**UBER Project**

Table Passengers

Table Drivers

*Name* VARCHAR (50)

**PassengerID** INT PRIMARY KEY

*Phone* VARCHAR (15)

*Email* VARCHAR (50)

**DriverID** INT Primary Key

*Name* VARCHAR (50)

*VehicleType* VARCHAR (20) VehiclePlate VARCHAR(15),

VehiclePlate VARCHAR(15),

Rating FLOAT

*VehiclePlate* VARCHAR (15)

*Rating* FLOAT

*TotalTrips* INT

**TripID INT** PRIMARY KEY

*StartLocation* VARCHAR (50)

*PassengerID* INT

*DriverID* INT

*Fare* DECIMAL (8, 2)

*EndLocation* VARCHAR (50)

Table

Trips

*FOREIGN KEY (DriverID) REFERENCES Drivers (DriverID)*

*FOREIGN KEY (PassengerID) REFERENCES Passengers (PassengerID)*

CREATE TABLE Drivers (

DriverID INT PRIMARY KEY,

Name VARCHAR(50),

VehicleType VARCHAR(20),

VehiclePlate VARCHAR(15),

Rating FLOAT

);

INSERT INTO `` (`DriverID`,`Name`,`VehicleType`,`VehiclePlate`,`Rating`) VALUES (1,'John Doe','Sedan','ABC123',4.8);

INSERT INTO `` (`DriverID`,`Name`,`VehicleType`,`VehiclePlate`,`Rating`) VALUES (2,'Jane Smith','SUV','XYZ456',4.9);

INSERT INTO `` (`DriverID`,`Name`,`VehicleType`,`VehiclePlate`,`Rating`) VALUES (3,'swarna vengal','Mini','LMO567',4.7);

INSERT INTO `` (`DriverID`,`Name`,`VehicleType`,`VehiclePlate`,`Rating`) VALUES (4,'Richard Williams','Tourister','FYI001',4.2);

INSERT INTO `` (`DriverID`,`Name`,`VehicleType`,`VehiclePlate`,`Rating`) VALUES (5,'Asha paran','SUV','PPO999',4.4);

INSERT INTO `` (`DriverID`,`Name`,`VehicleType`,`VehiclePlate`,`Rating`) VALUES (6,'Pushkar Dass','Primer','PFO666',4.5);

INSERT INTO `` (`DriverID`,`Name`,`VehicleType`,`VehiclePlate`,`Rating`) VALUES (7,'Dharun Krish','Compact','FAO654',4.6);

INSERT INTO `` (`DriverID`,`Name`,`VehicleType`,`VehiclePlate`,`Rating`) VALUES (8,'Rajesh Kumar','Mini','ILU143',4.6);

INSERT INTO `` (`DriverID`,`Name`,`VehicleType`,`VehiclePlate`,`Rating`) VALUES (9,'Pradeep Kumar','SUV','ILP012',4.8);

INSERT INTO `` (`DriverID`,`Name`,`VehicleType`,`VehiclePlate`,`Rating`) VALUES (10,'Bob Johnson','Compact','JKL789',4.7);

CREATE TABLE Passengers (

PassengerID INT PRIMARY KEY,

Name VARCHAR(50),

Email VARCHAR(50),

Phone VARCHAR(15),

TotalTrips INT

);

INSERT INTO `` (`PassengerID`,`Name`,`Email`,`Phone`,`TotalTrips`) VALUES (101,'Alice Johnson','alice@example.com','123-456-7890',15);

INSERT INTO `` (`PassengerID`,`Name`,`Email`,`Phone`,`TotalTrips`) VALUES (102,'Bob Smith','bob@example.com','987-654-3210',20);

INSERT INTO `` (`PassengerID`,`Name`,`Email`,`Phone`,`TotalTrips`) VALUES (110,'Charlie Brown','charlie@example.com','555-555-5555',10);

INSERT INTO `` (`PassengerID`,`Name`,`Email`,`Phone`,`TotalTrips`) VALUES (111,'David Williams','david@example.com','111-222-3333',12);

INSERT INTO `` (`PassengerID`,`Name`,`Email`,`Phone`,`TotalTrips`) VALUES (112,'Emma Davis','emma@example.com','444-555-6666',18);

INSERT INTO `` (`PassengerID`,`Name`,`Email`,`Phone`,`TotalTrips`) VALUES (113,'Frank Turner','frank@example.com','777-888-9999',25);

INSERT INTO `` (`PassengerID`,`Name`,`Email`,`Phone`,`TotalTrips`) VALUES (114,'Grace Wilson','grace@example.com','111-111-1111',8);

INSERT INTO `` (`PassengerID`,`Name`,`Email`,`Phone`,`TotalTrips`) VALUES (115,'Henry Miller','henry@example.com','999-888-7777',14);

INSERT INTO `` (`PassengerID`,`Name`,`Email`,`Phone`,`TotalTrips`) VALUES (116,'Ivy Parker','ivy@example.com','333-444-5555',21);

CREATE TABLE Trips (

TripID INT PRIMARY KEY,

DriverID INT,

PassengerID INT,

StartLocation VARCHAR(50),

EndLocation VARCHAR(50),

Fare DECIMAL(8, 2),

FOREIGN KEY (DriverID) REFERENCES Drivers(DriverID),

FOREIGN KEY (PassengerID) REFERENCES Passengers(PassengerID)

);

INSERT INTO `` (`TripID`,`DriverID`,`PassengerID`,`StartLocation`,`EndLocation`,`Fare`) VALUES (1001,1,101,'123 Main St','456 Oak St',25.00);

INSERT INTO `` (`TripID`,`DriverID`,`PassengerID`,`StartLocation`,`EndLocation`,`Fare`) VALUES (1002,2,102,'789 Pine St','321 Elm St',30.50);

INSERT INTO `` (`TripID`,`DriverID`,`PassengerID`,`StartLocation`,`EndLocation`,`Fare`) VALUES (1013,3,110,'555 Birch St','666 Pine St',20.00);

INSERT INTO `` (`TripID`,`DriverID`,`PassengerID`,`StartLocation`,`EndLocation`,`Fare`) VALUES (1014,1,111,'777 Elm St','888 Pine St',15.25);

INSERT INTO `` (`TripID`,`DriverID`,`PassengerID`,`StartLocation`,`EndLocation`,`Fare`) VALUES (1015,2,112,'999 Maple St','000 Oak St',28.90);

INSERT INTO `` (`TripID`,`DriverID`,`PassengerID`,`StartLocation`,`EndLocation`,`Fare`) VALUES (1016,3,113,'111 Birch St','222 Elm St',19.00);

INSERT INTO `` (`TripID`,`DriverID`,`PassengerID`,`StartLocation`,`EndLocation`,`Fare`) VALUES (1017,1,114,'333 Pine St','444 Oak St',24.50);

INSERT INTO `` (`TripID`,`DriverID`,`PassengerID`,`StartLocation`,`EndLocation`,`Fare`) VALUES (1018,2,115,'555 Elm St','666 Maple St',32.25);

DDL

1. \*\*CREATE:\*\*

- The CREATE command in SQL is a powerful statement used to initialize and establish various elements within a database. It enables the creation of fundamental database components such as tables, indexes, functions, views, stored procedures, and triggers. By utilizing the CREATE command, a database administrator can define the structure and characteristics of these objects, paving the way for efficient data management.

2. \*\*DROP:\*\*

- The DROP command is employed in SQL to perform the deletion of objects from the database. This command allows the removal of tables, indexes, functions, views, stored procedures, and triggers. It is a critical tool for database maintenance, enabling the elimination of unnecessary or obsolete elements, thus optimizing the database's overall structure and performance.

3. \*\*ALTER:\*\*

- SQL's ALTER command provides a flexible means to modify and enhance the structure of a database. With the ALTER command, database administrators can make adjustments to existing tables, columns, or other database objects. This includes altering data types, adding or dropping columns, and implementing changes that align with evolving requirements without the need to recreate the entire database.

4. \*\*TRUNCATE:\*\*

- The TRUNCATE command in SQL is specifically designed to swiftly remove all records from a table, effectively resetting it to an empty state. Unlike the DELETE command, which removes rows one by one, TRUNCATE is a more efficient option for clearing large amounts of data. It not only removes the records but also releases the allocated storage space, contributing to improved database performance.

5. \*\*COMMENT:\*\*

- The COMMENT command in SQL facilitates the addition of descriptive comments to the data dictionary, enhancing the documentation and understanding of database objects. This command allows developers and administrators to include annotations that provide insights into the purpose, usage, or other relevant information about tables, columns, or other database components.

DDL Continue..

6. \*\*RENAME:\*\*

- The RENAME command in SQL is utilized to modify the name of an existing database object. This can include renaming tables, indexes, functions, views, stored procedures, or triggers. The RENAME command is valuable for maintaining a coherent and organized database structure, especially when adjustments to naming conventions or object identification become necessary over time.

DML

\*\*SELECT:\*\*

The SELECT command in SQL is a fundamental and versatile statement that serves as the primary mechanism for retrieving data from a database. It is a cornerstone of the querying process, allowing users to specify the columns they want to retrieve, the tables they want to query, and various conditions to filter and sort the results. The SELECT statement facilitates a wide range of operations, enabling users to aggregate data, perform calculations, and join multiple tables to create comprehensive result sets.

The syntax of the SELECT command encompasses several clauses, including:

- \*\*SELECT Clause:\*\* Specifies the columns to be retrieved from the database tables.

- \*\*FROM Clause:\*\* Specifies the tables from which the data should be retrieved.

- \*\*WHERE Clause:\*\* Enables the imposition of conditions to filter the rows based on specific criteria.

- \*\*GROUP BY Clause:\*\* Facilitates the grouping of results based on one or more columns, often used in conjunction with aggregate functions like COUNT, SUM, AVG, etc.

- \*\*HAVING Clause:\*\* Acts as a filter for grouped data, allowing conditions to be applied after the GROUP BY operation.

- \*\*ORDER BY Clause:\*\* Defines the order in which the result set should be presented, either in ascending or descending order based on specified columns.

- \*\*JOIN Clause:\*\* Enables the combination of rows from two or more tables based on a related column between them.

The SELECT command is not limited to simple data retrieval; it empowers users to perform complex queries, analyze data, and derive valuable insights from the database. Its flexibility and robust functionality make it an indispensable tool for interacting with and extracting meaningful information from relational databases.

DML

1. \*\*INSERT:\*\*

- The INSERT command in SQL is employed to add new records or data into a table within a database. It plays a pivotal role in populating tables with information, allowing for the organized and systematic storage of data. By utilizing the INSERT command, database users can specify the values for each column, ensuring accurate and meaningful data entry.

2. \*\*UPDATE:\*\*

- The UPDATE command in SQL serves as a powerful tool for modifying existing data within a table. It enables users to make targeted changes to specific records based on specified criteria, ensuring the accuracy and currency of information. The UPDATE command is essential for maintaining data integrity and accommodating changes in business requirements over time.

3. \*\*DELETE:\*\*

- The DELETE command in SQL is employed to remove records from a table, offering a means to selectively eliminate data based on specified conditions. Unlike the TRUNCATE command, which removes all records from a table, DELETE provides a more granular approach, allowing users to delete specific rows. Proper use of the DELETE command is crucial for managing data retention and ensuring the relevance of stored information.

4. \*\*LOCK:\*\*

- The LOCK command in SQL is not a standalone command but is a concept related to managing concurrency and ensuring data consistency within a database. SQL databases use various types of locks, such as shared locks and exclusive locks, to control access to data during transactions. Locks prevent conflicts that may arise when multiple users or transactions attempt to read or modify the same data simultaneously. Proper use of locking mechanisms is essential for maintaining the integrity of the database and preventing data inconsistencies in multi-user environments. The specific syntax and behavior of locking depend on the database management system being used.

DCL

1. \*\*GRANT:\*\*

- The GRANT command in SQL is a powerful statement used to confer specific access privileges to users or roles within a database. It plays a crucial role in managing security and access control, allowing administrators to define precisely what actions (such as SELECT, INSERT, UPDATE, DELETE) a user or role is permitted to perform on particular database objects (such as tables, views, or procedures). The GRANT command enhances data security by providing a fine-grained approach to authorization, ensuring that only authorized individuals can interact with and manipulate the data as defined by the database administrator.

2. \*\*REVOKE:\*\*

- The REVOKE command in SQL acts as a counterpart to the GRANT command, enabling administrators to withdraw or revoke previously granted access privileges from users or roles. This is essential for maintaining a dynamic and controlled security environment. By using the REVOKE command, administrators can promptly restrict or modify user access permissions, responding to changes in user roles or security policies. It enhances the manageability of database security by allowing administrators to adapt access privileges based on evolving organizational needs, ensuring that access remains aligned with authorized requirements and data protection standards.

TCL

1. \*\*COMMIT:\*\*

- The COMMIT command is used to permanently save the changes made during the current transaction to the database. Once a COMMIT is issued, the changes become permanent and cannot be rolled back. COMMIT is essential for ensuring data consistency and integrity, as it finalizes a successful transaction.

2. \*\*ROLLBACK:\*\*

- The ROLLBACK command is used to undo the changes made during the current transaction. It is typically employed when an error occurs, and the changes need to be reverted to maintain the integrity of the database. ROLLBACK ensures that the database remains in a consistent state even if a part of the transaction fails.

TCL Continue..

3. \*\*SAVEPOINT:\*\*

- SAVEPOINT is used to set a point within a transaction to which you can later roll back. It provides a way to create intermediate points within a transaction, allowing for partial rollbacks in case of errors without affecting the entire transaction.

4. \*\*ROLLBACK TO SAVEPOINT:\*\*

- ROLLBACK TO SAVEPOINT is used to undo the changes made since a specific savepoint. It allows for a more granular rollback within a transaction, reverting to a designated point without affecting the entire transaction.

5. \*\*SET TRANSACTION:\*\*

- SET TRANSACTION allows the user to set properties for the current transaction, such as isolation level and other transaction characteristics. This command is used to configure the behavior of subsequent transactions.