25-May-2k25

Semester Project

CC-215, Lab Manual-MySQL

Sajawal Bukhary & Usama Ashraf

G2-BS(IT), SEMESTER 4, EVENING

Submitted To	
Madam Sehrish Khan	
Lecturer GGC SKP	
Submitted By	
Student Name(s)	
Syed Muhammad Sajawal Hussain	*Usama Ashraf*
Roll No(s)	
110 766	*110 769*
Reg.No(s)	
2023-KS-570	*2023-KS-569*
Contact(s)	
0319-7341432	*0319-9272955*
Feedback @	
bsitf23e110766@gcbskp.edu.pk	
bsitf23e110769@gcbskp.edu.pk	

---------- ----------

Instructure **HOD**

Table of contents

Lab Manual DBMS

Activities

- Pre-Lab activities
- During Lab Tasks
- Post-Lab Activities

Requirements

- Laptop/Desktop
- Internet Connection For setting up MySQL
- Installed MySQL (Full Setup)

Pre-Lab Tasks
• Introduction to basic terms of Database Systems • Installation of MySQL setup • Introduction to Semester Project Problem
During Lab Tasks
• Learn the usage of CLI for MySQL Client • Building ERDs and Normalize Project Statement • Implement basic functions
Post-Lab Tasks
• Check your database for Real-World scenarios. • Update your database for correctness and consistency • Review your system

Pre-Lab Tasks

Installation of MySQL

- *Click here to download setup*
- Extract WinRAR file
    - Run C++ Redistribution file
    - Now restart your PC

  - Run MySQL setup
- Complete installation steps
- Use easy to accessible password
- Enjoy Coding in MySQL

Basics of Database

- **Data:**
  Collection of Unprocessed facts and figures is called data.
- **Instructions:**
  The meta data that is used to treat data is called instructions.
- **Processing:**
  Processing is the step of performing different operations on data is called processing.
- **Information:**
  Processed data is called information.
- **Database:**
  A database is a structured collection of data that is stored and managed electronically. It allows efficient storage, retrieval, and manipulation of data. Databases are designed to manage huge data and provide a way to organise, update, and query data.
- **Types of Databases:**
  Relational Databases (MySQL, PostgreSQL), NoSQL Databases (MongoDB, Cassandra), Graph Databases (Neo4j), and Time-Series Databases (Influx DB) are some common types of databases.
- **Database System:**
  Database system is a software that allows for the definition, creation, maintenance, and manipulation of databases. It provides a structured way store, manage, and retrieve data. Database systems are designed to enable efficient data management and support various applications and users.

- **Components of Database:**
  Database Management System, Database, Data Dictionary, Schema, Query Processing, Storage Engine, Security and Access Control, Backup & Recovery, and User Interface are the basic components of Database System.
- **Entity:**
  Entity is an object, about which data is collected, and Table is developed.
- **Table:**
  It is a collection of rows and columns; related records are integrated in a single table.
- **Attributes:**
  A characteristic or property of entity is called the Attribute. Columns are created depending upon these attributes.
- **Column:**
  A single field or attribute of a table is called column.
- **Record:**
  A single entry or instance of entity is called record. In a record a single entity is explained using all the attributes.
- **Centralized Database:**
  A database that is stored and managed at a single location, such as a server or mainframe. All data is stored in one place, and users access it remotely.
- **Distributed Database:**
  A database that is spread across multiple locations, such as multiple places, and users can access it form any location. Distributed databases can improve performance, availability, and performance.
- **Normalization:**
  The process of organizing data in a database to minimize data

redundancy and dependency. It involves dividing large tables into smaller ones, each with a specific purpose, improve data integrity and reduce data duplication.

- **Denormalization:**
  The process of intentionally allowing some data redundancy or dependency in a database to improve performance, typically for Read-Heavy applications. Denormalization can speed up queries but may compromise data consistency.
- **Entity Relationship Diagram:**
  ERD is visual representation of the structure of a database, showing the relationships between entities and their attributes. ERDs help design and understand database schema, making it easier to identify entities, attributes, and relationships.
- **Forms:**
  A way to collect data from users typically in a graphical interface. Forms may be use for multiple purposes.
- **Users:**
  An individual who interacts with the system, applications or database is called user.
- **Views:**
  View is a virtual table that is used to represent data, often based on queries or specific conditions.
- **Trigger:**
  A set of actions that is automatically executed in response to specific database events, such as insert, update, or delete.
- **Query:**
  A request to get data from database is known as query. In simple words a request to retrieve data or for interaction with database.

- **Subquery:**
  A query nested inside another query, used to retrieve data that depends upon the result of outer query.
- **Data fragmentation:**
  The process of breaking down data into smaller pieces or fragments, often stored in different locations, which can lead to inefficiencies in data storage and retrieval.
- **Data replication:**
  The process of making and maintaining multiple copies of data in different locations, often to improve data availability, reliability, and performance.
- **DDL Commands:**
  Data Definition Commands, are those commands used to define and modify the database structure. Such as CREATE, TRUNCATE, ALTER, DROP.
- **DML Commands:**
  Data Manipulation Language, are those commands which are used to manage data within a database. Such as INSERT, UPDATE, DELETE.
- **DCL Commands:**
  Data Control Language, are those commands which are used to control access and permissions within a database. Such as GRANT, REVOKE.
- **SDLC:**
  SDLC stands for Software Development Life Cycle. A process of planning, designing, developing, testing, and implementing software applications.
- **DBLC:**
  DBLC stands for Database Life Cycle. A process of designing,

implementing, and maintaining database, including planning, analysis, design, implementation, and maintenance.

- **SQL vs NoSQL:**

  The difference between SQL and NoSQL is given below:

Feature	SQL Databases	NoSQL Databases
1. Data Structure	Relational, tabular (rows and columns)	Non-relational; document, key-value, graph, column-family
2. Schema	Predefined, fixed schema	Dynamic, flexible, schema-less
3. Query Language	SQL (Structured Query Language)	Varies depending on the NoSQL type (e.g., JSON, Cypher)
4. Scalability	Primarily vertical scaling (scale up hardware)	Primarily horizontal scaling (add more servers)
5. Transactions	ACID properties (Atomicity, Consistency, Isolation, Durability)	BASE properties (Basically Available, Soft state, eventually consistent) - some offer ACID

6. Relationships	Explicit relationships via foreign keys, joins	Relationships can be embedded, referenced, or graph-based
7. Data Integrity	Strong data integrity due to schema and constraints	Can vary; schema-less nature might lead to less strict integrity
8. Complexity	Can handle complex queries with joins	Complex queries can be challenging, often denormalized for performance
9. Flexibility	Less flexible for evolving data structures	Highly flexible for handling diverse and changing data
10. Use Cases	Structured data, complex transactions, reporting, data warehousing	Unstructured/semi-structured data, high traffic, real-time applications, big data

Problem

We must design a database for a company which runs Intercity Routes and want to track their company record. Their case is like the given statement.

Statement

A bus company runs intercity routes. Passengers book seats online. Buses are equipped with GPS and expected arrival times are updated in real time. Maintenance logs include mileage, engine checks, and incidents. Drivers are assigned routes in shifts. Passengers rate cleanliness, driver behaviour, and punctuality.

Let's break it down for better understanding and to understand the structure of working of company we must also observe some real-time scenarios. So, that we can solve this problem in unique and acceptable manners.

Detailed Overview

Passengers initiate their journey by securing their seats through an online booking system. Once on board, the buses are tracked in real time using GPS technology. This allows for the dynamic updating of expected arrival times, keeping passengers informed about their journey's progress.

Behind the scenes, a meticulous maintenance schedule ensures the buses remain in optimal condition. Detailed logs are maintained, recording the mileage accumulated by each bus, the dates and outcomes of engine checks, and any incidents that may have occurred.

The operation also involves the careful assignment of drivers to specific routes, organized in shifts to ensure continuous service.

Upon reaching their destination, passengers can provide valuable feedback. They can rate various aspects of their experience, including the cleanliness of the bus, the behaviour of the driver, and the punctuality of the service. This feedback loop is crucial for the company to monitor and improve its services.

During Lab Tasks

Normalization

The process of organizing data in simpler way to minimize data redundancy, eliminate anomalies, and improve integrity. Normalization involves dividing large tables into smaller ones, each with a specific purpose, and linking them through relationships. Following goals are considered when we normalize a database.

- Reducing data redundancy
- Improving data consistency
- Enhancing scalability and flexibility

When we normalize a database, we must pass through following three phases which are also called Normalization Forms.

- First Normal Form
- Second Normal Form
- Third Normal Form

Above steps are about removing different levels of removing dependencies from data and a larger table is converted into smaller multiple tables.

- **First Normal Form**
  Removing repeating groups, and arrays, setting up primary key, creating separate columns for each attribute.
- **Second Normal Form**
  After satisfying the requirements of first normal form. A database or table is moved for satisfaction of criteria for Second Normal Form. Partial dependencies are removed. Dependent columns are moved with their relevant columns.
- **Third Normal Form**
  After satisfaction of second normal form criteria. A database or table is moved for satisfaction of criteria for Third Normal Form. Where Transitive dependencies are removed and finally a normalized form is received.

Example:

STUDENT (stdno, dptno, stdname, dptname, course, teacher)

- **First NF**
  STUDENT (stdno, dptno, stdname, dptname, crsno, course, teacher)
- **Second NF**
  STD (stdno, stdname), DPT (dptno, dptname, crsno, course, teacher)
- **Third NF**
  STD (stdno, stdname), DPT (dptno, dptname), CRS (crsno, course, teacher)
- **Result:**
  STD (stdno, stdname, dptno, crsno), DPT (dptno, dptname), CRS (crsno, course, teacher)

Designing ERDs

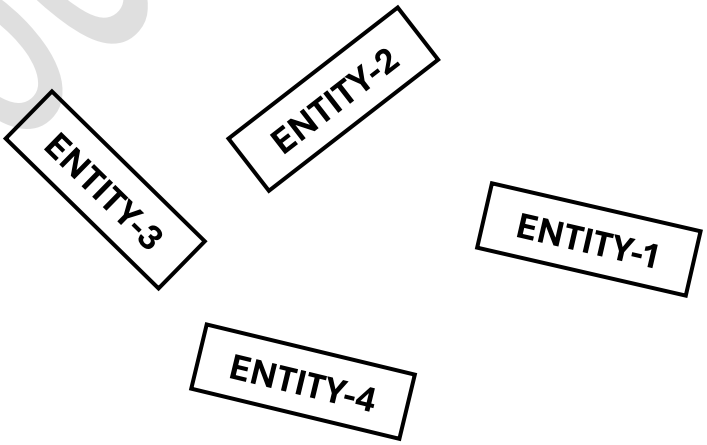
Designing ERD process involves multiple steps. These steps are enlisted below.

- Defining Problem (has done)
- Identifying Entities
- Identifying Attributes
- Identifying Relationships
- Creating/Drawing diagrams
    - Chen's Model ERD
    - Crow's Foot Model ERD

Entities

After a deep observation following entities are found from above given statement. These entities will become tables later when we will turn our graphical project into coding form and will write MySQL queries to develop a full structure.

- INTERCITY ROUTES
- BUSSES
- MAINTENANCE LOGS
- DRIVERS



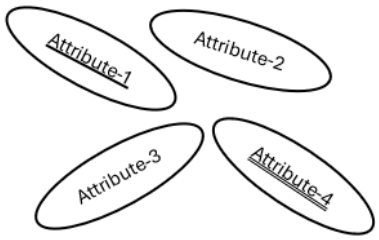
It is necessary to know about entity. Entity is a real-world object which can have properties, and about which we can collect data to discuss it. Some examples of entities are Person, Student, Car, Chair, Book, or may be any name that may have properties to describe it in a well-mannered way. An entity is graphically represented by using a rectangle inside which entity name is written as given in diagram.

Attributes

Deeply observation revealed that this statement includes only few attributes. But it's impossible to create a database for managing a huge system without identifying its participations. To identify all the entities, it's necessary to find out their related attributes. We found these attributes for above entities.

Entities	Attributes					
	A1	**A2**	**A3**	**A4**	**A5**	**A6**
ROUTES	*Route_ID*	*Distance*	*Time*	*Driver_id*		
BUSES	*Bus_ID*	*Bus_Type*	*Bus_Capa*	*Maint_id*	*Route_id*	*Driver_id*
MAINT_LOGS	*Maint_ID*	*Engine*	*Mileage*	*Maitenan*		
DRIVERS	*Driver_ID*	*Name*	*Contact*	*Salary*	*HireDate*	*Address*

Before moving forward it's necessary to know about attributes. An attribute is a property of entity that describes the entity. Such as its name, its adjective. Examples of attributes for a person are Name, Contact, CNIC, Address etc. Attributes are represented by using oval shape in diagram. Inside oval attribute name is written. There may be multiple types of attributes such as derived attributes, multi-valued attributes etc. It is important to note that an attribute which is primary key is underlined as "Attribute-1", and a foreign key is also underlined but its double underlined as "Attribute-4".

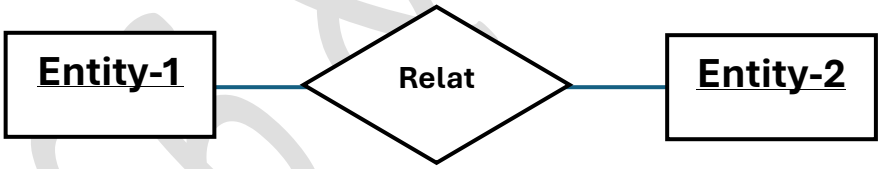


Relationships

After deep observation we concluded following relationships from above scenario. Where entities are related to each other to develop a full system. To develop a full Real-World acceptable system, it's necessary to find out authentic relationships.

No	Entity-1	Relationship	Entity-2
1	**Drivers**	*Assign*	**Routes**
2	**Drivers**	*Assign*	**Buses**
3	**Buses**	*Has*	**Maintenance_Logs**
4	**Buses**	*Has*	**Routes**

Before moving forward even a single step it's necessary to know about a relationship in a Database Management System. A relationship describes how two or more entities interact or are connected in a system. It defines the association, such as "owns," "belongs to," or "works for," between them.



Relational Schema

Relational Schema for above problem is given below:

1. **BUSES** (Bus_ID, Bus_Type, Bus_Capacity, Driver_ID, Route_ID, Maintenance_ID)

2. **ROUTES** (Route_ID, Route_Distance, Route_Time, Driver_ID)

3. **MAINTENANCE_LOGS** (Maint_ID, Engine_Check, Mileage, Maint_Desc)

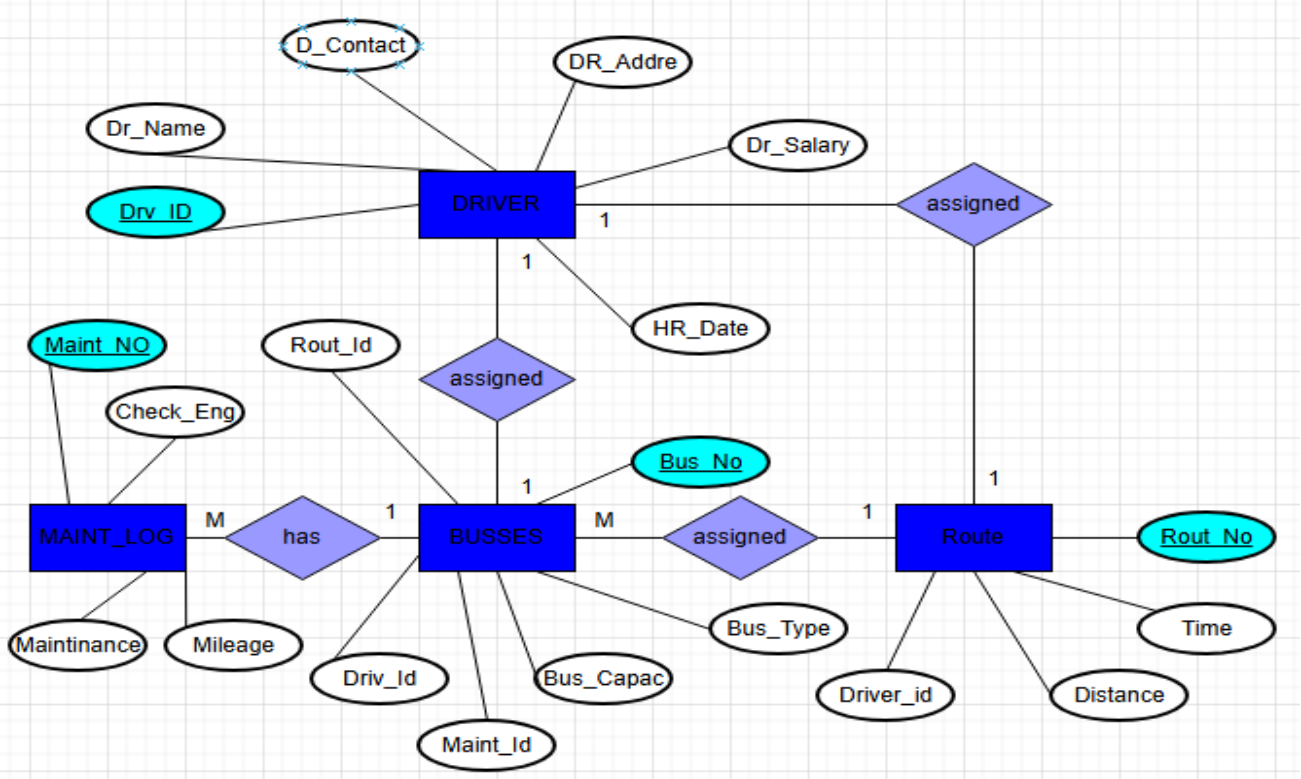
4. **DRIVERS** (Driver_ID, Driver_Name, Driver_Contact, Driver_Salary, Driver_Address, Driver_HireDate)

Diagrams Drawing

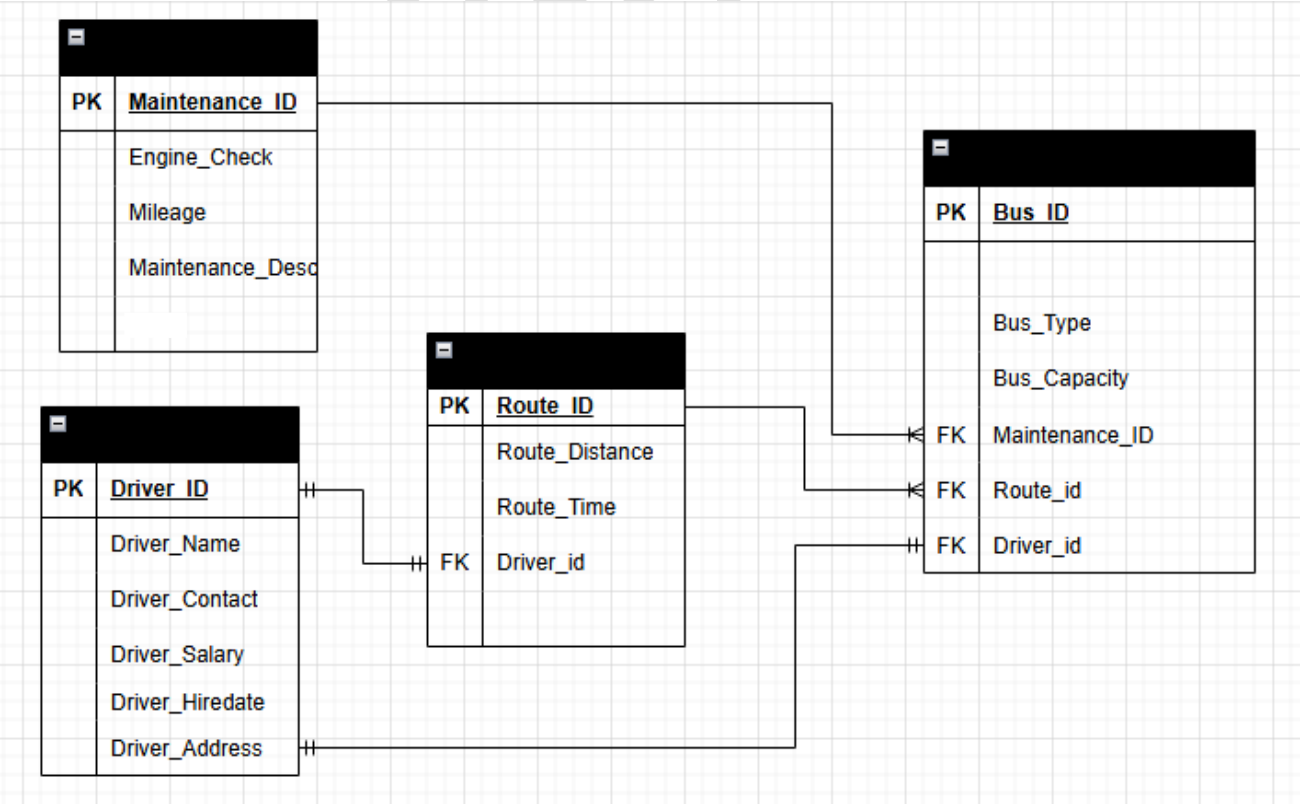
To draw diagrams for above statement. We need to use drawing tool "draw.io" which is easy to use and free (Open Source) platform for users to develop millions of diagrams.

Because we have done few basic steps already. So, it easy to draw diagram now. By using any online/offline tool we can also complete this step.

Chen's Model ERD



Crow's Foot Model ERD



Database Creation/Deletion Queries

These are commonly used MySQL queries. Which are used for developing structure of database.

- **Show database**
  This query shows all the available databases. It helps us in looking for a specific database if it exist or not.
  Syntax:
  SHOW DATABASES;

```
mysql> show databases;
+------------------------+
| Database               |
+------------------------+
| bus_reservation_system |
| college                |
```

- **Create Database**
  This query is used to create database. It is first query in creating a new database.
  Syntax:
  CREATE YOUR_DATABASE_NAME ;

```
mysql> create database BusReservationSystem;
Query OK, 1 row affected (0.03 sec)

mysql> show databases;
+------------------------+
| Database               |
+------------------------+
| bus_reservation_system |
| busreservationsystem   |
| college                |
```

- **Dropping a Database**
  This query is used to drop a database. It helps us to delete a database when it is not needed more.

Syntax:

DROP DATABASE YOUR_DATABASE_NAME;

```
mysql> drop database college;
Query OK, 3 rows affected (0.20 sec)
```

Rename query is not supported in MySQL for updating the name of whole database. So, for this you must delete whole database if any name related mistake is in your database's name.

Tables Creation/Deletion Queries

These queries are used to deal tables in database. We can use these queries for creating table, dropping table, deleting table, updating table, altering table and few more operations on table.

- **Show Tables**

  This query will also enlist tables that are created before you if you are using any old database. You can also use it when you want to check how many tables are available in your database.

  Syntax:

  SHOW TABLES;

```
mysql> Show tables;
Empty set (0.01 sec)
```

- **Create tables**

  This query is used to create a new table. It helps us in creating a table in MySQL. In following command "N" represents that you can use variable number of columns.

  Syntax:

  CREATE TABLE TABLE_NAME(COL_1 DATATYPE(DOMAIN)....N) ;

```
mysql> create table buses(busno INT(5), buscolor varchar(10));
Query OK, 0 rows affected, 1 warning (0.04 sec)

mysql> show tables;
+-------------------------------+
| Tables_in_busreservationsystem |
+-------------------------------+
| buses                         |
+-------------------------------+
1 row in set (0.00 sec)
```

- **Describe Table**

  This query is used to describe table. It describes the structure of table, constraints of table etc.

  Synatx-1:

  DESCRIBE TABLE YOUR_TABLE_NAME;

```
mysql> describe table buses;
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------+
|  1 | SIMPLE      | buses | NULL       | ALL  | NULL          | NULL | NULL    | NULL |    1 |   100.00 | NULL  |
+----+-------------+-------+------------+------+---------------+------+---------+------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)
```

  Syntax-2:

  DESCRIBE YOUR_TABLE_NAME;

```
mysql> describe buses;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| busno    | int         | YES  |     | NULL    |       |
| buscolor | varchar(10) | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
2 rows in set (0.01 sec)
```

- **Inserting Value in Tables**

  This query is used in two ways, for inserting values in tables of database. You can also enter values for multiple rows at a tme.

  Syntax-1:

  INSERT INTO YOUR_TABLE(COL_1,COL_2...COL_N)
  VALUE(CVAL_1, CVAL_2....CVAL_N) ;

```
mysql> insert into buses (busno, buscolor) value (1, 'Red');
Query OK, 1 row affected (0.01 sec)
```

Syntax-2:

INSERT INTO YOUR_TABLE VALUE (CVAL_1, CVAL_2....CVAL_N);

```
mysql> insert into buses value (2, 'Yellow');
Query OK, 1 row affected (0.01 sec)
```

- **Showing up your table's values**

  This query is used to show values entered in your table.

  Syntax-1:

  *SELECT * FROM TABLE_NAME ;*

```
mysql> select * from buses;
+-------+----------+
| busno | buscolor |
+-------+----------+
|     1 | Red      |
|     2 | Yellow   |
+-------+----------+
2 rows in set (0.00 sec)
```

Syntax-2:

SELECT COL_1, COL_2....COL_N FROM YOUR_TABLE_NAME ;

```
mysql> SELECT buscolor from buses;
+----------+
| buscolor |
+----------+
| Red      |
| Yellow   |
+----------+
2 rows in set (0.00 sec)
```

- **Update table**

  This query is used to update any value of a table. Such as we want to change any value from tables one cell we can change it easily by using this query.

  Syntax:

  UPDATE TABLE_NAME SET COL = 'COLVAL' ;

```
mysql> update buses set buscolor = 'Green' ;
Query OK, 1 row affected (0.01 sec)
Rows matched: 2  Changed: 1  Warnings: 0

mysql> select * from buses;
+-------+----------+
| busno | buscolor |
+-------+----------+
|     1 | Green    |
|     2 | Green    |
+-------+----------+
2 rows in set (0.00 sec)
```

But we observed that all the value from table are changed with given colour. To handle this error, we have a clause 'WHERE CLAUSE' which allow us to us condition.

- **WHERE Clause**
  This clause is used at the end of a query and is used to give a condition for a process.
  Syntax:
  UPDATE YOUR_TABLE SET COL='COVAL' WHERE CONDITION;

```
mysql> update buses set buscolor = 'RED' where busno=1 ;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from buses;
+-------+----------+
| busno | buscolor |
+-------+----------+
|     1 | RED      |
|     2 | Green    |
+-------+----------+
2 rows in set (0.00 sec)
```

- **Distinct Query**
  It is used to get distinct values. We use it when we are looking for such records where there is no repeating values in rows. It does not show repeated rows.
  Syntax:
  SELECT DISTINCT COL_NAME FROM TABLE_NAME ;

```
mysql> select * from buses;
+-------+----------+
| busno | buscolor |
+-------+----------+
|     1 | RED      |
|     2 | Green    |
|     3 | Green    |
|     3 | RED      |
+-------+----------+
4 rows in set (0.00 sec)

mysql> select distinct buscolor from buses;
+----------+
| buscolor |
+----------+
| RED      |
| Green    |
+----------+
2 rows in set (0.00 sec)
```

- **Alter Table:**
  This query is used to alter table. It is used for ways of altering a table.
  Syntax-1:
  ALTER TABLE TABLE_NAME ADD COL_NAME DATATYPE(DOMAIN) CONSTRAINT ;

```
mysql> ALTER table buses add column busstop varchar(10) not null;
Query OK, 0 rows affected (0.04 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> select * from buses;
+-------+----------+---------+
| busno | buscolor | busstop |
+-------+----------+---------+
|     1 | RED      |         |
|     2 | Green    |         |
|     3 | Green    |         |
|     3 | RED      |         |
+-------+----------+---------+
4 rows in set (0.00 sec)
```

Here constraint may be, NULL, NOT NULL OR UNIQUE.

- **Delete Command**
  This command is used to delete a single row from a table for this command WHERE clause is also used to give condition.
  Syntax:
  DELETE FROM TABLE_NAME WHERE CONDITION ;

```
mysql> delete from buses where busno=3;
Query OK, 2 rows affected (0.01 sec)

mysql> select * from buses;
+-------+----------+---------+
| busno | buscolor | busstop |
+-------+----------+---------+
|     1 | RED      |         |
|     2 | Green    |         |
+-------+----------+---------+
2 rows in set (0.00 sec)
```

- **Truncate Command**

  This command/query is used to delete all the data available in the table. Table remains available but its data is deleted.

  Syntax:

  TRUNCATE TABLE_NAME ;

```
mysql> truncate buses;
Query OK, 0 rows affected (0.04 sec)

mysql> show tables;
+--------------------------------+
| Tables_in_busreservationsystem |
+--------------------------------+
| buses                          |
+--------------------------------+
1 row in set (0.01 sec)

mysql> select * from buses;
Empty set (0.00 sec)
```

- **Rename a table**

  This query is used to rename a table when you misspelled its name, or you want to rename it due to another reason.

  Syntax:

  RENAME TABLE TABLE_OlD_NAME TO TABLE_NEW_NAME ;

```
mysql> show tables;
+--------------------------------+
| Tables_in_busreservationsystem |
+--------------------------------+
| buses                          |
+--------------------------------+
1 row in set (0.00 sec)

mysql> rename table buses to vehicles;
Query OK, 0 rows affected (0.03 sec)

mysql> show tables;
+--------------------------------+
| Tables_in_busreservationsystem |
+--------------------------------+
| vehicles                       |
+--------------------------------+
1 row in set (0.00 sec)
```

- **Drop a table**

  This query is used to delete a table permanently. It drops a table, its data and related information about table.

Syntax:

DROP TABLE YOUR_TABLE;

```
mysql> drop table vehicles;
Query OK, 0 rows affected (0.02 sec)

mysql> show tables;
Empty set (0.00 sec)
```

- **Setting-Up Primary key**

  This is table creation query which also helps us to set up primary key for a table.

  Syntax:

  CREATE TABLE YOUR_TABLE(COL_1 DATATYPE(DOMAIN) PRIMARY KEY, COL_2.....COL_N);

```
mysql> create table bus(vno INT(5) primary key, color varchar(5));
Query OK, 0 rows affected, 1 warning (0.03 sec)
```

- **Setting up foreign key**

  This query is used to set up foreign key for a table. Be care full when setting up foreign key. Both tables should contain at lest one common column in them. Such as we have two tables both have a common column which is “vno INT(5)”.

  Syntax:

  ALTER TABLE YOUR_TABLE ADD CONSTRAINT FK_NAME FOREIGN KEY (COL) REFERENCES PARENT_TABLE(COL) ;

```
mysql> alter table vehicle add constraint fk_BUS foreign key (vno) references bus(vno);
Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

- **Alias (Report Name)**
  This is used to give a temporary name to a column of a table.
  Syntax:
  SELECT DRIVER_NAME AS NAME, DRIVER_SALARY AS SALARY FROM DRIVERS;

```
mysql> select driver_name AS NAME, driver_salary AS SALARY from drivers;
+--------------+--------+
| NAME         | SALARY |
+--------------+--------+
| Ahad Hasan   |  50000 |
| Wahad Hasam  |  45000 |
| Ghufran Gul  |  55000 |
| Shafi Hasan  |  60000 |
| Haiedar Ali  |  40000 |
+--------------+--------+
5 rows in set (0.00 sec)
```

Clauses

- **Where clause**

  This clause is used to provide some condition according to given condition data is fetched from database. Such as we want to see a driver's info who has driver_id1 or other than one (Which is unique for driver), we will use this clause.

  Syntax:

  *SELECT * FROM YOUR_TABLE WHERE CONDITION;*

```
mysql> select * from drivers where driver_id=4;
+-----------+-------------+----------------+---------------+---------------------+-----------------+
| driver_id | driver_name | driver_contact | driver_salary | driver_address      | driver_hiredate |
+-----------+-------------+----------------+---------------+---------------------+-----------------+
|         4 | Shafi Hasan | +923001100110  |         60000 | 111 street city xyz | 2012-01-02      |
+-----------+-------------+----------------+---------------+---------------------+-----------------+
1 row in set (0.00 sec)

mysql> select * from drivers where driver_id=0;
Empty set (0.01 sec)

mysql>
```

- **Having clause**

  Having clause is used with the combination of Order BY clause. It is also used to put some condition for presenting data. It filters results based some condition.

  Syntax:

  SELECT DRIVER_SALARY AS SALARY, AVG(DRIVER_SALARY) AS AVG_SALARY FROM DRIVERS GROUP BY DRIVER_SALARY HAVING AVG(DRIVER_SALARY) = 50000 ;

```
mysql> select driver_salary AS SALARY, AVG(driver_salary) AS AVG_SALARY from drivers group by driver_salary having AVG(driver_salary) = 50000;
+--------+------------+
| SALARY | AVG_SALARY |
+--------+------------+
|  50000 | 50000.0000 |
+--------+------------+
1 row in set (0.00 sec)

mysql>
```

- **Order By Clause**
  Order by clause is used to arrange data in an order commonly. Order of data may be Ascending or Descending. It is about sorting of data in a unique order. You have to arrange a record of students in order, or drivers in order you will use this.
  Syntax:
  SELECT DRIVER_NAME, DRIVER_CONTACT FROM DRIVERS ORDER BY DRIVER_NAME ASC ;

```
mysql> select driver_name, driver_contact from drivers ORDER BY driver_name asc;
+-------------+----------------+
| driver_name | driver_contact |
+-------------+----------------+
| Ahad Hasan  | +923001122334  |
| Ghufran Gul | +923996677330  |
| Haiedar Ali | +923999777444  |
| Shafi Hasan | +923001100110  |
| Wahad Hasam | +923110022991  |
+-------------+----------------+
5 rows in set (0.01 sec)
```

By changing order from ASC to DESC:

```
mysql> select driver_name, driver_contact from drivers ORDER BY driver_name desc;
+-------------+----------------+
| driver_name | driver_contact |
+-------------+----------------+
| Wahad Hasam | +923110022991  |
| Shafi Hasan | +923001100110  |
| Haiedar Ali | +923999777444  |
| Ghufran Gul | +923996677330  |
| Ahad Hasan  | +923001122334  |
+-------------+----------------+
5 rows in set (0.00 sec)
```

- **Between**

  This command is used to return records between given range; it gets two values from user higher(1st) and lower(2nd) and returns result. Where clause is also used with it.

  Syntax:

  SELECT DRIVER_ID, DRIVER_NAME, DRIVER_SALARY FROM DRIVERS WHERE DRIVER_SALARY BETWEEN 40000 AND 50000 ;

```
mysql> select driver_id, driver_name, driver_salary from drivers where driver_salary BETWEEN 40000 AND 50000;
+-----------+-------------+---------------+
| driver_id | driver_name | driver_salary |
+-----------+-------------+---------------+
|         1 | Ahad Hasan  |         50000 |
|         2 | Wahad Hasam |         45000 |
|         5 | Haiedar Ali |         40000 |
+-----------+-------------+---------------+
3 rows in set (0.00 sec)
```

- **Use Between for Date**

  This query gives dates between range a range.

  Syntax:

  SELECT DRIVER_NAME AS NAME, DRIVER_HIREDATE AS HIRED_ON FROM DRIVERS WHERE DRIVER_HIREDATE BETWEEN '2014-03-03' AND '2016-02-02' ;

```
mysql> select driver_name AS NAME, driver_hiredate AS Hired_ON from drivers where driver_hiredate BETWEEN '2014-03-03' AND '2016-02-02';
+-------------+------------+
| NAME        | Hired_ON   |
+-------------+------------+
| Ahad Hasan  | 2015-01-01 |
| Wahad Hasam | 2016-02-02 |
| Ghufran Gul | 2014-03-03 |
+-------------+------------+
3 rows in set (0.00 sec)
```

- **In Clause**

  This query is used to find out a specific record based on the provided record's list.

  Such as you will provide many values after IN and matching record will be printed.

Syntax:

*SELECT * FROM DRIVERS WHERE DRIVER_SALARY IN (50000, 40000, 250000);*

```
mysql> select * from drivers where driver_salary IN (50000, 40000, 250000);
+-----------+-------------+----------------+---------------+--------------------+-----------------+
| driver_id | driver_name | driver_contact | driver_salary | driver_address     | driver_hiredate |
+-----------+-------------+----------------+---------------+--------------------+-----------------+
|         1 | Ahad Hasan  | +923001122334  |         50000 | 123 street city abc | 2015-01-01      |
|         5 | Haiedar Ali | +923999777444  |         40000 | 222 street city jkl | 2017-05-10      |
+-----------+-------------+----------------+---------------+--------------------+-----------------+
2 rows in set (0.00 sec)
```

- **Like ‘’**

  It is used to find out the specific name based on length, first letter, or last letter etc.

  Syntax-1:

  *SELECT * FROM DRIVERS WHERE DRIVER_NAME LIKE 'A%';*

```
mysql> select * from drivers where driver_name like 'A%';
+-----------+-------------+----------------+---------------+--------------------+-----------------+
| driver_id | driver_name | driver_contact | driver_salary | driver_address     | driver_hiredate |
+-----------+-------------+----------------+---------------+--------------------+-----------------+
|         1 | Ahad Hasan  | +923001122334  |         50000 | 123 street city abc | 2015-01-01      |
+-----------+-------------+----------------+---------------+--------------------+-----------------+
1 row in set (0.00 sec)
```

Syntax-2:

*SELECT * FROM DRIVERS WHERE DRIVER_NAME LIKE '%N';*

```
mysql> select * from drivers where driver_name like '%n';
+-----------+-------------+----------------+---------------+--------------------+-----------------+
| driver_id | driver_name | driver_contact | driver_salary | driver_address     | driver_hiredate |
+-----------+-------------+----------------+---------------+--------------------+-----------------+
|         1 | Ahad Hasan  | +923001122334  |         50000 | 123 street city abc | 2015-01-01      |
|         4 | Shafi Hasan | +923001100110  |         60000 | 111 street city xyz | 2012-01-02      |
+-----------+-------------+----------------+---------------+--------------------+-----------------+
2 rows in set (0.00 sec)
```

Syntax-3:

*SELECT * FROM DRIVERS WHERE DRIVER_NAME LIKE '_____';*

```
mysql> select * from drivers where driver_name like '_________';
+-----------+-------------+----------------+---------------+--------------------+-----------------+
| driver_id | driver_name | driver_contact | driver_salary | driver_address     | driver_hiredate |
+-----------+-------------+----------------+---------------+--------------------+-----------------+
|         1 | Ahad Hasan  | +923001122334  |         50000 | 123 street city abc | 2015-01-01      |
+-----------+-------------+----------------+---------------+--------------------+-----------------+
1 row in set (0.00 sec)
```

Functions

Functions are also available in MySQL which are used to perform many tasks. It is necessary to note that a function gets parameters. And performs some operation on these parameters.

- Format Functions
- Aggregate Functions

Format Functions

- **Date Format**
  This function is used to specify the format of function.
  Syntax-1:
  SELECT DRIVER_NAME AS NAME,
  DATE_FORMAT(DRIVER_HIREDATE, '%D %M %Y') AS HIRED
  FROM DRIVERS ;

```
mysql> select driver_name AS NAME, DATE_FORMAT(driver_hiredate, '%d %m %y') AS HIRED from drivers;
+-------------+----------+
| NAME        | HIRED    |
+-------------+----------+
| Ahad Hasan  | 01 01 15 |
| Wahad Hasam | 02 02 16 |
| Ghufran Gul | 03 03 14 |
| Shafi Hasan | 02 01 12 |
| Haiedar Ali | 10 05 17 |
+-------------+----------+
5 rows in set (0.00 sec)
```

Syntax-2:

SELECT DRIVER_NAME AS NAME,
DATE_FORMAT(DRIVER_HIREDATE, '%D %M %Y') AS HIRED
FROM DRIVERS;

```
mysql> select driver_name AS NAME, DATE_FORMAT(driver_hiredate, '%d %M %Y') AS HIRED from drivers;
+-------------+------------------+
| NAME        | HIRED            |
+-------------+------------------+
| Ahad Hasan  | 01 January 2015  |
| Wahad Hasam | 02 February 2016 |
| Ghufran Gul | 03 March 2014    |
| Shafi Hasan | 02 January 2012  |
| Haiedar Ali | 10 May 2017      |
+-------------+------------------+
5 rows in set (0.00 sec)
```

Syntax-3:

SELECT DRIVER_NAME AS NAME,
DATE_FORMAT(DRIVER_HIREDATE, '%D %M %Y') AS HIRED
FROM DRIVERS ;

```
mysql> select driver_name AS NAME, DATE_FORMAT(driver_hiredate, '%D %M %Y') AS HIRED from drivers;
+-------------+-------------------+
| NAME        | HIRED             |
+-------------+-------------------+
| Ahad Hasan  | 1st January 2015  |
| Wahad Hasam | 2nd February 2016 |
| Ghufran Gul | 3rd March 2014    |
| Shafi Hasan | 2nd January 2012  |
| Haiedar Ali | 10th May 2017     |
+-------------+-------------------+
5 rows in set (0.00 sec)
```

- **Time Format**
  This function is used to define format for time. We can manage how can time be presented.

  Syntax-1:

  SELECT ROUTE_DISTANCE AS DISTANCE,
  TIME_FORMAT(ROUTE_TIME, '%H:%I %P') FROM ROUTES;

```
mysql> select route_distance AS Distance, TIME_FORMAT(route_time, '%h:%i %p') from routes;
+----------+-----------------------------------+
| Distance | TIME_FORMAT(route_time, '%h:%i %p') |
+----------+-----------------------------------+
| 20km     | 10:15 AM                          |
| 30km     | 10:30 AM                          |
| 25km     | 10:45 AM                          |
| NULL     | 11:00 AM                          |
| NULL     | NULL                              |
+----------+-----------------------------------+
5 rows in set (0.00 sec)
```

Syntax-2:

SELECT ROUTE_DISTANCE AS DISTANCE,
TIME_FORMAT(ROUTE_TIME, '%H:%I:%S %P') FROM ROUTES;

```
mysql> select route_distance AS Distance, TIME_FORMAT(route_time, '%h:%i:%s %p') from routes;
+----------+--------------------------------------+
| Distance | TIME_FORMAT(route_time, '%h:%i:%s %p') |
+----------+--------------------------------------+
| 20km     | 10:15:00 AM                          |
| 30km     | 10:30:00 AM                          |
| 25km     | 10:45:00 AM                          |
| NULL     | 11:00:00 AM                          |
| NULL     | NULL                                 |
+----------+--------------------------------------+
5 rows in set (0.00 sec)
```

Aggregate Functions

- **SUM() Function**
  Sum function is used to find out the total summation of an attribute. Such as if you want to find out the total amount that you spent in giving salaries to your employee, you will use this function.
  Syntax:
  SELECT SUM(DRIVER_SALARY) AS TOTAL_SALARY FROM DRIVERS;

```
mysql> select sum(driver_salary) AS Total_Salary from drivers;
+--------------+
| Total_Salary |
+--------------+
|       250000 |
+--------------+
1 row in set (0.00 sec)
```

- **MIN() Function**
  MIN function is used to find out the minimum value from any integer value. Based on this function you can find out the month from year in which you spent your minimum salary, or you can find out your employee with minimum salary.
  Syntax:
  SELECT MIN(DRIVER_SALARY) AS TOTAL_SALARY FROM DRIVERS;

```
mysql> select MIN(driver_salary) AS Total_Salary from drivers;
+--------------+
| Total_Salary |
+--------------+
|        40000 |
+--------------+
1 row in set (0.00 sec)
```

- **MAX() Function**
  Max function is used to find out maximum amount from a table. Based on this function we can find out the maximum expenses of a house from all months of an year.

Syntax:

SELECT MAX(DRIVER_SALARY) AS MAX_SALARY FROM DRIVERS;

```
mysql> select MAX(driver_salary) AS Max_Salary from drivers;
+------------+
| Max_Salary |
+------------+
|      60000 |
```

- **Count(*) Function**

  It is used to count all the entered records of a table. We can count that how many records are recorded in database.

  Syntax-1:

  SELECT COUNT(*) FROM DRIVERS;

```
mysql> select count(*) from drivers;
+----------+
| count(*) |
+----------+
|        5 |
+----------+
1 row in set (0.01 sec)
```

Syntax-2:

SELECT COUNT(ROUTE_DISTANCE) FROM ROUTES;

```
mysql> select count(route_distance) from routes;
+-----------------------+
| count(route_distance) |
+-----------------------+
|                     3 |
+-----------------------+
1 row in set (0.00 sec)
```

Operators

In MySQL operators are also used which are used to perform multiple tasks.

- Arithmetic Operators
- Logical Operators
- Relational Operators

Arithmetic Operators

- **"+" & "-" Operators:**
  These are used to show increment/decrement for report representation. Such as you are presenting that you are going to increase/decrease salaries of employees what will be their next salary.
  Syntax:
  SELECT DRIVER_NAME, DRIVER_SALARY + 10000 AS INCREASED, DRIVER_SALARY - 10000 AS DECREASED FROM DRIVERS;

```
mysql> select driver_name, driver_salary + 10000 AS INCREASED, driver_salary - 10000 AS DECREASED from drivers;
+-------------+-----------+-----------+
| driver_name | INCREASED | DECREASED |
+-------------+-----------+-----------+
| Ahad Hasan  |     60000 |     40000 |
| Wahad Hasam |     55000 |     35000 |
| Ghufran Gul |     65000 |     45000 |
| Shafi Hasan |     70000 |     50000 |
| Haiedar Ali |     50000 |     30000 |
+-------------+-----------+-----------+
5 rows in set (0.01 sec)
```

- **"*" & "/" Operators:**
  These are used to find out per day salary, and yearly salary of an employee if we are discussing only employee case. But these are also useful for multiple uses.
  Syntax:
  *SELECT DRIVER_NAME, DRIVER_SALARY * 12 AS YEARLY_SALARY, DRIVER_SALARY / 30 AS PER_DAY_SALARY FROM DRIVERS;*

```
mysql> select driver_name, driver_salary * 12 AS YEARLY_SALARY, driver_salary / 30 AS PER_DAY_SALARY from drivers;
+-------------+---------------+-----------------+
| driver_name | YEARLY_SALARY | PER_DAY_SALARY |
+-------------+---------------+-----------------+
| Ahad Hasan  |        600000 |       1666.6667 |
| Wahad Hasam |        540000 |       1500.0000 |
| Ghufran Gul |        660000 |       1833.3333 |
| Shafi Hasan |        720000 |       2000.0000 |
| Haiedar Ali |        480000 |       1333.3333 |
+-------------+---------------+-----------------+
5 rows in set (0.00 sec)
```

- **AND Operator:**
  It is used to give condition in combination of two conditions. It is widely used operator.
  Syntax:
  *SELECT * FROM DRIVERS WHERE DRIVER_SALARY = 50000 AND DRIVER_HIREDATE = '20150101';*

```
mysql> select * from drivers where driver_salary = 50000 AND driver_hiredate = '20150101';
+-----------+-------------+----------------+---------------+---------------------+-----------------+
| driver_id | driver_name | driver_contact | driver_salary | driver_address      | driver_hiredate |
+-----------+-------------+----------------+---------------+---------------------+-----------------+
|         1 | Ahad Hasan  | +923001122334  |         50000 | 123 street city abc | 2015-01-01      |
+-----------+-------------+----------------+---------------+---------------------+-----------------+
1 row in set (0.00 sec)

mysql> select * from drivers where driver_salary = 50000 AND driver_hiredate = '20140303';
Empty set (0.00 sec)
```

- **OR Operator:**
  It is used to give condition in combination of two conditions. Where one of them is satisfied than result is returned.

Syntax:

*SELECT * FROM DRIVERS WHERE DRIVER_SALARY = 50000 OR DRIVER_HIREDATE = '20140303';*

```
mysql> select * from drivers where driver_salary = 50000 OR driver_hiredate = '20140303';
+-----------+-------------+----------------+---------------+---------------------+-----------------+
| driver_id | driver_name | driver_contact | driver_salary | driver_address      | driver_hiredate |
+-----------+-------------+----------------+---------------+---------------------+-----------------+
|         1 | Ahad Hasan  | +923001122334  |         50000 | 123 street city abc | 2015-01-01      |
|         3 | Ghufran Gul | +923996677330  |         55000 | 789 street city ghi | 2014-03-03      |
+-----------+-------------+----------------+---------------+---------------------+-----------------+
2 rows in set (0.00 sec)
```

- **"<" & ">" Operators:**

  These operators are used to compare values. And are used in logics widely.

  Syntax:

  *SELECT * FROM DRIVERS WHERE DRIVER_SALARY > 40000 AND DRIVER_SALARY < 50000;*

```
mysql> select * from drivers where driver_salary > 40000 AND driver_salary < 50000;
+-----------+-------------+----------------+---------------+---------------------+-----------------+
| driver_id | driver_name | driver_contact | driver_salary | driver_address      | driver_hiredate |
+-----------+-------------+----------------+---------------+---------------------+-----------------+
|         2 | Wahad Hasam | +923110022991  |         45000 | 456 street city def | 2016-02-02      |
+-----------+-------------+----------------+---------------+---------------------+-----------------+
1 row in set (0.00 sec)
```

Views

Views are virtual tables. When you are dealing with multiple users you need to create views to restrict users from accessing irrelevant information.

- Create View
- Replace/Update View
- Enlist all the Views

The above three terms are implemented below.

- **Create View:**
  It is used to create view for users with less limits of data accessing.
  Syntax:
  CREATE VIEW VIEW_DRIVER1 AS SELECT DRIVER_ID, DRIVER_NAME, DRIVER_ADDRESS FROM DRIVERS;

```
mysql> create view view_driver1 AS select driver_id, driver_name, driver_address from drivers;
Query OK, 0 rows affected (0.02 sec)

mysql> select * from view_driver1;
+-----------+-------------+--------------------+
| driver_id | driver_name | driver_address     |
+-----------+-------------+--------------------+
|         1 | Ahad Hasan  | 123 street city abc |
|         2 | Wahad Hasam | 456 street city def |
|         3 | Ghufran Gul | 789 street city ghi |
|         4 | Shafi Hasan | 111 street city xyz |
|         5 | Haiedar Ali | 222 street city jkl |
+-----------+-------------+--------------------+
5 rows in set (0.01 sec)
```

- **Replace View**

  This is updating of view for a user when it has given a new data access.

  Syntax:

  CREATE OR REPLACE VIEW VIEW_DRIVER1 AS SELECT DRIVER_ID, DRIVER_NAME, DRIVER_SALARY,DRIVER_ADDRESS FROM DRIVERS;

```
mysql> create or replace view view_driver1 AS select driver_id, driver_name, driver_salary,driver_address from drivers;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from view_driver1;
+-----------+-------------+---------------+--------------------+
| driver_id | driver_name | driver_salary | driver_address     |
+-----------+-------------+---------------+--------------------+
|         1 | Ahad Hasan  |         50000 | 123 street city abc |
|         2 | Wahad Hasam |         45000 | 456 street city def |
|         3 | Ghufran Gul |         55000 | 789 street city ghi |
|         4 | Shafi Hasan |         60000 | 111 street city xyz |
|         5 | Haiedar Ali |         40000 | 222 street city jkl |
+-----------+-------------+---------------+--------------------+
5 rows in set (0.00 sec)
```

- **Enlist all views**

  To get list of all the views we will use following command.

  Syntax:

  SELECT TABLE_NAME AS VIEW_NAME FROM INFORMATION_SCHEMA.VIEWS WHERE TABLE_SCHEMA = 'BUS_RESERVATION_SYSTEM';

```
mysql> select table_name as view_name from information_schema.views where table_schema = 'bus_reservation_system';
+--------------+
| view_name    |
+--------------+
| view_bus     |
| view_driver1 |
+--------------+
2 rows in set (0.01 sec)
```

Joins In MySQL

Joins are used to retrieve data from more than two tables which are related to each other. There are many types of joins which are used in MySQL. These types are discussed below.

- **Natural Join**
  It is commonly used type of join. It is used to retrieve data from two tables. Suppose you must retrieve data of a driver and type of bus which is assigned to him. You can use Natural Join.
  Syntax:
  SELECT DRIVERS.DRIVER_ID, DRIVERS.DRIVER_NAME, BUSES.BUS_ID, BUSES.BUS_TYPE FROM DRIVERS NATURAL JOIN BUSES WHERE DRIVERS.DRIVER_ID = BUSES.BUS_ID;

```
mysql> select drivers.driver_id, drivers.driver_name, buses.bus_id, buses.bus_type from drivers natural join buses where drivers.driver_id = buses.bus_id;
+-----------+-------------+--------+----------+
| driver_id | driver_name | bus_id | bus_type |
+-----------+-------------+--------+----------+
|         1 | Ahad Hasan  |      1 | AC-Time  |
|         2 | Wahad Hasam |      2 | NULL     |
|         3 | Ghufran Gul |      3 | AC-Time  |
|         4 | Shafi Hasan |      4 | NULL     |
+-----------+-------------+--------+----------+
4 rows in set (0.00 sec)

mysql>
```

- **Left Join**
  It returns all the records of left table and matched records of right table. If there is no match than result will be NULL.
  Syntax:
  SELECT DRIVERS.DRIVER_ID, DRIVERS.DRIVER_NAME, BUSES.BUS_ID, BUSES.BUS_TYPE FROM DRIVERS LEFT JOIN BUSES ON DRIVERS.DRIVER_ID = BUSES.BUS_ID;

```
mysql> select drivers.driver_id, drivers.driver_name, buses.bus_id, buses.bus_type from drivers left join buses on drivers.driver_id = buses.bus_id;
+-----------+-------------+--------+----------+
| driver_id | driver_name | bus_id | bus_type |
+-----------+-------------+--------+----------+
|         1 | Ahad Hasan  |      1 | AC-Time  |
|         2 | Wahad Hasam |      2 | NULL     |
|         3 | Ghufran Gul |      3 | AC-Time  |
|         4 | Shafi Hasan |      4 | NULL     |
|         5 | Haiedar Ali |      5 | N-AC     |
+-----------+-------------+--------+----------+
5 rows in set (0.00 sec)
```

- **Right Join**

  Right join returns all values from right table and matching values from left table. As under definition of join it works as same as a join works mean showing up values from multiple tables.

  Syntax:

  SELECT DRIVERS.DRIVER_ID, DRIVERS.DRIVER_NAME, BUSES.BUS_ID, BUSES.BUS_TYPE FROM DRIVERS RIGHT JOIN BUSES ON DRIVERS.DRIVER_ID = BUSES.BUS_ID;

```
mysql> select drivers.driver_id, drivers.driver_name, buses.bus_id, buses.bus_type from drivers right join buses on drivers.driver_id = buses.bus_id;
+-----------+-------------+--------+----------+
| driver_id | driver_name | bus_id | bus_type |
+-----------+-------------+--------+----------+
|         1 | Ahad Hasan  |      1 | AC-Time  |
|         2 | Wahad Hasam |      2 | NULL     |
|         3 | Ghufran Gul |      3 | AC-Time  |
|         4 | Shafi Hasan |      4 | NULL     |
|         5 | Haiedar Ali |      5 | N-AC     |
+-----------+-------------+--------+----------+
5 rows in set (0.00 sec)
```

- **Inner Join**

  Inner join is used to return the records that has matching values. From both/multiple tables.

  Syntax:

  SELECT DRIVERS.DRIVER_ID, DRIVERS.DRIVER_NAME, BUSES.BUS_ID, BUSES.BUS_TYPE FROM DRIVERS INNER JOIN BUSES ON DRIVERS.DRIVER_ID = BUSES.BUS_ID;

```
mysql> select drivers.driver_id, drivers.driver_name, buses.bus_id, buses.bus_type from drivers inner join buses on drivers.driver_id = buses.bus_id;
+-----------+-------------+--------+----------+
| driver_id | driver_name | bus_id | bus_type |
+-----------+-------------+--------+----------+
|         1 | Ahad Hasan  |      1 | AC-Time  |
|         2 | Wahad Hasam |      2 | NULL     |
|         3 | Ghufran Gul |      3 | AC-Time  |
|         4 | Shafi Hasan |      4 | NULL     |
|         5 | Haiedar Ali |      5 | N-AC     |
+-----------+-------------+--------+----------+
5 rows in set (0.00 sec)
```

- **Full / Outer Join**
  It returns all the records from both tables and returns NULL value where no match is found. MySQL does not support this join. So, we will skip its syntax and further details.
- **Cross Join**
  Cross join is well known type of join which returns the cartesian product of both tables.
  Syntax:
  SELECT DRIVERS.DRIVER_ID, DRIVERS.DRIVER_NAME, ROUTES.ROUTE_ID, ROUTES.ROUTE_DISTANCE FROM DRIVERS CROSS JOIN ROUTES WHERE DRIVERS.DRIVER_ID = ROUTES.DRIVER_ID;

```
mysql> select drivers.driver_id, drivers.driver_name, routes.route_id, routes.route_distance from drivers cross join routes where drivers.driver_id = routes.driver_id;
+-----------+-------------+----------+----------------+
| driver_id | driver_name | route_id | route_distance |
+-----------+-------------+----------+----------------+
|         2 | Wahad Hasam |        1 | 20km           |
|         4 | Shafi Hasan |        2 | 30km           |
|         3 | Ghufran Gul |        3 | 25km           |
|         1 | Ahad Hasan  |        4 | NULL           |
|         1 | Ahad Hasan  |        5 | NULL           |
+-----------+-------------+----------+----------------+
5 rows in set (0.00 sec)
```

User Management in MySQL

In MySQL there is facility of managing different users. User management allows the usage of database by many users and ensures data security related issues.

- **Create a USER**
  First, when we want to allow access to any person there must be someone. So, we will create a user first.
  Syntax:
  CREATE USER 'USERNAME'@'HOSTNAME' IDENTIFIEDBY 'PASSWORD' ;

```
mysql> create user 'SAJAWAL'@'localhost' identified by 'pswd';
Query OK, 0 rows affected (0.02 sec)
```

In above syntax 'pswd' represents the password for user.

- **Drop a USER**
  After creating a user there may be need of removing/deleting a user. For such need we can also drop a user.
  Syntax:
  DROP USER 'USERNAME'@'HOSTNAME' ;

```
mysql> drop user 'SAJAWAL'@'localhost';
Query OK, 0 rows affected (0.01 sec)
```

- **Grant Privileges to USER**
  Now, you can create a user and drop a user. But consider if you must assign jobs to users of doing some interactions with your database. You must read out and observe this pdf again for further queries (HaHaHa). Before sitting in a CEOs chair or a manager's chair carefully read and implement this Manual. Now its turn to grant access to many users depending upon their jobs.

1. **Full Access**

   Full access allows a user to interact with database fully. It mean a single user can View, Update, Alter database.

   Syntax:

   *GRANT ALL PRIVILEGES ON YOUR_DATABASE. * TO 'USERNAME'@'HOSTNAME';*

```
mysql> grant all privileges on bus_reservation_system.* TO 'SAJAWAL'@'localhost';
Query OK, 0 rows affected (0.01 sec)
```

2. **Only View database**

   Only view database allows a user to view a database. User cannot perform any change on database. Such as when you searched your result. You have given this access by your university.

   Syntax:

   *GRANT SPECIFIC_PRIVILEGE ON YOUR_DATABASE. * TO 'USERNAME'@'HOSTNAME';*

```
mysql> create user 'SAJAWAL1'@'localhost' identified by 'pswd';
Query OK, 0 rows affected (0.02 sec)

mysql> grant select on bus_reservation_system.* TO 'SAJAWAL1'@'localhost';
Query OK, 0 rows affected (0.01 sec)
```

3. **Update database/Data Entry Level**

   Data Entry Level (Updating database), this access will allow a user to update database, insert new data in database.

   Syntax:

   *GRANT SPECIFIC_PRIVILEGE_1, SPECIFIC_PRIVILEGE_2 ON YOUR_DATABASE. * TO 'USERNAME'@'HOSTNAME';*

```
mysql> create user 'SAJAWAL2'@'localhost' identified by 'pswd';
Query OK, 0 rows affected (0.02 sec)

mysql> grant update, insert on bus_reservation_system.* TO 'SAJAWAL1'@'localhost';
Query OK, 0 rows affected (0.01 sec)
```

4. **Alter database**

   This is alternation access. It will allow a user to alter database. In the absence of real developer (My Absence). This user will alter database if needed. But for accessing database there should be access of my Laptop (HaHaHa).

   Syntax:

   *GRANT SPECIFIC_PRIVILEGE ON YOUR_DATABASE. * TO 'USERNAME'@'HOSTNAME' ;*

```
mysql> create user 'SAJAWAL3'@'localhost' identified by 'pswd';
Query OK, 0 rows affected (0.02 sec)

mysql> grant alter on bus_reservation_system.* TO 'SAJAWAL1'@'localhost';
Query OK, 0 rows affected (0.01 sec)
```

Now, you have assigned jobs to your employees. But unfortunately, you enjoyed a holydays trip, then you returned. You forgot the jobs and given access to your employees (uff…What to do…?). No thing special…! Just scroll this manual and carefully read further.

- **Show the list of USERS**

  This query will enlist all the users that you created before your trip. So, use this and remember it for later.

  Syntax:

  SELECT USER , HOST FROM MYSQL.USER WHERE USER NOT I ('ROOT' , 'MYSQL.SYS' , 'MYSQL.SESSION') AND HOST = 'HOSTNAME';

```
mysql> select user , host from mysql.user where user not in ('root', 'mysql.sys', 'mysql.session') and host = 'localhost';
+------------------+-----------+
| user             | host      |
+------------------+-----------+
| SAJAWAL          | localhost |
| SAJAWAL1         | localhost |
| SAJAWAL2         | localhost |
| SAJAWAL3         | localhost |
| mysql.infoschema | localhost |
+------------------+-----------+
5 rows in set (0.00 sec)
```

Now, we have discovered all the users. So, what's about jobs of these users.

To explore this concept. We need to dive into it. By getting about info..

- **Showing privileges for a unique user**
  Now, you must get info of a user its job-related info.
  Syntax:
  SHOW GRANTS FOR 'USERNAME'@'HOSTNAME' ;

```
mysql> show grants for 'SAJAWAL'@'localhost';
+-----------------------------------------------------------------------------+
| Grants for SAJAWAL@localhost                                                |
+-----------------------------------------------------------------------------+
| GRANT USAGE ON *.* TO `SAJAWAL`@`localhost`                                 |
| GRANT ALL PRIVILEGES ON `bus_reservation_system`.* TO `SAJAWAL`@`localhost` |
+-----------------------------------------------------------------------------+
2 rows in set (0.01 sec)
```

Now, you want to stuck off an employee from your firm. Then you will use this query to remove grants for that employee.

- **REVOKE**
  This query is used to remove all the grants for a user. Let suppose you assigned a job of data entry to an employee. Now, want stuck off that employee from your firm. You will remove employee's access to database. You will use this query.
  Syntax:
  *REVOKE ALL PRIVILEGES ON *.* FROM 'USERNAME'@'HOSTNAME' ;*

```
mysql> REVOKE ALL PRIVILEGES ON *.* FROM 'SAJAWAL2'@'localhost';
Query OK, 0 rows affected (0.01 sec)
```

Now, stuck off the users:

```
mysql> drop user 'SAJAWAL'@'localhost';
Query OK, 0 rows affected (0.01 sec)

mysql> drop user 'SAJAWAL1'@'localhost';
Query OK, 0 rows affected (0.01 sec)
```

Post Lab Activities

In Post-Lab Tasks following tasks are included for practice from home.

- Revise theory
- Revise diagram designing
- Revise schema designing
- Revise database development steps
- Implement above all queries
- Implement all the mentioned stuff
- Design a database # 1
  An online store tracks products, staff members, income/expenses, and suppliers of products.
- Design a database # 2
  A library tracks books, staff members, publishers, book readers, also offers membership to students.
- Design a database # 3
  A Software house tracks Staff members, Software products, Clients, Resources providers, and also has multiple branches to track in database.