occurrences of a substring within a string with a new substring, you use the REPLACE() function as follows:

```
REPLACE(input_string, substring, new_substring);
```

1. Write a query to replace character ‘A’ from all first name to ‘@’

```
SELECT  
REPLACE(first_name, 'a', '@')  
as RL  
FROM employee;
```

## **Output:**

RL
kelly
tom
mike
@ndy
@njel
r@m
roh@n
john

If you observe from the above output here all the character a is replaced with character @ using substring.

2. Write a query to change ‘SOON’ to ‘MOON’

```
SELECT REPLACE('SOON', 'S', 'M') as RP;
```

**Output:**

RP
MOON

3. Write a query to change ‘BOAT’ to ‘FLOAT’

```
SELECT REPLACE('BOAT', 'B', 'FL') as RP;
```

**Output:**

RP
FLOAT

4. Write a query to change ‘JACK AND JILL’ to ‘HACK AND HILL’

```
SELECT REPLACE('JACK AND JILL', 'J', 'H') as RP;
```

**Output:**

RP
HACK AND HILL

5. Write a query to change ‘JACK AND JUE’ to ‘BLACK AND BLUE’

```
SELECT REPLACE('JACK AND JUE', 'J', 'BL') as RP;
```

**Output:**

RP

HACK AND HILL

6. Write a query to correct the spelling of gmail in email column

```
UPDATE  
    employee  
SET  
    email = replace(email, 'gamil', 'gmail');
```

To verify you can execute the query below

```
SELECT email FROM sql_notes.employee ;
```

**Output:**

email
davis@gmail.com
tom@gmail.com
mike@gmail.com
andy@gmail.com
anj@gmail.com
ram@gmail.com
ro@gmail.com
jo__%%%\$@gmail.com

7. Write a query to format the number (987)6783457 to 9876783457

```
SELECT REPLACE(REPLACE('(987)6783457', '(', ''), ')', '') as  
RP;
```

**Output:**

RP
9876783457

## INSTR() Function

This function in MySQL is used to return the location of the first occurrence of a substring within a given string.

### Syntax :

`INSTR(string_1, string_2)`

### Parameters :

This function accepts 2 parameters.

- **string\_1 –**

The string where searching takes place.

- **string\_2 –**

The string/substring which will be searched in string\_1.

**Returns :** It returns the position of the first occurrence of a substring within a given string.

Let us understand INSTR() by taking employee table below

emp_id	first_name	last_name	email	hire_date	salary	dept_id
1	kelly	davis	davis@gmail.com	2021-01-22	78000	80
2	tom	taylor	tom@gmail.com	2020-09-22	84200	70
3	mike	whalen	mike@gmail.com	2021-06-30	98200	50
4	andy	lumb	andy@gmail.com	2021-02-27	42200	80
5	anjel	nair	anj@gmail.com	2019-09-26	42200	40
6	ram	kumar	ram@gmail.com	2018-12-26	64200	40

7	rohan	sharma	ro@gmail.com	2021-02-09	84200	20	
8	john	king	jo__%\$@gmail.com	2021-02-09	124200	20	

1. Write a query to display the position of character ‘N’ in all firstname

```
SELECT
    first_name, INSTR(first_name, 'n')
FROM
    employee;
```

Output:

first_name	INSTR(first_name, 'n')
kelly	0
Tom	0
mike	0
andy	2
anjel	2
ram	0
rohan	5
john	4

As you can see from the above output if the substring that you are searching for is not present in the given string that time it will return you 0

2. Query the position of ‘easy’ in ‘mysqliseeasy’

```
SELECT INSTR( 'MYSQLISEASY' , 'EASY' );
```

Output:

INSTR('MYSQLISEASY', 'EASY')
8

3. Display the first name of all the employees that contains ‘n’ in it without using LIKE operator

```
SELECT  
    first_name  
FROM  
    employee  
WHERE  
    INSTR(first_name, 'n');
```

Output:

first_name
andy
anjel

rohan
john

4. Query the number of occurrences of character ‘N’ in the first name that contains ‘n’ in it

There is no function which will give the number of occurrences of n directly. To get that you can make use of length and replace function. Using length function get the length of the string, using replace replace all occurrences of n. Now subtract total length of the string with the string after removing n in it

```
SELECT
    first_name,
    LENGTH(first_name) - LENGTH(REPLACE(first_name, 'n',''))
as ln
FROM
    employee
WHERE
    INSTR(first_name, 'n');
```

### **Output:**

first_name	ln
andy	1

anjel	1
rohan	1
john	1

5. Query the number of occurrences of character ‘A’ in the first name that contains ‘A’ in it

```

SELECT
    first_name,
    LENGTH(first_name) - LENGTH(REPLACE(first_name, 'a',''))
as ln
FROM
    employee
WHERE INSTR(first_name, 'a');

```

**Output:**

first_name	ln
andy	1
anjel	1
ram	1
rohan	1

Here in this you can avoid cases by converting it to lowercase or uppercase as well

6. Query the first name of employee that contain at least two occurrences of 'l'

```
SELECT
    first_name
FROM
    employee
WHERE (LENGTH(first_name) - LENGTH(REPLACE(first_name,
    'l',''))) >= 2;
```

Output:

first_name
kelly

## **LOCATE(),REVERSE(),REPEAT() Function**

**LOCATE()** Function:

**LOCATE()** function in MySQL is used for finding the location of a substring in a string. It will return the location of the first occurrence of the substring in the string. If the substring is not present in the string then it will return 0. When searching for the location of a substring in a string it does not perform a case-sensitive search.

**Syntax :**

`LOCATE(substring, string, start)`

**Parameters :**

This method accepts three parameters.

- **substring –**  
The string whose position is to be retrieved.
- **string –**  
The string within which the position of the substring is to be retrieved.
- **start –**  
The starting position for the search. It is optional .Position 1 is default.

**Returns :**

The location of the first occurrence of the substring in the string.

1. Write a query to display the position of character ‘n’ in all firstname

```
SELECT  
LOCATE('n', first_name)  
FROM  
employee;
```

**Output:**

LOCATE('n', first_name)
0
0
0
2
2
0
5
4

Here we have not provided any starting position from which it needs to search so it will start from index 0

2. Write a query to display the position of character ‘n’ in all firstname from index 3

```
SELECT  
LOCATE('n', first_name,3)  
FROM  
employee;
```

**Output:**

LOCATE('n', first_name,3)
0
0
0
0
0
0
5
4

## REVERSE() FUNCTION:

This function could be used to reverse a string and find the result. The REVERSE() function takes the input parameter as a string and results in the reverse order of that string.

### Syntax:

REVERSE(string)

3. Query the first name of all employees in reverse order

```
SELECT  
REVERSE(first_name)  
FROM  
employee;
```

### Output:

REVERSE(first_name)
yllek
moT
ekim
ydna
lejna
mar
nahor
nhoj

4. Query the first name of all employees if it is not a palindrome

```
SELECT  
    first_name  
FROM  
    employee  
WHERE REVERSE(first_name) != first_name;
```

**Output:**

first_name
kelly
Tom
mike
andy
anjel
ram
rohan
john

**REPEAT() FUNCTION:**

This function in MySQL is used to repeat a string a specified number of times.

**Syntax :**

REPEAT(str, count)

**Parameters :**

This method accepts two parameters.

- **str** –Input String which we want to repeat.
- **count** –It will describe how many times to repeat the string.

**Returns :**

It returns a repeated string.

5. Query the first\_name of all employees twice

```
SELECT  
REPEAT(first_name, 2)  
FROM  
employee;
```

**Output:**

REPEAT(first_name,2)
kellykelly
TomTom
mikemike
andyandy

anjelanjal

ramram

rohanrohan

johnjohn

6. Write a query to display the following output:

\*\*\* \_ \*\*\*\*\* \_ \*\*\*\*\* \_ \*\*\*

```
SELECT REPEAT('*** _ ***', 3) as rp;
```

Output:

rp

\*\*\* \_ \*\*\*\*\* \_ \*\*\*\*\* \_ \*\*\*

## LEFT(),RIGHT(),LPAD(),RPAD(),SOUNDEX() FUNCTION

- Query the first name of all employees whose first name begins with vowel**

```

SELECT
    first_name
FROM
    employee
WHERE
SUBSTRING(first_name, 1, 1) IN ('a', 'e', 'i', 'o', 'u');

```

**Output:**

first_name
andy
anjel

Here we have used substring() to extract the first character from the string, instead of this we can make use of **LEFT()** to extract the first character. The syntax looks as shown below

**Syntax:**

LEFT(string, number\_of\_chars)

```

SELECT
    first_name
FROM
    employee
WHERE LEFT(first_name,1) IN ('a','e','i','o','u');

```

**1. Display 'MYSQL' from 'MYSQL IS EASY'**

```
SELECT  
LEFT('MYSQL IS EASY',5) as LT;
```

**Output:**

LT
MYSQL

**2. Query the first name of all the employees whose first name ends with vowel**

```
SELECT  
first_name  
FROM  
employee  
WHERE SUBSTRING(first_name, -1) IN ('a','e','i','o','u');
```

**Output:**

first_name
mike

Instead of making use of `substring()` to fetch the last characters in the given string you can make use of `RIGHT()`. The syntax of `RIGHT()` function is shown below

**Syntax:**

```
RIGHT(string, number_of_chars)
```

```
SELECT  
    first_name  
FROM  
    employee  
WHERE RIGHT(first_name, 1) IN ('a','e','i','o','u');
```

**3. Display 'EASY' from 'MYSQL IS EASY'**

```
SELECT  
RIGHT('MYSQL IS EASY', 4) as RT;
```

**Output:**

RT
EASY

**LPAD() Function:**

**LPAD()** function in MySQL is used to pad or add a string to the left side of the original string.

**Syntax:**

```
LPAD(str, len, padstr)
```

- **str** –

The actual string which is to be padded. If the length of the original string is larger than the len parameter, this function removes the overflowing characters from the string.

- **len** –

This is the length of a final string after the left padding.

- **padstr** –

String that to be added to the left side of the Original Str.

**Returns**: It returns a new string of length len after padding.

#### 4. Query the string 'MYSQL' by adding '\*\*\*' at the beginning of the string

```
SELECT  
LPAD('MYSQL', 7, '***') as LP;
```

**Output:**

LP
**MYSQL

Now let's see if the length is equal to or less than the given string

```
SELECT LPAD('MYSQL', 5, '***') as  
LP;
```

**Output:**

LP
MYSQL

As you can see from the above output here length of the string is same given string so there is no padding happens as shown above

SELECT LPAD('MYSQL', 4, '**') as LP;
--------------------------------------

**Output:**

LP
MYSQ

Here the length of the original string is larger than the len parameter in that case, string length will be reduced.

**5. Query the first name of all the employees by adding '\*' at the beginning with 15 as total length**

SELECT LPAD(first_name, 15, '*') as LP FROM employee;
--

**Output:**

LP
*****kelly
*****Tom
*****mike
*****andy
*****anjel
*****ram
*****rohan
*****john

As you can see from the above output for all first name in the employee table \* is added at the beginning till the length of the string becomes 15.

**RPAD() Function:**

**RPAD()** function in MySQL is used to pad or add a string to the right side of the original string.

**Syntax:**

RPAD(str, len, padstr)

- **str** –

The actual string which is to be padded. If the length of the original string is larger than the len parameter, this function removes the

overflowing characters from the string.

- **len** –

This is the length of a final string after the right padding.

- **padstr** –

String that to be added to the right side of the Original Str.

**Returns** : It returns a new string of length len after padding.

## 6. Query the first name of all the employees by adding '\*' at the ending with 15 as total length

```
SELECT
    RPAD(first_name, 15, '*') as RP
FROM
    employee;
```

**Output:**

RP
kelly*****
tom*****
mike*****
andy*****
anjel*****
ram*****

rohan\*\*\*\*\*

john\*\*\*\*\*

**7. Display the string 'MYSQL IS EASY' as \*\*\*MYSQL IS EASY\*\*\***

```
SELECT  
RPAD(LPAD('MYSQL IS EASY', 16, '*'), 19, '*') as padding;
```

**Output:****padding**

\*\*\*MYSQL IS EASY\*\*\*

Here left padding is done on the string and making a string of length 16, for that string right padding is done making the string length 19.

**SOUNDEX() Function:**

The SOUNDEX() function accepts a string and converts it to a four-character code based on how the string sounds when it is spoken.

The following shows the syntax of the SOUNDEX() function:

**SOUNDEX(input\_string);****8. Query to check 'DAMN' and 'DAM' sounds same**

```
SELECT SOUNDEX('DAMN'), SOUNDEX('DAM');
```

**Output:****SOUNDEX('DAMN')****SOUNDEX('DAM')**

D500	D500
------	------

As you can see from the above output, both string is returning the same value it clearly indicates that both strings sounds the same

### **9. Query the first name of all employees that are homophones of 'KELLIE'**

```

SELECT
    first_name
FROM
    employee
WHERE SOUNDEX(first_name)
=SOUNDEX('KELLIE');
```

#### **Output:**

first_name
kelly

As you can observe from the above it will match whether the first name sounds exactly the same as 'KELLIE'. Here it matches with one string kelly which sounds exactly same as 'KELLIE'

#### **HACKERANK QUERIES**

##### **1. Weather Observation station 12**

**Link:**

<https://www.hackerrank.com/challenges/weather-observation-station-12/problem?isFullScreen=true>

Query the list of CITY names from STATION that do not start with vowels and

do not end with vowels. Your result cannot contain duplicates.

## Input Format

The STATION table is described as follows:

STATION	
Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT\_N is the northern latitude and LONG\_W is the western longitude.

## Solution:

```

SELECT
    DISTINCT(CITY)
FROM
    STATION
WHERE
    (LEFT(CITY,1) NOT IN ('a', 'e', 'i', 'o', 'u'))
    AND
    RIGHT(CITY,1) NOT IN ('a', 'e', 'i', 'o', 'u'))

```

## 2. Weather Observation Station 7

### Link:

<https://www.hackerrank.com/challenges/weather-observation-station-7/problem>

Query the list of CITY names ending with vowels (a, e, i, o, u) from STATION.

Your result cannot contain duplicates.

## Input Format

The STATION table is described as follows:

**STATION**

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT\_N is the northern latitude and LONG\_W is the western longitude.

## Solution:

```

SELECT
    DISTINCT(CITY)
FROM
    STATION
WHERE (RIGHT(CITY,1) IN ('a', 'e', 'i',
'o', 'u'))

```

## Numeric Functions

### ROUND() Function:

The ROUND() function rounds a number to a specified number of decimal places.

#### Syntax:

ROUND(number, decimals)

1. Write a query to round off the number 56.3436 at second decimal place

```
SELECT ROUND(56.3436,2);
```

#### Output:

<b>ROUND(56.3436)</b>
56.34

If you observe from the above output the value is been rounded off to two decimal places

If you don't specify the number of decimal values that time by default it will be considered as 0.

Now let's understand how the precision works for negative numbers

round(45.65, 1) gives result = 45.7

round(45.65, -1) gives result = 50

because the precision in this case is calculated from the decimal point. If positive then it'll consider the right side number and round it upwards if it's  $\geq 5$ , and if  $\leq 4$  then round is downwards. and similarly if it's negative then the precision is calculated for the left hand side of the decimal point. If it's  $\geq 5$

For example, round(44.65, -1) gives 40 but round(45.65, -1) gives 50... (Here it is checking with left hand side)

2. Predict the output ROUND(563.3436, -1)

```
SELECT ROUND(563.3436, -1);
```

**Output:**

<b>ROUND(563.3436)</b>
560

As you can see the left side value is less than 4, precision is decremented

3. Predict the output ROUND(-16.56,-1)

```
SELECT ROUND(-16.56, -1);
```

**Output:**

<b>ROUND(-16.56,-1)</b>
-20

**TRUNCATE() Function:**

The TRUNCATE() function truncates a number to the specified number of decimal places.

**Syntax:****TRUNCATE(number, decimals)**

4. Write a query to truncate 456.432 to 2nd decimal places

```
SELECT TRUNCATE(456.432,2);
```

**Output:**

<b>TRUNCATE(456.432,2)</b>
456.43

5. Guess the output of TRUNCATE(456.556, -1)

```
SELECT TRUNCATE(456.556, -1);
```

**Output:**

<b>TRUNCATE(456.556,-1)</b>
450

### MOD Function:

The MOD() function returns the remainder of a number divided by another number.

### Syntax:

MOD(x, y)

1. Write a query to display remainder of a number 241 when divided by 2

```
SELECT MOD(241, 2);
```

### Output:

MOD(241, 2)
1

2. Write a query to display the details of even number of rows in employee table

```
SELECT
    emp_id, first_name
FROM
    employee
WHERE
    MOD(emp_id, 2) = 0; //here we are checking whether the
    number perfectly divided by 2
```

### Output:

emp_id	first_name
2	Tom
4	andy
6	ram

8	john
---	------

3. Write a query to display the details of odd number of rows in employee table

```
SELECT
    emp_id, first_name
FROM
    employee
WHERE
    NOT MOD(emp_id, 2) = 0;
```

**Output:**

emp_id	first_name
1	kelly
3	mike
5	anjel
7	rohan

**CEIL() Function:**

CEIL() function is used to get the smallest integer which is greater than, or equal to, the specified numeric expression.

**Syntax:**

CEIL(number)

1. Write a query to get the ceil of 2.88

```
SELECT CEIL(2.88);
```

**Output:**

CEIL(2.88)
3

Here it is rounded off to the next greater number which is 3 here.

2. Write a query to get the ceil of 2

```
SELECT CEIL(2);
```

**Output:**

CEIL(2)
2

3. Write a query to get the ceil of -2.88

```
SELECT CEIL(-2.88);
```

**Output:**

CEIL(-2.88)
2

4. Query the salary of employees by incrementing it to 23.33% in integer format

```
SELECT  
    CEIL(salary + salary * 23.33/100) as ceil  
FROM  
    employee;
```

**Output:**

ceil
96198
103844
121111
52046
52046
79178
103844
153176

**FLOOR Function:**

The FLOOR() function returns the largest integer value that is smaller than or equal to a number.

**Syntax:**

`FLOOR(number)`

1. Write a query to get the floor of 2.33

```
SELECT FLOOR(2.33);
```

**Output:**

<b>FLOOR(2.33)</b>
2

2. Write a query to get the floor of -2.33

```
SELECT FLOOR(-2.33);
```

**Output:**

<b>FLOOR(2.33)</b>
-3

**POWER() Function:**

The POWER() function returns the value of a number raised to the power of another number.

**Syntax:**

POWER(a, b)

1. Write a query to find cube of a number 3

```
SELECT POWER(3,3);
```

**Output:**

<b>POWER(3,3)</b>
27

**SQRT() Function:**

The SQRT() function returns the square root of a number.

**Syntax:**

SQRT(number)

1. Write a query to to find the square root of a number 16

```
SELECT SQRT(16);
```

**Output:**

<b>SQRT(16)</b>
4

## Date and Time Function

### CURDATE():

This function in Mysql is used to return the current date.

The date is returned to the format of “YYYY-MM-DD” (string) or as YYYYMMDD (numeric).

This function equals the CURRENT\_DATE() function.

### Syntax:

CURDATE()

1. Write a query to display the current date

```
SELECT CURDATE();
```

### Output:

CURDATE()
2022-04-21

**NOW():**

The NOW() function returns the current date and time.

**Syntax:**

NOW()

1. Write a query to display the current date and time

```
SELECT NOW();
```

**Output:**

<b>NOW()</b>
2022-04-21 17:11:44

2022-04-21 17:11:44
---------------------

**DAYNAME() Function:**

This function in Mysql is used to return the weekday name for a specified date.

**Syntax:**

DAYNAME(date)

1. Display the week day of today's date

```
SELECT DAYNAME('2021-04-21');
```

Or

```
SELECT DAYNAME(NOW());
```

**Output:**

<b>DAYNAME('2021-04-21')</b>
Wednesday

Wednesday
-----------

**MONTH() Function:**

The MONTH() function returns the month part for a specified date (a number from 1 to 12).

**Syntax:**

MONTH(date)

1. Write a query to display the current month

```
SELECT MONTH(NOW());
```

**Output:**

<b>MONTH(NOW())</b>
4

**YEAR() Function:**

The YEAR() function returns an integer value which represents the year of the specified date.

**Syntax:**

YEAR(input\_date)

1. Write a query to display the working experience of an employee in terms of years

```
SELECT  
    first_name, 2022 - YEAR(hire_date) as exp  
FROM  
    employee;
```

**Output:**

first_name	exp
kelly	1
Tom	2
mike	1
andy	1
anjel	3
ram	4
rohan	1
john	1

2. Write a query to display the details of employee who have at least 2year experience

```

SELECT
    first_name, 2022 - YEAR(hire_date) as exp
FROM
    employee
WHERE
    2022 - YEAR(hire_date) >= 2;

```

**Output:**

first_name	exp

Tom	2
anjel	3
ram	4

### **DAY() Function:**

The DAY() function returns the day of the month (from 1 to 31) for a specified date.

Syntax:

DAY(date)

3. Write a query to extract day from hired date

```
SELECT
    DAY(hire_date)
FROM
    employee;
```

**Output:**

DAY(hire_date)
22
22
30
27
26
26

9

9

The same can be achieved using **EXTRACT()** Function

### **EXTRACT() Function:**

This function in MySQL is used to extract a part from a specified date.

#### **Syntax:**

`EXTRACT(part FROM date)`

#### **Parameter :**

This method accepts two parameters which are illustrated below:

**part** – Specified part to extract like SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, YEAR, etc

**Date** – Specified date to extract a part from

1. Write a query to extract day from hired date

```
SELECT  
    EXTRACT(DAY FROM hire_date)  
FROM  
    employee;
```

#### **Output:**

<b>EXTRACT(DAY FROM hire_date)</b>
22
22

30

27

26

26

9

9

2. Write a query to extract Year from hired date

```
SELECT  
    EXTRACT(YEAR FROM hire_date)  
FROM  
    employee;
```

**Output:****EXTRACT(YEAR FROM hire\_date)**

2021

2020

2021

2021

2019

2018

2021

2021

3. Write a query to extract year, month, day using EXTRACT()

```
SELECT  
EXTRACT(YEAR FROM hire_date) as YEAR,  
EXTRACT(MONTH FROM hire_date) as MONTH,  
EXTRACT(DAY FROM hire_date) as DAY  
FROM  
employee;
```

**Output:**

YEAR	MONTH	DAY
2021	1	22
2020	9	22
2021	6	30
2021	2	27
2019	9	26
2018	12	26
2021	2	9

2021	2	9
------	---	---

### **LAST\_DAY() Function:**

The LAST\_DAY() function extracts the last day of the month for a given date.

#### **Syntax:**

LAST\_DAY(date)

1. Write a query to display the last day of the current month

```
SELECT LAST_DAY(NOW()) ;
```

Output:

```
LAST_DAY(NOW())
```

```
2022-04-30
```

Here you are getting 30 as the output it is because in the april month we have 30days and that is the last day

### **DATEDIFF() Function:**

The DATEDIFF() function compares two dates and returns the difference. The DATEDIFF() function is specifically used to measure the difference between two dates in years, months, weeks, and so on.

#### **Syntax:**

DATEDIFF(date\_part,start\_date\_value1, end\_date\_value2);

1. Write a query to display the number of days between 14th february 2020 to 15th august 2020

```
SELECT DATEDIFF("2020-08-15" , "2020-02-14") as DF;
```

**Output:**

DF
183

As you can see there is 183 days difference between 14th february to 15th august

2. Write a query to display the number of days employee has worked for the company

```
SELECT  
    DATEDIFF(NOW(), hire_date) as DD  
FROM  
    employee;
```

**Output:**

DD
456
578
297
420

940

1214

438

438

**DATE\_ADD() Function:**

The DATE\_ADD() function adds a time/date interval to a date and then returns the date.

**Syntax:**

DATE\_ADD(date, INTERVAL value unit)

**Parameters:**

**date:** It is used to specify the date to which the interval should be added.

**value:** It is used to specify the value of the time/date interval to be added.

**unit:** It is used to specify the unit type of the interval such as DAY, MONTH, MINUTE, etc.

1. Write a query to display the date 2020-12-31 by adding 1 day to it

```
SELECT  
DATE_ADD('2020-12-31', INTERVAL 1 DAY) as DA;
```

**Output:**

DA

2021-01-01
------------

As you can see from the above output the desired date interval has been added to the start value of the date. we have given last day of 2020 for that when you add 1 day to that we have got 1st january 2021

2. Now let's see by adding year interval to the existing date

```
SELECT
DATE_ADD('2020-12-31', INTERVAL 2 YEAR) as DA;
```

**Output:**

DA
2022-12-31

When two years added to the existing date '2020-12-31' then you will get '2022-12-31'

3. Adding 10 seconds to the existing time

```
SELECT
DATE_ADD('2020-12-31 10:00:00', INTERVAL 10 SECOND) as
DA;
```

**Output:**

DA
2020-12-31 10:00:10

The output has come as 10 AM 10 seconds, of 31th december 2020.

4. Add 1 minute and 1 second to 2020-12-31 10:00:00

```
SELECT
DATE_ADD('2020-12-31 10:00:00', INTERVAL '1:1'
MINUTE_SECOND)
as DA;
```

**Output:**

DA
2020-12-31 10:01:01

5. Add -1 day and 5 hours to 2020-12-31 10:00:00

```
SELECT  
DATE_ADD('2020-12-31 10:00:00', INTERVAL '-1 5'  
DAY_HOUR)  
as DA;
```

**Output:**

DA
2020-12-30 05:00:00

6. Write a query to display the date by removing 2018-12-31 by removing 1 day from it

```
SELECT  
DATE_SUB('2018-12-31', INTERVAL 1 DAY) as DA;
```

**Output:**

DA
2018-12-30

## Control Flow Function

The control flow functions allow you to add if-then-else logic to SQL queries without using the procedural code. The following show the most commonly used MySQL control flows functions:

- **CASE** – return the corresponding result in THEN branch if the condition in the WHEN branch is satisfied, otherwise, return the result in the ELSE branch.

**Syntax:**

```
CASE value
    WHEN [compare_value] THEN result
        [WHEN [compare_value] THEN result ...]
        [ELSE result]
END CASE
```

**Or**

```
CASE
    WHEN expression1 THEN result
        [WHEN expression2 THEN result ...]
        [ELSE result]
END CASE
```

- **IF** – return a value based on a given condition.

**Syntax:**

IF(condition, value\_if\_true, value\_if\_false)

- **IFNULL**– return the first argument if it is not NULL , otherwise returns the second argument

**Syntax:**

IFNULL(*expression*, *alt\_value*)

- **NULLIF**— return NULL if the first argument is equal to the second argument, otherwise, returns the first argument.

### Syntax:

`NULLIF(expr1, expr2)`

1. Write a query to display first name and departments of all employees

```
SELECT first_name,
CASE dept_id
WHEN 20 THEN 'SALES'
WHEN 30 THEN 'EXE'
WHEN 40 THEN 'SHIP'
WHEN 50 THEN 'MKT'
WHEN 70 THEN 'ACC'
ELSE 'I_T'
END as DEPT
FROM employee;
```

### Output:

first_name	DEPT
kelly	I_T
tom	ACC
mike	MKT
andy	I_T
anjel	SHIP
ram	SHIP
rohan	SALES

john	SALES
------	-------

Here you have fetched the values by specifying the value, let's see the same example using expression

```
SELECT first_name,
CASE
WHEN dept_id = 20 THEN 'SALES'
WHEN dept_id IN (30,40) THEN 'E/S'
WHEN dept_id > 40 THEN 'M/A/I'
ELSE 'N/A'
END as DEPT
FROM employee;
```

### Output:

first_name	DEPT
kelly	M/A/I
tom	M/A/I
mike	M/A/I
andy	M/A/I
anjel	E/S
ram	E/S
rohan	SALES
john	SALES

2. Write a query to display first name and salary and salary as high salary if

the salary is greater than 80000

```
SELECT first_name,
CASE
WHEN salary > 80000 THEN 'HIGH_SALARY'
ELSE 'LOW_SALARY'
END as SAL
FROM sql_notes.employee;
```

Let's see the same example IF()

```
SELECT
    first_name, IF(salary > 80000, 'HIGH_SALARY',
'LOW_SALARY') as SAL
FROM
    employee;
```

**Output:**

first_name	SAL
kelly	LOW_SALARY
tom	HIGH_SALARY
mike	HIGH_SALARY
andy	LOW_SALARY
anjel	LOW_SALARY
ram	LOW_SALARY
rohan	HIGH_SALARY
john	HIGH_SALARY

2. Write a query to display as new employee for those who hired in the year 2021 else old employee

```
SELECT
    first_name, IF(YEAR(hire_date) = '2021',
'NEW_EMPLOYEE', 'OLD_EMPLOYEE') as emp
FROM
    employee;
```

**Output:**

first_name	emp
kelly	NEW_EMPLOYEE
tom	OLD_EMPLOYEE
mike	NEW_EMPLOYEE
andy	NEW_EMPLOYEE
anjel	OLD_EMPLOYEE
ram	OLD_EMPLOYEE
rohan	NEW_EMPLOYEE
john	NEW_EMPLOYEE

3. Write a query to display null values in email id column as ‘no value entered’

```
SELECT
    IF(EMAIL IS NOT NULL, email, 'No Values Entered') as
email
```

```
FROM  
employee;
```

**Output:**

email
davis@gmail.com
tom@gmail.com
mike@gmail.com
andy@gmail.com
anj@gmail.com
ram@gmail.com
ro@gmail.com
No Values Entered

Now let's use IFNULL() to achieve the same

```
SELECT  
    IFNULL(email, 'NO Values Entered') as email  
FROM  
    employee;
```

4. Query the email id's of all the employees with john's email id as null

```
SELECT NULLIF(email, 'ro@gmail.com') as email FROM  
employee;
```

**Output:**

email
-------

davis@gmail.com

tom@gmail.com

mike@gmail.com

andy@gmail.com

anj@gmail.com

ram@gmail.com

null

null

Here if you observe from the output now mail of [ro@gmail.com](mailto:ro@gmail.com) is updated with null.

## Conversion Function

In some cases, the Server uses data of one type where it expects data of a different data type. This can happen when the Server can automatically convert the data to the expected data type. This data type conversion can be done implicitly by the Server, or explicitly by the user.

1. Write a query to convert integer value 150 to character type data

This can be achieved using **CONVERT()**. Syntax of **CONVERT()** as shown below

Syntax:

```
CONVERT(value, type)
```

```
SELECT CONVERT(150, CHAR);
```

**Output:**

```
CONVERT(150, CHAR)
```

```
150
```

2. Query the year from hire date in character format

Here year is present in first 4 character in hire date, to fetch that make use of **CHAR()** and pass number of characters to be fetched inside that

```
SELECT  
    CONVERT(hire_date, CHAR(4))  
FROM  
    employee;
```

**Output:**

```
CONVERT(hire_date, CHAR(4))
```

```
2021
```

2020
2021
2021
2019
2018
2021
2021

3. Query the given string value ‘12-12-12’ into time format

```
SELECT
CONVERT('12-12-12', TIME) as time;
```

Output:

time
00:00:12

Here it will consider first two values and will be treated as seconds

4. Convert the given string ‘14:12:14’ into date format

```
SELECT CONVERT('14-12-14', DATE) as date;
```

Output:

date
2014-12-14

5. Convert the given string ‘70:12:12’ into date format

```
SELECT CONVERT( '70-12-12' , DATE) as date;
```

Output:

date
1970-12-12

Here the year is 70. That is the reason it is considered as 19's means above 70 will be considered as 19's below 70 will be considered as 20's.

6. Convert the given string '14:12:12' into datetime format

```
SELECT CONVERT( '14:12:14' , DATETIME) as date_time;
```

Output:

date_time
2014-12-14 00:00:00

7. Extract time from '12:12:12 21:03:06'

```
SELECT CONVERT( '12:12:12 21:03:06' , TIME) as time;
```

Output:

time
21:03:06

### CAST() Function:

The CAST() function converts a value (of any type) into a specified datatype.

#### Syntax:

```
CAST(expression AS datatype(length))
```

```
SELECT CAST(150 AS CHAR);
```

**Output:****CAST(150 AS CHAR)**

150

As you can see from the output here 150 is converted to character using cast().

## Aggregate Function

In database management an aggregate function is a function where the values of multiple rows are grouped together as input on certain criteria to form a single value of more significant meaning.

Various aggregate functions are

- **AVG()** – returns the average of a set.
- **COUNT()** – returns the number of items in a set.
- **MAX()** – returns the maximum value in a set.
- **MIN()** – returns the minimum value in a set
- **SUM()** – returns the sum of all or distinct values in a set

1. Query the highest salary from the employee table

For this let's sort the salary in descending order using order by clause and then access the first value using limit keyword.

```
SELECT
    salary
FROM
    employee
ORDER BY
    salary
DESC LIMIT 1;
```

Output:

salary
124200

The same can be achieved using one function MAX(). The syntax of that is as shown below

**Syntax:**

```
SELECT MAX(expression)
```

```
FROM table_name
```

```
[WHERE restriction];
```

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

**Output:**

MAX(salary)
124200

2. Write a query to get highest salary paid to the employee by adding employee id to the salary

```
SELECT  
    MAX(salary + emp_id)  
FROM  
    employee;
```

**Output:**

MAX(salary)
124208

### 3. Query the lowest salary from the employee table

You can get the lowest salary using an order by clause. Instead of that you can make use of MIN() Function

```
SELECT MIN(salary) FROM sql_notes.employee;
```

#### Output:

<b>MIN(salary)</b>
42200

MAX() and MIN() function works along with DISTINCT or ALL.

If in case you want maximum or minimum salary of only distinct data that time you can make use of distinct keyword

If in case you want maximum or minimum salary of by considering all salary that time you can make use of all keyword

### 4. Write a query to display total salary paid by the company to the employees

This can be achieved using SUM() in sql. Syntax of SUM() is shown below

```
SUM(expression)
```

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

#### Output:

<b>SUM(salary)</b>
617400

5. Write a query to display sum of distinct salary paid to the employees

```
SELECT  
    SUM(DISTINCT salary)  
FROM  
    employee;
```

**Output:**

<b>SUM(DISTINCT salary)</b>
491000

6. Write a query to display the number of employees working in a company

This can be achieved using COUNT().

**COUNT():**

The COUNT() function returns the number of records returned by a select query.

Syntax:

COUNT(expression)

```
SELECT  
    COUNT(emp_id)  
FROM  
    employee;
```

**Output:**

<b>COUNT(emp_id)</b>
8

7. Write a query to display the count of non duplicate salary from employee table

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

**Output:**

<b>COUNT(DISTINCT emp_id)</b>
8

8. Query the number of employees working in sales department  
Dept\_id of sales is 2, let's use the department id and fetch the number

```
SELECT  
    COUNT(emp_id)  
FROM  
employee  
WHERE  
dept_id = 20;
```

**Output:**

<b>COUNT(emp_id)</b>
2

9. Query the average salary from the employee table

Here we can make use of AVG() Function. Syntax of AVG() Function is shown below

**AVG(expression)**

```
SELECT
    AVG(salary)
FROM
    employee;
```

**Output:**

<b>AVG(salary)</b>
77175.0000

10. Display the number of employees and maximum salary paid to sales department

```
SELECT
    COUNT(dept_id), MAX(salary)
FROM
    employee
WHERE
    dept_id = 20;
```

**Output:**

COUNT(dept_id)	MAX(salary)
2	124200

11. Display highest and lowest salary paid to sales department

```
SELECT  
    MAX(salary), MIN(salary)  
FROM  
    employee  
WHERE  
    dept_id = 20;
```

**Output:**

MAX(salary)	MIN(salary)
124200	84200

## Group By Clause

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

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

Syntax:

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

1. Write a query to display the dept\_id and average salary of all employees in each department

```
SELECT
    dept_id, AVG(salary)
FROM
    employee
GROUP BY dept_id;
```

**Output:**

dept_id	AVG(salary)
20	104200.0000
40	53200.0000
50	98200.0000

70	84200.0000
80	60100.0000

2. Query the different department in the employee table

```
SELECT
    dept_id
FROM
    employee
GROUP BY
    dept_id;
```

Output:

dept_id
20
40
50
70
80

3. Write a query to retrieve the dept\_id, salary of employees having different salaries in each department

```
SELECT
    dept_id, salary
FROM
    employee
GROUP BY
    dept_id, salary;
```

**Output:**

dept_id	salary
80	78000
70	84200
50	98200
80	42200
40	42200
40	64200
20	84200
20	124200

Here unique value with the combination of dept\_id and salary is retrieved.

4. Query the number of employees in each department

```
SELECT
    count(*)
FROM
    employee
GROUP BY
    dept_id;
```

**Output:**

count(*)
2
2

1
1
2

5. Write a query to display department id and least salary of all employees in each department

```
SELECT
    dept_id, min(salary)
FROM
    employee
GROUP BY
    dept_id;
```

**Output:**

dept_id	min(salary)
20	84200
40	42200
50	98200
70	84200
80	42200

6. Query the total salary paid to the employees of each department whose id is greater than 20

```

SELECT
    SUM(salary)
FROM
    employee
WHERE
    dept_id > 20
GROUP BY
    dept_id ;

```

**Output:**

SUM(salary)
106400
98200
84200
120200

7. Query the number of employees, min salary of all employees in each department except sales department  
Dept\_id of sales is 20.

```

SELECT
    COUNT(*), MIN(salary)
FROM
    employee
WHERE
    dept_id != 20
GROUP BY
    dept_id ;

```

**Output:**

COUNT(*)	salary

2	42200
1	98200
1	84200
2	42200

8. Query the number of employees hired in each year

```
SELECT
    COUNT(*), YEAR(hire_date)
FROM
    employee
GROUP BY
    YEAR(hire_date);
```

**Output:**

COUNT(*)	YEAR(hire_date)
5	2021
1	2020
1	2019
1	2018

9. Display the dept\_id, number of employees in each department in descending order

```
SELECT
    dept_id, COUNT(emp_id) as c
FROM
    employee
GROUP BY
```

```
dept_id  
ORDER by c DESC ;
```

**Output:**

dept_id	c
20	2
40	2
80	2
50	1
70	1

As you can see from the above output count of emp\_id of each department is in decreased order

## QUERIES ON GROUP BY CLAUSE AND HAVING CLAUSE

1. Query the dept\_id and number of employees of the department having highest number of employees
  - Here you can retrieve dept\_id, number of employees of the department using group by clause
  - Here you also need the highest number of employees so you will be using order by clause along with group by.
  - In that it is required to fetch only 1 data which is highest from the table for that we need to make use of limit keyword

```
SELECT
    dept_id, COUNT(emp_id) as cn
FROM
    employee
GROUP BY
    dept_id
ORDER by cn DESC
LIMIT 1;
```

### Output:

dept_id	cn
20	2

2. Write a query to display the department names, maximum salary, average salary paid to employees in each department in alphabetical order
  - Here let's make use of cases to get department names
  - Then MAX(), AVG() Function to get maximum and average salary of employee and grouped by department
  - Next retrieve in alphabetical order using order by clause

```
SELECT
CASE dept_id
WHEN 20 THEN 'Sales'
```

```

WHEN 30 THEN 'EXE'
WHEN 40 THEN 'SHP'
WHEN 50 THEN 'MRK'
ELSE 'IT'
END as dept,
MAX(salary), AVG(salary)
FROM employee GROUP BY dept_id
ORDER BY dept;

```

**Output:**

dept	MAX(salary)	AVG(salary)
IT	84200	84200.0000
IT	78000	60100.0000
MRK	98200	98200.0000
Sales	124200	104200.0000
SHP	64200	53200.0000

As you can see from the above table dept column is sorted in alphabetical order

3. Query the total salary, average salary of all employees with average salary greater than 60000 in each department

```

SELECT
    SUM(salary), AVG(salary) as av
FROM
    employee
WHERE
    av > 60000
GROUP BY
    dept_id;

```

## Output:

```
0 3 12:06: SELECT SUM(salary),
35      AVG(salary) as av FROM
sql_notes.employee WHERE av >
60000 GROUP BY dept_id LIMIT
0, 1000
```

Error Code: 0.00  
**1054. Unknown column 'av' in 'where clause'**

Here you are getting error it is because where clause cannot be used with aggregate functions so to work with aggregate function we can make use of **HAVING CLAUSE**

## HAVING CLAUSE:

The HAVING Clause enables you to specify conditions that filter which group results appear in the results.

The WHERE clause places conditions on the selected columns, whereas the HAVING clause places conditions on groups created by the GROUP BY clause.

### Syntax:

```
SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY
```

```
SELECT
    SUM(salary), AVG(salary) as av
FROM
    employee
GROUP BY
    dept_id
HAVING
    av > 60000;
```

**Output:**

SUM(salary)	av
208400	104200.0000
98200	98200.0000
84200	84200.0000
120200	60100.0000

4. Query the dept\_id, total salary, average salary of all employees with average salary greater than 60000 in each department and order by dept\_id

```

SELECT
    dept_id, SUM(salary), AVG(salary) as av
FROM
    employee
GROUP BY
    dept_id
HAVING
    AVG(salary) > 60000
ORDER BY
    dept_id;

```

**Output:**

dept_d	SUM(salary)	av
20	208400	104200.0000
50	98200	98200.0000

70	84200	84200.0000
80	120200	60100.0000

5. Query the average salary of all employees in each department having at least two employees in each department and average salary in the range of 50000 to 60000 in descending order of average salary

```

SELECT
    dept_id, SUM(salary), AVG(salary) as av
FROM
    employee
GROUP BY
    dept_id
HAVING
    COUNT(emp_id) >1 AND
    AVG(salary) BETWEEN 50000 AND 60000
ORDER BY
    AVG(salary) DESC;

```

**Output:**

dept_id	SUM(salary)	av
40	106400	53200.0000

6. Write a query to display dept id, average salary of all employees in each department with dept\_id > 20, having average salary > 40000 and sort in descending order with respect to dept id

```

SELECT
    dept_id, AVG(salary) as av
FROM
    employee
GROUP BY
    dept_id
HAVING
    dept_id > 20

```

```
AND  
    AVG(salary) > 40000  
ORDER BY  
    dept_id DESC;
```

**Output:**

dept_id	av
80	60100.0000
70	84200.0000
50	98200.0000
40	53200.0000

As you can see from the above output dept\_id is sorted in descending order

## HACKERRANK Queries

### 1. Employee Salaries

**Link:** <https://www.hackerrank.com/challenges/salary-of-employees/problem>

Write a query that prints a list of employee names (i.e.: the name attribute) for employees in Employee having a salary greater than \$2000 per month who have been employees for less than 10 months. Sort your result by ascending employee\_id.

#### Input Format

The Employee table containing employee data for a company is described as follows:

Column	Type
employee_id	Integer
name	String
months	Integer
salary	Integer

where employee\_id is an employee's ID number, name is their name, months is the total number of months they've been working for the company, and salary is their monthly salary.

#### Sample Input

employee_id	name	months	salary
12228	Rose	15	1968
33645	Angela	1	3443
45692	Frank	17	1608
56118	Patrick	7	1345
59725	Lisa	11	2330
74197	Kimberly	16	4372
78454	Bonnie	8	1771
83565	Michael	6	2017
98607	Todd	5	3396
99989	Joe	9	3573

## Sample Output

Angela  
Michael  
Todd  
Joe

## Explanation

Angela has been an employee for 1 month and earns \$3443 per month.

Michael has been an employee for 6 months and earns \$2017 per month.

Todd has been an employee for 5 months and earns \$3396 per month.

Joe has been an employee for 9 months and earns \$3573 per month.

We order our output by ascending employee\_id.

## Solution:

```
SELECT
    name
FROM
    EMPLOYEE
WHERE
    salary > 2000
and
    months < 10
ORDER BY
    employee_id;
```

## 2. Higher Than 75 Marks

Link:

<https://www.hackerrank.com/challenges/more-than-75-marks/problem>

Query the Name of any student in STUDENTS who scored higher than 75 Marks. Order your output by the last three characters of each name. If two or more students both have names ending in the same last three characters (i.e.: Bobby, Robby, etc.), secondary sort them by ascending ID.

## Input Format

The STUDENTS table is described as follows:

Column	Type
<i>ID</i>	<i>Integer</i>
<i>Name</i>	<i>String</i>
<i>Marks</i>	<i>Integer</i>

The Name column only contains uppercase (A-Z) and lowercase (a-z) letters.

## Sample Input

<i>ID</i>	<i>Name</i>	<i>Marks</i>
1	Ashley	81
2	Samantha	75
4	Julia	76
3	Belvet	84

## Sample Output

Ashley

Julia

Belvet

## Explanation

Only Ashley, Julia, and Belvet have Marks > 75. If you look at the last three characters of each of their names, there are no duplicates and 'ley' < 'lia' < 'vet'.

### Solution:

```

SELECT
    name
FROM
    Students
WHERE
    Marks > 75
ORDER BY
    RIGHT(name, 3), ID ASC;

```

Using where clause you can retrieve data where marks of students is greater than 75.

By making use of order by clause you can sort the data in ascending order, here they have asked to sort using the last 3 characters of name and id.

### 3. Weather Observation Station 5

**Link:**

<https://www.hackerrank.com/challenges/weather-observation-station-5/problem>

Query the two cities in STATION with the shortest and longest CITY names, as well as their respective lengths (i.e.: number of characters in the name). If there is more than one smallest or largest city, choose the one that comes first when ordered alphabetically.

The STATION table is described as follows:

**STATION**

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT\_N is the northern latitude and LONG\_W is the western longitude.

### Sample Input

For example, CITY has four entries: DEF, ABC, PQRS and WXY.

### Sample Output

ABC 3

PQRS 4

### Explanation

When ordered alphabetically, the CITY names are listed as ABC, DEF, PQRS, and WXY, with lengths 3,3,4 and 3. The longest name is PQRS, but there are 3 options for the shortest named city. Choose ABC, because it comes first alphabetically.

### Note

You can write two separate queries to get the desired output. It need not be a single query.

### Solution:

```
SELECT city, LENGTH(city) FROM STATION  
ORDER BY LENGTH(city), city LIMIT 1;  
  
SELECT city, LENGTH(city) FROM STATION  
ORDER BY LENGTH(city) DESC, city DESC LIMIT 1;
```

First we need to print the shortest city name and its length for that sort of cities and length(this you can get using length()) in ascending order and using limit to retrieve first data.

Next we need to print the largest city name and its length for that sort of cities and length(this you can get using length()) in Descending order and using limit to retrieve first data.

## 4. Population Density Difference

Link:

[https://www.hackerrank.com/challenges/population-density-difference/problem?h\\_r=internal-search](https://www.hackerrank.com/challenges/population-density-difference/problem?h_r=internal-search)

Query the difference between the maximum and minimum populations in CITY.

**Input Format**

The CITY table is described as follows:

**CITY**

Field	Type
ID	NUMBER
NAME	VARCHAR2(17)
COUNTRYCODE	VARCHAR2(3)
DISTRICT	VARCHAR2(20)
POPULATION	NUMBER

Solution:

```
SELECT MAX(POPULATION) - MIN(POPULATION) FROM CITY;
```

## 5. Average Population

Link:

<https://www.hackerrank.com/challenges/average-population/problem?isFullScreen=true>

Query the average population for all cities in CITY, rounded down to the nearest integer.

**Input Format**

The CITY table is described as follows:

**CITY**

Field	Type
ID	NUMBER
NAME	VARCHAR2(17)
COUNTRYCODE	VARCHAR2(3)
DISTRICT	VARCHAR2(20)
POPULATION	NUMBER

Solution:

```
SELECT  
    ROUND(AVG(POPULATION))  
FROM  
    CITY
```

## 6. Weather Observation Station 13

Link:

<https://www.hackerrank.com/challenges/weather-observation-station-13/problem>

Query the sum of Northern Latitudes (LAT\_N) from STATION having values greater than 38.7880 and less than 137.2345. Truncate your answer to 4 decimal places.

### Input Format

The STATION table is described as follows:

STATION	
Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT\_N is the northern latitude and LONG\_W is the western longitude.

### Solution:

```
SELECT
    TRUNCATE(SUM(LAT_N),4)
FROM
    STATION
WHERE
    LAT_N > 38.7880 AND LAT_N < 137.2345;
```

## 7. Weather Observation Station 14

Link:

<https://www.hackerrank.com/challenges/weather-observation-station-14/problem>

Query the greatest value of the Northern Latitudes (LAT\_N) from STATION that is less than 137.2345. Truncate your answer to 4 decimal places.

**Input Format**

The STATION table is described as follows:

STATION	
Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT\_N is the northern latitude and LONG\_W is the western longitude.

**Solution:**

```
SELECT
    TRUNCATE(MAX(LAT_N),4)
FROM
    STATION
WHERE
    LAT_N < 137.2345;
```

## 8. Weather Observation Station 16

Link:

<https://www.hackerrank.com/challenges/weather-observation-station-16/problem>

Query the smallest Northern Latitude (LAT\_N) from STATION that is greater than 38.7780. Round your answer to 4 decimal places.

### Input Format

The STATION table is described as follows:

**STATION**

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT\_N is the northern latitude and LONG\_W is the western longitude.

### Solution:

```
SELECT
    ROUND(MIN(LAT_N),4)
FROM
    STATION
WHERE
    LAT_N > 38.7780;
```

## 9. Weather Observation Station 16

Link:

<https://www.hackerrank.com/challenges/weather-observation-station-17/problem>

Query the Western Longitude (LONG\_W) where the smallest Northern Latitude (LAT\_N) in STATION is greater than . Round your answer to decimal places.

### Input Format

The STATION table is described as follows:

**STATION**

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT\_N is the northern latitude and LONG\_W is the western longitude.

### Solution:

```
SELECT
    ROUND(LONG_W, 4)
FROM
    STATION
WHERE
    LAT_N > 38.7780
ORDER BY
    LAT_N
LIMIT 1;
```

## 10. Weather Observation Station 15

Link:

<https://www.hackerrank.com/challenges/weather-observation-station-15/problem>

Query the Western Longitude (LONG\_W) for the largest Northern Latitude (LAT\_N) in STATION that is less than . Round your answer to decimal places.

### Input Format

The STATION table is described as follows:

**STATION**

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT\_N is the northern latitude and LONG\_W is the western longitude.

### Solution:

```
SELECT
    ROUND(LONG_W, 4)
FROM
    STATION
WHERE
    LAT_N < 137.2345
ORDER BY
    LAT_N
DESC LIMIT 1 ;
```

## 11. Weather Observation Station 17

Link:

<https://www.hackerrank.com/challenges/weather-observation-station-17/problem>

Query the Western Longitude (LONG\_W) where the smallest Northern Latitude (LAT\_N) in STATION is greater than 38.7780. Round your answer to 4 decimal places.

### Input Format

The STATION table is described as follows:

**STATION**

Field	Type
ID	NUMBER
CITY	VARCHAR2(21)
STATE	VARCHAR2(2)
LAT_N	NUMBER
LONG_W	NUMBER

where LAT\_N is the northern latitude and LONG\_W is the western longitude.

Solution

```

SELECT
    ROUND(LONG_W, 4)
FROM
    STATION
WHERE
    LAT_N > 38.7780
ORDER BY
    LAT_N
LIMIT 1 ;

```

## 12.Type Of Triangle

Link:

<https://www.hackerrank.com/challenges/what-type-of-triangle/problem>

Write a query identifying the type of each record in the TRIANGLES table using its three side lengths. Output one of the following statements for each record in the table:

- Equilateral: It's a triangle with sides of equal length.
- Isosceles: It's a triangle with sides of equal length.
- Scalene: It's a triangle with sides of differing lengths.
- Not A Triangle: The given values of A, B, and C don't form a triangle.

Input Format

The TRIANGLES table is described as follows:

<i>Column</i>	<i>Type</i>
A	<i>Integer</i>
B	<i>Integer</i>
C	<i>Integer</i>

Each row in the table denotes the lengths of each of a triangle's three sides.

Sample Input

A	B	C
20	20	23
20	20	20
20	21	22
13	14	30

### Sample Output

Isosceles  
Equilateral  
Scalene  
Not A Triangle

### Explanation

Values in the tuple form an Isosceles triangle, because .

Values in the tuple form an Equilateral triangle, because . Values in the tuple form a Scalene triangle, because .

Values in the tuple cannot form a triangle because the combined value of sides and is not larger than that of side .

### Solution:

```
SELECT
CASE
WHEN A+B > C THEN
CASE
WHEN A=B AND B=C THEN 'Equilateral'
WHEN A=B OR B=C OR C=A THEN 'Isosceles'
WHEN A!=B OR B!=C OR C!=A THEN 'Scalene'
END
ELSE 'Not A Triangle'
END
FROM TRIANGLES;
```

## Subquery

A subquery is a SQL query nested inside a larger query.

A subquery may occur :

- A SELECT clause
- A FROM clause
- A WHERE clause

The subquery can be nested inside a SELECT, INSERT, UPDATE, or DELETE statement or inside another subquery.

A subquery is usually added within the WHERE Clause of another SQL SELECT statement.

You can use the comparison operators, such as `>`, `<`, or `=`. The comparison operator can also be a multiple-row operator, such as IN, ANY, or ALL.

A subquery is also called an inner query or inner select, while the statement containing a subquery is also called an outer query or outer select.

The inner query executes first before its parent query so that the results of an inner query can be passed to the outer query.

Syntax:

```
SELECT column_name  
FROM table_name  
WHERE column_name expression operator  
      ( SELECT COLUMN_NAME   from TABLE_NAME   WHERE ... );
```

1. Write a query to display the employee who earn less than rohan

Now first you need to know the salary of rohan using the query below

```
SELECT
    salary
FROM
    employee
WHERE
    first_name = 'ROHAN'
```

Next compare the salary of other employee with rohan salary and display the result as shown below

### Solution:

```
SELECT
    *
FROM
    employee
WHERE
    salary <
(SELECT
    salary
FROM
    employee
WHERE
    first_name = 'ROHAN');
```

Output

emp_id	first_name	last_name	email	hire_date	salary	dept_id
1	kelly	davis	davis@gmail.com	2021-01-22	78000	80
4	andy	lumb	andy@gmail.com	2021-02-27	42200	80
5	anjel	nair	anj@gmail.com	2019-09-26	42200	40

6	ram	kumar	ram@gmail.com	2018-12-26	64200	40
---	-----	-------	---------------	------------	-------	----

2. Write a query to display the employee details of sales department  
 Here first we should know the dept\_id of sales that can be fetched using the query below

```
SELECT
    dept_id
FROM
    department
WHERE
    dept_name = 'sales'
```

Now display all the employees details whose dept\_id is belong to sales using the query below

```
SELECT * FROM
    employee
WHERE
    dept_id =
(SELECT dept_id FROM department WHERE dept_name =
'sales');
```

Output:

emp_id	first_name	last_name	email	hire_date	salary	dept_id
7	rohan	sharma	ro@gmail.com	2021-02-09	84200	20
8	john	king		2021-02-09	124200	20



3. Query the employee details who work in the department in which john works

First need to fetch the dept\_id of John using the query below

```
SELECT
    dept_id
FROM
    employee
WHERE
    first_name = 'JOHN'
```

Now once you got john department id, the employees whose department id matches the john the department id need to be retrieved using the query below

```
SELECT
    *
FROM
    employee
WHERE
    dept_id =
    (SELECT dept_id FROM sql_notes.employee WHERE first_name
    = 'JOHN');
```

Output:

emp_id	first_name	last_name	email	hire_date	salary	dept_id
7	rohan	sharma	ro@gmail.com	2021-02-09	84200	20
8	john	king		2021-02-09	124200	20

4. Query the details of employees working in the department where department names begins with ‘s’

Here first you need to fetch department name starting with s using the query below

```
SELECT
    dept_id
FROM
    department
WHERE
    dept_name
LIKE 's%'
```

Now once you get dept\_id of the department starting with s you can fetch employee detail working in that department using the below query

```
SELECT
    *
FROM
    employee
WHERE
    dept_id =
(SELECT dept_id FROM department WHERE dept_name LIKE
's%');
```

Output:

<b>0 9 13:06:</b>	<b>SELECT * FROM</b>	<b>Error Code:</b>	<b>0.00</b>
<b>04</b>	<b>sql_notes.employee WHERE dept_id</b>	<b>1242.</b>	<b>0 sec</b>
	<b>= (SELECT dept_id FROM</b>	<b>Subquery</b>	
	<b>sql_notes.department WHERE</b>	<b>returns more</b>	
	<b>dept_name LIKE 's%'') LIMIT 0,</b>	<b>than 1 row</b>	
	<b>1000</b>		

Here you have got the error it is because in the inner query you are fetching the department id whose department name starts with s there might be more than one department whose name starts with s, but in the parent query you are trying to match with one department id using equal to here you need to make use of IN operator which helps you to match multiple department.

```

SELECT
*
FROM
    employee
WHERE
    dept_id
IN
(SELECT dept_id FROM sql_notes.department WHERE dept_name
LIKE 's%');

```

Output:

dept_id	first_name	last_name	email	hire_date	salary	dept_id
7	rohan	sharma	ro@gmail.com	2021-02-09	84200	20
8	john	king	null	2021-02-09	124200	20
5	anjel	nair	anj@gmail.com	2019-09-26	42200	40
6	ram	kumar	ram@gmail.com	2018-12-26	64200	40

5. Query the details of employees who earn more than kelly and belong to john's department

Here you need get the dept\_id of the employee john using the query below

```
SELECT dept_id FROM employee WHERE first_name = 'john'
```

Next need to get salary for kelly using the query below

```
SELECT salary FROM sql_notes.employee WHERE first_name = 'Kelly'
```

Review the detail of all employee who earn more than kelly and dept\_id is same as john by executing the following query

```
SELECT * FROM
    employee
WHERE
    dept_id =
(SELECT dept_id FROM sql_notes.employee WHERE first_name
= 'john')
and
    salary >
(SELECT salary FROM sql_notes.employee WHERE first_name =
'Kelly');
```

Output:

dept-id	first_name	last_name	email	hire_date	salary	dept_id
7	rohan	sharma	ro@gmail.com	2021-02-09	84200	20

8	john	king		2021-02-09	124200	20
---	------	------	--	------------	--------	----

6. Query the details of employees who belong to either sales or IT department earning salaries greater than 50000

First retrieve the dept\_id working in dept\_name sales or IT using the below query

```
SELECT dept_id FROM sql_notes.department WHERE dept_name = 'sales' OR dept_name = 'IT'
```

Next retrieve the employee whose salary > 50000 and dept\_id is sales or IT using the query below

```
SELECT * FROM
employee
WHERE
dept_id
IN
(SELECT dept_id FROM department WHERE dept_name = 'sales'
OR dept_name = 'IT')
and
salary > 50000;
```

### Output:

emp_id	first_name	last_name	email	hire_date	salary	dept_id
7	rohan	sharma	ro@gmail.com	2021-02-09	84200	20
8	john	king		2021-02-09	124200	20

1	kelly	davis	davis@gmail.com	2021-01-22	78000	80
---	-------	-------	-----------------	------------	-------	----

7. Query the first\_name of managers of all department earning less than john

First retrieve the manager id of the employees working as manager using the below query

```
SELECT mrg_id FROM department
```

Next retrieve the salary of john using the query below

```
SELECT salary FROM employee WHERE first_name = 'john'
```

Now retrieve the first name of employee Where the employee is working as a manager and salary is less than the john salary using the below query

```
SELECT
    first_name
FROM
    employee
WHERE
    emp_id
IN
    (SELECT mrg_id FROM department)
AND
    salary <
    (SELECT salary FROM sql_notes.employee WHERE first_name =
    'john') ;
```

### Output:

first_name
kelly
tom
mike
andy

ram

8. Predict the output?

```
SELECT DEPT_ID, MIN(SALARY) FROM EMPLOYEE GROUP BY
DEPT_ID HAVING MIN(SALARY) < (SELECT MAX(SALARY) FROM
EMPLOYEE WHERE DEPT_ID = 20) ORDER BY DEPT_ID;
```

Here the first maximum salary of an employee working in dept\_id 20 is fetched. Next dep\_id and minimum salary is fetched in each department where minimum salary is less than maximum salary of employees working in department id 20. It will be printed in ascending order of department id.

#### **Output:**

<b>dept_id</b>	<b>MIN(salary)</b>
20	84200
40	42200
50	98200
70	84200
80	42200

9. Write a query to check if there are employees working in executive department with dept\_id 30

#### **EXISTS:**

The EXISTS operator is used to test for the existence of any record in a subquery. The EXISTS operator returns TRUE if the subquery returns one or more records.

**Syntax:**

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition);
```

```
SELECT * FROM
employee
WHERE
EXISTS
(SELECT * FROM employee WHERE dept_id = 30);
```

**Output:**

emp_id	first_name	last_name	email	hire_date	salary	dept_id
1	kelly	davis	davis@gmail.com	2021-01-22	78000	80
2	tom	taylor	tom@gmail.com	2020-09-22	84200	30
3	mike	whalen	mike@gmail.com	2021-06-30	98200	50
4	andy	lumb	andy@gmail.com	2021-02-27	42200	80
5	anjel	nair	anj@gmail.com	2019-09-26	42200	40
6	ram	kumar	ram@gmail.com	2018-12-26	64200	40
7	rohan	sharma	ro@gmail.com	2021-02-09	84200	20
8	john	king	null	2021-02-09	124200	20

10. Write a query to display the employee whose salary is greater than the minimum salary of any of the department
- First get minimum salary present in each department using the below query

```
SELECT MIN(salary) FROM employee GROUP BY dept_id
```

- Next retrieve the details of the employee whose salary is greater than minimum salary present in each department using the query below

```
SELECT first_name, dept_id, salary
FROM
employee
WHERE salary >
ANY(SELECT MIN(salary) FROM employee GROUP BY dept_id);
```

11.Create a copy of the table using subquery?

```
CREATE TABLE EMP (SELECT * FROM employee);
```

To verify weather the new table is created or not, let's execute the below query

```
SELECT * FROM emp;
```

Output:

dept_id	first_name	last_name	email	hire_date	salary	dept_id
1	kelly	davis	davis@gmail.com	2021-01-22	78000	80
2	tom	taylor	tom@gmail.com	2020-09-22	84200	30
3	mike	whalen	mike@gmail.com	2021-06-30	98200	50
4	andy	lumb	andy@gmail.com	2021-02-27	42200	80
5	anjel	nair	anj@gmail.com	2019-09-26	42200	40
6	ram	kumar	ram@gmail.com	2018-12-26	64200	40
7	rohan	sharma	ro@gmail.com	2021-02-09	84200	20
8	john	king		2021-02-09	124200	20

If you observe from the above output the data, constraint everything is copied. If you want only table with those column to be created then you can achieve using LIKE Operator as shown below

```
CREATE TABLE sql_notes.emp3 LIKE employee;
```

Now table is there if you want to copy the data present in employee table to emp3 in that case you can execute the query below

```
INSERT INTO sql_notes.emp3(SELECT * FROM employee);
```

To verify weather all data is inserted or not execute the below query

```
SELECT * FROM emp3;
```

Output:

dept_id	first_name	last_name	email	hire_date	salary	dept_id
1	kelly	davis	davis@gmail.com	2021-01-22	78000	80
2	tom	taylor	tom@gmail.com	2020-09-22	84200	30
3	mike	whalen	mike@gmail.com	2021-06-30	98200	50
4	andy	lumb	andy@gmail.com	2021-02-27	42200	80
5	anjel	nair	anj@gmail.com	2019-09-26	42200	40
6	ram	kumar	ram@gmail.com	2018-12-26	64200	40
7	rohan	sharma	ro@gmail.com	2021-02-09	84200	20
8	john	king		2021-02-09	124200	20

12. Create a copy of the table where the department id is 40

```
CREATE TABLE
emp2(SELECT * FROM employee WHERE dept_id = 40;)
```

To verify whether the copy of table is created or not execute the query below

```
SELECT * FROM emp2;
```

**Output:**

emp_id	first_name	last_name	email	hire_date	salary	dept_id
5	anjel	nair	anj@gmail.com	2019-09-26	42200	40
6	ram	kumar	ram@gmail.com	2018-12-26	64200	40

13. Update the salary of employees in sales department by incrementing to 5000

```
UPDATE
employee
SET salary = salary + 5000
WHERE
dept_id =
(SELECT dept_id FROM sql_notes.department WHERE dept_name
= 'SALES');
```

To verify weather the salary of sales department is updated or not you can execute the below query

```
SELECT * FROM employee;
```

**Output:**

dept_id	first_name	last_name	email	hire_date	salary	dept_id
1	kelly	davis	davis@gmail.com	2021-01-22	78000	80
2	tom	taylor	tom@gmail.com	2020-09-22	84200	30
3	mike	whalen	mike@gmail.com	2021-06-30	98200	50
4	andy	lumb	andy@gmail.com	2021-02-27	42200	80
5	anjel	nair	anj@gmail.com	2019-09-26	42200	40
6	ram	kumar	ram@gmail.com	2018-12-26	64200	40
7	rohan	sharma	ro@gmail.com	2021-02-09	89200	20
8	john	king		2021-02-09	129200	20

The department id of sales is 20, as you can see from the above output rohan and john salary is increased

14. Write a query to delete the record of sales manager

- First fetch the manager id with department name sales using the query below

```
SELECT
    mrg_id
FROM
    department
WHERE
    dept_name = 'sales'
```

- Once you fetched the manager id then you delete the record with the emp\_id = mrg\_id using the query below

```
DELETE
FROM
    emp3
WHERE
    emp_id =
(SELECT mrg_id FROM department WHERE dept_name = 'sales');
```

To verify whether the employees got deleted execute the query below

```
SELECT * FROM emp3;
```

Output:

dept_id	first_name	last_name	email	hire_date	salary	dept_id
1	kelly	davis	davis@gmail.com	2021-01-22	78000	80
2	tom	taylor	tom@gmail.com	2020-09-22	84200	30
3	mike	whalen	mike@gmail.com	2021-06-30	98200	50
4	andy	lumb	andy@gmail.com	2021-02-27	42200	80
5	anjel	nair	anj@gmail.com	2019-09-26	42200	40
6	ram	kumar	ram@gmail.com	2018-12-26	64200	40

7	rohan	sharma	ro@gmail.com	2021-02-09	89200	20
---	-------	--------	--------------	------------	-------	----

Here john with emp\_id was working as manager in sales department that record got deleted

15. Write a query to delete the email id of IT Manager

```
Update
  employee
SET
  email = NULL
WHERE
  emp_id =
(SELECT mrg_id FROM department WHERE dept_name = 'IT');
```

To verify weather the mail id of IT manager is updated with null execute the below query

```
SELECT * FROM employee;
```

**Output:**

dept_id	first_name	last_name	email	hire_date	salary	dept_id
1	kelly	davis	null	2021-01-22	78000	80
2	tom	taylor	tom@gmail.com	2020-09-22	84200	30
3	mike	whalen	mike@gmail.com	2021-06-30	98200	50
4	andy	lumb	andy@gmail.com	2021-02-27	42200	80
5	anjel	nair	anj@gmail.com	2019-09-26	42200	40
6	ram	kumar	ram@gmail.com	2018-12-26	64200	40
7	rohan	sharma	ro@gmail.com	2021-02-09	89200	20

8	john	king		2021-02-09	129200	20
---	------	------	--	------------	--------	----