Date: 21/6/24 **EXPERIMENT-1**

1. Introduction to MySQL Workbench.
2. How to use MySQL Workbench to work with a database.
3. How to use MySQL Workbench to run SQL Statements.

1. Introduction to MySQL Workbench.

MySQL Workbench is a graphical tool for working with MySQL servers and databases. MySQL

Workbench fully supports MySQL server versions 5.5 and higher. It is also compatible with older

MySQL server 5.x versions, except in certain situations (like displaying the process list) due to changed system tables. It does not support MySQL server versions 4.x.

- MySQL Workbench is a graphical tool for working with MySQL servers and databases. MySQL Workbench fully supports MySQL server versions 5.5 and higher. It is also compatible with older
- MySQL server 5.x versions, except in certain situations (like displaying the process list) due to changed system tables. It does not support MySQL server versions 4.x.
- MySQL Workbench functionality covers five main topics:
- **SQL Development**: Enables you to create and manage connections to database servers. Along with enabling you to configure connection parameters, MySQL Workbench provides the capability to execute SQL queries on the database connections using the built-in SQL Editor.
- **Data Modeling (Design):** Enables you to create models of your database schema graphically, reverse and forward engineer between a schema and a live database, and edit all aspects of your database using the comprehensive Table Editor. The Table Editor provides easy-to-use facilities for editing Tables, Columns, Indexes, Triggers, Partitioning, Options, Inserts and Privileges, Routines and Views.
- **Server Administration:** Enables you to administer MySQL server instances by administering users, performing backup and recovery, inspecting audit data, viewing database health, and monitoring the MySQL server performance.
- **Data Migration**: Allows you to migrate from Microsoft SQL Server, Microsoft Access, Sybase ASE, SQLite, SQL Anywhere, PostreSQL, and other RDBMS tables, objects and data to MySQL. Migration also supports migrating from earlier versions of MySQL to the latest releases.
- **MySQL Enterprise Support**: Support for Enterprise products such as MySQL Enterprise Backup, MySQL Firewall, and MySQL Audit.
- **MySQL Workbench is available in two editions**: the Community Edition and the Commercial Edition.
- The Community Edition is available free of charge. The Commercial Edition provides additional
- Enterprise features, such as access to MySQL Enterprise Backup, MySQL Firewall, and MySQL Audit.
- MySQL Workbench is a Visual database designing and modeling access tool for MySQL server relational database. It facilitates creation of new physical data models and modification of existing MySQL databases with reverse/forward engineering and change management functions.

MySQL Workbench functionality covers five main topics

SQL Development: Enables you to create and manage connections to database servers. Along with

enabling you to configure connection parameters, MySQL Workbench provides the capability to execute SQL queries on the database connections using the built-in SQL Editor.

Data Modeling (Design): Enables you to create models of your database schema graphically, reverse

and forward engineer between a schema and a live database, and edit all aspects of your database using the comprehensive Table Editor. The Table Editor provides easy-to-use facilities for editing Tables, Columns, Indexes, Triggers, Partitioning, Options, Inserts and Privileges, Routines and Views.

Server Administration: Enables you to administer MySQL server instances by administering users,

performing backup and recovery, inspecting audit data, viewing database health, and monitoring the

MySQL server performance.

Data Migration: Allows you to migrate from Microsoft SQL Server, Microsoft Access, Sybase ASE,

SQLite, SQL Anywhere, PostreSQL, and other RDBMS tables, objects and data to MySQL. Migration also supports migrating from earlier versions of MySQL to the latest releases.

MySQL Enterprise Support: Support for Enterprise products such as MySQL Enterprise Backup,

MySQL Firewall, and MySQL Audit.

MySQL Workbench is available in two editions: the Community Edition and the Commercial Edition.

The Community Edition is available free of charge. The Commercial Edition provides additional

Enterprise features, such as access to MySQL Enterprise Backup, MySQL Firewall, and MySQL Audit.

MySQL Workbench is a Visual database designing and modeling access tool for MySQL server relational database. It facilitates creation of new physical data models and modification of existing MySQL databases with reverse/forward engineering and change management functions.

Getting Started MySQL workbench- Modeling and Design tool Models are at the core of most valid and high performance databases. MySQL workbench has tools that allow developers and database administrators visually create physical database design models that can be easily translated into MySQL databases using forward engineering.

MySQL workbench supports creation of multiple models in the same environment.

It supports all objects such as tables, views, stored procedures, triggers, etc. that make up a database.

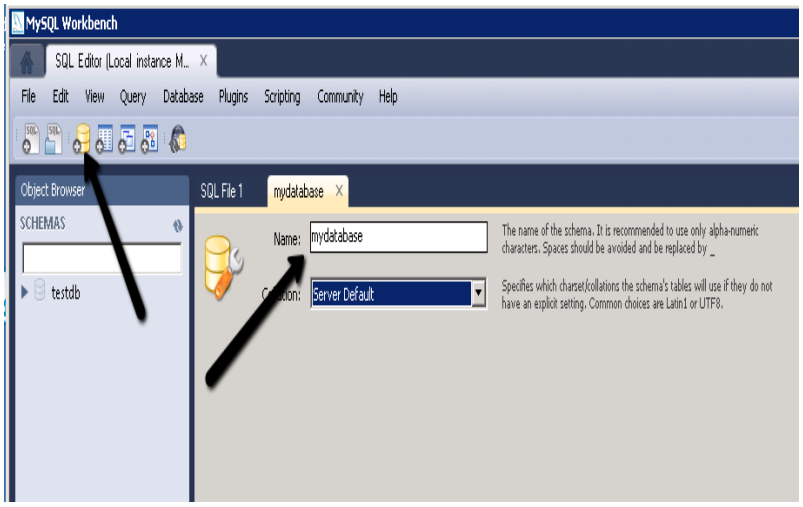
MySQL workbench has a built-in model validating utility that reports any issues that might be found to the data modeler.

It also allows for different modeling notations and can be extended by using LUA a scripting language.

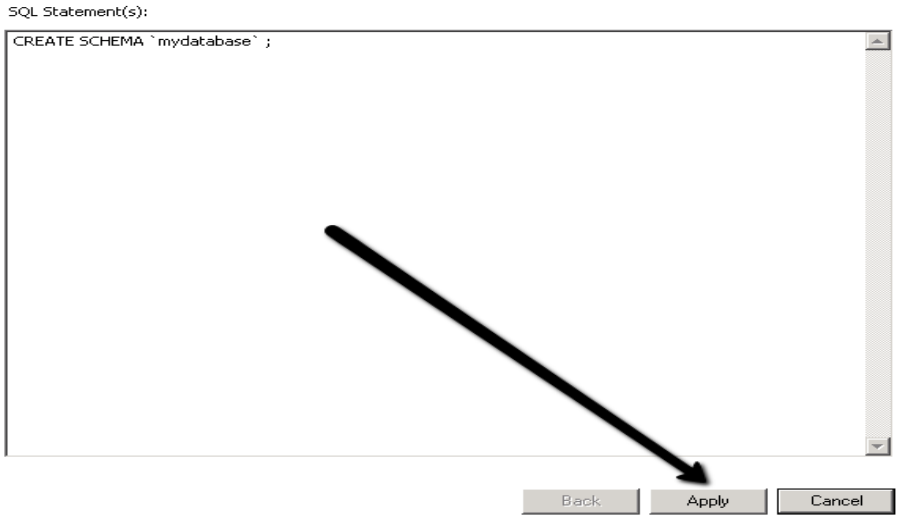
2. How to use MySQL Workbench to work with a database.

MySQL workbench may require a login to your MySQL server. Enter your root or user and password that has been assigned dba server privileges.

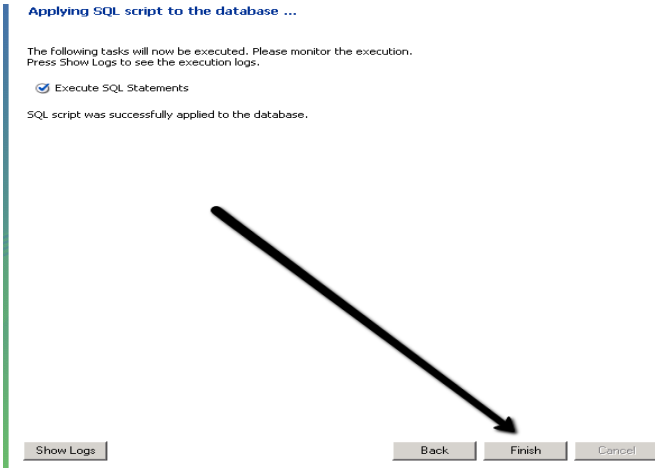
Click on the **New Schema** icon in the menu, and then enter a **name** for your new database in the field as shown. Click the **Apply** button to generate the SQL script.



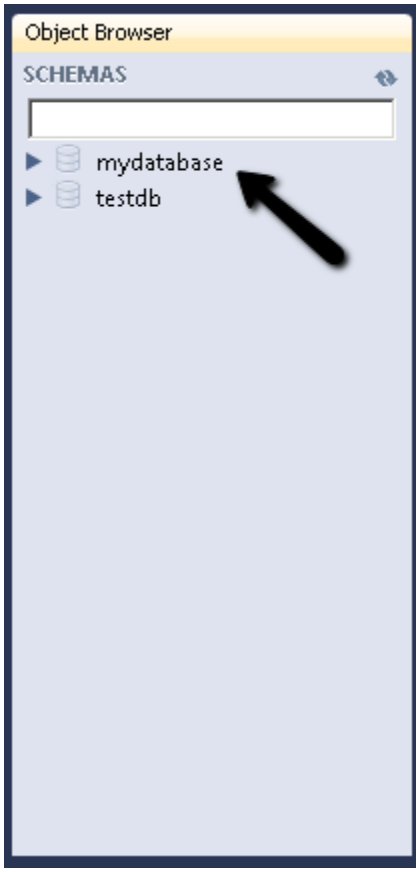
Click the **Apply** button again to execute the create database statement, and create your new database.



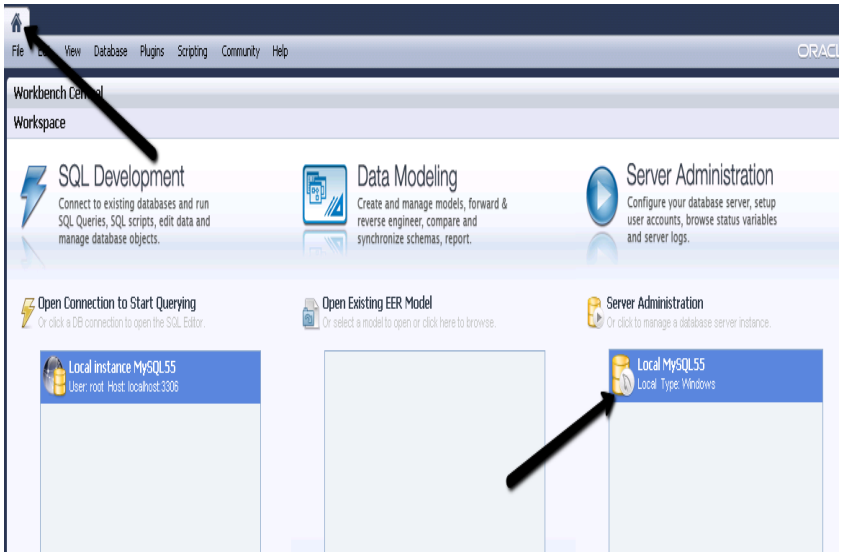
Click **Finish**.



Your database should now be listed on the left with your other database schemas.

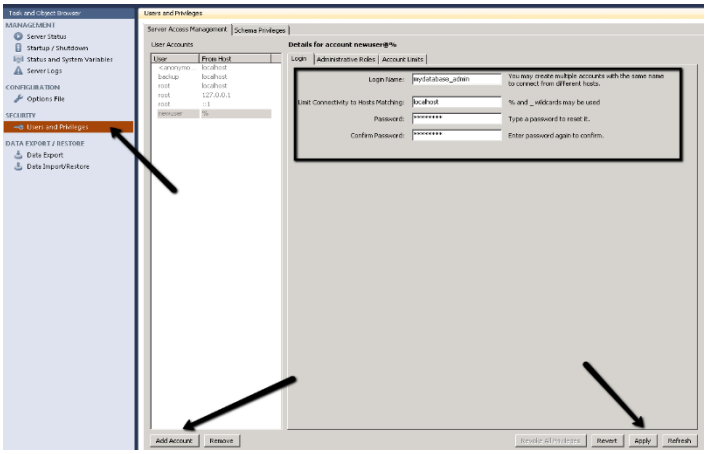


Click the **Home** icon in the top left corner to return to the Workbench Central screen. Click on your MySQL server instance under the **Server Administrator** section of MySQL workbench to create a new database user and assign privileges to your new database.

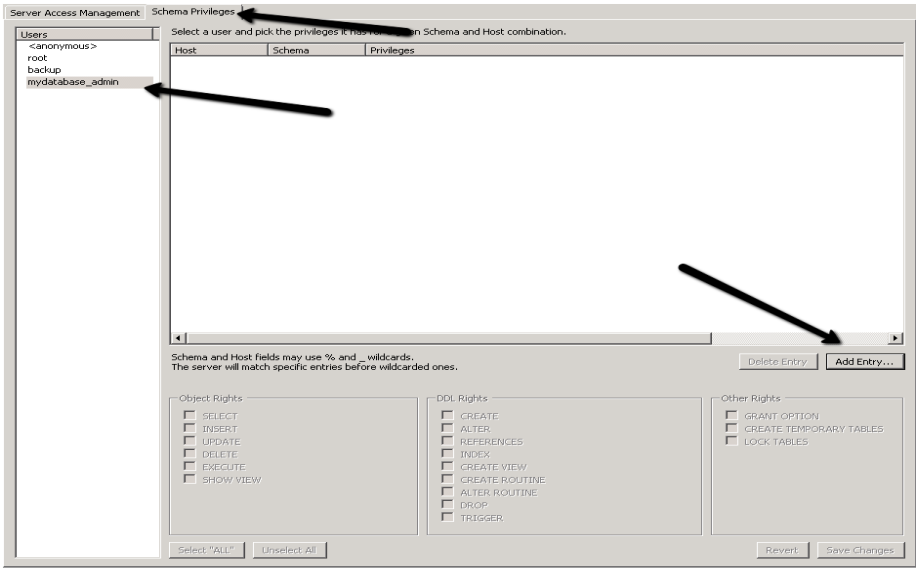


Click on **Users and Privileges**. Then click on **Add Account**. Enter a **login name** for the new user,

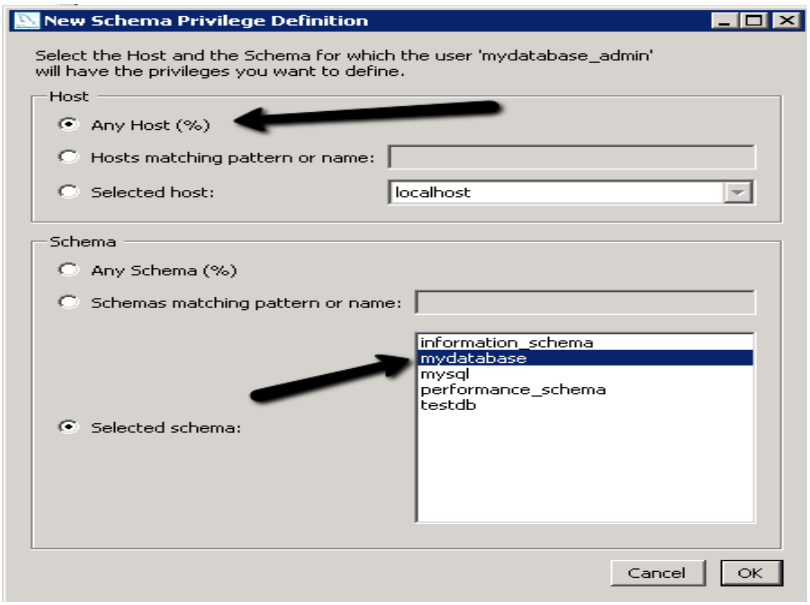
type **localhost** and a new **password** as shown. Click **Apply** to create the new user account.



To assign privileges for this user to access a specific database, click on the **Schema Privileges** tab. **Click the user account** from the list of users on the left. Click the **Add Entry** button.

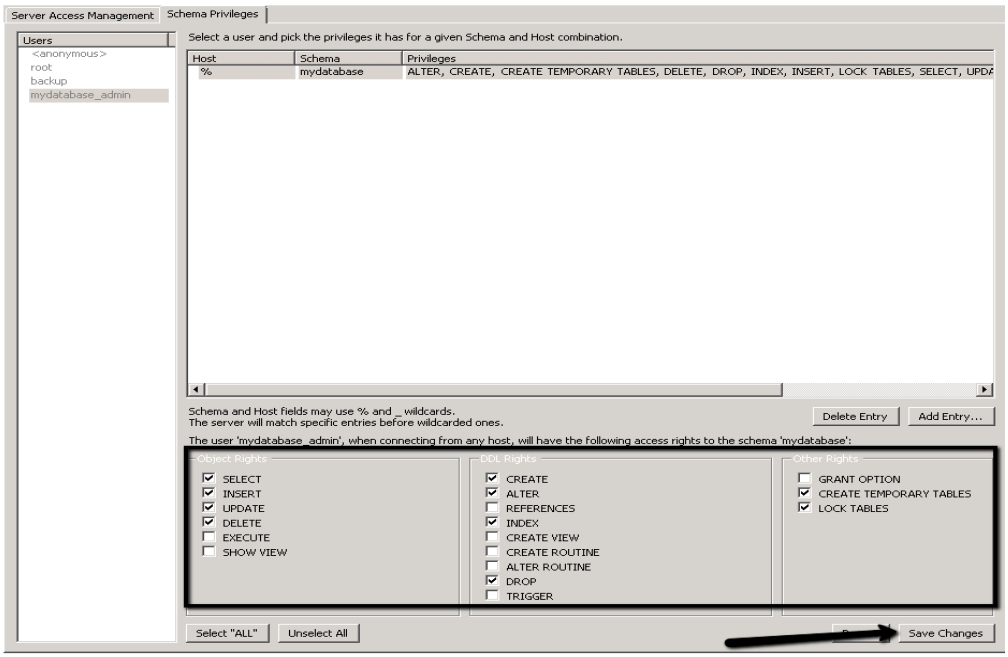


Select the **Selected Schema** radio option, and choose your database schema from the list.



Select the appropriate privileges to allow the user access to the selected database. Most modern website software will only require the permissions listed below. Click Save Changes to complete your new user setup.

Select, Insert, Update, Delete, Create, Alter, Index, Drop, Create Temporary Tables, Lock Tables



3. How to use MySQL Workbench to run SQL Statements.

SQL Query Tab

The SQL query secondary tab opens by default when you make a connection to a server from the Home screen. It includes a query editor area and a toolbar. You can enter SQL statements directly into the query editor area. The statements entered can be saved to a file or snippet for later use. At any point, you can also execute the statements you have entered.

To save a snippet of code entered into the query editor, click **Save SQL to Snippets List** ( ) from the SQL query toolbar, enter a name (optional), and click OK. The following figure shows the main elements of a query tab.

SQL Editor - SQL Query Tab

Executing a SELECT query will display the associated result set in the SQL View panel, directly below the SQL Query panel. These cells are editable if MySQL Workbench is able to determine how, as for example they are editable if a Primary or Unique key exists within the result set. If not, MySQL Workbench will display a "read-only" icon at the bottom-right corner of the SQL View panel, and hovering the mouse cursor over this icon will provide a hint as to why it's not editable.

SQL Query Toolbar

The SQL query toolbar provides actions that enable you to create and manage queries. The following figure shows the set buttons in the toolbar, located within the SQL query tab.

SQL query buttons (from left to right) include:

- **Open a Script File in this Editor**: Loads content from a saved SQL script into the SQL editor.
- **Save SQL Script to File**: Saves contents from the SQL editor into a file.
- **Execute SQL Script**: Executes the selected portion of the query, or the entire query if nothing is selected.
- **Execute Current SQL script**: Execute the statement under the keyboard cursor.
- **Explain (All or Selection)**: Execute the EXPLAIN command on the query under the keyboard cursor.

A result grid tab is also displayed when executing an EXPLAIN statement. Clicking it will execute the same query, as if **Execute SQL Script** was selected.

Alternatively, the Visual Explain plan is already available for all executed queries. Select **Execution Plan** from the results tab to view it.

➔ **Stop the query being executed**: Halts execution of the currently executing SQL script.

Note

The database connection will not be restarted, and open transactions will remain open.

➔ **Toggle whether execution of SQL script should continue after failed statements**: If the red "breakpoint" circle is displayed, the script terminates on a statement that fails. If you click the button so that the green arrow is displayed, execution continues past the failed code, possibly generating additional result sets. In either case, any error generated from attempting to execute the faulty statement is recorded in the Output tab sheet.

This behavior can also be set from the **SQL Execution** user preferences panel.

➔ **Commit**: Commits the current transaction.

Note

All query tabs in the same connection share the same transactions. To have independent transactions, a new connection must be opened.

➔ ***Rollback***: Rolls back the current transaction.

Note

All query tabs in the same connection share the same transactions. To have independent transactions, a new connection must be opened.

➔ **Toggle Auto-Commit Mode**: If selected, each statement will be committed independently.

Note

All query tabs in the same connection share the same transactions. To have independent transactions, a new connection must be opened.

Auto-commit is enabled by default, and this default behavior can be modified (disabled) under the **SQL Execution** user preferences panel.

➔ **Set Limit for Executed Queries**: The default value is 1000, which appends "LIMIT 0, 1000" to SELECT queries.

The default (1000) can be changed from the **SQL Execution** user preferences panel.

- **Save Snippet**: Save the current statement or selection to the active snippet list.
- **Beautify SQL**: Beautify/reformat the SQL script.

By default, SQL keywords are changed to UPPER CASE. This functionality can be changed from the **SQL Editor** user preferences window.

- **Find panel**: Show the Find panel for the editor.
- **Invisible characters**: Toggle display of invisible characters, such as newlines, tabs, spaces.

A new line is represented as ***[LF]***, a space as a single dot (.), and a tab as a right arrow.

- **Wrapping**: Toggles the wrapping of long lines in the SQL editor.

Introduction to Data Types in MySQL. (Character, Integer, Fixed, Floating, Date, Time, ENUM, SET, Large Objects).

Once you have identified all of the tables and columns that the database will need, you should determine each field's MySQL data type. When creating the database, as you will do in the next chapter, MySQL requires that you define what sort of information each field will contain. There are three primary categories, which is true for almost every database software:

- Text
- Numbers
- Dates and times

Within each of these, there are a number of variants—some of which are MySQL-specific—you can use. Choosing your column types correctly not only dictates what information can be stored and how, but also affects the database's overall performance. **Table 3.2** lists most of the available types for MySQL, how much space they take up, and a brief description.

Table 3.2 Here are most of the available column types for use with MySQL databases.

MySQL Datatypes		
Type	**Size**	**Description**
CHAR[Length]	Length bytes	A fixed-length field from 0 to 255 characters long.
VARCHAR(Length)	String length + 1 bytes	A fixed-length field from 0 to 255 characters long.
TINYTEXT	String length + 1 bytes	A string with a maximum length of 255 characters.
TEXT	String length + 2 bytes	A string with a maximum length of 65,535 characters.
MEDIUMTEXT	String length + 3 bytes	A string with a maximum length of 16,777,215 characters.
LONGTEXT	String length + 4 bytes	A string with a maximum length of 4,294,967,295 characters.
TINYINT[Length]	1 byte	Range of -128 to 127 or 0 to 255 unsigned.
SMALLINT[Length]	2 bytes	Range of -32,768 to 32,767 or 0 to 65535 unsigned.
MEDIUMINT[Length]	3 bytes	Range of -8,388,608 to 8,388,607 or 0 to 16,777,215 unsigned.
INT[Length]	4 bytes	Range of -2,147,483,648 to 2,147,483,647 or 0 to 4,294,967,295 unsigned.
BIGINT[Length]	8 bytes	Range of -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 or 0 to 18,446,744,073,709,551,615 unsigned.
FLOAT	4 bytes	A small number with a floating decimal point.
DOUBLE[Length, Decimals]	8 bytes	A large number with a floating decimal point.
DECIMAL[Length, ecimals]	Length + 1 or Length + 2 bytes	A DOUBLE stored as a string, allowing for a fixed decimal point.
DATE	3 bytes	In the format of YYYY-MM-DD.

DATETIME	8 bytes	In the format of YYYY-MM-DD HH:MM:SS.
TIMESTAMP	4 bytes	In the format of YYYYMMDDHHMMSS; acceptable range ends inthe year 2037.
TIME	3 bytes	In the format of HH:MM:SS
ENUM	1 or 2 bytes	Short for enumeration, which means that each column can haveone of several possible values.
SET	1, 2, 3, 4, or 8 bytes	Like ENUM except that each column can have more than one ofseveral possible values.

Date: 28/6/24 **EXPERIMENT-2**

AIM: Examples on
i) DDL Commands: CREATE, ALTER, DROP and TRUNCATE a Table
ii) Implementation of Constraints PRIMARY KEY, FOREIGN KEY, CHECK, NOT NULL, UNIQUE.

i. Examples on DDL Commands CREATE, ALTER, DROP, and TRUNCATE a Table.

DESCRIPTION: Data Definition Language (**DDL**) is a standard for **commands** that define the different structures in a database. **DDL statements** create, modify, and remove database objects such as tables, indexes, and users. Common **DDL statements** are CREATE, ALTER, DROP and TRUNCATE.

a) **Create:**

Syntax:

CREATE TABLE *<tablename> (column_1 datatype(size), column_2 datatype(size)…colum_n datatype(size));*

Example:

create table student(sname varchar(20), sid int(5), sclass int(1), sphone int(10));

b) **Alter:**

The Alter table command is used to add, delete, modify columns, add or drop various constraints on an existing table. In other words, Alter command is used to change the structure of the table.

➔ **Syntax to alter table and add column:**
ALTER TABLE *table_name* ADD *column_name datatype*;

Example:

Alter table student add slname varchar(30);

➔ **Syntax to alter table and delete a column:**
ALTER TABLE *table_name* DROP COLUMN *column_name;*

Example:

Alter table student drop column section;

➔ **Syntax to alter table and modify the column:**
ALTER TABLE *table_name* MODIFY COLUMN *column_name datatype*;

Example:
Alter table student modify column sid varchar(10);

c) **Drop:**

This statement is used to remove a table definition and all the data, indexes, triggers, constraints and permission specifications for that table.

➔ **Syntax:**
DROP TABLE <table_name>

Example:
Drop table student;

d) **Truncate:**
This command is used to delete complete data from table. However, the structure of table is retained.

➔ **Syntax:**
TRUNCATE TABLE <table_name>

Design the following Schema Diagram

Employee

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

Department

Dname	Dnumber	Mgr_ssn	Mgr_Start_date

Dept_locations

Dnumber	Dlocation

Project

Pname	Pnumber	Plocation	Dnum

Works_On

Essn	Pno	Hours

Dependent

Essn	Dependent_name	Sex	Bdate	Relationship

EMPLOYEE SCHEMA

employee:

create table Employee (Fname varchar(10), Minit char(1), Lname varchar(20), Ssn number(9) primary key ,Bdate date ,Address varchar(30) , Sex char(1) ,Salary decimal(5,0),Super_ssn char(9) , Dno int);

Table created

```
mysql> desc employee;
+-----------+--------------+------+-----+---------+-------+
| Field     | Type         | Null | Key | Default | Extra |
+-----------+--------------+------+-----+---------+-------+
| Fname     | varchar(10)  | NO   |     | NULL    |       |
| Minit     | char(1)      | YES  |     | NULL    |       |
| Lname     | varchar(20)  | NO   |     | NULL    |       |
| Ssn       | char(9)      | NO   | PRI | NULL    |       |
| Bdate     | date         | YES  |     | NULL    |       |
| address   | varchar(30)  | YES  |     | NULL    |       |
| Sex       | char(1)      | YES  |     | NULL    |       |
| Salary    | decimal(5,0) | YES  |     | NULL    |       |
| Super_ssn | char(9)      | YES  | MUL | NULL    |       |
| Dno       | int          | NO   |     | NULL    |       |
+-----------+--------------+------+-----+---------+-------+
10 rows in set (0.00 sec)
```

department:

create table department (Dname varchar(15), Dnum int primary key, Mgr_ssn char(9), Mgr_start_date date);

Table Created

```
mysql> desc department;
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| Dname          | varchar(15) | NO   | UNI | NULL    |       |
| Dnumber        | int         | NO   | PRI | NULL    |       |
| Mgr_ssn        | char(9)     | NO   | MUL | NULL    |       |
| Mgr_start_date | date        | YES  |     | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

dept_locations:

create table dept_locations(Dnumber int primary key, Dlocation varchar(15) primary key, phone_no int);

Table Created

```
mysql> desc dept_locations;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| Dnumber   | int         | NO   | PRI | NULL    |       |
| Dlocation | varchar(15) | NO   | PRI | NULL    |       |
| phone_no  | int         | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

project:

create table project (Pname varchar(15) , Pnumber int primary key ,Plocation varchar(15), Dnum int);

Table created

```
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| Pname     | varchar(15) | NO   | UNI | NULL    |       |
| Pnumber   | int         | NO   | PRI | NULL    |       |
| Plocation | varchar(15) | YES  |     | NULL    |       |
| Dnum      | int         | NO   | MUL | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

works_on:

create table works_on(Essn char(9) primary key, Pno int primary key, Hours decimal(3,1));

Table created

```
mysql> desc workers_on;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| Essn  | char(9)      | NO   | PRI | NULL    |       |
| Pno   | int          | NO   | PRI | NULL    |       |
| Hours | decimal(3,1) | NO   |     | NULL    |       |
+-------+--------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

Dependent:

create table dependent (Ssn number(10), Deptname varchar(20), sex varchar(2), Bdate date, Relationship varchar(20) primary key);

Table Created

```
mysql> desc dependent;
+----------------+-------------+------+-----+---------+-------+
| Field          | Type        | Null | Key | Default | Extra |
+----------------+-------------+------+-----+---------+-------+
| Essn           | char(9)     | NO   | PRI | NULL    |       |
| Dependent_name | varchar(15) | NO   | PRI | NULL    |       |
| Sex            | char(1)     | YES  |     | NULL    |       |
| Bdate          | date        | YES  |     | NULL    |       |
| Relationship   | varchar(8)  | YES  |     | NULL    |       |
+----------------+-------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

Cmd: alter

➔ **Syntax:**

- add new columns-alter table table_name add(new colname1 datatype(size),…..…….....colname datatype(size));
- modify the existing- table table_name modify (colname new datatype(new size));
- adding constraints-
    1. alter table table_name add (primary key (colname));
    2. alter table tablename add (foreign key(colname) references tablename (colname));

Use: modifies the structure of database.

Employee:

alter table emp modify (Ssn primary key, Dno references dept(Dnum));

Department:

alter table dept modify(mssn references emp(ssn),dnum primarykey));

Department_location:

alter table depl modify(foreignkey (dnum) references dept(dnum));

Project:

alter table project modify (foreignkey(numd) references dept(dnum));

Works_on:

alter table works_on modify(foreignkey(essn) references emp(ssn), foreignkey(pnum) references project(pno));

Dependent:

alter table dependent modify (foreignkey(ssn) references emp(ssn));

cmd: insert

➔ **Syntax:** insert into tablename values(colname1, colname2, colname3,…..);

➔ **Use:** This command is used to insert all the values into the table.

Employee:

INSERT INTO EMPLOYEE VALUES('John','B','Smith','123456789','1965-01-09','731 Fondren,Houston,TX','M',30000,'333445555',5);

INSERT INTO EMPLOYEE VALUES('Franklin','T','Wong','333445555','1955-12-08','638 Voss,Houston,TX','M',40000,'888665555',5);

INSERT INTO EMPLOYEE VALUES('Alicia','J','Zelaya','999887777','1968-01-19','3321 Castle,Spring,TX','F',25000,'987654321',4);

INSERT INTO EMPLOYEE VALUES('Jennifer','S','Wallace','987654321','1941-06-20','291 Berry,Bellaire,TX','F',43000,'888665555',4);

INSERT INTO EMPLOYEE VALUES('Ramesh','K','Narayan','666884444','1962-09-15','975 Fire Oak,Humble,TX','M',38000,'333445555',5);

INSERT INTO EMPLOYEE VALUES('Ahmad','V','Jabbar','987987987','1969-03-29','980 Dallas,Houston,TX','M',25000,'987654321',4);

INSERT INTO EMPLOYEE VALUES('James','E','Borg','888665555','1937-11-10','450 Stone,Houston,TX','M',55000,'NULL',1);

```
mysql> select * from employee;
+---------+------+---------+-----------+------------+----------------------+------+--------+-----------+-----+
| Fname   | Minit | Lname   | Ssn       | Bdate      | address              | Sex  | Salary | Super_ssn | Dno |
+---------+------+---------+-----------+------------+----------------------+------+--------+-----------+-----+
| John    | B    | Smith   | 123456789 | 1965-01-09 | 731 Fondren, Houston TX | M  |  30000 | 333445555 |   5 |
| Franklin | T   | Wong    | 333445555 | 1965-12-08 | 638 Voss, Houston TX   | M  |  35000 | 888665555 |   5 |
| Ramesh  | K    | Narayan | 666884444 | 1962-09-15 | 975 Fire Oak, Humble TX | M  |  38000 | 333445555 |   5 |
| James   | E    | Borg    | 888665555 | 1937-11-10 | 450 Stone, Houston TX  | M  |  55000 | NULL      |   1 |
| Jennifer | S   | Wallace | 987654321 | 1941-06-20 | 291 Berry, Bellaire TX | F  |  43000 | 888665555 |   4 |
| Ahmad   | V    | Jabbar  | 987987987 | 1969-03-29 | 980 Dallas, Houston TX | M  |  25000 | 987654321 |   4 |
| Alicia  | J    | Zelaya  | 999887777 | 1968-01-19 | 3321 Castle, Spring TX | F  |  25000 | 987654321 |   4 |
+---------+------+---------+-----------+------------+----------------------+------+--------+-----------+-----+
7 rows in set (0.00 sec)
```

Department:

INSERT INTO DEPARTMENT VALUES ('Research',5,'333445555','1988-05-22');

INSERT INTO DEPARTMENT VALUES('Administration',4,'987654321','1995-01-01');

INSERT INTO DEPARTMENT VALUES('Headquarters',1,'888665555','1981-06-19');

```
mysql> select * from department;
+----------------+---------+-----------+-----------------+
| Dname          | Dnumber | Mgr_ssn   | Mgr_start_date  |
+----------------+---------+-----------+-----------------+
| Headquarters   |       1 | 888665555 | 1981-06-19      |
| Administration |       4 | 987654321 | 1995-01-01      |
| Research       |       5 | 333445555 | 1988-05-22      |
+----------------+---------+-----------+-----------------+
3 rows in set (0.00 sec)
```

Dept_loc:

INSERT INTO DEPT_LOCATIONS VALUES(1,'Houston');

INSERT INTO DEPT_LOCATIONS VALUES(4,'Stafford');

INSERT INTO DEPT_LOCATIONS VALUES(5,'Bellaire');

INSERT INTO DEPT_LOCATIONS VALUES(5,'Sugarland');

INSERT INTO DEPT_LOCATIONS VALUES(5,'Houston');

```
mysql> select * from dept_locations;
+---------+-----------+----------+
| Dnumber | Dlocation | phone_no |
+---------+-----------+----------+
|       1 | Houston   |     NULL |
|       4 | Stafford  |     NULL |
|       5 | Bellaire  |     NULL |
|       5 | Houston   |     NULL |
|       5 | Sugarland |     NULL |
+---------+-----------+----------+
5 rows in set (0.09 sec)
```

Project:

INSERT INTO PROJECT VALUES('ProductX',1,'Bellaire',5);

INSERT INTO PROJECT VALUES('ProductY',2,'Sugarland',5);

INSERT INTO PROJECT VALUES('ProductZ',3,'Houston',5);

INSERT INTO PROJECT VALUES('Computerization',10,'Stafford',4);

INSERT INTO PROJECT VALUES('Reorganization',20,'Houston',1);

INSERT INTO PROJECT VALUES('Newbenefits',30,'Stafford',4);

```
mysql> select * from project;
+-----------------+---------+-----------+------+
| Pname           | Pnumber | Plocation | Dnum |
+-----------------+---------+-----------+------+
| ProductX        |       1 | Bellaire  |    5 |
| ProductY        |       2 | Sugarland |    5 |
| ProductZ        |       3 | Houston   |    5 |
| Computerization |      10 | Bellaire  |    5 |
| Reorganization  |      20 | Houston   |    1 |
| Newbenefits     |      30 | Stafford  |    4 |
+-----------------+---------+-----------+------+
6 rows in set (0.00 sec)
```

Works_on:

INSERT INTO WORKS_ON VALUES('123456789',1,32.5);

INSERT INTO WORKS_ON VALUES('123456789',2,7.5);

INSERT INTO WORKS_ON VALUES('666884444',3,40.0);

INSERT INTO WORKS_ON VALUES('453453453',1,20.0);

INSERT INTO WORKS_ON VALUES('453453453',2,20.0);

INSERT INTO WORKS_ON VALUES('333445555',2,10.0);

INSERT INTO WORKS_ON VALUES('333445555',3,10.0);

INSERT INTO WORKS_ON VALUES('333445555',10,10.0);

INSERT INTO WORKS_ON VALUES('333445555',20,10.0);

INSERT INTO WORKS_ON VALUES('999887777',30,30.0);

INSERT INTO WORKS_ON VALUES('999887777',10,10.0);

INSERT INTO WORKS_ON VALUES('987987987',10,35.0);

INSERT INTO WORKS_ON VALUES('987987987',30,5.0);

INSERT INTO WORKS_ON VALUES('987654321',30,20.0);

INSERT INTO WORKS_ON VALUES('987654321',20,15.0);

```
mysql> select * from workers_on;
+-----------+-----+-------+
| Essn      | Pno | Hours |
+-----------+-----+-------+
| 123456789 |   1 |  32.5 |
| 123456789 |   2 |   7.5 |
| 333445555 |   2 |  10.0 |
| 333445555 |   3 |  10.0 |
| 333445555 |  10 |  10.0 |
| 333445555 |  20 |  10.0 |
| 666884444 |   3 |  40.0 |
| 888665555 |  20 |  16.0 |
| 987654321 |  20 |  15.0 |
| 987654321 |  30 |  20.0 |
| 987987987 |  10 |  35.0 |
| 987987987 |  30 |   5.0 |
| 999887777 |  10 |  10.0 |
| 999887777 |  30 |  30.0 |
+-----------+-----+-------+
14 rows in set (0.00 sec)
```

Dependent:

INSERT INTO DEPENDENT VALUES('333445555','Alice','F','1986-04-05','Daughter');

INSERT INTO DEPENDENT VALUES('333445555','Theodore','M','1983-10-25','Son');

INSERT INTO DEPENDENT VALUES('333445555','Joy','F','1958-05-03','Spouse');

INSERT INTO DEPENDENT VALUES('987654321','Abner','M','1942-02-28','Spouse');

INSERT INTO DEPENDENT VALUES('123456789','Michael','M','1988-01-04','Son');

INSERT INTO DEPENDENT VALUES('123456789','Alice','F','1988-12-30','Daughter');

INSERT INTO DEPENDENT VALUES('123456789','Elizabeth','F','1967-05-05','Spouse');

```
mysql> select * from dependent;
+-----------+----------------+------+------------+--------------+
| Essn      | Dependent_name | Sex  | Bdate      | Relationship |
+-----------+----------------+------+------------+--------------+
| 123456789 | Alice          | F    | 1988-12-30 | Daughter     |
| 123456789 | Elizabeth      | F    | 1967-05-05 | Spouse       |
| 123456789 | Michael        | M    | 1988-01-04 | Son          |
| 333445555 | Alice          | F    | 1986-04-04 | Daughter     |
| 333445555 | Joy            | F    | 1958-05-03 | Spouse       |
| 333445555 | Theodore       | M    | 1983-10-25 | Son          |
| 987654321 | Abner          | M    | 1942-02-28 | Spouse       |
+-----------+----------------+------+------------+--------------+
7 rows in set (0.08 sec)
```

Cmd: drop

- ➔ **Syntax:** drop table tablename;
- ➔ **Use:** destroying of all tables is done using this command.

Employee:

drop table emp;

Department:

drop table dept;

Department_loc:

drop table depl;

Project:

drop table project;

Works_on:

drop table works_on;

Dependent:

drop table dependent;

mysql>>alter table employee drop column dept_location;
query OK, 0 rows affected (0.60 sec)
Records: 0 Duplicates: 0 Warnings: 0
mysql>>desc employee;

```
mysql> desc employee;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| Fname     | varchar(10) | NO   |     | NULL    |       |
| Minit     | char(1)     | YES  |     | NULL    |       |
| Lname     | varchar(20) | NO   |     | NULL    |       |
| Ssn       | char(9)     | NO   | PRI | NULL    |       |
| Bdate     | date        | YES  |     | NULL    |       |
| address   | varchar(30) | YES  |     | NULL    |       |
| Sex       | char(1)     | YES  |     | NULL    |       |
| Salary    | decimal(5,0)| YES  |     | NULL    |       |
| Super_ssn | char(9)     | YES  | MUL | NULL    |       |
| Dno       | int         | NO   |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
10 rows in set (0.00 sec)
```

mysql>> truncate table employee;
query OK, 0 rows affected (0.29 sec)

ii. Examples on Implementation of Constraints PRIMARY KEY, FOREIGN KEY, CHECK, NOT NULL, UNIQUE.

DESCRIPTION: Constraints are the rules that are used to limit the type of data that can go into a table, to maintain accuracy and integrity of the data inside table. There are two types of constraints: **Column level constraints** and **Table level constraints** depending upon the range up to which their conditions hold. Constraints are used to make sure that the integrity of data is maintained. The following are various constraints:

a) **Primary Key:** Primary key constraint is used to uniquely identify each record in a database. A primary key must contain a unique value and it must not contain a null value. Primary key is usually used to index data inside a table.

➔ **Syntax: (to add primary key constraint while creating a table)**
Create table <tablename> (column_1 datatype(size) Primary key, column_2 datatype(size)…column_n datatype(size));

Example:
Create table student (sidint(5) primary key, sname varchar(30), sdept varchar(4));

➔ **Syntax** : col_name datatype PRIMARYKEY;

➔ **Syntax: (to add primary key constraint using alter table command)**
Alter table <tablename> add primary key (column_name)

Example: Alter table employee add primary key (eid);

b) **Foreign Key:** Foreign Key is used to establish relation between two tables. It is used to ensure **referential integrity** between tables. The foreign key is usually a column that is linked with the primary key of parent table. And once a column is declared to be as foreign key, it doesn't allow inclusion of details which are not present in the primary key of parent table. In this way it ensures referential integrity.

➔ **Syntax: (to add foreign key using alter table command)**
Alter table <tablename> add foreign key (column_child table) references parent table (column_parent table);

Example:
Alter table employeebonus add foreign key(esuperid) references employee(eid);

➔ **Syntax**: (datatype) REFERENCES (datatype);

c) Not Null: NOT NULL constraint restricts a column from having a NULL value. Once **NOT NULL** constraint is applied to a column, you cannot pass a null value to that column. It enforces a column to contain a proper value.

➔ **Syntax:**
Create table <table-name>(column_1 datatype(size) NOT NULL, column_2 datatype(size)…column_n datatype(size));

Example:
Create table student (sidint(5) NOT NULL, sname varchar(30), sdept varchar(3));

d) Unique: Unique constraint prevents the user from adding duplicate values in the column.

➔ **Syntax: (to add unique constraint while creating a table)**
CREATE TABLE tablename(column_1datatype(size) NOT NULL,column_2 datatype(size)… column_n datatype(size), UNIQUE(columnname));

Example:
CREATE TABLE Persons (IDint NOT NULL, LastName varchar(25) NOT NULL,FirstName varchar(25),Age int,UNIQUE (ID));

➔ **Syntax: (to add unique constraint using alter command)**
Alter table <table-name> add unique (column-name);

Example:
Alter table persons add unique (ID);

e) Check: CHECK constraint is used to restrict the value of a column within a range. It performs check on the values, before storing them into the database. It is like condition checking before saving data into a column.

➔ **Example to apply check constraint while creating the table:**
Create table student(sidint(5) NOT NULL Check(sid>0), sname varchar(30) NOT NULL, age int(2));

Example to apply check constraint using alter command:
Alter table student add check (sid>0);

mysql>> create table department(Dname varchar(20), Dnumberint(1), Mgr_ssnint(10,Mgr_start_date date);
Query OK, 0 rows affected (0.29 sec)

mysql>>desc department;

```
mysql> select * from department;
+----------------+---------+-----------+----------------+
| Dname          | Dnumber | Mgr_ssn   | Mgr_start_date |
+----------------+---------+-----------+----------------+
| Headquarters   |       1 | 888665555 | 1981-06-19     |
| Administration |       4 | 987654321 | 1995-01-01     |
| Research       |       5 | 333445555 | 1988-05-22     |
+----------------+---------+-----------+----------------+
3 rows in set (0.00 sec)
```

mysql>>insert into department values(“ Research”,5,333445555,”1988-05-22”);
Query OK, 1 row affected (0.07 sec)

mysql>>insert into department values(“Administration”,4,987654321,”1955-01-01”);
Query OK, 1 row affected (0.03 sec)

mysql>>insert into department values(“Headquarters”,1,888665555,”1981-06-19”);
Query OK, 1 row affected (0.08 sec)

mysql>>create table dept_locations (Dnumberint(1), Dlocation varchar(20));
Query OK, 0 rows affected (0.32 sec)

mysql>> alter table department add primary key(Dnumber);
Query OK, 0 rows affected (0.64 sec)
Records: 0 Duplicates:0 Warnings:0

mysql>> alter table dept_locations add foreign key (Dnumber) references department (Dnumber);
Query OK, 0 rows affected (0.97 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql>insert into dept_locations values(1,”Houston”);
Query OK, 1 row affected (0.07 sec)

mysql> insert into dept_locations values(4,”Stafford”);

Query OK, 1 row affected (0.05 sec)

mysql>insert into dept_locations values(5,”Bellaire”);
Query OK, 1 row affected (0.07 sec)

mysql>insert into department values(“Development”,5,987654322,”1995-02-01”);
Query OK, 1 row affected (0.07 sec)

mysql>create table project (Pname varchar(20), Pnumberint(3) NOT NULL, Plocation varchar(30), Dnumint(1));
Query OK, 0 rows affected (0.32 sec)

mysql>insert into project values (“ProductX”,1,”Bellaire”,5);
Query OK, 1 row affected (0.05 sec)

mysql> insert into project values("ProductY",2,"Sugarland",5);
Query OK, 1 row affected (0.09 sec)

mysql> insert into project values("ProductZ",3,NULL,5);
Query OK, 1 row affected (0.06 sec)

mysql>alter table project add unique (Pnumber);
Query OK, 0 rows affected (0.79 sec)
Records: 0 Duplicates: 0 Warnings: 0

Date: 5/7/24 **EXPERIMENT- 3**

AIM: Examples on

i. DML - INSERT, UPDATE, DELETE.

ii. DCL - COMMIT, ROLLBACK, SAVEPOINT.

DESCRIPTION:

DML: SELECT, INSERT, UPDATE, DELETE

Cmd: Update

- **Syntax:** Update table set colname=expression, colname=exp;
- **Use:** It updates all the columns given based upon expression value.

1. Update the salary of employees with ssn '999887777' to 28000

update employee set salary=28000 where ssn=999887777;

```
mysql> update employee set salary=28000 where ssn=999887777;
Query OK, 1 row affected (0.07 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

2.Update the dep_no of employee '999887777' ssn to 1.

update department set dnumber=1 where ssn=999887777;

```
mysql> delete from employee where ssn=9998887777;
Query OK, 0 rows affected (0.00 sec)
```

3. Show that the resulting salaries of every emp working on the project 'productX' is given a 10% rise.

select salary+0.1*salary hike, dno from employee, project where employee.dno=project.dnum and pname="productX";

```
+---------+-----+
| hike    | dno |
+---------+-----+
| 33000.0 |   5 |
| 38500.0 |   5 |
| 41800.0 |   5 |
+---------+-----+
3 rows in set (0.00 sec)
```

4.Update salary of all employees to 40000 .

Update employee set salary=40000;

```
mysql> Update employee set salary=40000;
Query OK, 1 row affected (0.01 sec)
Rows matched: 7  Changed: 1  Warnings: 0
```

5.Change the location and controlling dept.no of project no 10 to Bellarie and 5.

Update project set plocation= "Bellarie",dnum=5 where project.pnumber=10;

```
mysql> Update project set plocation= "Bellarie",dnum=5 where project.pnumber=10;
Query OK, 0 rows affected (0.08 sec)
Rows matched: 1  Changed: 0  Warnings: 0
```

Cmd: Select

➔ **Syntax:**

- Select colname1,colname2,…..colnamen from tablename;
- Select * from tablename;
- Select colname1,colname2,….colnamen from tablename where cond;

1.Retrieve birth date and address of all employees whose name is 'John B Smith'

Select bdate , address from employee where fname="John" and minit="B" and lname="Smith";

```
+------------+-------------------------+
| bdate      | address                 |
+------------+-------------------------+
| 1965-01-09 | 731 Fondren, Houston TX |
+------------+-------------------------+
1 row in set (0.00 sec)
```

2.Retrieve the name and address of all employees whose dept no is 5.

Select fname , min, address from emp where dno=5;

```
+----------+-------+-------------------------+
| fname    | minit | address                 |
+----------+-------+-------------------------+
| John     | B     | 731 Fondren, Houston TX |
| Franklin | T     | 638 Voss, Houston TX    |
| Ramesh   | K     | 975 Fire Oak, Humble TX |
+----------+-------+-------------------------+
3 rows in set (0.00 sec)
```

3.For each employee, retrieve the emp's fname and lname and his /her immediate supervisor.

Select e.fname ,e.lname,(Select s.fname from employee s where s.ssn = e.super_ssn) as super_fname ,(select s.lname from employee s where s.ssn = e.super_ssn) as super_lname from employee e;

fname	lname	super_fname	super_lname
John	Smith	Franklin	Wong
Franklin	Wong	James	Borg
Ramesh	Narayan	Franklin	Wong
James	Borg	NULL	NULL
Jennifer	Wallace	James	Borg
Ahmad	Jabbar	Jennifer	Wallace
Alicia	Zelaya	Jennifer	Wallace

7 rows in set (0.00 sec)

4.Retrieve the sal of all employees .

Select salary from employee;

salary
40000
40000
40000
40000
40000
40000
40000

Cmd: delete

- ➔ **Syntax:** delete from tablename;

  delete from tablename where search condition;
- ➔ **Use:** To delete the particular row or all the columns data.

1. delete the employee tuple in ssn '999887777 '.

delete from employee where ssn=9998887777;

2. delete only those rows that have lname Smith.

delete from employee where lname="Smith";

3. delete the rows from works_on table where hours is 32.5.

delete from workers_on where hours=32.5;

4. delete the rows from project where pname is productX.

delete from project where pname=’productX’;

Cmd: in

- ➔ **Syntax:** select colname1, colname2 ….colnamen where colname in (condition);
- ➔ **wUse:** To retrieve the columns present in many conditions

1. Retrieve the social security numbers of all employees who works on project numbers 1, 2 ,3.

select ssn from employee ,project where employee.dno=project.dnum and pnumber in(1,2,3);

ssn
123456789
123456789
123456789
333445555
333445555
333445555
666884444
666884444
666884444

2. Retrieve the emp’s tuple where employee lname is smith or Wallace.

select fname,lname from employee where lname in ("smith" , "wallace");

fname	lname
John	Smith
Jennifer	Wallace

ii. DCL: COMMIT WORK, ROLLBACK,SAVEPOINT Commands.

Aim: Insert data into table using COMMIT, ROLLBACK, SAVE POINT, in PL/SQL block

Description: COMMIT command is used to make changes permanently save to database during the current transaction

ROLLBACK command is used to execute at the end of current transaction and undo any changes made since the begin transaction.

SAVE POINT saves current point with the unique name in the transaction.

1. Write a query to implement the save point

SAVEPOINT S1;

Savepoint created.

SQL> select * from employee;

Fname	Minit	Lname	Ssn	Bdate	address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston TX	M	40000	333445555	5
Franklin	T	Wong	333445555	1965-12-08	638 Voss, Houston TX	M	40000	888665555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble TX	M	40000	333445555	5
James	E	Borg	888665555	1937-11-10	450 Stone, Houston TX	M	40000	NULL	1
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire TX	F	40000	888665555	4
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston TX	M	40000	987654321	4
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring TX	F	40000	987654321	4

SQL>INSERT INTO EMPLOYEE VALUES('Amith','B','Smith','234567891','1967-01-09','734 Fondren,Houston,TX','M',30000,'333445556',5);

2.Write a query to implement the rollback

Ans: SQL> rollback s1;

SQL> select * from employee;

Fname	Minit	Lname	Ssn	Bdate	address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston TX	M	40000	333445555	5
Franklin	T	Wong	333445555	1965-12-08	638 Voss, Houston TX	M	40000	888665555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble TX	M	40000	333445555	5
James	E	Borg	888665555	1937-11-10	450 Stone, Houston TX	M	40000	NULL	1
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire TX	F	40000	888665555	4
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston TX	M	40000	987654321	4
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring TX	F	40000	987654321	4

3.Write a query to implement the commit

SQL> COMMIT;

Commit complete.

```
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

Date: 12/7/24 **EXPERIMENT -4**

AIM: Examples on retrieving data from a single table using

i) SELECT statement with where clause(AND, OR, NOT, IN, BETWEEN,LIKE)

ii) ORDER BY clause(sort by column name)

iii) LIMIT clause

i) SELECT statement with where clause(AND, OR, NOT, IN, BETWEEN,LIKE)

Description: AND and OR operators in SQL can be combined to test for multiple conditions in a SELECT, INSERT, UPDATE, or DELETE statement. When combining these conditions, it is important to use parentheses so that the database knows what order to evaluate each condition.

Cmd: AND

- **Syntax:** select colname1, colname2……. Colnamen from tablename where cond1 and cond2;
- **Use:** This selects the combined result from both conditions.

1. Retrieve details from works_on table whose ssn is 333445555 and pno is 10.

select * from workers_on where Essn=333445555 and Pno=10;

```
mysql> select * from workers_on where Essn=333445555 and Pno=10;
+-----------+-----+-------+
| Essn      | Pno | Hours |
+-----------+-----+-------+
| 333445555 |  10 |  10.0 |
+-----------+-----+-------+
1 row in set (0.01 sec)
```

Cmd: OR

- **Syntax:** select colname1, colname2….. colnamen from tablename where cond1 or cond2;
- **Use:** This selects the tuples which are common in both cond1, cond2.

2. Retrieve the emp's whose minit is A or B or K.

select fname,lname,minit from employee where minit='A' or minit='B' or minit='K';

```
mysql> select fname,lname,minit from employee where minit='A' or minit='B' or minit='K';
+--------+---------+-------+
| fname  | lname   | minit |
+--------+---------+-------+
| John   | Smith   | B     |
| Joyce  | English | A     |
| Ramesh | Narayan | K     |
+--------+---------+-------+
```

Cmd: Between

➔ **Syntax:** between value1 or/and value

➔ **Use**: It get the values between two ranges.

3. Retrieve all employees of dep whose salary is between 30000 and 40000.

select Ssn,Salary from employee e where e.Dno=5 and e.Salary between 30000 and 40000;

```
mysql> select Ssn,Salary from employee e where e.Dno=5 and e.Salary between 30000 and 40000;
+-----------+--------+
| Ssn       | Salary |
+-----------+--------+
| 123456789 |  30000 |
| 333445555 |  40000 |
| 666884444 |  38000 |
+-----------+--------+
3 rows in set (0.00 sec)
```

Cmd: Like

➔ **Syntax:**

- %: matching any string
- -: underscore for any character

➔ **Use:** it allows comparison of one string value with another string value which is not identical . This can be achieved by using wild card characters like %,-.\

4. Retrieve all employees whose address is' Houston,TX'.

select Ssn,Address from employee where Address like '%Houston TX';

```
mysql> select Ssn,Address from employee where Address like '%Houston TX';
+-----------+-------------------------+
| Ssn       | Address                 |
+-----------+-------------------------+
| 123456789 | 731 Fondren, Houston TX |
| 333445555 | 638 Voss, Houston TX    |
| 453453453 | 5631 Rice, Houston TX   |
| 888665555 | 450 Stone, Houston TX   |
| 987987987 | 980 Dallas, Houston TX  |
+-----------+-------------------------+
5 rows in set (0.00 sec)
```

5. Find all the employees who are born during 1950's.

select fname,ssn from employee where bdate BETWEEN '1950-01-01' AND '159-12-31';

```
mysql> select fname,ssn from employee where bdate BETWEEN '1950-01-01' AND '159-12-31';
Empty set (0.00 sec)
```

ii) ORDER BY clause(sort by column name)

Description : The **ORDER BY Clause in MySQL** is a powerful tool that allows you to sort the **result set** of a query in ascending or descending order based on one or more columns.

It is an essential part of querying databases when you want to retrieve data in a specific order. In this article, we will explore the syntax and usage of the MySQL ORDER BY Keyword.

Cmd: ORDERBY

- ➔ **Syntax**: select colname1 , colname2……..colnamen from tablename orderby colname1…..colnamen;
- ➔ **Use:** To arrange in an alphabetical order and desending and ascending order also.

6. Retrieve a list of emp's and the projects they are working on ordered by dept and within each department ordered by alphabetically by lname, fname.

select fname,pnumber from employee e,project p where e.dno=p.dnum order by pnumber,fname,lname;

```
mysql> select fname,pnumber from employee e,project p where e.dno=p.dnum order by pnumber,fname,lname;
+----------+---------+
| fname    | pnumber |
+----------+---------+
| Franklin |       1 |
| John     |       1 |
| Joyce    |       1 |
| Ramesh   |       1 |
| Franklin |       2 |
| John     |       2 |
| Joyce    |       2 |
| Ramesh   |       2 |
| Franklin |       3 |
| John     |       3 |
| Joyce    |       3 |
| Ramesh   |       3 |
| Ahmad    |      10 |
| Alicia   |      10 |
| Jennifer |      10 |
| James    |      20 |
| Ahmad    |      30 |
| Alicia   |      30 |
| Jennifer |      30 |
+----------+---------+
19 rows in set (0.00 sec)
```

7. Retrieve department number from department table ordered by mgr_start_date

select dnumber from department order by mgr_start_date;

```
mysql> select dnumber from department order by mgr_start_date;
+----------+
| dnumber |
+----------+
|        1 |
|        5 |
|        4 |
+----------+
```

Cmd: DISTINCT

- **Syntax**: select distinct * from tablename;
- **Use**: It retrives distinct set of values from the table.

8. Retrieve distinct salaries from employee table

select distinct Salary from employee;

```
mysql> select distinct Salary from employee;
+--------+
| Salary |
+--------+
|  30000 |
|  40000 |
|  25000 |
|  38000 |
|  55000 |
|  43000 |
+--------+
6 rows in set (0.00 sec)
```

iii)LIMIT clause

Description: LIMIT clause to constrain the number of rows returned by a query.
The LIMIT clause is used in the SELECT statement to constrain the number of rows to return. The LIMIT clause accepts one or two arguments. The values of both arguments must be zero or positive integers.

```
SELECT
    customerNumber,
    customerName,
    creditLimit
FROM
    customers
ORDER BY creditLimit DESC
LIMIT 5;
```

customernumber	customername	creditlimit
223	Natürlich Autos	0
168	American Souvenirs Inc	0
169	Porto Imports Co.	0
206	Asian Shopping Network, Co	0
125	Havel & Zbyszek Co	0

In this example:

- First, the ORDER BY clause sorts the customers by credits in low to high.
- Then, the LIMIT clause returns the first 5 rows.

Date: 19/7/24 **EXPERIMENT -5**

AIM: Examples on Functions in MySQL

i)String Functions

ii) Numeric Functions

iii)Date & Time

iv)Conversion Functions

i) String Functions

Description : String functions are used to manipulate the string values.

Cmd: INSTR

- ➔ **Syntax:** select INSTR(str1,str2,1,1) ;
- ➔ **Use:** This function returns the position of the first occurrence of a string in another string.

Example: select instr("Hello, World!","world");

```
mysql> select instr("Hello, World!","world");
+--------------------------------+
| instr("Hello, World!","world") |
+--------------------------------+
|                              8 |
+--------------------------------+
1 row in set (0.01 sec)
```

Cmd: Concat

- ➔ **Syntax:** select concat (<str1>,<str2>);
- ➔ **Use :** str1, str2 are the strings that are being combined.

Example: select concat(Fname," ",Lname) from EMPLOYEE;

```
mysql> select concat(Fname," ",Lname) from EMPLOYEE;
+-------------------------+
| concat(Fname," ",Lname) |
+-------------------------+
| John Smith              |
| Franklin Wong           |
| Ramesh Narayan          |
| James Borg              |
| Jennifer Wallace        |
| Ahmad Jabbar            |
| Alicia Zelaya           |
+-------------------------+
7 rows in set (0.04 sec)
```

Cmd: Least

➔ **Syntax:** select least(<number1>,<number2>);

➔ **Use:** It gives the least value of set of values

Example: select least(9,4,1,2,3,4,5,6);

```
mysql> select least(9,4,1,2,3,4,5,6);
+----------------------+
| least(9,4,1,2,3,4,5,6) |
+----------------------+
|                    1 |
+----------------------+
1 row in set (0.00 sec)
```

Cmd: Greatest

➔ **Syntax:** select greatest (<num1>,<num2>) ;

➔ **Use:** it gives greatest values among set of values

Example: select greatest(9,4,1,2,3,4,5,6,7);

```
mysql> select greatest(9,4,1,2,3,4,5,6,7);
+----------------------------+
| greatest(9,4,1,2,3,4,5,6,7) |
+----------------------------+
|                          9 |
+----------------------------+
1 row in set (0.00 sec)
```

Cmd : Truncate

➔ **Syntax:** select trunk(<num>,<decimal>) ;

➔ **Use:** Truncate the value to a special decimal number

Example: select truncate(123.4567,2);

```
mysql> select truncate(123.4567,2);
+---------------------+
| truncate(123.4567,2) |
+---------------------+
|              123.45 |
+---------------------+
1 row in set (0.00 sec)
```

Cmd: Round

➔ **Syntax:** select round (<num>,[rounded to places]);

➔ **Use:** It rounds the value

Example: select round(123.456,2);

```
mysql> select round(123.456,2);
+------------------+
| round(123.456,2) |
+------------------+
|           123.46 |
+------------------+
```

Cmd: LPAD

➔ **Syntax:** select lpad ("PVP", 5, "*");
➔ **Use:** Returns char1,leftpadded to length with sequence of character in char2.

Example :select lpad("samsung", 9, '*');

```
mysql> select lpad("samsung", 9, '*');
+------------------------+
| lpad("samsung", 9, '*') |
+------------------------+
| **samsung              |
+------------------------+
1 row in set (0.00 sec)
```

Cmd: RPAD

➔ **Syntax;** select rpad ("PVP", 5, "*");
➔ **Use:** returns right padded to length n with sequence of character in char2

Example: select rpad("samsung", 9, '*');

```
mysql> select rpad("samsung", 9, '*');
+------------------------+
| rpad("samsung", 9, '*') |
+------------------------+
| samsung**              |
+------------------------+
1 row in set (0.02 sec)
```

Cmd : LTRIM

➔ **Syntax:** select ltrim (" PVPSIT");
➔ **Use:** The LTRIM() function removes leading spaces from a string.

Example: select ltrim (“ samsung”);

```
mysql> select ltrim('  samsung');
+--------------------+
| ltrim('  samsung') |
+--------------------+
| samsung            |
+--------------------+
1 row in set (0.00 sec)
```

Cmd: Lower

- ➔ **Syntax:** select lower(‘PVPSIT’);
- ➔ **Use:** The LOWER() function converts a string to lower-case.

Example: select lower (‘PLACEMENTS’);

```
mysql> select lower('PLACEMENTS');
+--------------------+
| lower('PLACEMENTS') |
+--------------------+
| placements         |
+--------------------+
1 row in set (0.00 sec)
```

Cmd: Upper

- ➔ **Synatx:** select upper(‘training’);
- ➔ **Use:** The UPPER() function converts a string to upper-case.

Example: select upper (‘training’);

```
mysql> select upper('training');
+------------------+
| upper('training') |
+------------------+
| TRAINING         |
+------------------+
1 row in set (0.00 sec)
```

Cmd: Length

- ➔ **Syntax:** select length(‘PVPSIT’) as LengthofString;
- ➔ **Use :** The LENGTH() function returns the length of a string (in bytes).

Example: select length('Employee') as Length;

```
mysql> select length('Employee') as Length;
+--------+
| Length |
+--------+
|      8 |
+--------+
1 row in set (0.00 sec)
```

Cmd: Replace

- **Syntax:** select REPLACE(*string*, *from_string*, *new_string*);
- **Use:** The REPLACE() function replaces all occurrences of a substring within a string, with a new substring.

Example: select replace('PVP', 'VP', 'PSV');

```
mysql> select replace('PVP', 'VP', 'PSV');
+-----------------------------+
| replace('PVP', 'VP', 'PSV') |
+-----------------------------+
| PPSV                        |
+-----------------------------+
1 row in set (0.00 sec)
```

Cmd: Reverse

- **Syntax:** select reverse('string');
- **Use:** The REVERSE() function reverses a string and returns the result.

Example: select REVERSE('DBMS');

```
mysql> select REVERSE('DBMS');
+-----------------+
| REVERSE('DBMS') |
+-----------------+
| SMBD            |
+-----------------+
1 row in set (0.00 sec)
```

Cmd: STRCMP

- **Sytax:** select STRCMP(*string1, string2*)
- **Use:** The STRCMP() function compares two strings.

- If *string1 = string2*, this function returns 0
- If *string1 < string2*, this function returns -1
- If *string1 > string2*, this function returns 1

Example: select strcmp('SQL Tutorial', 'HTML Tutorial');

```
mysql> select strcmp('SQL Tutorial', 'HTML Tutorial');
+----------------------------------------+
| strcmp('SQL Tutorial', 'HTML Tutorial') |
+----------------------------------------+
|                                      1 |
+----------------------------------------+
1 row in set (0.01 sec)
```

ii)Numeric Functions

Cmd: Count

- ➔ **Synatx:** select COUNT(*expression*)
- ➔ **Use:** The COUNT() function returns the number of records returned by a select query.

Example: select count(fname) from EMPLOYEE;

```
mysql> select count(fname) from EMPLOYEE;
+--------------+
| count(fname) |
+--------------+
|            8 |
+--------------+
1 row in set (0.08 sec)
```

Cmd: Floor

- ➔ **Syntax:** select FLOOR(*number*)
- ➔ **Use:** The FLOOR() function returns the largest integer value that is smaller than or equal to a number.

Example: select floor (27.57);

```
mysql> select floor (27.57);
+---------------+
| floor (27.57) |
+---------------+
|            27 |
+---------------+
1 row in set (0.01 sec)
```

Cmd: CEIL

- ➔ **Syntax:** select CEIL(number);
- ➔ **Use:** The CEIL() function returns the smallest integer value that is bigger than or equal to a number.

Example: select CEIL(13.56);

```
mysql> select CEIL(13.56);
+-------------+
| CEIL(13.56) |
+-------------+
|          14 |
+-------------+
1 row in set (0.00 sec)
```

Cmd: MIN

- ➔ **Syntax:** SELECT MIN(*column_name*) FROM *table_name* WHERE *condition*;
- ➔ **Use:** The MIN() function returns the smallest value of the selected column.

Example: select min(salary) from EMPLOYEE;

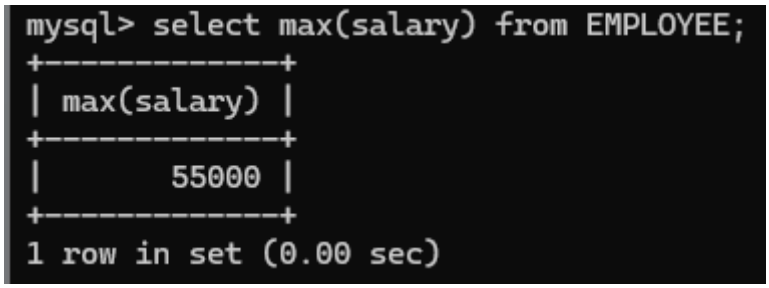
```
mysql> select min(salary) from EMPLOYEE;
+-------------+
| min(salary) |
+-------------+
|       25000 |
+-------------+
1 row in set (0.01 sec)
```

Cmd: MAX

- ➔ **Syntax:** SELECT MAX(*column_name*) FROM *table_name* WHERE *condition*;

➔ **Use:** The MAX() function returns the greatest value of the selected column.

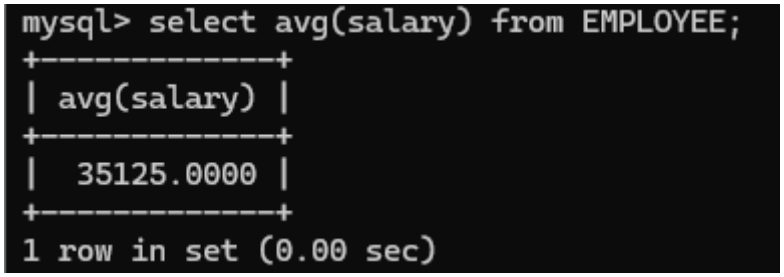
Example: select max(salary) from EMPLOYEE;



Cmd: AVG

➔ **Syntax:** SELECT AVG(*column_name*) FROM *table_name*
➔ **Use:** The AVG() function returns the average value of a numeric column.

Example: select avg(salary) from EMPLOYEE;

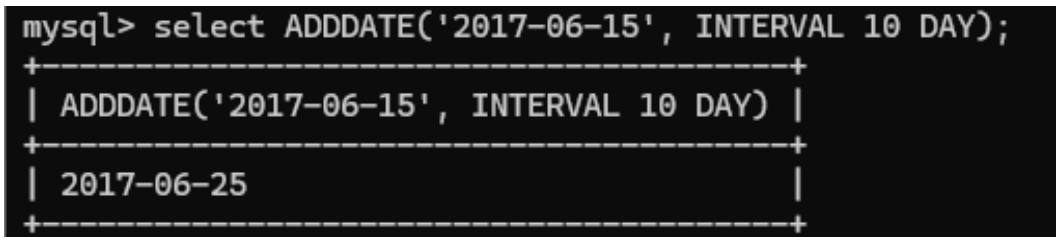


iii)Date Functions

Cmd: ADDDATE

➔ **Syntax:** select ADDDATE(*date*, INTERVAL *value addunit*)
➔ **Use:** The ADDDATE() function adds a time/date interval to a date and then returns the date.

Example: select ADDDATE("2017-06-15", INTERVAL 10 DAY);



Cmd: CURDATE

➔ **Syntax:** select CURDATE();
➔ **Use:** The CURDATE() function returns the current date.

Example: select `CURDATE() + 1;`

```
mysql> select CURDATE() + 1;
+---------------+
| CURDATE() + 1 |
+---------------+
|      20240926 |
+---------------+
1 row in set (0.00 sec)
```

iii)Time Functions

Cmd: CURRENTTIME

- ➔ **Syntax:** SELECT CURRENT_TIME();
- ➔ **Use:** The CURRENT_TIME() function returns the current time.

Example: SELECT CURRENT_TIME() + 1;

```
mysql> SELECT CURRENT_TIME() + 1;
+--------------------+
| CURRENT_TIME() + 1 |
+--------------------+
|             161301 |
+--------------------+
1 row in set (0.00 sec)
```

Cmd: DAYOFMONTH

- ➔ **Syntax:** select DAYOFMONTH(*date*)
- ➔ **Use:** The DAYOFMONTH() function returns the day of the month for a given date (a number from 1 to 31).

Example: select DAYOFMONTH("2017-06-15 09:34:21");

```
mysql> select DAYOFMONTH('2017-06-15 09:34:21');
+-----------------------------------+
| DAYOFMONTH('2017-06-15 09:34:21') |
+-----------------------------------+
|                                15 |
+-----------------------------------+
1 row in set (0.00 sec)
```

Cmd: DAYOFWEEK

- ➔ **Syntax :** select DAYOFWEEK(*date*)

➔ **Use :** The DAYOFWEEK() function returns the weekday index for a given date (a number 1-7).

Example: select DAYOFWEEK("2017-06-15 09:34:21");

```
mysql> select DAYOFWEEK("2017-06-15 09:34:21");
+----------------------------------+
| DAYOFWEEK("2017-06-15 09:34:21") |
+----------------------------------+
|                                5 |
+----------------------------------+
1 row in set (0.00 sec)
```

Cmd: DAYOFYEAR

➔ **Syntax:** select `DAYOFYEAR(`*`date`*`)`

➔ **Use :** The DAYOFYEAR() function returns the day of the year for a given date (a number from 1 to 366).

Example: select DAYOFYEAR("2017-01-01");

```
mysql> select DAYOFYEAR("2017-01-01");
+-------------------------+
| DAYOFYEAR("2017-01-01") |
+-------------------------+
|                       1 |
+-------------------------+
1 row in set (0.00 sec)
```

Cmd: WEEK

➔ **Syntax:** `select WEEK(`*`date, firstdayofweek`*`)`

➔ **Use:** The WEEK() function returns the week number for a given date (a number from 0 to 53).

Example: select WEEK("2017-10-25");

```
mysql> select WEEK("2017-10-25");
+--------------------+
| WEEK("2017-10-25") |
+--------------------+
|                 43 |
+--------------------+
1 row in set (0.00 sec)
```

Cmd: ISNULL

- ➔ **Syntax:** select `ISNULL(expression)`
- ➔ **Use:** The ISNULL() function returns 1 or 0 depending on whether an expression is NULL.

Example: select ISNULL(350);

```
mysql> select ISNULL(350);
+-------------+
| ISNULL(350) |
+-------------+
|           0 |
+-------------+
1 row in set (0.00 sec)
```

iii)Conversion Functions

Cmd: CONVERT

- ➔ **Syntax:** select `CONVERT(value, type)`
- ➔ **Use:** The CONVERT() function converts a value into the specified datatype or character set.

Example: select CONVERT("14:06:10", TIME);

```
mysql> select CONVERT('14:06:10', TIME);
+--------------------------+
| CONVERT('14:06:10', TIME) |
+--------------------------+
| 14:06:10                 |
+--------------------------+
1 row in set (0.01 sec)
```

Cmd: STR_TO_DATE

- ➔ **Syntax:** select STR_TO_DATE(*string*, *format*)
- ➔ **Use:** The STR_TO_DATE() function returns a date based on a string and a format.

Example: SELECT STR_TO_DATE("2017,8,14 10,40,10", "%Y,%m,%d %h,%i,%s");

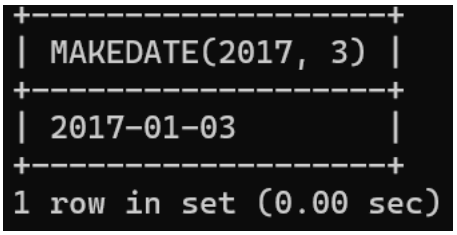
```
mysql> SELECT STR_TO_DATE("2017,8,14 10,40,10", "%Y,%m,%d %h,%i,%s");
+-------------------------------------------------------+
| STR_TO_DATE("2017,8,14 10,40,10", "%Y,%m,%d %h,%i,%s") |
+-------------------------------------------------------+
| 2017-08-14 10:40:10                                   |
+-------------------------------------------------------+
1 row in set (0.00 sec)
```

Cmd: Makedate

➔ **Syntax:** select MAKEDATE(*year*, *day*)

➔ **Use:** The MAKEDATE() function creates and returns a date based on a year and a number of days value.

Example: select MAKEDATE(2017, 3);



Date: 2/8/24 **EXPERIMENT -6**

AIM: Examples on Summary Queries: Queries using Aggregate functions,GROUP By and Having Clauses, ROLLUP Operator.

Aggregate functions: These are also called group functions; they include min, max, count, avg and sum.

Count: This function count no. of rows

1. Count the total number of employees whose salaries exceed 400000 in each department but only for department where more than 3 employees work

select count(*), dnumber from employee emp inner join department dep on emp.dno = dep.dnumber where emp.salary > 40000 group by dnumber;

```
mysql> select count(*), dnumber from employee emp inner join department dep on emp.dno = dep.dnumber where emp.salary > 40000 group by dnumber;
+----------+---------+
| count(*) | dnumber |
+----------+---------+
|        1 |       1 |
|        1 |       4 |
+----------+---------+
2 rows in set (0.00 sec)
```

2. For each department that has more than 3emps ,retrieve the depy. No. and no. of its emps who are more than 30000

select count(*), dno from employee emp,department dep where emp.dno = dep.dnumber and emp.salary>30000 group by dnumber having count(*)>1;

```
mysql> select count(*), dno from employee emp,department dep where emp.dno = dep.dnumber and emp.salary>30000 group by dnumber having count(*)>1;
+----------+-----+
| count(*) | dno |
+----------+-----+
|        2 |   5 |
+----------+-----+
1 row in set (0.01 sec)
```

3. Retrive the total no.of employees in the company

select count(ssn) from Employee;

```
mysql> select count(ssn) from Employee;
+-------------+
| count(ssn) |
+-------------+
|           8 |
+-------------+
1 row in set (0.00 sec)
```

4. Retrive the no. of emps in research dept.

select count(Dno) from employee e,department d where e.dno=d.dnumber and dname='Research';

```
mysql> select count(Dno) from employee e,department d where e.dno=d.dnumber and dname='Research';
+-------------+
| count(Dno) |
+-------------+
|           4 |
+-------------+
1 row in set (0.01 sec)
```

5. Count the no. of distinct salary values in the database employee

select count(distinct Salary) from Employee;

```
mysql> select count(distinct Salary) from Employee;
+------------------------+
| count(distinct Salary) |
+------------------------+
|                      6 |
+------------------------+
1 row in set (0.00 sec)
```

SUM, MIN, MAX,AVG:

1. Find the sum of salaries of all employees the maximum salary,minimum salary and the avg salary

select sum(Salary), max(Salary), min(Salary), avg(Salary) from Employee;

```
mysql> select sum(Salary), max(Salary), min(Salary), avg(Salary) from Employee;
+-------------+-------------+-------------+-------------+
| sum(Salary) | max(Salary) | min(Salary) | avg(Salary) |
+-------------+-------------+-------------+-------------+
|      281000 |       55000 |       25000 |  35125.0000 |
+-------------+-------------+-------------+-------------+
1 row in set (0.00 sec)
```

2. Find the sum of salaries of the research dept as well as the max,min,avg sal in dept

select sum(Salary),max(Salary),min(Salary),avg(Salary) from Employee e, Department d where e.Dno=d.Dnumber and Dname='Research';

```
mysql> select sum(Salary),max(Salary),min(Salary),avg(Salary) from Employee e, Department d where e.Dno=d.Dnumber and Dname='Research';
+-------------+-------------+-------------+-------------+
| sum(Salary) | max(Salary) | min(Salary) | avg(Salary) |
+-------------+-------------+-------------+-------------+
|      133000 |       40000 |       25000 |  33250.0000 |
+-------------+-------------+-------------+-------------+
1 row in set (0.00 sec)
```

Group by having:

1.for each department ,retrieve the dno ,no of emp's in the dept and there avg salary.

select Dno,count(Ssn),avg(Salary) from Department d,Employee e where e.Dno=d.Dnumber group by Dno;

```
mysql> select Dno,count(Ssn),avg(Salary) from Department d,Employee e where e.Dno=d.Dnumber group by Dno;
+-----+------------+-------------+
| Dno | count(Ssn) | avg(Salary) |
+-----+------------+-------------+
|   5 |          4 |  33250.0000 |
|   1 |          1 |  55000.0000 |
|   4 |          3 |  31000.0000 |
+-----+------------+-------------+
3 rows in set (0.00 sec)
```

2.for each project retrieve the pno,pname,&no of employes who works on that project from the project table

select Pno,Pname,count(Essn) from project p,workers_on w where p.Pnumber=w.Pno group by Pname,Pnumber;

```
mysql> select Pno,Pname,count(Essn) from project p,workers_on w where p.Pnumber=w.Pno group by Pname,Pnumber;
+-----+-----------------+-------------+
| Pno | Pname           | count(Essn) |
+-----+-----------------+-------------+
|  10 | Computerization |           3 |
|  30 | Newbenefits     |           3 |
|   1 | ProductX        |           2 |
|   2 | ProductY        |           3 |
|   3 | ProductZ        |           2 |
|  20 | Reorganization  |           3 |
+-----+-----------------+-------------+
6 rows in set (0.00 sec)
```

3.for each project,on which more than 2 employees work,retrieve the pno ,pname,no of emp's who work on project

select Pnumber,Pname,count(essn) from Project p,Workers_on w where p.pnumber=w.pno group by pname,pnumber having count(*)>2;

```
mysql> select Pnumber,Pname,count(essn) from Project p,Workers_on w where p.pnumber=w.pno group by pname,pnumber having count(*)>2;
+---------+-----------------+-------------+
| Pnumber | Pname           | count(essn) |
+---------+-----------------+-------------+
|      10 | Computerization |           3 |
|      30 | Newbenefits     |           3 |
|       2 | ProductY        |           3 |
|      20 | Reorganization  |           3 |
+---------+-----------------+-------------+
4 rows in set (0.00 sec)
```

4.for each project,retrieve the pno,name,no of emp's from dept 5 who work on the project

select pnumber,pname,count(ssn) from Project p,Workers_on w,Employee e,Department d where p.pnumber=w.pno and e.dno=d.dnumber and p.dnum=e.dno and e.dno=5 group by pno,pname;

```
mysql> select pnumber,pname,count(ssn) from Project p,Workers_on w,Employee e,Department d where p.pnumber=w.pno and e.dno=d.dnumber and p.dnum=e.
dno and e.dno=5 group by pno,pname;
+---------+----------+------------+
| pnumber | pname    | count(ssn) |
+---------+----------+------------+
|       1 | ProductX |          8 |
|       2 | ProductY |         12 |
|       3 | ProductZ |          8 |
+---------+----------+------------+
3 rows in set (0.04 sec)
```

ROLLUP Operator:

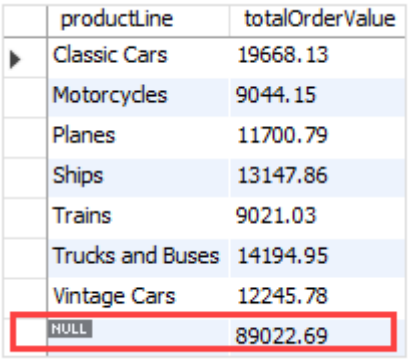
The ROLLUP clause is an extension of the GROUP BY clause with the following syntax:

```
SELECT
    select_list
FROM
    table_name
GROUP BY
    c1, c2, c3 WITH ROLLUP;
```

The ROLLUP generates multiple grouping sets based on the columns or expressions specified in the GROUP BY clause. For example:

```
SELECT
    productLine,
    SUM(orderValue) totalOrderValue
FROM
    sales
GROUP BY
    productline WITH ROLLUP;
```

Here is the output:



productLine	totalOrderValue
Classic Cars	19668.13
Motorcycles	9044.15
Planes	11700.79
Ships	13147.86
Trains	9021.03
Trucks and Buses	14194.95
Vintage Cars	12245.78
NULL	89022.69

As clearly shown in the output, the ROLLUP clause generates not only the subtotals but also the grand total of the order values.

Date: 26/7/24 **EXPERIMENT -7**

AIM: Examples on

i) INNER JOIN

ii)OUTER JOIN USING NATURAL KEYWORDS

i) INNER JOIN:

Description : A JOIN clause is used to combine rows from two or more tables, based on a related column between them**.**

a) EQUI JOIN:

1. Retrieve the names of employees and their salries who are part of any department.

select fname,salary from employee e , department d where e.dno=d.dnumber;

```
+----------+--------+
| fname    | salary |
+----------+--------+
| Jennifer |  43000 |
| Ahmad    |  25000 |
| Alicia   |  25000 |
| James    |  55000 |
| John     |  30000 |
| Franklin |  35000 |
| Ramesh   |  38000 |
+----------+--------+
7 rows in set (0.00 sec)
```

b)NON EQUI JOIN:

1. Retrieve the names of employee and project details based on his salary range.

select fname ,pname from employee e , project p where e.salary >30000 and e.salary< 45000 and e.dno =p.dnum;

```
+----------+-----------------+
| fname    | pname           |
+----------+-----------------+
| Ramesh   | ProductX        |
| Franklin | ProductX        |
| Ramesh   | ProductY        |
| Franklin | ProductY        |
| Ramesh   | ProductZ        |
| Franklin | ProductZ        |
| Ramesh   | Computerization |
| Franklin | Computerization |
| Jennifer | Newbenefits     |
+----------+-----------------+
9 rows in set (0.00 sec)
```

c) SELF JOIN:

1. Display all the employees who earn more than their managers.

```
+-----------+--------+--------+-----------+--------+
| ssn       | fname  | salary | super_ssn | salary |
+-----------+--------+--------+-----------+--------+
| 666884444 | Ramesh |  38000 | 888665555 |  35000 |
+-----------+--------+--------+-----------+--------+
1 row in set (0.00 sec)
```

EXAMPLES:

1. For every project located in Stafford , retrieve the project num controlling dept no ,manger last name , address and birth date using join.

select Pnumber,Dnum,Super_Ssn "Manager Ssn",e.Lname,e.Address,e.Bdate from EMPLOYEE e,PROJECT p WHERE p.Dnum=e.Dno AND p.Plocation='Stafford';

```
+---------+------+-------------+---------+-----------------------+------------+
| Pnumber | Dnum | Manager Ssn | Lname   | Address               | Bdate      |
+---------+------+-------------+---------+-----------------------+------------+
|      10 |    4 | 888665555   | Wallace | 291 Berry, Bellaire TX | 1941-06-20 |
|      10 |    4 | 987654321   | Jabbar  | 980 Dallas, Houston TX | 1969-03-29 |
|      10 |    4 | 987654321   | Zelaya  | 3321 Castle, Spring TX | 1968-01-19 |
|      30 |    4 | 888665555   | Wallace | 291 Berry, Bellaire TX | 1941-06-20 |
|      30 |    4 | 987654321   | Jabbar  | 980 Dallas, Houston TX | 1969-03-29 |
|      30 |    4 | 987654321   | Zelaya  | 3321 Castle, Spring TX | 1968-01-19 |
+---------+------+-------------+---------+-----------------------+------------+
6 rows in set (0.0176 sec)
```

2. Retrieve the names and address of all employees who works on research department using natural join.

select Fname,Lname,Address from EMPLOYEE e natural join DEPARTMENT d WHERE e.Dno=d.Dnumber and Dname="Research";

```
+----------+---------+-------------------------+
| Fname    | Lname   | Address                 |
+----------+---------+-------------------------+
| John     | Smith   | 731 Fondren, Houston TX |
| Franklin | Wong    | 638 Voss, Houston TX    |
| Joyce    | English | 5631 Rice, Houston TX   |
| Ramesh   | Narayan | 975 Fire Oak, Humble TX |
+----------+---------+-------------------------+
4 rows in set (0.0328 sec)
```

ii)OUTER JOIN

1. Retrieve the names of employees and supervisors using left outer join.

select e.Fname,e.Lname,s.Fname,s.Lname from EMPLOYEE e left outer join EMPLOYEE s ON s.Super_ssn=e.Ssn;

Fname	Lname	Fname	Lname
John	Smith	NULL	NULL
Franklin	Wong	John	Smith
Franklin	Wong	Joyce	English
Franklin	Wong	Ramesh	Narayan
Joyce	English	NULL	NULL
Ramesh	Narayan	NULL	NULL
James	Borg	Franklin	Wong
James	Borg	Jennifer	Wallace
Jennifer	Wallace	Ahmad	Jabbar
Jennifer	Wallace	Alicia	Zelaya
Ahmad	Jabbar	NULL	NULL
Alicia	Zelaya	NULL	NULL

12 rows in set (0.0025 sec)

2. Retrieve the names of employees and supervisor using right outer join.

select e.Fname,e.Lname,s.Fname,s.Lname from EMPLOYEE e right outer join EMPLOYEE s ON s.Super_ssn=e.Ssn;

Fname	Lname	Fname	Lname
Franklin	Wong	John	Smith
James	Borg	Franklin	Wong
Franklin	Wong	Joyce	English
Franklin	Wong	Ramesh	Narayan
NULL	NULL	James	Borg
James	Borg	Jennifer	Wallace
Jennifer	Wallace	Ahmad	Jabbar
Jennifer	Wallace	Alicia	Zelaya

8 rows in set (0.0010 sec)

Date: 2/8/24 **EXPERIMENT – 8**

AIM: Examples on
i)SUB/SUMMARY Queries Using IN, ANY, SOME,
ALL , EXISTS and NOT EXISTS functions.
ii)Retrieval of data from interrelated tables in the database in several ways

i)Examples on sub queries using IN, ANY, SOME, EXISTS and NON EXISTS functions
Cmd:

In: (nested queries)
1. select the project no fro projects that have an employee with lname smith as manager.
select pnumber from project where pnumber in (select pnumber from project p,employee e,department d,works_on w where e.dno=p.dnum and e.ssn=w.essn and e.lname="smith" and e.ssn=d.mgr_ssn);

```
Empty set (0.0176 sec)
```

2. List the essn of all employees who works on same project, hours as employees with essn 123456789 and 999887777 from works_on table.
select distinct w1.essn from works_on w1 join works_on w2 on w1.pno = w2.pno and w1.hours = w2.hours where w2.essn in ('123456789', '999887777') and w1.essn not in ('123456789', '999887777');

```
+-----------+
| Essn      |
+-----------+
| 333445555 |
+-----------+
1 row in set (0.0317 sec)
```

3. List the names of emp's whose salary is greater than the salary of all employees in dept '5'.
select fname,salary,dno from employee where salary>all(select max(salary) from employee where dno=5);

```
+----------+--------+-----+
| Fname    | Salary | Dno |
+----------+--------+-----+
| James    |  55000 |   1 |
| Jennifer |  43000 |   4 |
+----------+--------+-----+
2 rows in set (0.0016 sec)
```

Correlated Nested Queries:

Any:

List the names of employee whose salary is above average of their department.

select distinct(fname),lname,salary from employee e,department d where salary>any(select avg(salary) from employee group by dno);

Fname	Lname	Salary
Jennifer	Wallace	43000
James	Borg	55000
Ramesh	Narayan	38000
Franklin	Wong	40000

4 rows in set (0.0020 sec)

All:

1.Retrieve the name of emp whose salary is above salaries of all employees in the dept.

select Fname,Lname from EMPLOYEE where Salary>all(select Salary from EMPLOYEE);

Empty set (0.0015 sec)

Exists:

1.List the names of employees who have atleast one dependent

select e.fname from employee e where exists(select * from dependent d where e.ssn=d.essn);

Fname
John
Franklin
Jennifer

3 rows in set (0.0223 sec)

2.Retrive the names of employees whose dependent name is same as employee name

select e.fname from employee e where exists(select * from dependent d where e.ssn=d.essn and e.fname=d.dependent_name);

Empty set (0.0013 sec)

Not exists:

1.retrive the names of employess who have no dependents

select e.fname from employee e where not exists(select * from dependent d where e.ssn=d.essn);

Fname
Joyce
Ramesh
James
Ahmad
Alicia

5 rows in set (0.0013 sec)

Date: 9/8/24 **EXPERIMENT -9**

AIM: Examples on
i)Creating INDEXES and VIEWS
ii) INSERT, DELETE and DROP on VIEWS

i)Creating INDEXES
Description: Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

- **Syntax:** CREATE INDEX *index_name*
  ON *table_name* (*column1*, *column2*, ...);
- **Use :** The CREATE INDEX statement is used to create indexes in tables.

1. Create index on ssn of employee table
create index emp-ssn-ix on employee(ssn);

```
mysql> create index emp_ssn_ix on employee(ssn);
Query OK, 0 rows affected, 1 warning (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 1
```

2. Create index on ssn and dno of employee table.
create index emp-ssn-dno-ix on employee (ssn, dno);

```
mysql> create index emp_ssn_dno_ix on employee(ssn,dno);
Query OK, 0 rows affected (0.07 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Cmd: View
Description: It is a virtual table. If a table is derived from another table, then a view is created.

- **Syntax:** create view_name as select (colname1, colname2…. colnamen) from tablename;
- **Use:** A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. View can be created on multiple tables and can serve as security purpose which is dynamic.

1. Create a view with emp details of emp's who are working in dep5.
create view emp_view as select * from employee where dno=5;

```
mysql> create view emp_view as select * from employee where dno=5;
Query OK, 0 rows affected (0.01 sec)

mysql> select fname,dno from emp_view;
+----------+-----+
| Fname    | Dno |
+----------+-----+
| John     |   5 |
| Franklin |   5 |
| Ramesh   |   5 |
+----------+-----+
3 rows in set (0.00 sec)
```

2. Create a view to list the details of the dept with name 'research'.

create view dept_view as select * from employee e,department d where e.dno=d.dnumber and dname='research';

```
mysql> create view dept_view as
    -> select * from employee e,department d
    -> where e.dno=d.dnumber and dname="research";
Query OK, 0 rows affected (0.07 sec)

mysql> select fname,dname from dept_view;
+----------+----------+
| Fname    | Dname    |
+----------+----------+
| John     | Research |
| Franklin | Research |
| Ramesh   | Research |
+----------+----------+
3 rows in set (0.00 sec)
```

3. Create a view to list the names of emp's the project names under where they are workg and the no of hrs

create view proj as select fname,pname,hours from employee e, project p, works_no w where e.dno=p.dnum and p.pnumber=w.pno and e.ssn=w.Essn;

```
mysql> create view proj_view as
    -> select fname,pname,hours from employee e,project p,works_no w
    -> where e.dno=p.dnum and p.pnumber=w.pno and e.ssn=w.Essn;
Query OK, 0 rows affected (0.01 sec)

mysql> select fname,pname,hours from proj_view;
+----------+-----------------+-------+
| fname    | pname           | hours |
+----------+-----------------+-------+
| John     | ProductX        |  32.5 |
| John     | ProductY        |   7.5 |
| Franklin | ProductY        |  10.0 |
| Franklin | ProductZ        |  10.0 |
| Franklin | Computerization |  10.0 |
| Ramesh   | ProductZ        |  40.0 |
| James    | Reorganization  |  16.0 |
| Jennifer | Newbenefits     |  20.0 |
| Ahmad    | Newbenefits     |   5.0 |
| Alicia   | Newbenefits     |  30.0 |
+----------+-----------------+-------+
10 rows in set (0.01 sec)
```

4. Create a view to list the details of emp's with dept 'research'.

create view emp_res as select * from employee e, department d where e.dno=d.dnumber and dname='research';

```
mysql> create view emp_res as
    -> select * from employee e,department d
    -> where e.dno=d.dnumber and dname='research';
Query OK, 0 rows affected (0.01 sec)

mysql> select fname,ssn,dname from emp_res;
+----------+-----------+----------+
| Fname    | Ssn       | Dname    |
+----------+-----------+----------+
| John     | 123456789 | Research |
| Franklin | 333445555 | Research |
| Ramesh   | 666884444 | Research |
+----------+-----------+----------+
3 rows in set (0.00 sec)
```

5. Create a view to the dept, to list the sum of the salaries of employees in that dept.

create view dept_sal as select dname,sum(salary) from employee e,department d where e.dno=d.dnumber group by dname;

```
mysql> create view dept_sal as
    -> select dname,sum(salary) from employee e,department d
    -> where e.dno=d.dnumber group by dname;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from dept_sal;
+----------------+-------------+
| dname          | sum(salary) |
+----------------+-------------+
| Administration |       93000 |
| Headquarters   |       55000 |
| Research       |      103000 |
+----------------+-------------+
3 rows in set (0.00 sec)
```

6. Retrive the lname,fname of the emp who workson the project ‘productX’ as the name.

create view proj_name as select fname,lname from employee e,project p where p.dnum=e.dno and pname='productX';

```
mysql> create view proj_name as
    -> select fname,lname from employee e,project p
    -> where p.dnum=e.dno and pname='productX';
Query OK, 0 rows affected (0.01 sec)

mysql> select * from proj_name;
+----------+---------+
| fname    | lname   |
+----------+---------+
| John     | Smith   |
| Franklin | Wong    |
| Ramesh   | Narayan |
+----------+---------+
3 rows in set (0.00 sec)
```

ii)INSERT, DELETE and DROP on Views

Cmd: Insert

➔ **Syntax:** INSERT INTO view_name (column1, column2, ...)
VALUES (value1, value2, ...);

➔ **Use:** Inserting on views manages related data from multiple tables through a single interface.

Cmd: Delete

➔ **Syntax:** DELETE FROM view_name WHERE condition;

➔ **Use:** Deleting from views allows you to remove records while maintaining data integrity and restrict access to underlying tables.

Cmd: Drop

➔ **Syntax:** DROP VIEW view_name;

➔ **Use:** Dropping views is used to remove a view from the database when it is no longer needed.

1. Update sal of emp's where fname is 'john'.

update emp_view set salary=0 where fname='john';

```
mysql> update emp_view set salary=0 where fname='john';
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1  Changed: 0  Warnings: 0

mysql> select fname,salary from emp_view;
+----------+--------+
| Fname    | Salary |
+----------+--------+
| John     |      0 |
| Franklin |  35000 |
| Ramesh   |  38000 |
+----------+--------+
3 rows in set (0.00 sec)
```

2. Drop a view with emp details who are working in dept5.

drop view emp_view;

```
mysql> drop view emp_view;
Query OK, 0 rows affected (0.01 sec)
```

3. Drop a view with the details of dept with name 'reasearch'

drop a view dept_view;

```
mysql> drop view dept_view;
Query OK, 0 rows affected (0.01 sec)
```

4. Drop a view with project names, the names of emp's where there are working and no of hrs.

drop view proj_view;

```
mysql> drop view proj_view;
Query OK, 0 rows affected (0.01 sec)
```

5. Drop a view with the details of dept with name 'research'

drop view emp_res;

```
mysql> drop view emp_res;
Query OK, 0 rows affected (0.01 sec)
```

6. Drop a view to list the sum of sal's od emp's in the dept

drop view dept_sal;

```
mysql> drop view dept_sal;
Query OK, 0 rows affected (0.01 sec)
```

7. Drop a view with details of names who work on project 'productX'

drop view proj_name;

```
mysql> drop view proj_name;
Query OK, 0 rows affected (0.01 sec)
```

VIEW WITH CHECK OPTION:

1. Create a view with employee details who are working in departments with check option.

create view emp-views as select ssn,salary,dno from employee where dno=5 with check option.

```
mysql> create view emp_views as
    -> select ssn,salary,dno from employee
    -> where dno=5 with check option;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from emp_views;
+-----------+--------+-----+
| ssn       | salary | dno |
+-----------+--------+-----+
| 123456789 |      0 |   5 |
| 333445555 |  35000 |   5 |
| 666884444 |  38000 |   5 |
+-----------+--------+-----+
3 rows in set (0.00 sec)
```

2. Create a view to list the details of department with name 'RESEARCH' using check option.

create view dept-view as select fname,lname,dno, dnumber and dname='Research' with check option.

```
mysql> create view dept_view as
    -> select fname,lname,dno,dnumber,dname
    -> from employee e,department d
    -> where e.dno=d.dnumber and d.dname='research' with check option;
Query OK, 0 rows affected (0.01 sec)

mysql> select * from dept_view;
+----------+---------+-----+---------+----------+
| fname    | lname   | dno | dnumber | dname    |
+----------+---------+-----+---------+----------+
| John     | Smith   |   5 |       5 | Research |
| Franklin | Wong    |   5 |       5 | Research |
| Ramesh   | Narayan |   5 |       5 | Research |
+----------+---------+-----+---------+----------+
3 rows in set (0.00 sec)
```

Date: 13/9/24 **EXPERIMENT -10**

AIM: Examples on
i.) Create and Call STORED PROCEDURE (IN, OUT, INOUT Parameters), Drop a STORED PROCEDURE.
ii.) Create, call and Drop a FUNCTION.
iii.) Create and Drop a TRIGGER

i) Create and Call STORED PROCEDURE (IN, OUT, INOUT Parameters), Drop a STORED PROCEDURE.

Description: PL/SQL process procedural capabilities and it includes declaration section where we declare variables and these types which is optional and execution of code takes place between begin and end and exception handling section where we handle exceptions.

PROCEDURES:

1. Create a procedure to display salary of an employe using in, out parameters.

```
DELIMITER //
CREATE PROCEDURE get_salary(IN emp_no CHAR(9), OUT out_salary DECIMAL(5,0))
BEGIN
SELECT Salary INTO out_salary FROM employee WHERE Ssn = emp_no;
END;
//
DELIMITER ;
CALL get_salary(&amp;#39;333445555&amp;#39;, @out_salary);
SELECT @out_salary;
```

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE get_salary(IN emp_no CHAR(9), OUT out_salary DECIMAL(5,0))
    -> BEGIN
    ->     SELECT Salary INTO out_salary FROM employee WHERE Ssn = emp_no;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> CALL get_salary('333445555', @out_salary);
Query OK, 1 row affected (0.00 sec)

mysql> SELECT @out_salary;
+-------------+
| @out_salary |
+-------------+
|       35000 |
+-------------+
1 row in set (0.00 sec)
```

2. Create a procedure to raise (10%) salary of an employee using in parameter

```
DELIMITER //
CREATE PROCEDURE raise_salary(IN emp_no INT)
BEGIN
DECLARE emp_salary INT;
SELECT salary INTO emp_salary FROM employee WHERE ssn = emp_no;
IF emp_salary &amp;lt; 35000 THEN
SET emp_salary = emp_salary + (emp_salary * 0.1);
SELECT &amp;quot;Salary Updated&amp;quot; AS status, emp_salary AS updated_salary;
```

ELSE
SELECT "Salary Not Updated" AS status;
END IF;
END;
//
DELIMITER ;
CALL raise_salary('999887777');

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE raise_salary(IN emp_no INT)
    -> BEGIN
    ->     DECLARE emp_salary INT;
    ->     SELECT salary INTO emp_salary FROM employee WHERE ssn = emp_no;
    ->     IF emp_salary < 35000 THEN
    ->         SET emp_salary = emp_salary + (emp_salary * 0.1);
    ->         SELECT "Salary Updated" AS status, emp_salary AS updated_salary;
    ->     ELSE
    ->         SELECT "Salary Not Updated" AS status;
    ->     END IF;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> CALL raise_salary('999887777');
+----------------+----------------+
| status         | updated_salary |
+----------------+----------------+
| Salary Updated |          27500 |
+----------------+----------------+
1 row in set (0.00 sec)
```

3. Display name of an employee whose ssn is 123456789.

DELIMITER //
CREATE PROCEDURE get_employee_name(IN emp_ssn CHAR(9))
BEGIN
DECLARE emp_fname VARCHAR(10);
SELECT Fname INTO emp_fname FROM employees WHERE Ssn = emp_ssn;
SELECT emp_fname AS employee_name;
END;
//
DELIMITER ;
CALL get_employee_name('123456789&#39);

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE get_emp_name(IN emp_ssn CHAR(9))
    -> BEGIN
    ->     DECLARE emp_fname VARCHAR(10);
    ->     SELECT Fname INTO emp_fname FROM employee WHERE Ssn = emp_ssn;
    ->     SELECT emp_fname AS employee_name;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> CALL get_emp_name('123456789');
+---------------+
| employee_name |
+---------------+
| John          |
+---------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.01 sec)
```

4. Create a procedure to display salary of an employe

DELIMITER //

```
CREATE PROCEDURE get_sal(IN emp_ssn CHAR(9))
BEGIN
DECLARE emp_salary DECIMAL(5,0);
SELECT Salary INTO emp_salary FROM employee WHERE Ssn = emp_ssn;
SELECT emp_salary AS salary;
END;
//
DELIMITER ;
CALL get_sal('999887777');
```

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE get_sal(IN emp_ssn CHAR(9))
    -> BEGIN
    ->     DECLARE emp_salary DECIMAL(5,0);
    ->         SELECT Salary INTO emp_salary FROM employee WHERE Ssn = emp_ssn;
    ->         SELECT emp_salary AS salary;
    -> END;
    -> //
Query OK, 0 rows affected (0.02 sec)

mysql> DELIMITER ;
mysql> CALL get_sal('999887777');
+--------+
| salary |
+--------+
|  25000 |
+--------+
1 row in set (0.01 sec)
```

ii.) Create, call and Drop a FUNCTION.

1. Create a function to display sum of salaries of employees belonging to a particular department.

```
DELIMITER //
CREATE FUNCTION get_sal(dept_no INT)
RETURNS DECIMAL(10,2)
DETERMINISTIC
READS SQL DATA
BEGIN
DECLARE total_sal DECIMAL(10,2);
SELECT SUM(Salary) INTO total_sal FROM employee WHERE Dno = dept_no;
RETURN total_sal;
END;
//
DELIMITER ;
SELECT get_sal(5) as dept_sal;
```

```
mysql> DELIMITER //
mysql>
mysql> CREATE FUNCTION get_sal(dept_no INT)
    -> RETURNS DECIMAL(10,2)
    -> DETERMINISTIC
    -> READS SQL DATA
    -> BEGIN
    ->     DECLARE total_sal DECIMAL(10,2);
    ->     SELECT SUM(Salary) INTO total_sal FROM employee WHERE Dno = dept_no;
    ->     RETURN total_sal;
    -> END;
    -> //
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> DELIMITER ;
mysql> SELECT get_sal(5) as dept_sal;
+-----------+
| dept_sal  |
+-----------+
| 73000.00  |
+-----------+
1 row in set (0.01 sec)
```

2. Create a function to count the no. of employees belonging to department 4

```
DELIMITER //
CREATE FUNCTION get_count(dept_no INT) RETURNS INT
```

DETERMINISTIC
READS SQL DATA
BEGIN
DECLARE emp_count INT;
SELECT COUNT(*) INTO emp_count FROM employee WHERE Dno = dept_no;
RETURN emp_count;
END;
//

```
mysql> DELIMITER //
mysql> CREATE FUNCTION get_count(dept_no INT) RETURNS INT
    -> DETERMINISTIC
    -> READS SQL DATA
    -> BEGIN
    ->     DECLARE emp_count INT;
    ->     SELECT COUNT(*) INTO emp_count FROM employee WHERE Dno = dept_no;

    ->     RETURN emp_count;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> SELECT get_count(4) AS no_of_emps;
+------------+
| no_of_emps |
+------------+
|          3 |
+------------+
1 row in set (0.01 sec)
```

3. Create a function to display the name of employee for a given employee number

DELIMITER //
CREATE FUNCTION get_name(emp_ssn CHAR(9)) RETURNS VARCHAR(10)
DETERMINISTIC
READS SQL DATA
BEGIN
DECLARE emp_fname VARCHAR(10);
SELECT Fname INTO emp_fname FROM employee WHERE Ssn = emp_ssn;
RETURN emp_fname;
END;
//
DELIMITER ;
SELECT get_name('123456789') AS emp_name;

```
mysql> DELIMITER //
mysql> CREATE FUNCTION get_name(emp_ssn CHAR(9)) RETURNS VARCHAR(10)
    -> DETERMINISTIC
    -> READS SQL DATA
    -> BEGIN
    ->     DECLARE emp_fname VARCHAR(10);
    ->     SELECT Fname INTO emp_fname FROM employee WHERE Ssn = emp_ssn;
    ->     RETURN emp_fname;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> SELECT get_name('123456789') AS emp_name;
+-----------+
| emp_name  |
+-----------+
| John      |
+-----------+
1 row in set (0.00 sec)
```

4. Create a function to display the count of number of projects a given employee is working on

DELIMITER //
CREATE FUNCTION get_proj(emp_ssn CHAR(9)) RETURNS INT
DETERMINISTIC

READS SQL DATA
BEGIN
DECLARE proj_count INT;
SELECT COUNT(Pno) INTO proj_count FROM works_no WHERE essn = emp_ssn;
RETURN proj_count;
END;
//
DELIMITER ;
SELECT get_proj_count('123456789') AS project_count;

```
mysql> DELIMITER //
mysql> CREATE FUNCTION get_proj(emp_ssn CHAR(9)) RETURNS INT
    -> DETERMINISTIC
    -> READS SQL DATA
    -> BEGIN
    ->     DECLARE proj_count INT;
    ->     SELECT COUNT(Pno) INTO proj_count FROM works_no WHERE essn = emp_ssn;
    ->     RETURN proj_count;
    -> END;
    -> //
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;
mysql> SELECT get_proj('123456789') as project_count;
+---------------+
| project_count |
+---------------+
|             2 |
+---------------+
1 row in set (0.01 sec)
```

iii.) Create and Drop a TRIGGER

Description: It is a special type of stored procedure that is automatically executed in response to certain database events such as an INSERT, UPDATE, or DELETE operation.In a database management system (DBMS), a trigger is a set of instructions that automatically runs in response to a specific event, such as a change to a table's data.

➔ **Syntax:** CREATE TRIGGER trigger_name
{BEFORE | AFTER} {INSERT | UPDATE | DELETE}
ON table_name
FOR EACH ROW
BEGIN
-- Trigger logic (SQL statements) go here
END;

➔ **Example: Create table account with attributes account number and salary.**
CREATE TABLE account (
acct_num INT, salary DECIMAL(10,2));

```
mysql> create table account(acct_num int,salary decimal(10,2));
Query OK, 0 rows affected (0.04 sec)
```

1. Create a trigger that displays the sum of salaries inserted into the table.

CREATE TRIGGER saldrop_sum BEFORE INSERT ON account
FOR EACH ROW
SET @sum = @sum + NEW.salary;

SET @sum=0;
INSERT INTO account values(137,14.98),(141,1937.50),(97,-100.00);
SELECT @sum as 'Total amount inserted';

```
mysql> CREATE TRIGGER sal_sum BEFORE INSERT ON account
    -> FOR EACH ROW
    ->     SET @sum = @sum + NEW.salary;
Query OK, 0 rows affected (0.03 sec)
```

```
mysql> SET @sum=0;
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO account values(137,14.98),(141,1937.50),(97,-100.00);
Query OK, 3 rows affected (0.02 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT @sum as 'Total amount inserted';
+-----------------------+
| Total amount inserted |
+-----------------------+
|               1852.48 |
+-----------------------+
1 row in set (0.00 sec)
```

2. Drop Trigger sal_sum

DROP TRIGGER sal_sum;

```
mysql> DROP TRIGGER sal_sum;
Query OK, 0 rows affected (0.02 sec)
```

3. Create a trigger that sets the salary to 0 if it is updated to a negative value and caps it at 10,000 if it exceeds that amount.

DELIMITER //
CREATE TRIGGER upd_check BEFORE UPDATE ON employee
FOR EACH ROW
BEGIN
IF NEW.salary < 0 THEN
SET NEW.salary = 0;
ELSEIF NEW.salary > 10000 THEN
SET NEW.salary = 10000;
END IF;
END;
//
DELIMITER ;
UPDATE employee set salary = 20000 WHERE ssn=333445555;
SELECT salary,ssn FROM EMPLOYEE WHERE ssn=333445555;
UPDATE employee set salary = -20 WHERE ssn=333445555;
SELECT salary,ssn FROM EMPLOYEE WHERE ssn=333445555;

```
mysql> DELIMITER //
mysql> CREATE TRIGGER upd_check BEFORE UPDATE ON employee
    -> FOR EACH ROW
    -> BEGIN
    ->     IF NEW.salary < 0 THEN
    ->         SET NEW.salary = 0;
    ->     ELSEIF NEW.salary > 10000 THEN
    ->         SET NEW.salary = 10000;
    ->     END IF;
    -> END;
    -> //
Query OK, 0 rows affected (0.04 sec)

mysql> DELIMITER ;
mysql> UPDATE employee set salary = 20000 WHERE ssn=333445555
Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT salary,ssn FROM EMPLOYEE WHERE ssn=333445555;
+--------+-----------+
| salary | ssn       |
+--------+-----------+
|  10000 | 333445555 |
+--------+-----------+
1 row in set (0.01 sec)
```

```
mysql> UPDATE employee set salary = -20 WHERE ssn=333445555;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT salary,ssn FROM EMPLOYEE WHERE ssn=333445555;
+--------+-----------+
| salary | ssn       |
+--------+-----------+
|      0 | 333445555 |
+--------+-----------+
1 row in set (0.00 sec)
```

Date: 27/9/24 **EXPERIMENT-11**

CASE STUDY 1

Railway Reservation

Aim:

The railway reservation system facilitates the passengers to enquire about the trains available on the basis of source and destination, booking and cancellation of tickets, enquire about the status of the booked ticket, etc.

The aim of case study is to design and develop a database maintaining the records of different trains, train status, and passengers. The record of train includes its number, name, source, destination, and days on which it is available, whereas record of train status includes dates for which tickets can be booked, total number of seats available, and number of seats already booked. The database has been developed and tested on the MySQL Workbench.

Description:

Passengers can book their tickets for the train in which seats are available. For this, passenger has to provide the desired train number and the date for which ticket is to be booked. Before booking a ticket for a passenger, the validity of train number and booking date is checked. Once the train number and booking date are validated, it is checked whether the seat is available. If yes, the ticket is booked with confirm status and corresponding ticket ID is generated which is stored along with other details of the passenger. After all the available tickets are booked, certain numbers of tickets are booked with waiting status. If waiting lot is also finished, then tickets are not booked and a message of non-availability of seats is displayed. The ticket once booked can be cancelled at any time. For this, the passenger has to provide the ticket ID (the unique key). The ticket ID is searched and the corresponding record is deleted. With this, the first ticket with waiting status also gets confirmed.

List of Assumptions:

Since the reservation system is very large in reality, it is not feasible to develop the case study to that extent and prepare documentation at that level. Therefore, a small sample case study has been created to demonstrate the working of the reservation system. To implement this sample case study, some assumptions have been made, which are as follows:

1. The number of trains has been restricted to five.
2. The booking is open only for next seven days from the current date.
3. Only two categories of tickets can be booked, namely, AC and General.
4. The total number of tickets that can be booked in each category (AC and General) is ten.
5. The total number of tickets that can be given the status of waiting is two.
6. The in-between stoppage stations and their bookings are not considered.

Analysis of data required:

List of trains has to be maintained.

Detailed Passenger information is to be maintained.

In the booking procedure, the train number, train date, and category are read from the passenger.

On the basis of the values provided by the passenger, corresponding record is retrieved from the Train_Status.

If the desired category is AC, then total number of AC seats and number of booked AC seats are compared in order to find whether ticket can be booked or not.

Similarly, it can be checked for the general category.

If ticket can be booked, then passenger details are read and stored in the Passenger table.

Expected flow:

The expected flow of case study for Railway Reservation is as follows:

1. Analyze the data required.

2. Create the conceptual Schema (ER diagrams).

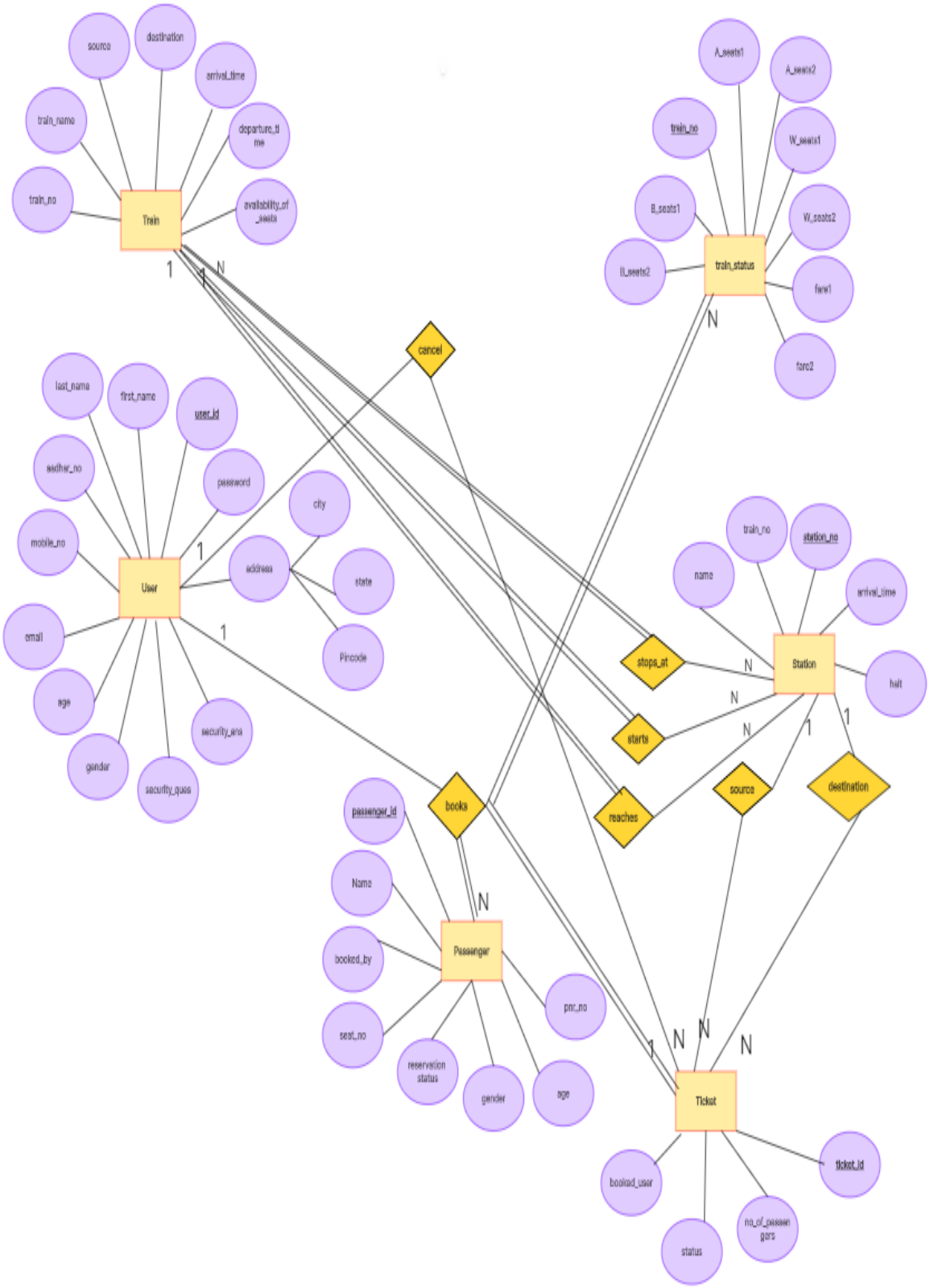
3. Create the relational Schema and apply normalization if required.

4. Implement the relational schema in MySQL.

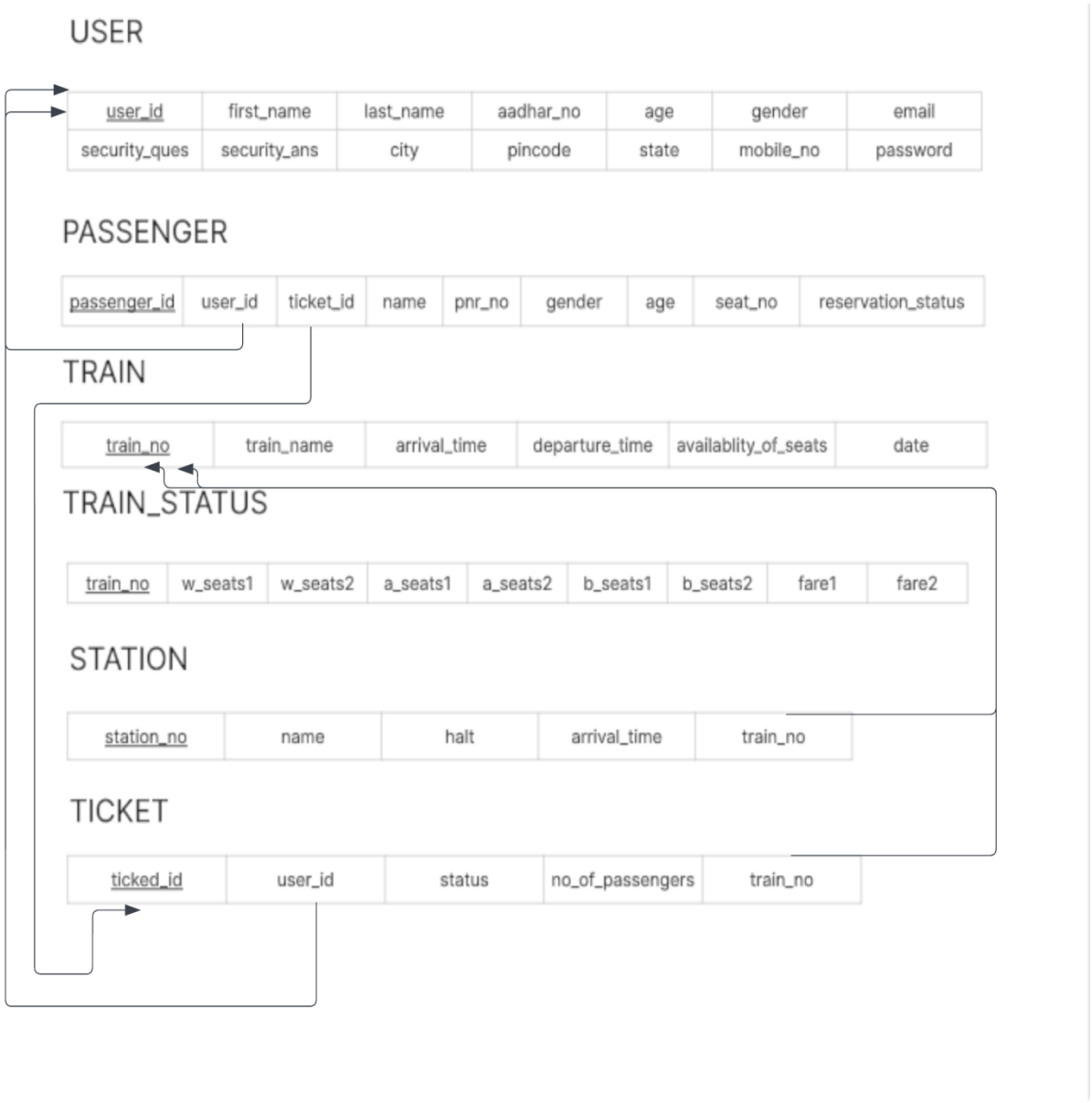
5. Query the database.

6. Generate necessary reports from the database.

Conceptual Schema of Railway Reservation System:



Normalised Relational Schema of Railway Reservation System:



Relationships:

- books - Ternary relationship between USER, TRAIN, PASSENGER and TICKET.
- starts - Between TRAIN and STATION
- reaches - Between TRAIN and STATION
- cancel - Between USER and TICKET
- stops_at - Between TRAIN and STATION

Implementation of Relational Schema in MySQL:

CREATION OF THE TABLES

1. **Users Table**

Query:

```
CREATE TABLE Users (
User_id INT PRIMARY KEY AUTO_INCREMENT,
Password VARCHAR(50),
First_name VARCHAR(50),
Last_name VARCHAR(50),
Gender CHAR(1),
Age INT,
Email VARCHAR(100),
Aadhar_no VARCHAR(12),
Mobile_no VARCHAR(10),
City VARCHAR(50),
State VARCHAR(50),
Pincode VARCHAR(6),
Security_ques VARCHAR(255),
Security_ans VARCHAR(255)  );
desc Users;
```

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

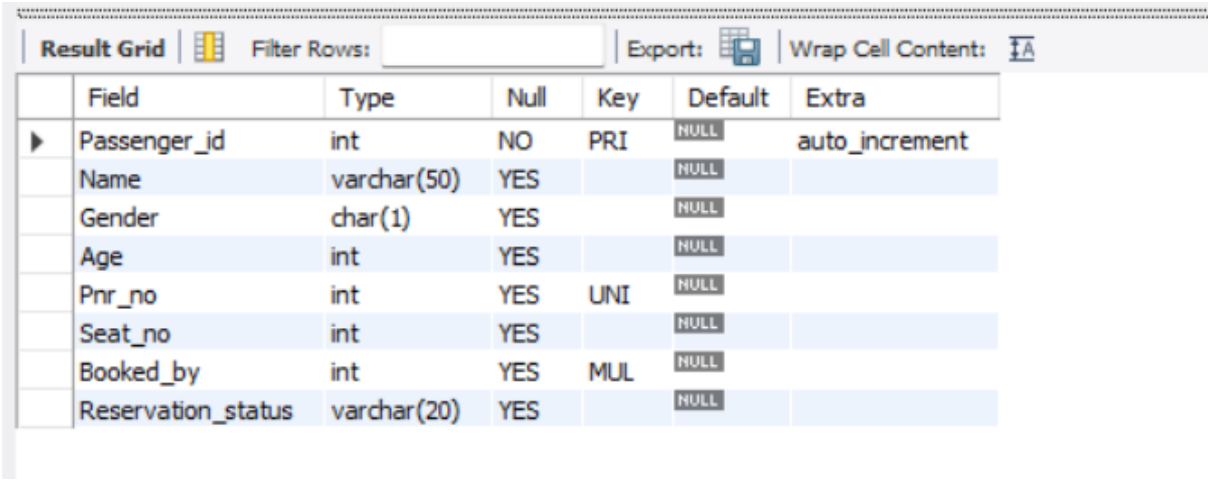
Field	Type	Null	Key	Default	Extra
User_id	int	NO	PRI	NULL	auto_increment
Password	varchar(50)	YES		NULL	
First_name	varchar(50)	YES		NULL	
Last_name	varchar(50)	YES		NULL	
Gender	char(1)	YES		NULL	
Age	int	YES		NULL	
Email	varchar(100)	YES		NULL	
Aadhar_no	varchar(12)	YES		NULL	
Mobile_no	varchar(10)	YES		NULL	
City	varchar(50)	YES		NULL	
State	varchar(50)	YES		NULL	
Pincode	varchar(6)	YES		NULL	
Security_...	varchar(255)	YES		NULL	
Security_ans	varchar(255)	YES		NULL	

2. **Passengers Table**

Query:

```
CREATE TABLE Passengers (
Passenger_id INT PRIMARY KEY AUTO_INCREMENT,
Name VARCHAR(50),
Gender CHAR(1),
Age INT,
Pnr_no INT UNIQUE,
Seat_no INT,
Booked_by INT,
Reservation_status VARCHAR(20),
FOREIGN KEY (Booked_by) REFERENCES Users(User_id)
);
desc Passengers;
```

Output:



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
Passenger_id	int	NO	PRI	NULL	auto_increment
Name	varchar(50)	YES		NULL	
Gender	char(1)	YES		NULL	
Age	int	YES		NULL	
Pnr_no	int	YES	UNI	NULL	
Seat_no	int	YES		NULL	
Booked_by	int	YES	MUL	NULL	
Reservation_status	varchar(20)	YES		NULL	

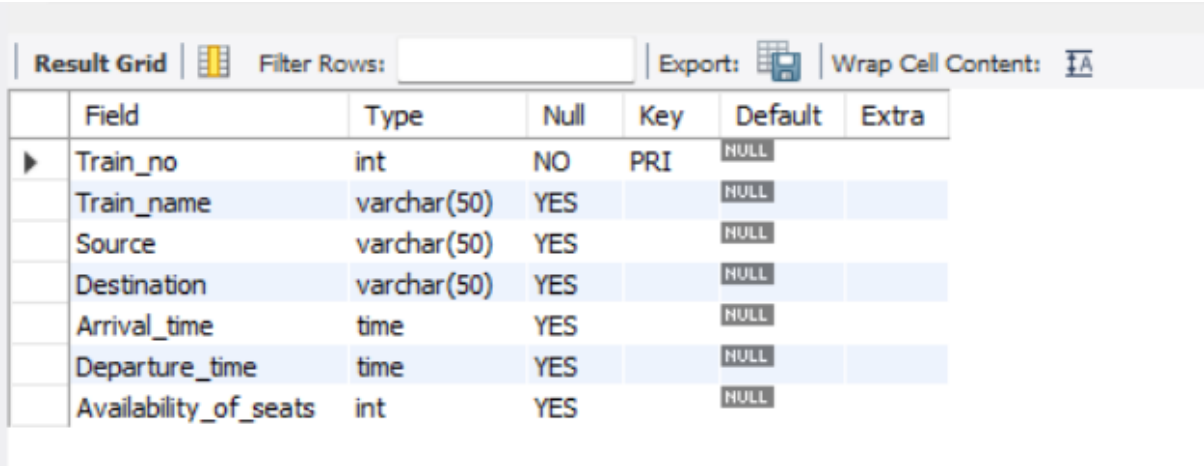
3. **Trains Table**

Query:

```
CREATE TABLE Trains (
Train_no INT PRIMARY KEY,
Train_name VARCHAR(50),
Source VARCHAR(50),
```

```
    Destination VARCHAR(50),
    Arrival_time TIME,
    Departure_time TIME,
    Availability_of_seats INT
);
desc Trains;
```

Output:



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
Train_no	int	NO	PRI	NULL	
Train_name	varchar(50)	YES		NULL	
Source	varchar(50)	YES		NULL	
Destination	varchar(50)	YES		NULL	
Arrival_time	time	YES		NULL	
Departure_time	time	YES		NULL	
Availability_of_seats	int	YES		NULL	

4. Train_Status Table

Query:

```
CREATE TABLE Train_status (
Train_no INT,
A_seats1 INT,
A_seats2 INT,
B_seats1 INT,
B_seats2 INT,
W_seats1 INT,
W_seats2 INT,
FOREIGN KEY (Train_no) REFERENCES Trains(Train_no)
);
desc Train_status;
```

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell C

Field	Type	Null	Key	Default	Extra
Train_no	int	YES	MUL	NULL	
A_seats1	int	YES		NULL	
A_seats2	int	YES		NULL	
B_seats1	int	YES		NULL	
B_seats2	int	YES		NULL	
W_seats1	int	YES		NULL	
W_seats2	int	YES		NULL	

5. Stations Table

Query:

```
CREATE TABLE Stations (
Name VARCHAR(50),
Station_no INT,
Train_no INT,
Arrival_time TIME,
Halt INT,
PRIMARY KEY (Station_no, Train_no),
FOREIGN KEY (Train_no) REFERENCES Trains(Train_no)
);
desc Stations;
```

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
Name	varchar(50)	YES		NULL	
Station_no	int	NO	PRI	NULL	
Train_no	int	NO	PRI	NULL	
Arrival_time	time	YES		NULL	
Hault	int	YES		NULL	

6. Tickets Table

Query:

```
CREATE TABLE Tickets (
Ticket_id INT PRIMARY KEY AUTO_INCREMENT,
Train_no INT,
Booked_user INT,
Status VARCHAR(20),
No_of_passengers INT,
FOREIGN KEY (Train_no) REFERENCES Trains(Train_no),
FOREIGN KEY (Booked_user) REFERENCES Users(User_id)
 );
desc Tickets;
```

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
Ticket_id	int	NO	PRI	NULL	auto_increment
Train_no	int	YES	MUL	NULL	
Booked_user	int	YES	MUL	NULL	
Status	varchar(20)	YES		NULL	
No_of_passengers	int	YES		NULL	

INSERTION OF DATA INTO TABLES

1. **Users Table**

Query:

```
INSERT INTO Users (Password, First_name, Last_name, Gender, Age, Email, Aadhar_no,
Mobile_no, City, State, Pincode, Security_ques, Security_ans)
VALUES
('pass123', 'John', 'Doe', 'M', 30, 'john@example.com', '123456789012', '9876543210',
'Mumbai', 'Maharashtra', '400001', 'Your first pet?', 'Rocky'),
('pass456', 'Jane', 'Smith', 'F', 28, 'jane@example.com', '098765432109', '9123456780', 'Delhi',
'Delhi', '110001', 'Your best friend?', 'Lucy'),
('pass789', 'Alice', 'Johnson', 'F', 24, 'alice@example.com', '567890123456', '9876541234',
'Chennai', 'Tamil Nadu', '600001', 'Your birthplace?', 'Chennai'),
```

('pass321', 'Bob', 'Brown', 'M', 35, 'bob@example.com', '456789012345', '8765432109', 'Bangalore', 'Karnataka', '560001', 'Your favorite food?', 'Pizza'),

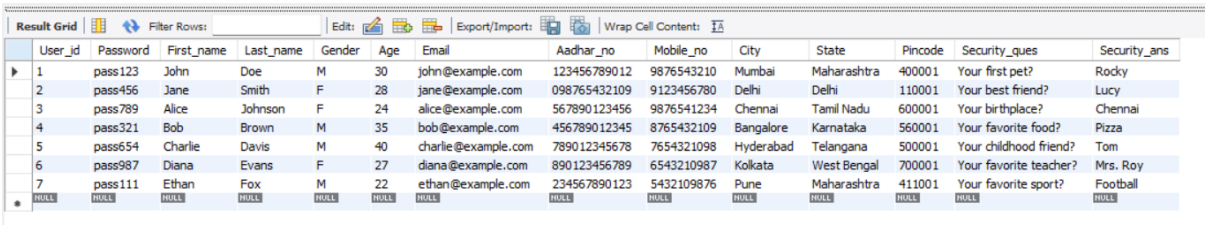
('pass654', 'Charlie', 'Davis', 'M', 40, 'charlie@example.com', '789012345678', '7654321098', 'Hyderabad', 'Telangana', '500001', 'Your childhood friend?', 'Tom'),

('pass987', 'Diana', 'Evans', 'F', 27, 'diana@example.com', '890123456789', '6543210987', 'Kolkata', 'West Bengal', '700001', 'Your favorite teacher?', 'Mrs. Roy'),

('pass111', 'Ethan', 'Fox', 'M', 22, 'ethan@example.com', '234567890123', '5432109876', 'Pune', 'Maharashtra', '411001', 'Your favorite sport?', 'Football');

select * from Users;

Output:



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

User_id	Password	First_name	Last_name	Gender	Age	Email	Aadhar_no	Mobile_no	City	State	Pincode	Security_ques	Security_ans
1	pass123	John	Doe	M	30	john@example.com	123456789012	9876543210	Mumbai	Maharashtra	400001	Your first pet?	Rocky
2	pass456	Jane	Smith	F	28	jane@example.com	098765432109	9123456780	Delhi	Delhi	110001	Your best friend?	Lucy
3	pass789	Alice	Johnson	F	24	alice@example.com	567890123456	9876541234	Chennai	Tamil Nadu	600001	Your birthplace?	Chennai
4	pass321	Bob	Brown	M	35	bob@example.com	456789012345	8765432109	Bangalore	Karnataka	560001	Your favorite food?	Pizza
5	pass654	Charlie	Davis	M	40	charlie@example.com	789012345678	7654321098	Hyderabad	Telangana	500001	Your childhood friend?	Tom
6	pass987	Diana	Evans	F	27	diana@example.com	890123456789	6543210987	Kolkata	West Bengal	700001	Your favorite teacher?	Mrs. Roy
7	pass111	Ethan	Fox	M	22	ethan@example.com	234567890123	5432109876	Pune	Maharashtra	411001	Your favorite sport?	Football
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

2. **Passengers Table**

Query:

INSERT INTO Passengers (Name, Gender, Age, Pnr_no, Seat_no, Booked_by, Reservation_status)

VALUES

('John Doe', 'M', 30, 123456, 1, 1, 'Confirmed'),

('Jane Smith', 'F', 28, 123457, 2, 2, 'Confirmed'),

('Alice Johnson', 'F', 25, 123458, 3, 3, 'Confirmed'),

('Bob Lee', 'M', 40, 123459, 4, 4, 'Waiting'),

('Charlie Brown', 'M', 35, 123460, 5, 5, 'Confirmed'),

('Eve Adams', 'F', 32, 123461, 6, 1, 'Confirmed'),

('Frank Wright', 'M', 45, 123462, 7, 2, 'Waiting');

select * from Passengers;

Output:

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

Passenger_id	Name	Gender	Age	Pnr_no	Seat_no	Booked_by	Reservation_status
1	John Doe	M	30	123456	1	1	Confirmed
2	Jane Smith	F	28	123457	2	2	Confirmed
3	Alice Johnson	F	25	123458	3	3	Confirmed
4	Bob Lee	M	40	123459	4	4	Waiting
5	Charlie Brown	M	35	123460	5	5	Confirmed
6	Eve Adams	F	32	123461	6	1	Confirmed
7	Frank Wright	M	45	123462	7	2	Waiting
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

3. Trains Table

Query:

```
INSERT INTO Trains (Train_no, Train_name, Source, Destination, Arrival_time,
Departure_time, Availability_of_seats)

VALUES

(101, 'Express 1', 'Mumbai', 'Delhi', '10:00:00', '18:00:00', 20),

(102, 'Express 2', 'Delhi', 'Chennai', '12:00:00', '22:00:00', 20),

(103, 'Express 3', 'Mumbai', 'Kolkata', '09:00:00', '20:00:00', 20),

(104, 'Express 4', 'Bangalore', 'Delhi', '11:00:00', '21:00:00', 20),

(105, 'Express 5', 'Hyderabad', 'Pune', '06:00:00', '15:00:00', 20);

select * from Trains;
```

Output:

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

Train_no	Train_name	Source	Destination	Arrival_time	Departure_time	Availability_of_seats
101	Express 1	Mumbai	Delhi	10:00:00	18:00:00	20
102	Express 2	Delhi	Chennai	12:00:00	22:00:00	20
103	Express 3	Mumbai	Kolkata	09:00:00	20:00:00	20
104	Express 4	Bangalore	Delhi	11:00:00	21:00:00	20
105	Express 5	Hyderabad	Pune	06:00:00	15:00:00	20
NULL	NULL	NULL	NULL	NULL	NULL	NULL

4. Train_Status Table

Query:

```
INSERT INTO Train_status (Train_no, A_seats1, A_seats2, B_seats1, B_seats2, W_seats1,
W_seats2)

VALUES

(101, 5, 4, 6, 5, 1, 0),
```

(102, 3, 2, 8, 6, 2, 1),

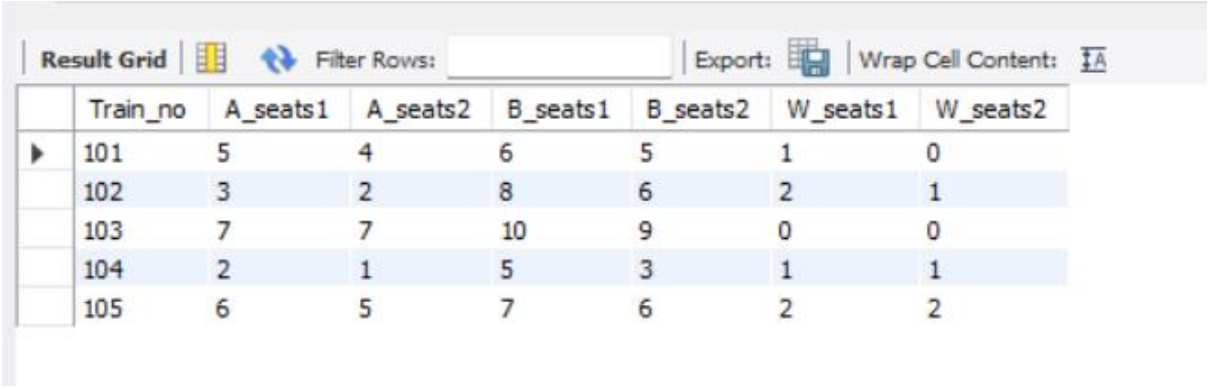
(103, 7, 7, 10, 9, 0, 0),

(104, 2, 1, 5, 3, 1, 1),

(105, 6, 5, 7, 6, 2, 2);

select * from Train_status;

Output:



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Train_no	A_seats1	A_seats2	B_seats1	B_seats2	W_seats1	W_seats2
101	5	4	6	5	1	0
102	3	2	8	6	2	1
103	7	7	10	9	0	0
104	2	1	5	3	1	1
105	6	5	7	6	2	2

5. **Stations Table**

Query:

INSERT INTO Stations (Name, Station_no, Train_no, Arrival_time, Hault)

VALUES

('Mumbai', 1, 101, '10:00:00', 5),

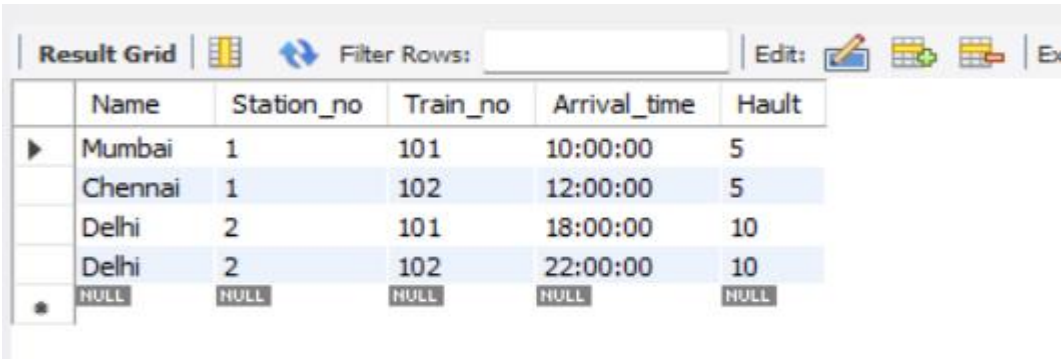
('Delhi', 2, 101, '18:00:00', 10),

('Chennai', 1, 102, '12:00:00', 5),

('Delhi', 2, 102, '22:00:00', 10);

select * from Stations;

Output:



Result Grid | Filter Rows: | Edit:

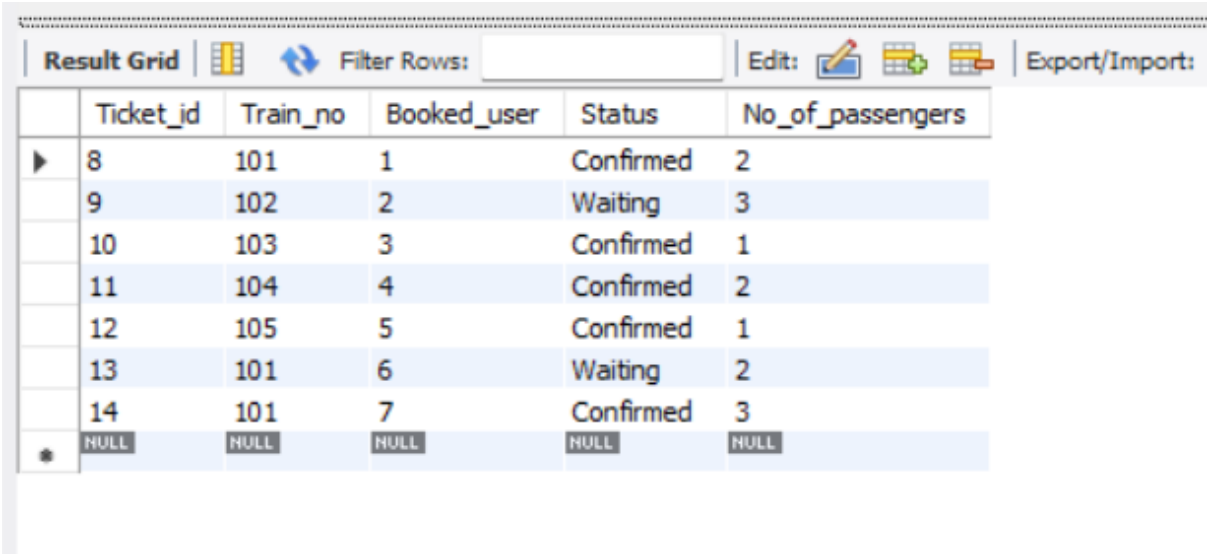
Name	Station_no	Train_no	Arrival_time	Hault
Mumbai	1	101	10:00:00	5
Chennai	1	102	12:00:00	5
Delhi	2	101	18:00:00	10
Delhi	2	102	22:00:00	10
NULL	NULL	NULL	NULL	NULL

6. Tickets Table

Query:

```
INSERT INTO Tickets (Train_no, Booked_user, Status, No_of_passengers)
VALUES
(101, 1, 'Confirmed', 2),
(102, 2, 'Waiting', 3),
(103, 3, 'Confirmed', 1),
(104, 4, 'Confirmed', 2),
(105, 5, 'Confirmed', 1),
(101, 6, 'Waiting', 2),
(101, 7, 'Confirmed', 3);
select * from Tickets;
```

Output:



Result Grid | Filter Rows: | Edit: | Export/Import:

Ticket_id	Train_no	Booked_user	Status	No_of_passengers
8	101	1	Confirmed	2
9	102	2	Waiting	3
10	103	3	Confirmed	1
11	104	4	Confirmed	2
12	105	5	Confirmed	1
13	101	6	Waiting	2
14	101	7	Confirmed	3
NULL	NULL	NULL	NULL	NULL

Querying the Database:

1. Display the passenger details travelling by "Express 1" train.

Query:

```
SELECT p.Passenger_id, p.Name, p.Pnr_no, t.Train_name, t.Source, t.Destination
FROM Passengers p
JOIN Tickets tk ON p.Booked_by = tk.Booked_user
JOIN Trains t ON t.Train_no = tk.Train_no
WHERE t.Train_name = 'Express 1';
```

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Passenger_id	Name	Pnr_no	Train_name	Source	Destination
1	John Doe	123456	Express 1	Mumbai	Delhi
6	Eve Adams	123461	Express 1	Mumbai	Delhi

2. Get the list of trains along with the available seats.

Query:

```
SELECT t.Train_name, ts.A_seats1, ts.A_seats2, ts.B_seats1, ts.B_seats2, ts.W_seats1,
ts.W_seats2

FROM Trains t

JOIN Train_status ts ON t.Train_no = ts.Train_no;
```

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Train_name	A_seats1	A_seats2	B_seats1	B_seats2	W_seats1	W_seats2
Express 1	5	4	6	5	1	0
Express 2	3	2	8	6	2	1
Express 3	7	7	10	9	0	0
Express 4	2	1	5	3	1	1
Express 5	6	5	7	6	2	2

3. Find all users who have booked tickets and their booking status

Query:

```
SELECT u.First_name, u.Last_name, tk.Status, t.Train_name, t.Source, t.Destination

FROM Users u

JOIN Tickets tk ON u.User_id = tk.Booked_user

JOIN Trains t ON t.Train_no = tk.Train_no;
```

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

First_name	Last_name	Status	Train_name	Source	Destination
John	Doe	Confirmed	Express 1	Mumbai	Delhi
Diana	Evans	Waiting	Express 1	Mumbai	Delhi
Ethan	Fox	Confirmed	Express 1	Mumbai	Delhi
Jane	Smith	Waiting	Express 2	Delhi	Chennai
Alice	Johnson	Confirmed	Express 3	Mumbai	Kolkata
Bob	Brown	Confirmed	Express 4	Bangalore	Delhi
Charlie	Davis	Confirmed	Express 5	Hyderabad	Pune

4. Find all trains where the total number of available seats in AC class (both A_seats1 and A_seats2) is greater than 10.

Query:

```
SELECT Train_name
FROM Trains
WHERE Train_no IN (
   SELECT Train_no
   FROM Train_status
   WHERE (A_seats1 + A_seats2) > 10
);
```

Output:

Train_name
Express 3
Express 5

5. Retrieve the names of those passengers with 'Confirmed' reservations

Query:

```
SELECT Name, Reservation_status
FROM Passengers
WHERE Reservation_status = 'Confirmed';
```

Output:

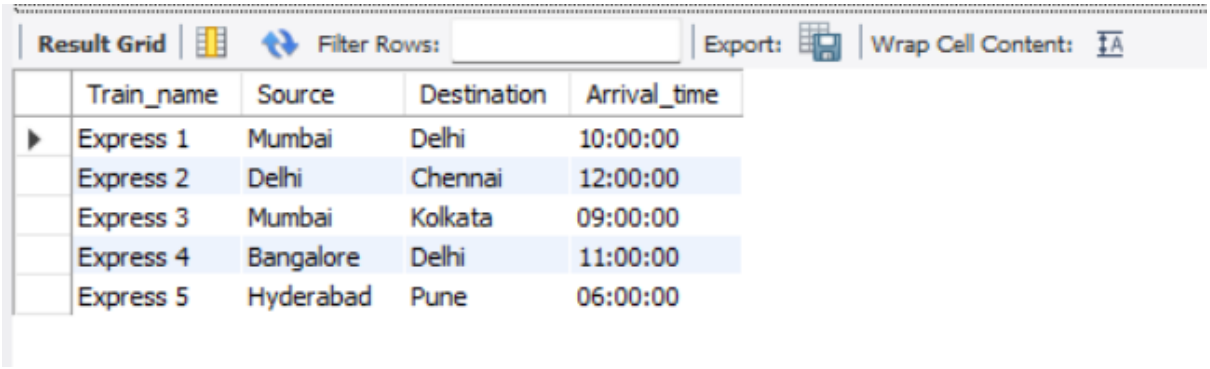
Name	Reservation_status
John Doe	Confirmed
Jane Smith	Confirmed
Alice Johnson	Confirmed
Charlie Brown	Confirmed
Eve Adams	Confirmed

6. Display the details of trains that arrive anytime between 6:00am and 12:00pm

Query:

```
SELECT Train_name, Source, Destination, Arrival_time
FROM Trains
WHERE Arrival_time BETWEEN '06:00:00' AND '12:00:00';
```

Output:



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Train_name	Source	Destination	Arrival_time
Express 1	Mumbai	Delhi	10:00:00
Express 2	Delhi	Chennai	12:00:00
Express 3	Mumbai	Kolkata	09:00:00
Express 4	Bangalore	Delhi	11:00:00
Express 5	Hyderabad	Pune	06:00:00

7.List trains leaving from Mumbai along with their departure times.

Query:

```
SELECT Train_name, Departure_time
FROM Trains
WHERE Source = 'Mumbai';
```

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Train_name	Departure_time
Express 1	18:00:00
Express 3	20:00:00

CASE STUDY 2

Hospital Management System

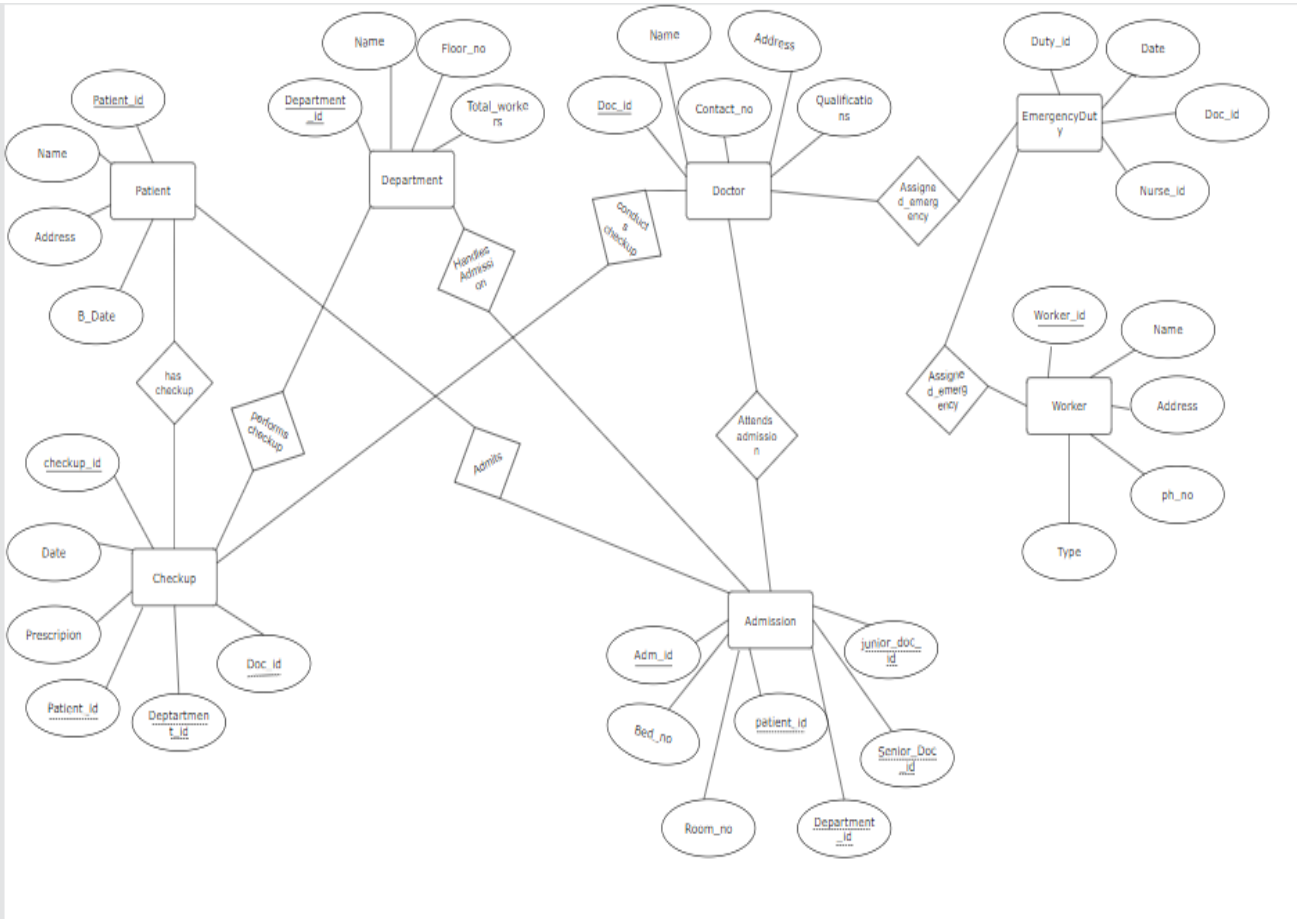
Whenever a new patient is either admitted or comes for outdoor check-up, a uniquepatient id is generated after storing the name, address and date of birth of thepatient. For furthervisits, the patient uses his unique id. There are severaldepartments in the Hospital. Adepartment characterized by a unique id, name,floor number and total workers. Thedoctors have a unique employee id, and theirname, address, contact number, qualifications are stored with this id.Similarly other workers like nurses, ward boys, ambulance drivers also have aunique employee id. Also, each worker is characterized by name, address and type.Doctors and workers can be associated with multiple departments with differentschedules. Whenever patient is admitted in the hospital various details arerecorded. Thepatient id, name and address are stored, the department number andname in which thepatient is admitted along with the bed number and room numberis also stored. Also, forevery patient, a senior doctor and junior doctor areappointed. The details of the doctors, like name, id, contact number is recorded.The prescribed medicines are also stored for apatient. In case of outdoor checkups,the patient id is stored, along with the departmentnumber, the employee id of thedoctor, the prescription and the date of checkup. Foremergency duty every night,the employee_id of doctor and the nurse is stored along withdate.

1. Data Analysis:

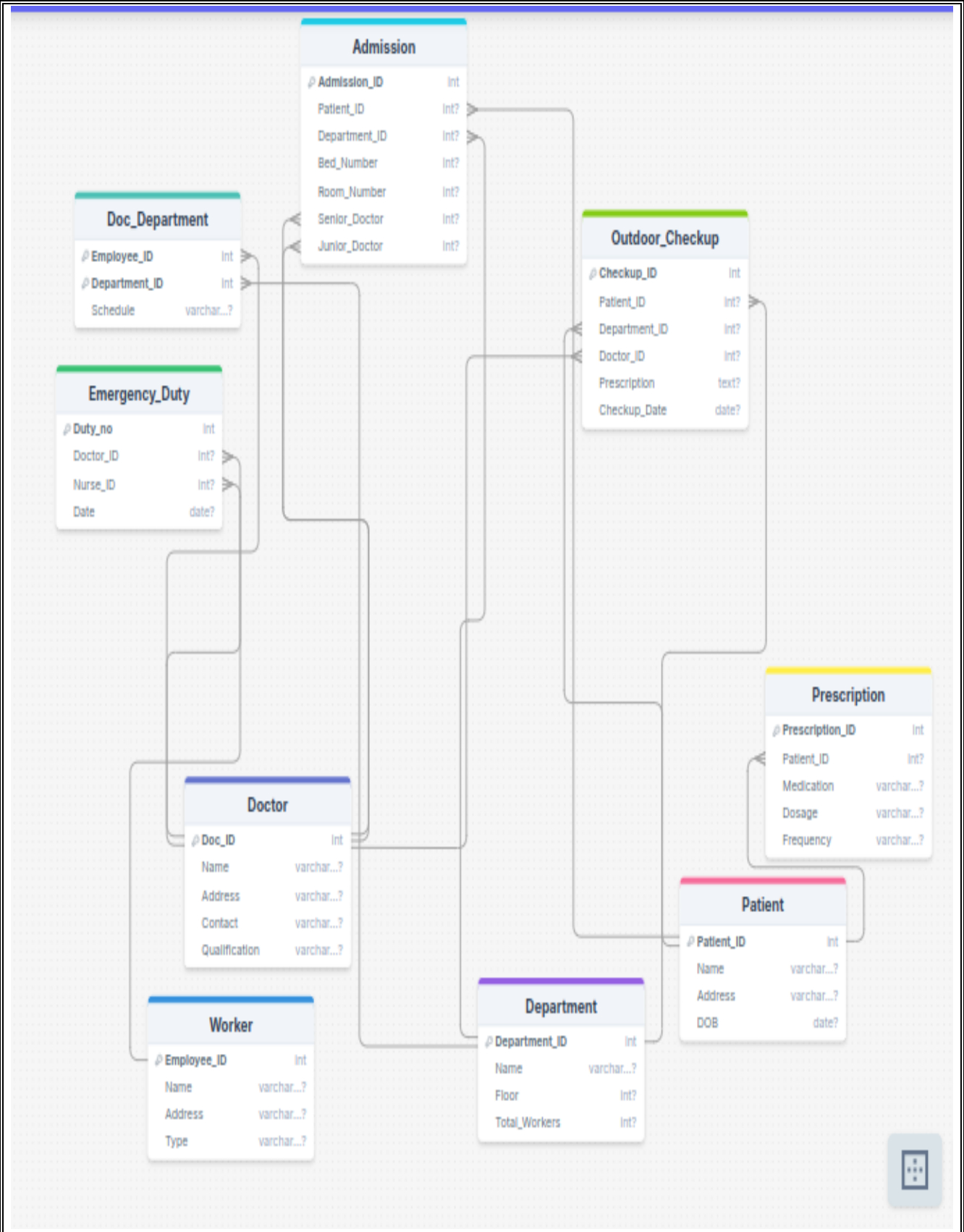
The data required for the Hospital Management System can be categorized into the following entities:

1. **Patient:** Each patient has a unique ID, name, address, and date of birth. For admitted patients, additional data includes department number, bed number, room number, and assigned doctors.
2. **Department:** Each department has a unique ID, name, floor number, and total workers.
3. **Doctor:** Each doctor has a unique employee ID, name, address, contact number, qualifications, and is associated with one or more departments.
4. **Worker:** Workers (nurses, ward boys, ambulance drivers) also have a unique employee ID, name, address, and type. They can be associated with multiple departments.
5. **Admissions:** When a patient is admitted, details such as patient ID, department, room, and assigned doctors are recorded.
6. **Outdoor Checkup:** For outdoor checkups, patient ID, department, doctor ID, prescription, and date of checkup are stored.
7. **Emergency Duty:** Employee IDs of doctors and nurses are recorded along with the date for night duties.
8. **Prescriptions:** Each patient's prescription, which includes medications and associated information, is stored.

2. Conceptual Schema (ER Diagram):



3. Relational Schema and Normalization:



3.2 Normalization Process:

The relational schema is normalized to the 3rd Normal Form (3NF).

1NF (First Normal Form): All tables have **"atomic"** columns with unique rows.

2NF (Second Normal Form): Each non-key attribute is **"fully functionally dependent"** on the primary key.

For example: doctor-department relationships are stored in a separate table, Doc_Department, to prevent partial dependency.

3NF (Third Normal Form): Transitive dependencies are removed. Each non-key attribute **"depends solely"** on the primary key of its table.

4. Implementing in MySQL:

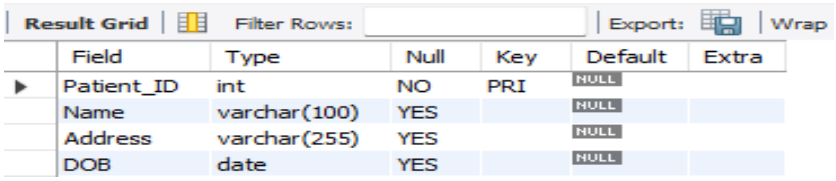
Creating tables:

1. **Patient:**

Query:

```
CREATE TABLE Patient (
 Patient_ID INT PRIMARY KEY,
Name VARCHAR(100),
Address VARCHAR(255),
DOB DATE );
```

Output:



Field	Type	Null	Key	Default	Extra
Patient_ID	int	NO	PRI	NULL	
Name	varchar(100)	YES		NULL	
Address	varchar(255)	YES		NULL	
DOB	date	YES		NULL	

2. **Department:**

Query:

```
CREATE TABLE Department (
Department_ID INT PRIMARY KEY,
 Name VARCHAR(100),
Floor INT,
Total_Workers INT );
```

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Con

Field	Type	Null	Key	Default	Extra
Department_ID	int	NO	PRI	NULL	
Name	varchar(100)	YES		NULL	
Floor	int	YES		NULL	
Total_Workers	int	YES		NULL	

3. **Doctor:**

Query:

```
CREATE TABLE Doctor (
Employee_ID INT PRIMARY KEY,
Name VARCHAR(100),
Address VARCHAR(255),
Contact VARCHAR(15),
Qualification VARCHAR(100) );
```

Output:

Result Grid | Filter Rows: | Export: | Wrap

Field	Type	Null	Key	Default	Extra
Employee_ID	int	NO	PRI	NULL	
Name	varchar(100)	YES		NULL	
Address	varchar(255)	YES		NULL	
Contact	varchar(15)	YES		NULL	
Qualification	varchar(100)	YES		NULL	

4. **Worker:**

Query:

```
CREATE TABLE Worker (
Employee_ID INT PRIMARY KEY,
Name VARCHAR(100),
Address VARCHAR(255),
Type VARCHAR(50) );
```

Output:

Result Grid | Filter Rows: | Export: | Wrap C

Field	Type	Null	Key	Default	Extra
Employee_ID	int	NO	PRI	NULL	
Name	varchar(100)	YES		NULL	
Address	varchar(255)	YES		NULL	
Type	varchar(50)	YES		NULL	

5. **Doc_Department:**

Query:

CREATE TABLE Doc_Department(

Employee_ID INT,

Department_ID INT,

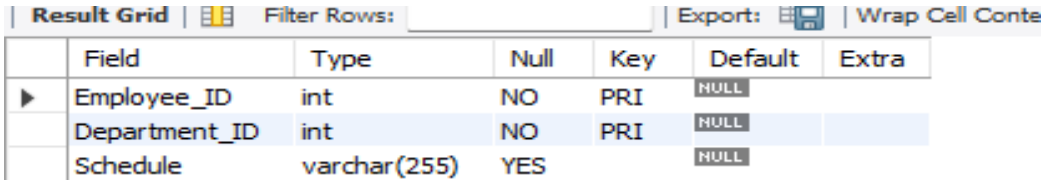
Schedule VARCHAR(255),

FOREIGN KEY (Employee_ID) REFERENCES Doctor(Employee_ID),

FOREIGN KEY (Department_ID) REFERENCES Department(Department_ID),

PRIMARY KEY (Employee_ID, Department_ID));

Output:



Result Grid | Filter Rows: | Export: | Wrap Cell Conte

Field	Type	Null	Key	Default	Extra
Employee_ID	int	NO	PRI	NULL	
Department_ID	int	NO	PRI	NULL	
Schedule	varchar(255)	YES		NULL	

6. **Admission:**

Query:

CREATE TABLE Admission (

Admission_ID INT PRIMARY KEY,

Patient_ID INT,

Department_ID INT,

Bed_Number INT,

Room_Number INT,

Senior_Doctor INT,

Junior_Doctor INT,

FOREIGN KEY (Patient_ID) REFERENCES Patient(Patient_ID),

FOREIGN KEY (Department_ID) REFERENCES Department(Department_ID),

FOREIGN KEY (Senior_Doctor) REFERENCES Doctor(Employee_ID),

FOREIGN KEY (Junior_Doctor) REFERENCES Doctor(Employee_ID)

);

Output:

Result Grid | Filter Rows: | Export:

Field	Type	Null	Key	Default	Ex
Admission_ID	int	NO	PRI	NULL	
Patient_ID	int	YES	MUL	NULL	
Department_ID	int	YES	MUL	NULL	
Bed_Number	int	YES		NULL	
Room_Number	int	YES		NULL	
Senior_Doctor	int	YES	MUL	NULL	
Junior_Doctor	int	YES	MUL	NULL	

7. **Outdoor_checkup:**

Query:

```
CREATE TABLE Outdoor_Checkup( ‘
Checkup_ID INT PRIMARY KEY,
Patient_ID INT,
Department_ID INT,
 Doctor_ID INT,
Prescription TEXT,
Checkup_Date DATE,
FOREIGN KEY (Patient_ID) REFERENCES Patient(Patient_ID),
FOREIGN KEY (Department_ID) REFERENCES Department(Department_ID),
FOREIGN KEY (Doctor_ID) REFERENCES Doctor(Employee_ID) );
```

Output:

Result Grid | Filter Rows: | Export:

Field	Type	Null	Key	Default	Extra
Checkup_ID	int	NO	PRI	NULL	
Patient_ID	int	YES	MUL	NULL	
Department_ID	int	YES	MUL	NULL	
Doc_ID	int	YES	MUL	NULL	
Prescription	text	YES		NULL	
Checkup_Date	date	YES		NULL	

8. **Emergency_Duty:**

Query:

```
CREATE TABLE Emergency_Duty(
Duty_ID INT PRIMARY KEY,
Doctor_ID INT,
Nurse_ID INT,
Date DATE,
```

FOREIGN KEY (Doctor_ID) REFERENCES Doctor(Employee_ID),

FOREIGN KEY (Nurse_ID) REFERENCES Worker(Employee_ID));

Output:

	Field	Type	Null	Key	Default	Extra
▶	Duty_ID	int	NO	PRI	NULL	
	Doctor_ID	int	YES	MUL	NULL	
	Nurse_ID	int	YES	MUL	NULL	
	Date	date	YES		NULL	

9. **Prescription:**

Query:

CREATE TABLE Prescription (

Prescription_ID INT PRIMARY KEY,

Patient_ID INT,

Medication VARCHAR(255),

Dosage VARCHAR(50),

Frequency VARCHAR(50),

FOREIGN KEY (Patient_ID) REFERENCES Patient(Patient_ID));

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Con

	Field	Type	Null	Key	Default	Extra
▶	Prescription_ID	int	NO	PRI	NULL	
	Patient_ID	int	YES	MUL	NULL	
	Medication	varchar(255)	YES		NULL	
	Dosage	varchar(50)	YES		NULL	
	Frequency	varchar(50)	YES		NULL	

Inserting data into tables:

1. **Patient**

Query:

INSERT INTO Patient (Patient_ID, Name, Address, DOB)

VALUES

(1, 'John Doe', '123 Main St, City A', '1985-06-12'),

(2, 'Jane Smith', '456 Oak St, City B', '1990-03-25');

Output:

Result Grid | Filter Rows: | Edit:

Patient_ID	Name	Address	DOB
1	John Doe	123 Main St, City A	1985-06-12
2	Jane Smith	456 Oak St, City B	1990-03-25
NULL	NULL	NULL	NULL

2. **Department**

Query:

```
INSERT INTO Department (Department_ID, Name, Floor, Total_Workers)
VALUES (101, 'Cardiology', 2, 20), (102, 'Neurology', 3, 15);
```

Output:

Result Grid | Filter Rows: | Edit

Department_ID	Name	Floor	Total_Workers
101	Cardiology	2	20
102	Neurology	3	15
NULL	NULL	NULL	NULL

3. **Doctor**

Query:

```
INSERT INTO Doctor (Employee_ID, Name, Address, Contact, Qualification)
VALUES
(201, 'Dr. Alan Turner', '789 Pine St, City A', '1234567890', 'MD Cardiology'),
(202, 'Dr. Emily Clark', '321 Cedar St, City B', '0987654321', 'MD Neurology');
```

Output:

Result Grid | Filter Rows: | Edit: | Export/Import:

Employee_ID	Name	Address	Contact	Qualification
201	Dr. Alan Turner	789 Pine St, City A	1234567890	MD Cardiology
202	Dr. Emily Clark	321 Cedar St, City B	0987654321	MD Neurology
NULL	NULL	NULL	NULL	NULL

4. **Worker**

Query:

```
INSERT INTO Worker (Employee_ID, Name, Address, Type)
VALUES (301, 'Nurse Amy Green', '567 Birch St, City C', 'Nurse'),
 (302, 'Ward Boy Mike Blue', '890 Maple St, City D', 'Ward Boy');
```

Output:

Result Grid | Filter Rows: | Edit:

Employee_ID	Name	Address	Type
301	Nurse Amy Green	567 Birch St, City C	Nurse
302	Ward Boy Mike Blue	890 Maple St, City D	Ward Boy
NULL	NULL	NULL	NULL

5. **Doc_Department:**

Query:

```
INSERT INTO Doc_Department (Employee_ID, Department_ID, Schedule)

VALUES (201, 101, 'Mon-Fri, 9am-5pm'),(202, 102, 'Mon-Wed, 10am-6pm');
```

Output:

Result Grid | Filter Rows: | Edit:

Employee_ID	Department_ID	Schedule
201	101	Mon-Fri, 9am-5pm
202	102	Mon-Wed, 10am-6pm
NULL	NULL	NULL

6. **Admission**

Query:

```
INSERT INTO Admission (Admission_ID, Patient_ID, Department_ID, Bed_Number,
Room_Number, Senior_Doctor, Junior_Doctor)

VALUES (1, 1, 101, 12, 101, 201, 202), (2, 2, 102, 8, 102, 202, 201);
```

Output:

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Conten

Admission_ID	Patient_ID	Department_ID	Bed_Number	Room_Number	Senior_Doctor	Junior_Doctor
1	1	101	12	101	201	202
2	2	102	8	102	202	201
NULL	NULL	NULL	NULL	NULL	NULL	NULL

7. **Outdoor_Checkup**

Query:

```
INSERT INTO Outdoor_Checkup (Checkup_ID, Patient_ID, Department_ID, Doc_ID,
Prescription, Checkup_Date)

VALUES

(1, 1, 101, 201, 'Medication A', '2024-09-10'),

(2, 2, 102, 202, 'Medication B', '2024-09-11');
```

Output:

Result Grid | Filter Rows: | Edit: | Export/Import:

Checkup_ID	Patient_ID	Department_ID	Doc_ID	Prescription	Checkup_Date
1	1	101	201	Medication A	2024-09-10
2	2	102	202	Medication B	2024-09-11
NULL	NULL	NULL	NULL	NULL	NULL

8. **Emergency_Duty:**

Query:

```
INSERT INTO Emergency_Duty (Duty_ID, Doctor_ID, Nurse_ID, Date)
VALUES (1, 201, 301, '2024-09-12'),(2, 202, 301, '2024-09-13');
```

Output:

Result Grid | Filter Rows: | Edit:

Duty_ID	Doctor_ID	Nurse_ID	Date
1	201	301	2024-09-12
2	202	301	2024-09-13
NULL	NULL	NULL	NULL

9. **Prescription:**

Query:

```
INSERT INTO Prescription (Prescription_ID, Patient_ID, Medication, Dosage, Frequency)
VALUES (1, 1, 'Medication A', '500mg', 'Twice a Day'),
(2, 2, 'Medication B', '250mg', 'Once a Day');
```

Output:

Result Grid | Filter Rows: | Edit:

Prescription_ID	Patient_ID	Medication	Dosage	Frequency
1	1	Medication A	500mg	Twice a Day
2	2	Medication B	250mg	Once a Day
NULL	NULL	NULL	NULL	NULL

5. Query the Database:

1. Retrive all Patients

Query:

```
select * from Patients;
```

Output:

Result Grid | Filter Rows: | Edit:

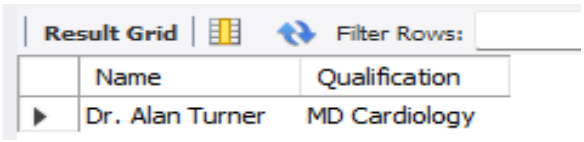
Patient_ID	Name	Address	DOB
1	John Doe	123 Main St, City A	1985-06-12
2	Jane Smith	456 Oak St, City B	1990-03-25
NULL	NULL	NULL	NULL

2. List all Doctors working in the Cardiology Department (Department_ID = 101)

Query:

SELECT D.Name, D.Qualification FROM Doctor D JOIN Doc_Department DD ON D.Employee_ID = DD.Employee_ID WHERE DD.Department_ID = 101;

Output:



Result Grid | Filter Rows:

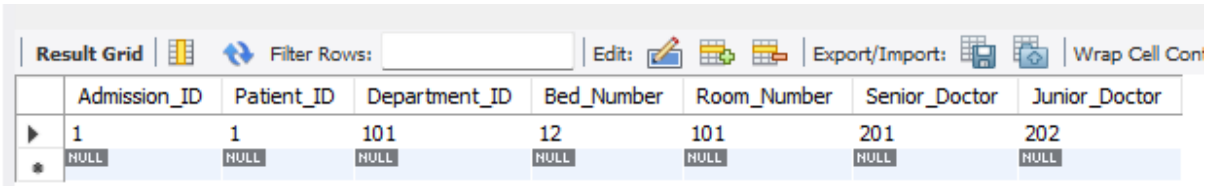
	Name	Qualification
▶	Dr. Alan Turner	MD Cardiology

3. Find all Admissions for a specific Patient (Patient_ID = 1)

Query:

SELECT * FROM Admission WHERE Patient_ID = 1;

Output:



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Con

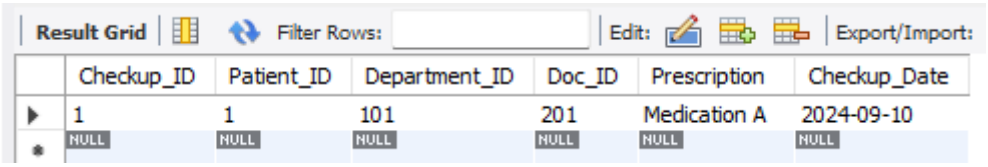
	Admission_ID	Patient_ID	Department_ID	Bed_Number	Room_Number	Senior_Doctor	Junior_Doctor
▶	1	1	101	12	101	201	202
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

4. Retrieve all Checkups for a specific Patient (Patient_ID = 1)

Query:

SELECT * FROM Outdoor_CheckupWHERE Patient_ID = 1;

Output:



Result Grid | Filter Rows: | Edit: | Export/Import:

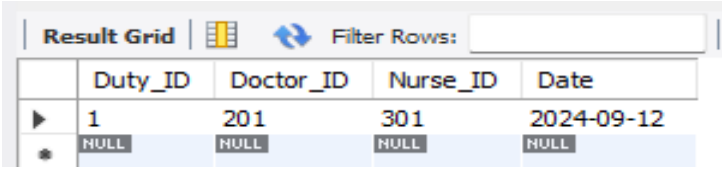
	Checkup_ID	Patient_ID	Department_ID	Doc_ID	Prescription	Checkup_Date
▶	1	1	101	201	Medication A	2024-09-10
*	NULL	NULL	NULL	NULL	NULL	NULL

5. List all Emergency Duties of a specific Doctor (Doctor_ID = 201)

Query:

SELECT * FROM Emergency_Duty WHERE Doctor_ID = 201;

Output:



Result Grid | Filter Rows:

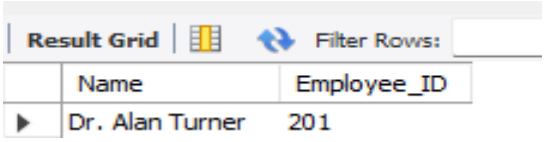
	Duty_ID	Doctor_ID	Nurse_ID	Date
▶	1	201	301	2024-09-12
*	NULL	NULL	NULL	NULL

6. Find doctors who are on emergency duty on a specific date

Query:

SELECT D.Name, D.Employee_ID FROM Doctor D JOIN Emergency_Duty ED ON D.Employee_ID = ED.Doctor_ID WHERE ED.Date = '2024-09-12';

Output:



Result Grid | Filter Rows:

Name	Employee_ID
Dr. Alan Turner	201

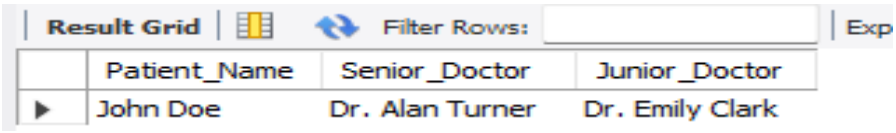
7. Retrieve the names of all patients admitted to a specific department along with their senior and junior doctors' names

Query:

SELECT P.Name AS Patient_Name, D1.Name AS Senior_Doctor, D2.Name AS Junior_Doctor FROM Admission AJOIN Patient P ON A.Patient_ID = P.Patient_ID

JOIN Doctor D1 ON A.Senior_Doctor = D1.Employee_ID JOIN Doctor D2 ON A.Junior_Doctor = D2.Employee_ID WHERE A.Department_ID = 101;

Output:



Result Grid | Filter Rows: | Exp

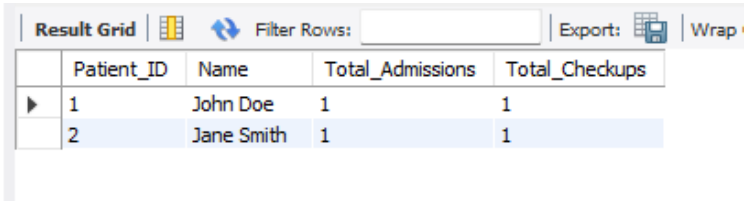
Patient_Name	Senior_Doctor	Junior_Doctor
John Doe	Dr. Alan Turner	Dr. Emily Clark

8.Find the total number of checkups and admissions for each patient

Query:

SELECT P.Patient_ID, P.Name, (SELECT COUNT(*) FROM Admission A WHERE A.Patient_ID = P.Patient_ID) AS Total_Admissions,(SELECT COUNT(*) FROM Outdoor_Checkup OC WHERE OC.Patient_ID = P.Patient_ID) AS Total_Checkups FROM Patient P;

Output:



Result Grid | Filter Rows: | Export: | Wrap

Patient_ID	Name	Total_Admissions	Total_Checkups
1	John Doe	1	1
2	Jane Smith	1	1

CASE STUDY 3

AirLine Reservation System

There are 6 different airlines in 6 different countries: Canada – AirCan, USA - USAir, UK - BritAir, France - AirFrance, Germany - LuftAir, Italy - ItalAir.

Their flights involve the following 12 cities: Toronto and Montreal in Canada, New York and Chicago in US, London and Edinburgh in UK, Paris and Nice in France, Bonn and Berlin in Germany, Rome and Naples in Italy.

In each of the 12 cities, there is a (single) booking office.

You are going to design a central air-reservation database to be used by all booking offices.

The flight has a unique flight number, air line code, business class indicator, smoking allowed indicator.

Flight availability has flight number, date + time of departure, number of total seats available in business class, number of booked seats in business class, number of total seats available in economy class, and number of booked seats in economy class.

The customers may come from any country, not just the 6 above, and from any province/state, and from any city. Customer has first & last name, mailing address, zero or more phone numbers, zero or more fax numbers, and zero or more email addresses.

Mailing address has street, city, province or state, postal code and country. Phone/fax number has country code, area code and local number.

Email address has only one string, and no structure is assumed. A customer can book one or more flights.

Two or more customers may have same mailing address and/or same phone number(s) and/or same fax number(s). But the email address is unique for each customer. First and last names do not have to be unique.

Booking has an unique booking number, booking city, booking date, flight number, date + time of departure (in local time, and time is always in hours and minutes), date + time of arrival (in local time), class indicator, total price (airport tax in origin + airport tax in destination + flight price – in local currency. The flight price for business class is 1.5 times of the listed flight price), status indicator (three types: booked. Canceled – the customer canceled the booking, scratched – the customer had not paid in full 30 days prior to the departure), customer who is responsible for payment, amount-paid-so far (in local currency), outstanding balance (in local currency), the first & last names to be printed on the ticket.

The airport taxes must be stored in local currencies (i.e. Canadian Master Thesis – Weiguang Zhang McMaster University- Computing and Software 26 dollars, US dollars, British Pounds, French francs, German marks, and Italian Liras).

Since the exchange rates change daily, they also must be stored for calculations of all prices involved. Though France, Germany, and Italy have had a common currency for a while, we used the names of their original currencies to involve in this exercise currency exchange rates and their changes.

1. Data Analysis:

1. Airline ↔ Flight (Operates)

- Airline: Total participation (Every airline must operate at least one flight).
- Flight: Partial participation (A flight must be operated by one airline, but not all flights are necessarily operated by every airline).

2. City ↔ BookingOffice (LocatedIn)

- City: Total participation (Every city must have one booking office).
- BookingOffice: Total participation (Every booking office must be located in one city).

3. Flight ↔ FlightAvailability (Scheduled)

- Flight: Total participation (Every flight must have at least one availability record).
- FlightAvailability: Total participation (Every flight availability record must be associated with one flight).

4. Customer ↔ MailingAddress (ResidesAt)

- Customer: Partial participation (Not every customer must have a mailing address).
- MailingAddress: Partial participation (Not every mailing address must be associated with a customer).

5. Customer ↔ PhoneFaxNumber (HasContact)

- Customer: Partial participation (Not every customer must have a phone/fax number).
- PhoneFaxNumber: Total participation (Every phone/fax number must be associated with one customer).

6. Flight ↔ Booking (BookedBy)

- Flight: Partial participation (Not every flight must be booked).
- Booking: Total participation (Every booking must be associated with one flight).

7. Customer ↔ Booking (Makes)

- Customer: Partial participation (Not every customer must make a booking).
- Booking: Total participation (Every booking must be made by one customer).

8. City ↔ AirportTax (Imposes)

- City: Total participation (Every city must impose an airport tax).
- AirportTax: Total participation (Every airport tax must be associated with one city).

9. Country ↔ Airline (Belongs)

- Country: Partial participation (Not every country must have an airline).
- Airline: Total participation (Every airline must belong to one country).

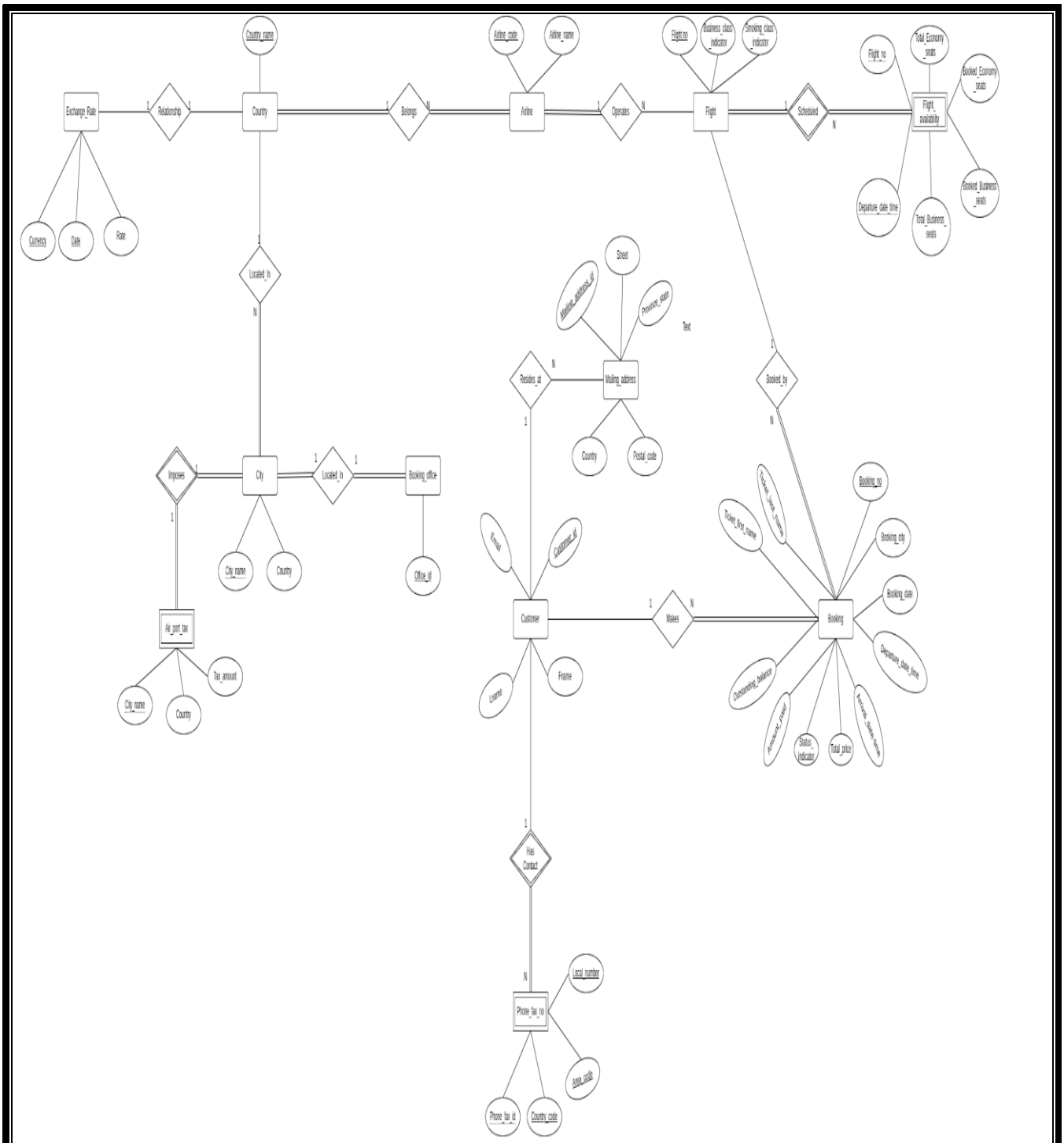
10. Country ↔ City (LocatedIn)

- Country: Total participation (Every country must have at least one city).
- City: Total participation (Every city must be located in one country).

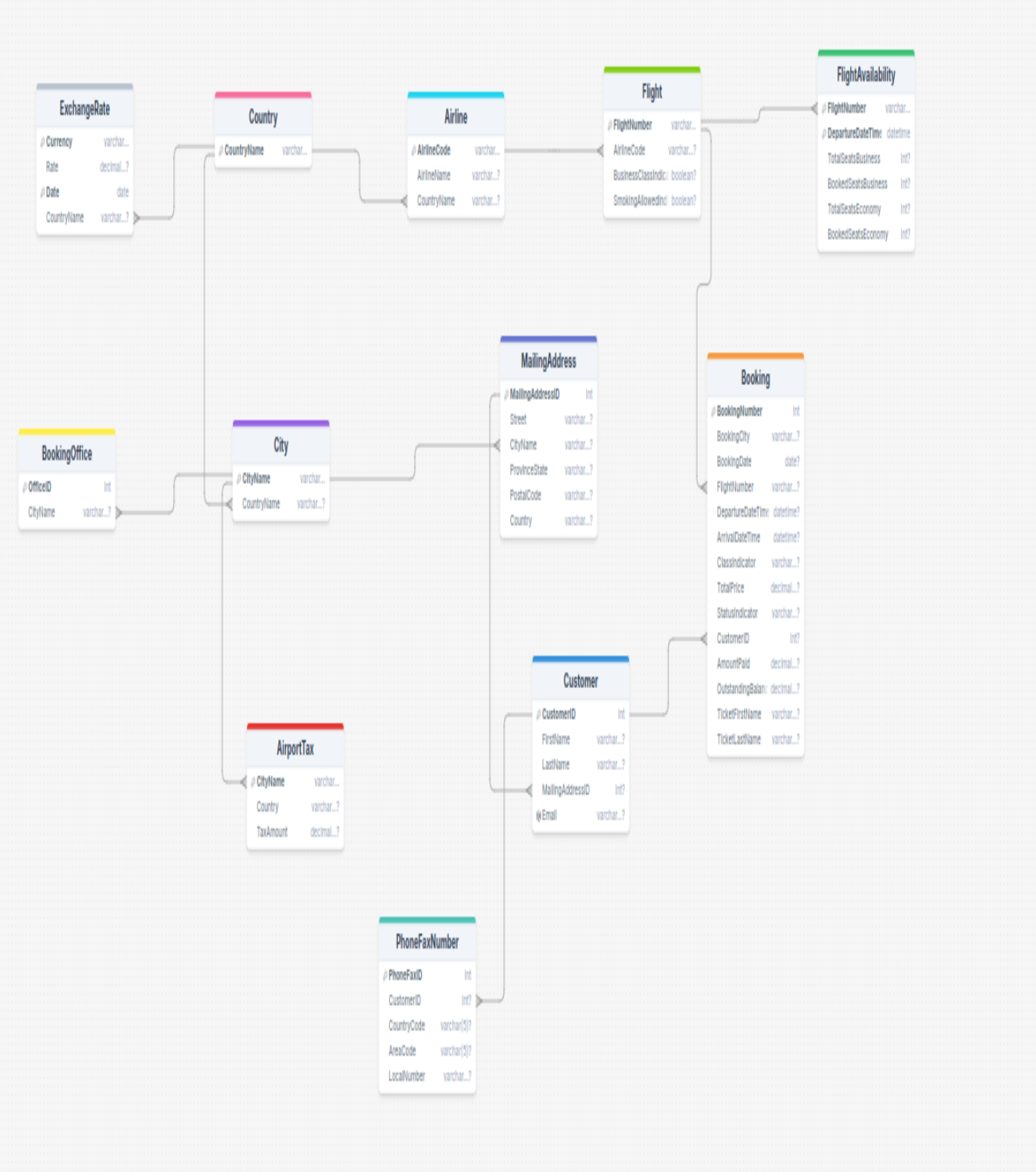
11. Country ↔ ExchangeRate (Gives)

- Country: Partial participation (Not every country must have an exchange rate with another country).
- ExchangeRate: Total participation (Every exchange rate must be associated with one country)

2. Conceptual Schema (ER Diagram):


Country_name
Airline_code
Airline_name
Flight no
Business class indicator
Smoking class indicator
Flight no
Total Economy seats
Booked Economy seats
Exchange Rate
Relationship
Country
Belongs
Airline
Operates
Flight
Scheduled
Flight availability
Currency
Date
Rate
Departure date time
Total Business seats
Booked Business seats
Located In
Street
Mailing address id
Province state
Text
Resides at
Mailing address
Booked by
Country
Postal code
Imposes
City
Located In
Booking office
Booking no
Ticket key name
Booking city
Booking date
City name
Country
Office id
Air port tax
Customer
Makes
Booking
Departure date time
City name
Country
Tax amount
Outstanding balance
Amount paid
Status indicator
Total price
Has Contact
Phone fax no
Local number
Area code
Phone fax id
Country code

3. Relational Schema and Normalization:



4. Implementing in MySQL:

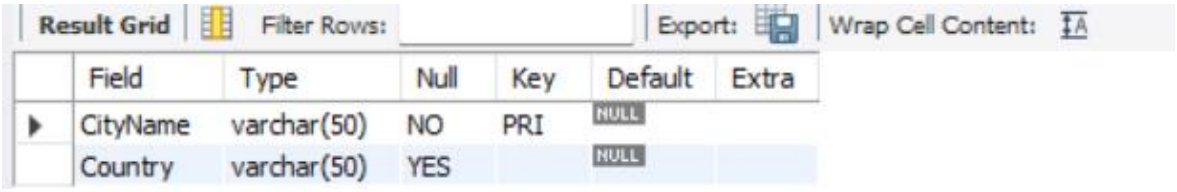
CREATION OF THE TABLES

1. **City Table**

Query:

```
CREATE TABLE City (
   CityName VARCHAR(50) PRIMARY KEY,
   Country VARCHAR(50)
);
desc city;
```

Output:



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
CityName	varchar(50)	NO	PRI	NULL	
Country	varchar(50)	YES		NULL	

2. **Mailing Address Table**

Query :

```
CREATE TABLE MailingAddress (
   MailingAddressID INT PRIMARY KEY AUTO_INCREMENT,
   Street VARCHAR(100),
   CityName VARCHAR(50),
   ProvinceState VARCHAR(50),
   PostalCode VARCHAR(20),
   Country VARCHAR(50),
   FOREIGN KEY (CityName) REFERENCES City(CityName)
);
desc MailingAddress;
```

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
MailingAddressID	int	NO	PRI	NULL	auto_increment
Street	varchar(100)	YES		NULL	
CityName	varchar(50)	YES	MUL	NULL	
ProvinceState	varchar(50)	YES		NULL	
PostalCode	varchar(20)	YES		NULL	
Country	varchar(50)	YES		NULL	

3. Customer Table

Query:

```
CREATE TABLE Customer (
    CustomerID INT PRIMARY KEY AUTO_INCREMENT,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    MailingAddressID INT,
    Email VARCHAR(100) UNIQUE,
    FOREIGN KEY (MailingAddressID) REFERENCES MailingAddress(MailingAddressID)
);
desc Customer;
```

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
CustomerID	int	NO	PRI	NULL	auto_increment
FirstName	varchar(50)	YES		NULL	
LastName	varchar(50)	YES		NULL	
MailingAddressID	int	YES	MUL	NULL	
Email	varchar(100)	YES	UNI	NULL	

4. PhoneFaxNumber Table

Query:

```
CREATE TABLE PhoneFaxNumber (
    PhoneFaxID INT PRIMARY KEY AUTO_INCREMENT,
    CustomerID INT,
    CountryCode VARCHAR(5),
    AreaCode VARCHAR(5),
```

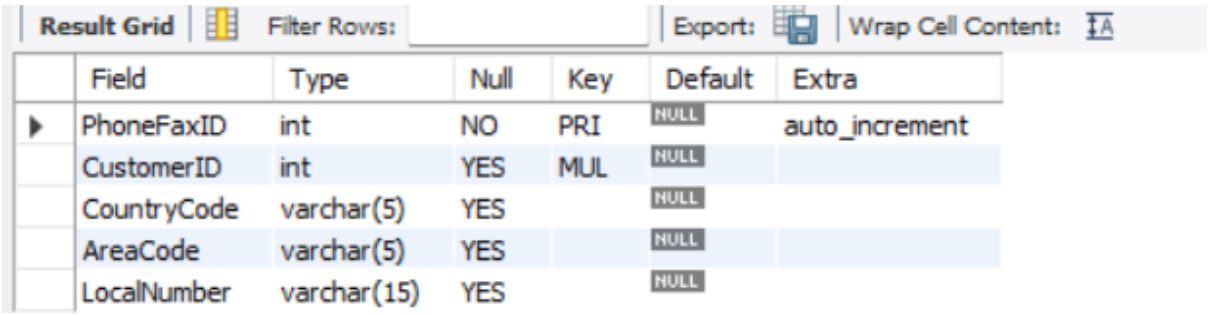
LocalNumber VARCHAR(15),

FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)

);

desc PhoneFaxNumber;

Output:



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
PhoneFaxID	int	NO	PRI	NULL	auto_increment
CustomerID	int	YES	MUL	NULL	
CountryCode	varchar(5)	YES		NULL	
AreaCode	varchar(5)	YES		NULL	
LocalNumber	varchar(15)	YES		NULL	

5. Airline Table

Query:

CREATE TABLE Airline (

AirlineCode VARCHAR(10) PRIMARY KEY,

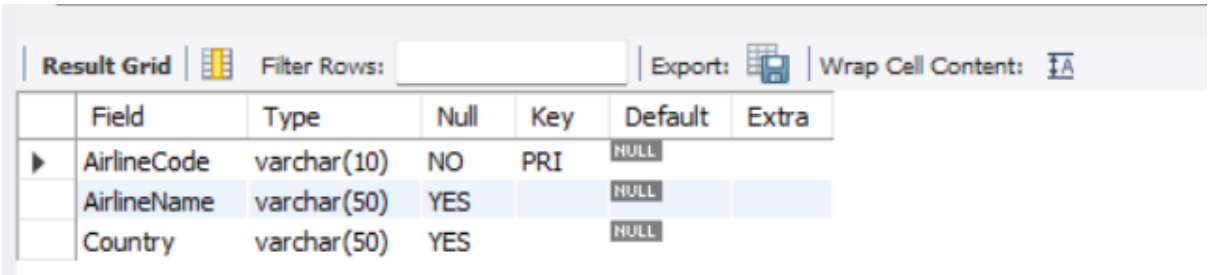
AirlineName VARCHAR(50),

Country VARCHAR(50)

);

desc airline;

Output:



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
AirlineCode	varchar(10)	NO	PRI	NULL	
AirlineName	varchar(50)	YES		NULL	
Country	varchar(50)	YES		NULL	

6. Flight Table

Query:

CREATE TABLE Flight (

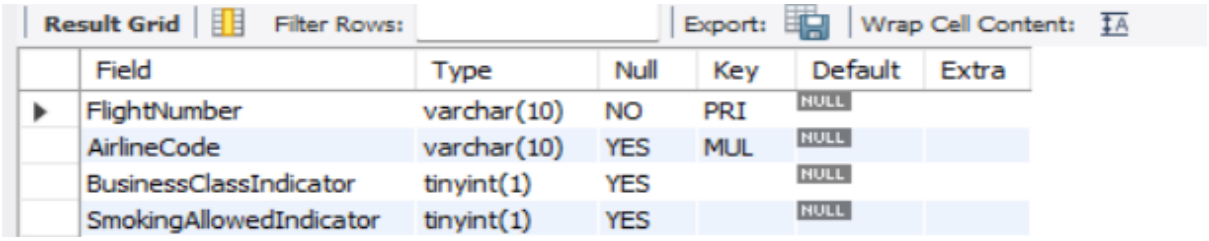
FlightNumber VARCHAR(10) PRIMARY KEY,

AirlineCode VARCHAR(10),

BusinessClassIndicator BOOLEAN,

```
    SmokingAllowedIndicator BOOLEAN,
    FOREIGN KEY (AirlineCode) REFERENCES Airline(AirlineCode)
);
desc flight;
```

Output:



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

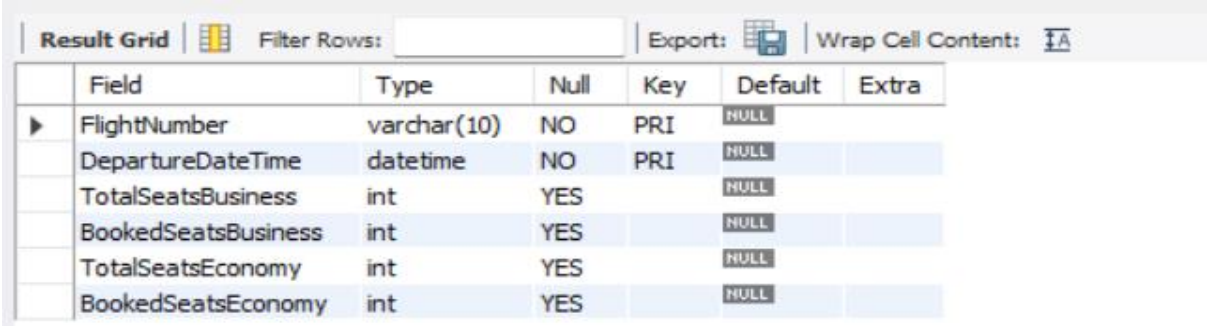
Field	Type	Null	Key	Default	Extra
FlightNumber	varchar(10)	NO	PRI	NULL	
AirlineCode	varchar(10)	YES	MUL	NULL	
BusinessClassIndicator	tinyint(1)	YES		NULL	
SmokingAllowedIndicator	tinyint(1)	YES		NULL	

7. FlightAvailability Table

Query:

```
CREATE TABLE FlightAvailability (
    FlightNumber VARCHAR(10),
    DepartureDateTime DATETIME,
    TotalSeatsBusiness INT,
    BookedSeatsBusiness INT,
    TotalSeatsEconomy INT,
    BookedSeatsEconomy INT,
    PRIMARY KEY (FlightNumber, DepartureDateTime),
    FOREIGN KEY (FlightNumber) REFERENCES Flight(FlightNumber)
);
desc FlightAvailability;
```

Output:



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

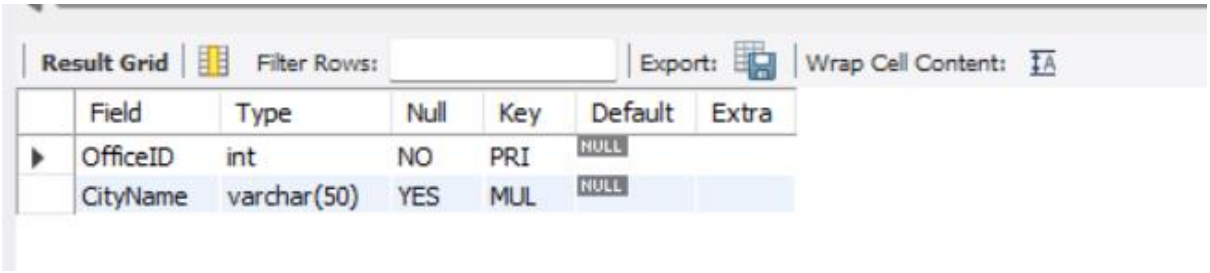
Field	Type	Null	Key	Default	Extra
FlightNumber	varchar(10)	NO	PRI	NULL	
DepartureDateTime	datetime	NO	PRI	NULL	
TotalSeatsBusiness	int	YES		NULL	
BookedSeatsBusiness	int	YES		NULL	
TotalSeatsEconomy	int	YES		NULL	
BookedSeatsEconomy	int	YES		NULL	

8. BookingOffice Table

Query:

```
CREATE TABLE BookingOffice (
   OfficeID INT PRIMARY KEY,
   CityName VARCHAR(50),
   FOREIGN KEY (CityName) REFERENCES City(CityName)
);
desc BookingOffice;
```

Output:



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Field	Type	Null	Key	Default	Extra
OfficeID	int	NO	PRI	NULL	
CityName	varchar(50)	YES	MUL	NULL	

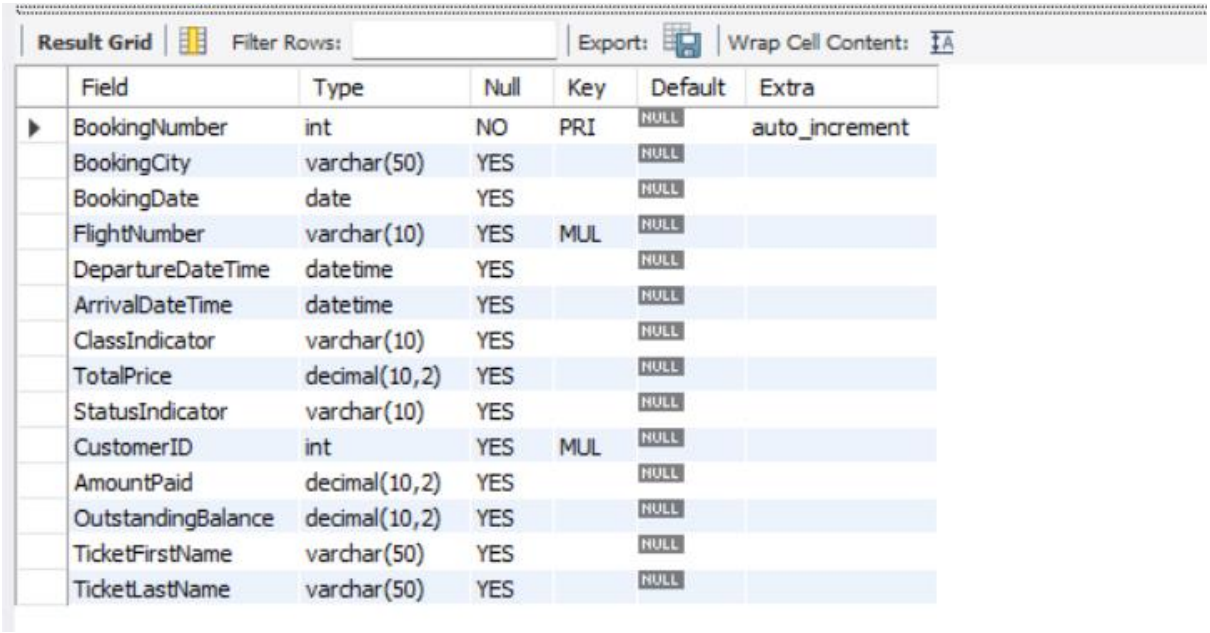
9. Booking Table

Query:

```
CREATE TABLE Booking (
BookingNumber INT PRIMARY KEY AUTO_INCREMENT,
BookingCity VARCHAR(50),
BookingDate DATE,
FlightNumber VARCHAR(10),
DepartureDateTime DATETIME,
ArrivalDateTime DATETIME,
ClassIndicator VARCHAR(10),
TotalPrice DECIMAL(10, 2),
StatusIndicator VARCHAR(10),
CustomerID INT,
AmountPaid DECIMAL(10, 2),
OutstandingBalance DECIMAL(10, 2),
TicketFirstName VARCHAR(50),
TicketLastName VARCHAR(50),
```

```
    FOREIGN KEY (FlightNumber) REFERENCES Flight(FlightNumber),
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)
);
desc Booking;
```

Output:



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

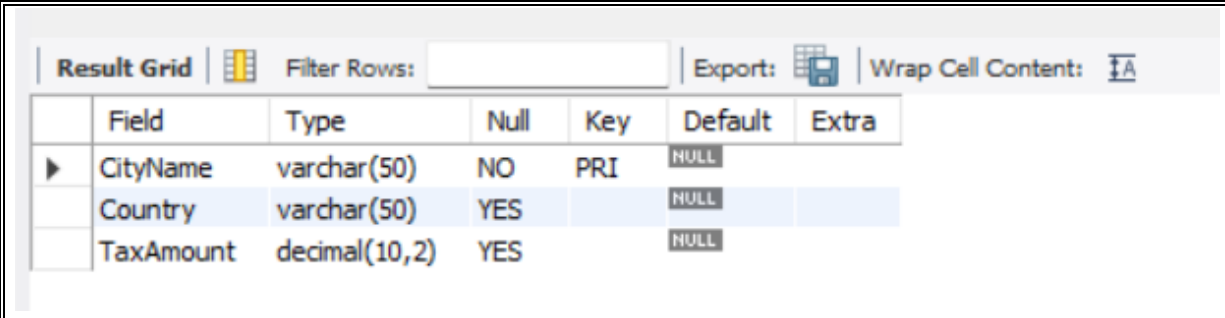
Field	Type	Null	Key	Default	Extra
BookingNumber	int	NO	PRI	NULL	auto_increment
BookingCity	varchar(50)	YES		NULL	
BookingDate	date	YES		NULL	
FlightNumber	varchar(10)	YES	MUL	NULL	
DepartureDateTime	datetime	YES		NULL	
ArrivalDateTime	datetime	YES		NULL	
ClassIndicator	varchar(10)	YES		NULL	
TotalPrice	decimal(10,2)	YES		NULL	
StatusIndicator	varchar(10)	YES		NULL	
CustomerID	int	YES	MUL	NULL	
AmountPaid	decimal(10,2)	YES		NULL	
OutstandingBalance	decimal(10,2)	YES		NULL	
TicketFirstName	varchar(50)	YES		NULL	
TicketLastName	varchar(50)	YES		NULL	

10. AirportTax Table

Query:

```
CREATE TABLE AirportTax (
    CityName VARCHAR(50),
    Country VARCHAR(50),
    TaxAmount DECIMAL(10, 2),
    PRIMARY KEY (CityName),
    FOREIGN KEY (CityName) REFERENCES City(CityName)
);
desc AirportTax;
```

Output:



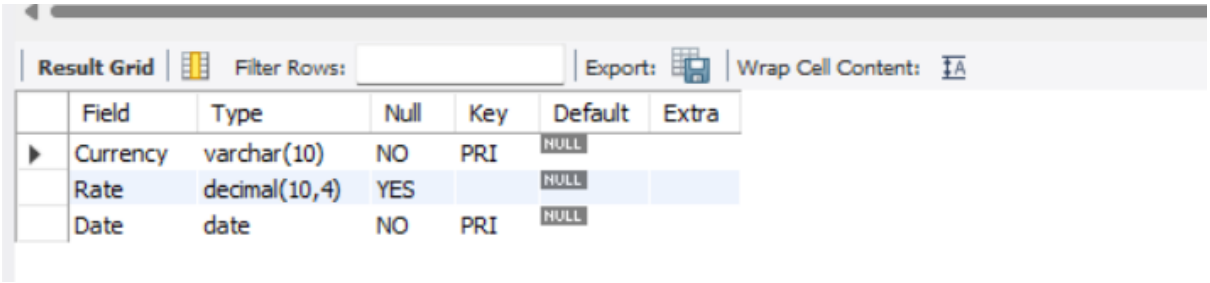
Field	Type	Null	Key	Default	Extra
CityName	varchar(50)	NO	PRI	NULL	
Country	varchar(50)	YES		NULL	
TaxAmount	decimal(10,2)	YES		NULL	

11. ExchangeRateTable

Query:

```
CREATE TABLE ExchangeRate (
    Currency VARCHAR(10),
    Rate DECIMAL(10, 4),
    Date DATE,
    PRIMARY KEY (Currency, Date)
);
desc ExchangeRate;
```

Output:



Field	Type	Null	Key	Default	Extra
Currency	varchar(10)	NO	PRI	NULL	
Rate	decimal(10,4)	YES		NULL	
Date	date	NO	PRI	NULL	

INSERTION OF DATA INTO TABLES

1. **Airline Table**

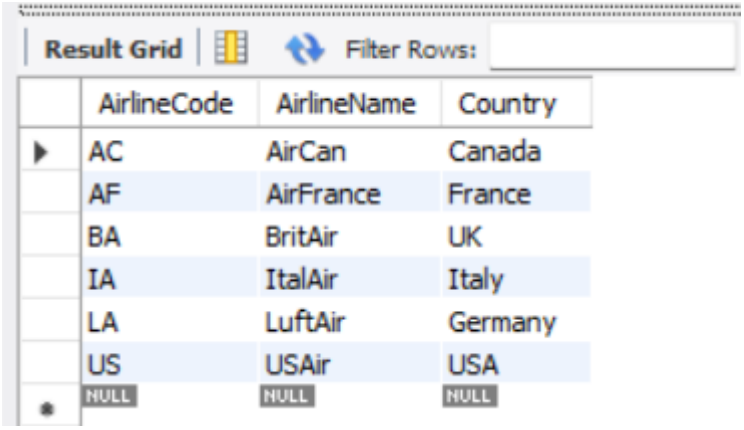
Query:

```
INSERT INTO Airline (AirlineCode, AirlineName, Country)
VALUES
('AC', 'AirCan', 'Canada'),
('US', 'USAir', 'USA'),
('BA', 'BritAir', 'UK'),
('AF', 'AirFrance', 'France'),
('LA', 'LuftAir', 'Germany'),
```

('IA', 'ItalAir', 'Italy');

Select * from Airline;

Output:



Result Grid | Filter Rows:

AirlineCode	AirlineName	Country
AC	AirCan	Canada
AF	AirFrance	France
BA	BritAir	UK
IA	ItalAir	Italy
LA	LuftAir	Germany
US	USAir	USA
NULL	NULL	NULL

2. **City Table**

Query:

INSERT INTO City (CityName, Country)

VALUES

('Toronto', 'Canada'),

('Montreal', 'Canada'),

('New York', 'USA'),

('Chicago', 'USA'),

('London', 'UK'),

('Edinburgh', 'UK'),

('Paris', 'France'),

('Nice', 'France'),

('Bonn', 'Germany'),

('Berlin', 'Germany'),

('Rome', 'Italy'),

('Naples', 'Italy');

select * from city;

Output:

Result Grid | Filter Rows:

CityName	Country
Berlin	Germany
Bonn	Germany
Chicago	USA
Edinburgh	UK
London	UK
Montreal	Canada
Naples	Italy
New York	USA
Nice	France
Paris	France
Rome	Italy
Toronto	Canada
NULL	NULL

3. MailingAddress Table

Query:

```
INSERT INTO MailingAddress (Street, CityName, ProvinceState, PostalCode, Country)
VALUES
('123 Queen St', 'Toronto', 'ON', 'M5V1Z4', 'Canada'),
('456 Rue St', 'Montreal', 'QC', 'H2Y1W9', 'Canada'),
('789 Broadway', 'New York', 'NY', '10003', 'USA'),
('101 Michigan Ave', 'Chicago', 'IL', '60611', 'USA'),
('102 Baker St', 'London', 'England', 'SW1A 2DR', 'UK'),
('103 Princes St', 'Edinburgh', 'Scotland', 'EH2 2ER', 'UK');
select * from MailingAddress;
```

Output:

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

MailingAddressID	Street	CityName	ProvinceState	PostalCode	Country
1	123 Queen St	Toronto	ON	M5V1Z4	Canada
2	456 Rue St	Montreal	QC	H2Y1W9	Canada
3	789 Broadway	New York	NY	10003	USA
4	101 Michigan Ave	Chicago	IL	60611	USA
5	102 Baker St	London	England	SW1A 2DR	UK
6	103 Princes St	Edinburgh	Scotland	EH2 2ER	UK
NULL	NULL	NULL	NULL	NULL	NULL

4. Customer Table

Query:

```
INSERT INTO Customer (FirstName, LastName, MailingAddressID, Email)
VALUES
('Alice', 'Smith', 1, 'alice.smith@example.com'),
```

('Bob', 'Jones', 2, 'bob.jones@example.com'),

('Charlie', 'Brown', 3, 'charlie.brown@example.com'),

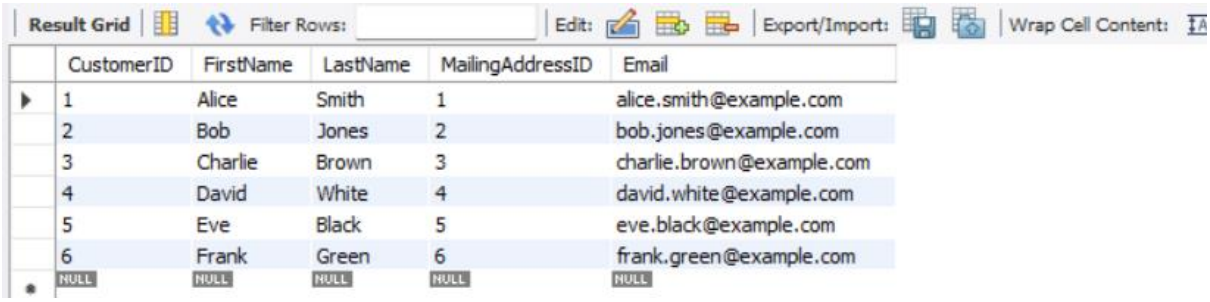
('David', 'White', 4, 'david.white@example.com'),

('Eve', 'Black', 5, 'eve.black@example.com'),

('Frank', 'Green', 6, 'frank.green@example.com');

select * from Customer;

Output:



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

CustomerID	FirstName	LastName	MailingAddressID	Email
1	Alice	Smith	1	alice.smith@example.com
2	Bob	Jones	2	bob.jones@example.com
3	Charlie	Brown	3	charlie.brown@example.com
4	David	White	4	david.white@example.com
5	Eve	Black	5	eve.black@example.com
6	Frank	Green	6	frank.green@example.com
NULL	NULL	NULL	NULL	NULL

5. **PhoneFaxNumber Table**

Query:

INSERT INTO PhoneFaxNumber (CustomerID, CountryCode, AreaCode, LocalNumber) VALUES

(1, '+1', '416', '5551234'),

(2, '+1', '514', '5555678'),

(3, '+1', '212', '5557890'),

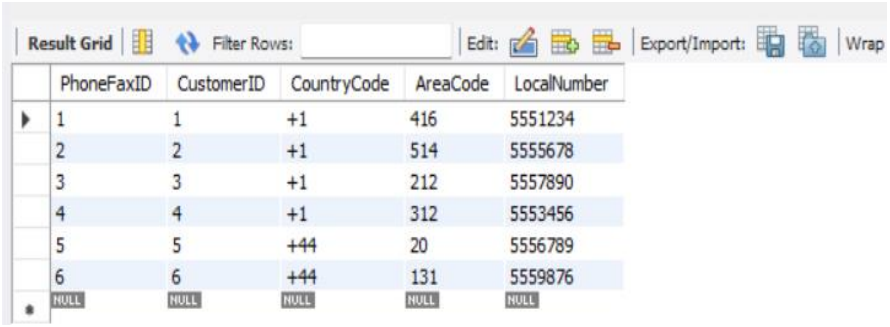
(4, '+1', '312', '5553456'),

(5, '+44', '20', '5556789'),

(6, '+44', '131', '5559876');

select * from PhoneFaxNumber;

Output:



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap

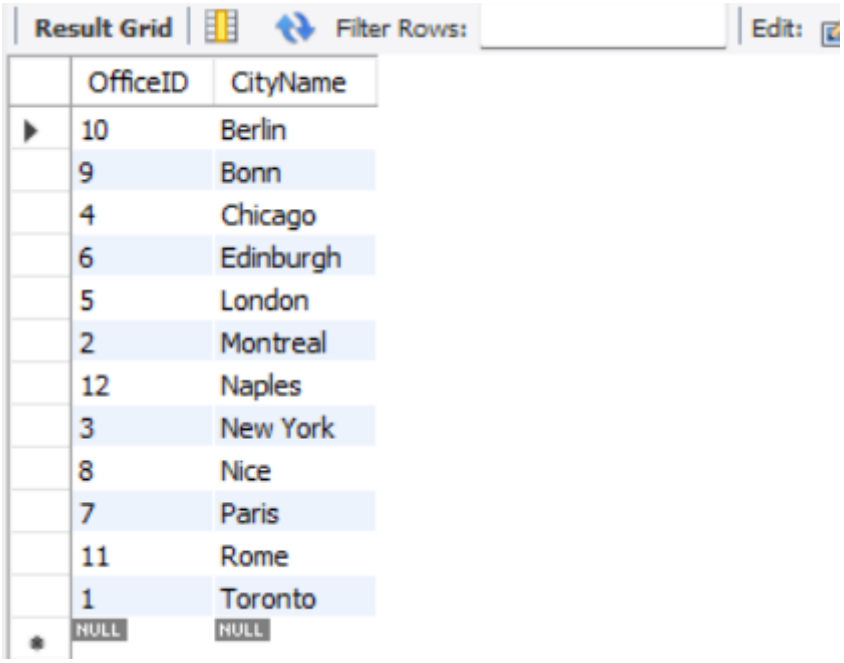
PhoneFaxID	CustomerID	CountryCode	AreaCode	LocalNumber
1	1	+1	416	5551234
2	2	+1	514	5555678
3	3	+1	212	5557890
4	4	+1	312	5553456
5	5	+44	20	5556789
6	6	+44	131	5559876
NULL	NULL	NULL	NULL	NULL

6. **BookingOffice Table**

Query:

```
INSERT INTO BookingOffice (OfficeID, CityName) VALUES
(1, 'Toronto'), (2, 'Montreal'), (3, 'New York'), (4, 'Chicago'), (5, 'London'),
(6, 'Edinburgh'), (7, 'Paris'), (8, 'Nice'), (9, 'Bonn'), (10, 'Berlin'), (11, 'Rome'),
(12, 'Naples');
select * from BookingOffice;
```

Output:



Result Grid | Filter Rows: | Edit:

OfficeID	CityName
10	Berlin
9	Bonn
4	Chicago
6	Edinburgh
5	London
2	Montreal
12	Naples
3	New York
8	Nice
7	Paris
11	Rome
1	Toronto
NULL	NULL

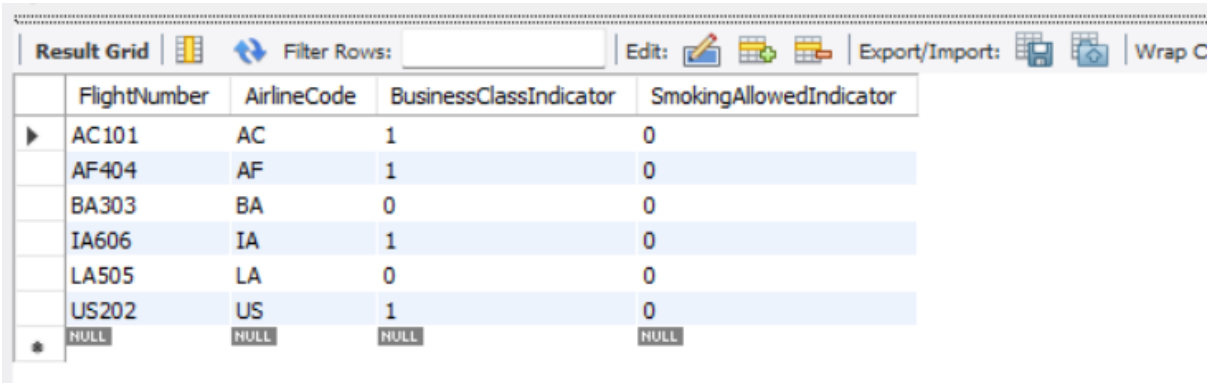
7. **Flight Table**

Query:

```
INSERT INTO Flight (FlightNumber, AirlineCode, BusinessClassIndicator,
SmokingAllowedIndicator)
VALUES
('AC101', 'AC', TRUE, FALSE),
('US202', 'US', TRUE, FALSE),
('BA303', 'BA', FALSE, FALSE),
('AF404', 'AF', TRUE, FALSE),
('LA505', 'LA', FALSE, FALSE),
('IA606', 'IA', TRUE, FALSE);
```

select * from Flight;

Output:



FlightNumber	AirlineCode	BusinessClassIndicator	SmokingAllowedIndicator
AC101	AC	1	0
AF404	AF	1	0
BA303	BA	0	0
IA606	IA	1	0
LA505	LA	0	0
US202	US	1	0
NULL	NULL	NULL	NULL

8. Booking Table

Query:

INSERT INTO Booking (BookingCity, BookingDate, FlightNumber, DepartureDateTime, ArrivalDateTime, ClassIndicator, TotalPrice, StatusIndicator, CustomerID, AmountPaid, OutstandingBalance, TicketFirstName, TicketLastName)

VALUES

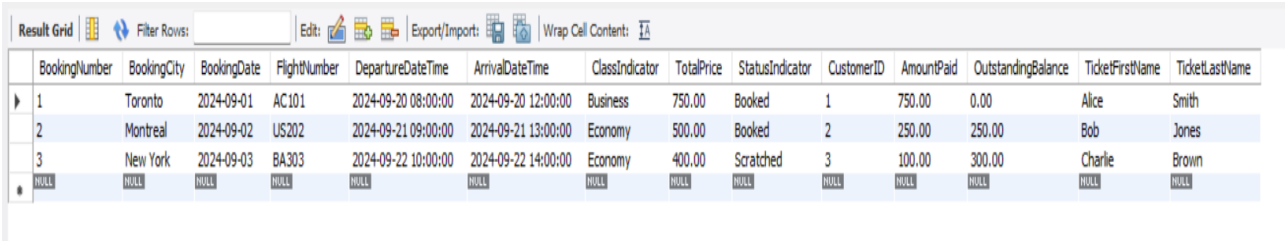
('Toronto', '2024-09-01', 'AC101', '2024-09-20 08:00:00', '2024-09-20 12:00:00', 'Business', 750.00, 'Booked', 1, 750.00, 0.00, 'Alice', 'Smith'),

('Montreal', '2024-09-02', 'US202', '2024-09-21 09:00:00', '2024-09-21 13:00:00', 'Economy', 500.00, 'Booked', 2, 250.00, 250.00, 'Bob', 'Jones'),

('New York', '2024-09-03', 'BA303', '2024-09-22 10:00:00', '2024-09-22 14:00:00', 'Economy', 400.00, 'Scratched', 3, 100.00, 300.00, 'Charlie', 'Brown');

select * from Booking;

Output:



BookingNumber	BookingCity	BookingDate	FlightNumber	DepartureDateTime	ArrivalDateTime	ClassIndicator	TotalPrice	StatusIndicator	CustomerID	AmountPaid	OutstandingBalance	TicketFirstName	TicketLastName
1	Toronto	2024-09-01	AC101	2024-09-20 08:00:00	2024-09-20 12:00:00	Business	750.00	Booked	1	750.00	0.00	Alice	Smith
2	Montreal	2024-09-02	US202	2024-09-21 09:00:00	2024-09-21 13:00:00	Economy	500.00	Booked	2	250.00	250.00	Bob	Jones
3	New York	2024-09-03	BA303	2024-09-22 10:00:00	2024-09-22 14:00:00	Economy	400.00	Scratched	3	100.00	300.00	Charlie	Brown
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

9. AirportTaxTable

Query:

INSERT INTO AirportTax (CityName, Country, TaxAmount)

VALUES

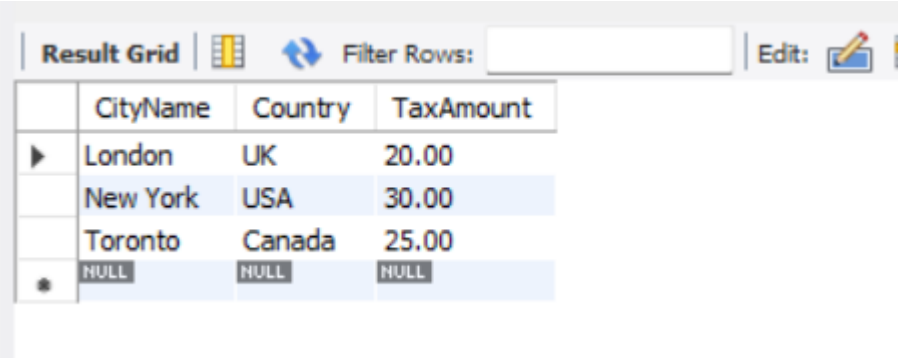
('Toronto', 'Canada', 25.00),

('New York', 'USA', 30.00),

('London', 'UK', 20.00);

select * from AirportTax;

Output:



CityName	Country	TaxAmount
London	UK	20.00
New York	USA	30.00
Toronto	Canada	25.00
NULL	NULL	NULL

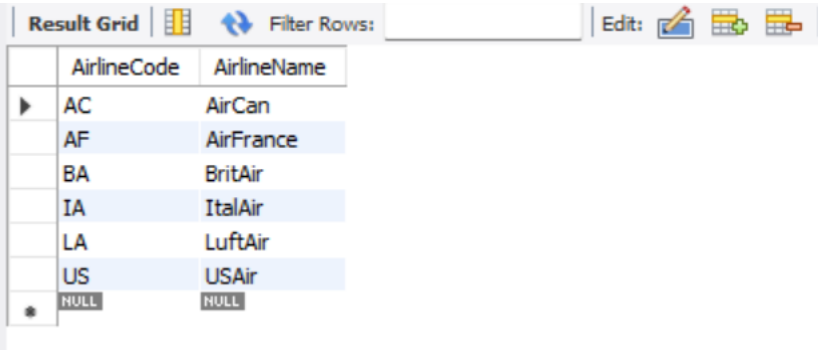
5. Query the Database:

1. **Display the airline code and airline name of all airlines in the system.**

Query:

SELECT AirlineCode, AirlineName FROM Airline;

Output:



AirlineCode	AirlineName
AC	AirCan
AF	AirFrance
BA	BritAir
IA	ItalAir
LA	LuftAir
US	USAir
NULL	NULL

2. **Find the names of all citie Query located in Canada.**

Query:

SELECT CityName

FROM City

WHERE Country = 'Canada';

Output:

Result Grid | Filter Rows: | E

CityName
Montreal
Toronto
NULL

3. Retrieve all booking office IDs and the city they are located in.

Query:

```
SELECT OfficeID, CityName
FROM BookingOffice;
```

Output:

Result Grid | Filter Rows: | Edit:

OfficeID	CityName
10	Berlin
9	Bonn
4	Chicago
6	Edinburgh
5	London
2	Montreal
12	Naples
3	New York
8	Nice
7	Paris
11	Rome
1	Toronto
NULL	NULL

4. Find the first name, last name, and email address of all customers.

Query:

```
SELECT FirstName, LastName, Email
FROM Customer;
```

Output:

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

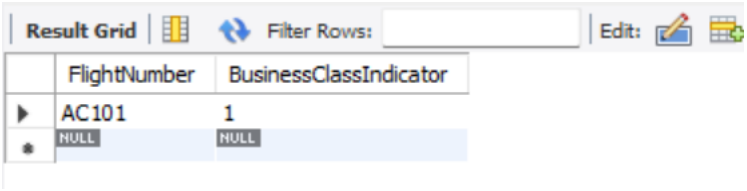
FirstName	LastName	Email
Alice	Smith	alice.smith@example.com
Bob	Jones	bob.jones@example.com
Charlie	Brown	charlie.brown@example.com
David	White	david.white@example.com
Eve	Black	eve.black@example.com
Frank	Green	frank.green@example.com

5. List all flights that are operated by 'AirCan'. Show the flight number and business class indicator.

Query:

```
SELECT FlightNumber, BusinessClassIndicator
FROM Flight
WHERE AirlineCode = 'AC';
```

Output:

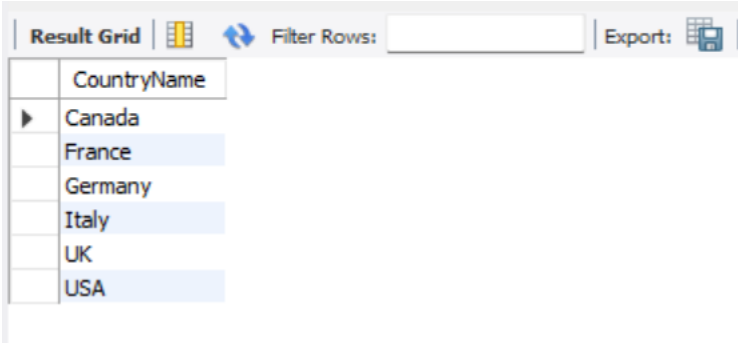


6. List all unique countries where airlines operate.

Query:

```
SELECT DISTINCT Country FROM Airline;
```

Output:

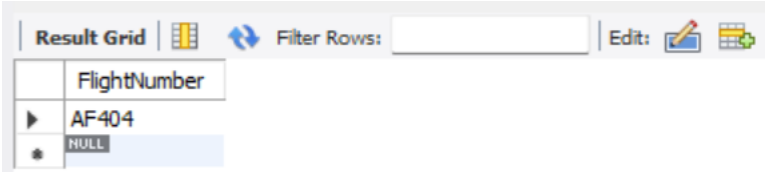


7. List the flight numbers for all flights operated by 'AirFrance'. Assume `AirlineCode` is referenced in `Flight`.

Query:

```
SELECT FlightNumber  FROM Flight
WHERE AirlineCode = (SELECT AirlineCode FROM Airline WHERE AirlineName = 'AirFrance');
```

Output:



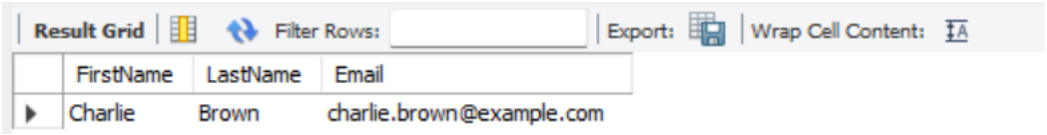
8. Retrieve the first name, last name, and email of all customers who live in 'New York'. Assume `MailingAddressID` in `Customer` references the `MailingAddress` table.

Query:

```
SELECT FirstName, LastName, Email  FROM Customer

WHERE MailingAddressID = (SELECT MailingAddressID FROM MailingAddress
WHERE CityName = 'New York');
```

Output:



FirstName	LastName	Email
Charlie	Brown	charlie.brown@example.com

9. Display the flight numbers where the number of booked seats in economy class i Query ess than the total available seats.

Query:

```
SELECT FlightNumber  FROM FlightAvailability

WHERE TotalSeatsEconomy > BookedSeatsEconomy;
```

Output:

FlightNumber
AC101
AF404
BA303
IA606
LA505
US202

10. List the first name and last name of all customers who have booked flight 'AC101'.

Query:

```
SELECT FirstName, LastName FROM Customer

WHERE CustomerID IN (SELECT CustomerID FROM Booking WHERE FlightNumber =
'AC101');
```

Output:

FirstName	LastName
Alice	Smith