

Ola Project from Top Varsity

- ⇒ Cleaning and Filtering (Better on Google SpreadSheet)
- ⇒ Exporting data as CSV for Analysing on SQL
- ⇒ Adding data on SQL (Create DB (Ola) ⇒ Table (Right Click) ⇒ Table data import Wizard)

Qns(1) ⇒ Retrieve all Successful Bookings

```
Create View Successful_Bookings As  
SELECT*FROM bookings  
WHERE Booking_Status = 'Success';  
Select *From Successful_Bookings;
```

The screenshot shows the MySQL Workbench interface. In the top navigation bar, the database 'Ola Project Top Varsity' is selected. The 'Query Grid' tab is active, displaying the following SQL code:

```
1 • CREATE DATABASE ola;  
2 • Use ola;  
3 • #Retrieve all Successful Bookings;  
4 • SELECT*FROM bookings  
5 • WHERE Booking_Status = 'Success';
```

Below the code, the 'Result Grid' shows the results of the query. The table has the following columns: Date, Time, Booking_ID, Booking_Status, Customer_ID, Vehicle_Type, Pickup_Location, Drop_Location, V_TAT, C_TAT, Canceled_Rides_by_Customer, Canceled_Rides_by_Driver, Incomplete_Rides, and Incomplete_TAT. The data consists of 10 rows of booking information.

In the bottom left, the 'Information' pane shows the definition of the 'bookings' table, which includes columns for Date, Time, Booking_ID, Booking_Status, Customer_ID, Vehicle_Type, Pickup_Location, Drop_Location, V_TAT, C_TAT, and Canceled_Rides.

What is a View in SQL?

A **view** in SQL is a virtual table that is based on the result of a SQL query. Unlike a regular table, a view does not store data itself; instead, it displays the data stored in other tables.

Views are created using the `CREATE VIEW` statement and can be queried just like a regular table.

Role of Views in Data Analysis

Views play a significant role in data analysis by:

- 1. Simplifying Complex Queries:** They can encapsulate complex SQL queries, making them reusable and easier to understand.
- 2. Data Security:** Views can restrict access to specific columns or rows, exposing only the required data to users.
- 3. Data Abstraction:** Views provide a layer of abstraction, so users don't need to know the underlying table structures.
- 4. Aggregation and Summarization:** Views can summarize data (e.g., using `GROUP BY`) for analytical purposes.
- 5. Enhancing Performance:** Although views themselves don't enhance performance, materialized views (a special type) can store query results for faster access.

#Qns(2) ⇒ Find the Average ride distance for Each Vehicle Type:

```
Create View ride_distance_for_Each_Vehicle_type As
Select Vehicle_Type, Avg(Ride_Distance)
As avg_distance From bookings
Group By Vehicle_Type;
Select * From ride_distance_for_Each_Vehicle_type;
```

The screenshot shows the MySQL Workbench interface. In the top-left pane, the Navigator displays the schema 'ola' with tables like 'bookings', 'Customer', 'Vehicle', etc. The main area, 'Query 1', contains the SQL code for creating a view. The 'Result Grid' shows the average ride distance for each vehicle type:

Vehicle_Type	avg_distance
Prime Sedan	15.7205
Bike	15.7145
Prime SUV	15.2064
eBike	15.6532
Mini	15.5903
Prime Plus	15.3797
Auto	6.2151

The bottom pane, 'ride_distance_for_Each_Vehicle...', shows the history of actions taken to create the view, including the creation of the view itself and the subsequent SELECT statement.

Qns(3) ⇒ Get the total number of Canceled Rides by Customers:

```
Select Count(*) From bookings
Where Booking_Status = 'Canceled by Customer';
```

Qns(4) ⇒ List the top 5 customers who booked the highest number of rides

#top 5 customers who booked the highest number of rides

```
Create View Top_5_Customer As
Select Customer_ID, Count(Booking_ID)
as total_rides
From bookings
Group by Customer_ID
Order by total_rides DESC LIMIT 5;
select * from Top_5_Customer;
```

The screenshot shows the MySQL Workbench interface with the following details:

- Schemas:** The schema tree shows the `ola` database selected, containing tables like `bookings`, `views`, and `stored procedures`.
- Query Editor:** The query `Create View Top_5_Customer As` is being typed into the editor. The code is as follows:


```
18 • Create View Top_5_Customer As
19   Select Customer_ID, Count(Booking_ID)
20     as total_rides
21   From bookings
22   Group by Customer_ID
23   Order by total_rides DESC LIMIT 5;
24 • select * from Top_5_Customer;
```
- Result Grid:** The result grid displays the top 5 customers with their total rides:

Customer_ID	total_rides
CID340854	4
CID989500	3
CID558349	3
CID275322	3
CID333767	3
- Output Window:** The output window shows the history of actions:

Action Output	Time	Action	Message	Duration / Fetch
1	08:17:23	Use ola	0 row(s) affected	0.00 sec
2	08:45:58	Create View Top_5_Customer As Select Customer_ID, Count(Booking_ID) as total_rides From bookings Group b...	0 row(s) affected	0.109 sec
3	08:46:18	select * from Top_5_Customer LIMIT 0, 1000	5 row(s) returned	0.516 sec / 0.000 sec
4	08:46:33	select * from Top_5_Customer LIMIT 0, 1000	5 row(s) returned	0.703 sec / 0.000 sec

- **Create View Top_5_Customer As :**
 - This creates a **view** named `Top_5_Customer`. A view is a virtual table based on the result set of a SQL query. It does not store data physically but provides a way to save and reuse queries.
- **Select Customer_ID, Count(Booking_ID) as total_rides :**
 - Customer_ID** : Selects the customer identifier from the `bookings` table.
 - Count(Booking_ID) as total_rides** : Counts the number of bookings (`Booking_ID`) for each customer and labels this count as `total_rides`.
- **From bookings :**

- Specifies the source table, which is `bookings`.
- `Group by Customer_ID` :
- Groups the data by `Customer_ID`, ensuring that the count of bookings (`Booking_ID`) is calculated per customer.
- `Order by total_rides DESC` :
- Orders the grouped results in descending order based on the total number of rides (`total_rides`), so customers with the highest number of rides appear first.
- `LIMIT 5` :
- Restricts the output to only the top 5 rows, effectively returning the top 5 customers with the most rides.

Qns(5)⇒ Get the number of rides canceled by drivers due to personal and car-related issues:

```
Create View Canceled_by_drivers As
SELECT Count(*) From bookings
Where Canceled_Rides_by_driver = 'Personal & Car related issue';
Select * From Canceled_by_drivers;
```

Qns(6)⇒ Find the Maximum & Minimum driver ratings for Prime Sedan bookings:

```
Create View Driver_rating_PS as
Select Max(Driver_Ratings) as Max_ratings,
Min(Driver_Ratings) as Min_ratings
From bookings Where Vehicle_Type = "Prime Sedan";
Select * From Driver_rating_PS;
```

The screenshot shows the MySQL Workbench interface. In the top-left corner, there's a 'MySQL Workbench' window title bar. Below it, the 'File Edit View Query Database Server Tools Scripting Help' menu is visible. On the left side, there's a 'Navigator' pane with a 'SCHEMAS' section containing various tables like Time, Booking_ID, Booking_Status, Customer_ID, Vehicle_Type, Pickup_Location, Drop_Location, V_TAT, C_TAT, Cancelled_Rides_by_Cust, Cancelled_Rides_by_Drv, Incomplete_Rides, Incomplete_Rides_Reason, Booking_Value, Payment_Method, Ride_Distance, Driver_Ratings, Customer_Rating, MyUnknownColumn, Indexes, and EnvironVar. Below the Navigator is an 'Administration' section with tabs for 'Schemas' and 'Information'. The main area is titled 'Query 1' and contains the following SQL code:

```

29 • Select * From Canceled_by_drivers;
30 #6 Find the Maximum & Minimum driver ratings for Prime Sedan bookings:
31 • Create View Driver_rating_PS as
32 Select Max(Driver_Ratings) as Max_ratings,
33 Min(Driver_Ratings) as Min_ratings,
34 From bookings Where Vehicle_Type = "Prime Sedan";
35 • Select * From Driver_rating_PS;

```

Below the code is a 'Result Grid' table with two columns: 'Max_ratings' and 'Min_ratings'. The values shown are 5 and 3 respectively. To the right of the result grid are buttons for 'Filter Rows', 'Export', and 'Wrap Cell Content'. At the bottom of the result grid, there are buttons for 'Max_ratings' and 'Min_ratings'.

On the right side of the interface, there's a 'Driver_rating_PS 6' tab under the 'Information' section. It has a 'Read Only' status indicator. Below it is a 'Action Output' table with several rows of log entries. The columns in the log table are '#', 'Time', 'Action', 'Message', and 'Duration / Fetch'. The log entries are:

#	Time	Action	Message	Duration / Fetch
8	09:05:07	Select * From Canceled_by_drivers LIMIT 0, 1000	1 row(s) returned	0.172 sec / 0.000 sec
9	09:13:10	Create View Driver_rating_PS as Select Max(Driver_Ratings) as Max_ratings, Min(Driver_Ratings) as Min_ratings	Error Code: 1054: Unknown column 'Driver_Ratings' in field list'	0.000 sec
10	09:14:29	Create View Driver_rating_PS as Select Max(Driver_Ratings) as Max_ratings, Min(Driver_Ratings) as Min_ratings	Error Code: 1054: Unknown column 'Driver_Ratings' in field list'	0.000 sec
11	09:15:57	Create View Driver_rating_PS as Select Max(Driver_Ratings) as Max_ratings, Min(Driver_Ratings) as Min_ratings	0 row(s) affected	0.016 sec
12	09:16:01	Select * From Driver_rating_PS LIMIT 0, 1000	1 row(s) returned	0.328 sec / 0.000 sec
13	09:18:54	Select * From Driver_rating_PS LIMIT 0, 1000	1 row(s) returned	0.219 sec / 0.000 sec

Qns(7)⇒ Retrieve all rides where payment was made using UPI:

```

SELECT * FROM bookings
WHERE Payment Method = 'UPI';

```

#Qns(8)⇒ Find the average customer rating per vehicle type:

```

Create View Vehicle_ratings_customer As
SELECT Vehicle_Type, AVG (Customer_Rating) as avg_customer_rating
FROM bookings
GROUP BY Vehicle_Type
Order BY avg_customer_rating ASC;
Select *from Vehicle_ratings_customer;

```

MySQL Workbench - Ola Project Top Versity

Query 1:

```

39 • #8 Find the average customer rating per vehicle type:
40 • Create View Vehicle_ratings_customer As
41   SELECT Vehicle_Type, AVG (Customer_Rating) as avg_customer_rating
42   FROM bookings
43   GROUP BY Vehicle_Type
44   Order By avg_customer_rating ASC;
45 • Select *from Vehicle_ratings_customer;

```

Result Grid:

Vehicle_Type	avg_customer_rating
eBike	3.981074168798
Bike	3.9923679999999977
Mini	3.99651482081499
Prime SUV	3.99860385040274
Auto	3.998531571218683
Prime Sedan	4.00284860189574
Prime Plus	4.008297802558229

Vehicle_ratings_customer 11:

Action Output:

#	Time	Action	Message	Duration / Fetch
15	09:30:22	Action	1000 row(s) returned	0.000 sec / 0.015 sec
16	10:02:26	SELECT *from bookings Where Payment_Method='UPI' LIMIT 0, 1000	7 row(s) returned	0.484 sec / 0.000 sec
17	10:04:26	SELECT Vehicle_Type, AVG (Customer_Rating) as avg_customer_rating FROM bookings GROUP BY Vehicle_Type	Error Code: 1146. Table 'ola.vehicle_ratings_customer' doesn't exist	0.032 sec
18	10:04:41	Create View Vehicle_ratings_customer As SELECT Vehicle_Type, AVG (Customer_Rating) as avg_customer_rating FROM bookings GROUP BY Vehicle_Type	0 row(s) affected	0.016 sec
19	10:04:45	Select *from Vehicle_ratings_customer LIMIT 0, 1000	7 row(s) returned	0.328 sec / 0.000 sec
20	10:08:37	Select *from Vehicle_ratings_customer LIMIT 0, 1000	7 row(s) returned	0.578 sec / 0.000 sec

#Qns(9)⇒ Calculate the total booking value of rides completed successfully

```

Create View Successful_rides As
Select SUM(Booking_Value) as total_successful_value
From bookings
Where Booking_Status = 'Success';
Select*From Successful_rides;

```

MySQL Workbench - Ola Project Top Versity

Query 1:

```

45 • Select *from Vehicle_ratings_customer;
46 • #9 Calculate the total booking value of rides completed successfully
47 • Create View Successful_rides As
48   Select SUM(Booking_Value) as total_successful_value
49   From bookings
50   Where Booking_Status = 'Success';
51 • Select*From Successful_rides;

```

Result Grid:

total_successful_value
23642787

Successful_rides 13:

Action Output:

#	Time	Action	Message	Duration / Fetch
18	10:04:41	Create View Vehicle_ratings_customer As SELECT Vehicle_Type, AVG (Customer_Rating) as avg_customer_rating FROM bookings GROUP BY Vehicle_Type	0 row(s) affected	0.016 sec
19	10:04:45	Select *from Vehicle_ratings_customer LIMIT 0, 1000	7 row(s) returned	0.328 sec / 0.000 sec
20	10:08:37	Select *from Vehicle_ratings_customer LIMIT 0, 1000	7 row(s) returned	0.578 sec / 0.000 sec
21	10:18:28	Select SUM(Booking_Value) as total_successful_value From bookings Where Booking_Status = 'Success' Li...	1 row(s) returned	0.235 sec / 0.000 sec
22	10:19:05	Create View Successful_rides As Select SUM(Booking_Value) as total_successful_value From bookings Wher...	0 row(s) affected	0.031 sec
23	10:19:23	Select*From Successful_rides LIMIT 0, 1000	1 row(s) returned	0.172 sec / 0.000 sec

#Qns(10)⇒ List all incomplete rides along with the reason:

```

CREATE VIEW incomplete_rides_list As
SELECT Booking_ID, Incomplete_Rides_Reason
FROM bookings
Where Incomplete_Rides = 'Yes';
SELECT*FROM incomplete_rides_list;

```

The screenshot shows the MySQL Workbench interface with the following details:

- Query Editor:** Contains the SQL code for creating the view and selecting from it.
- Result Grid:** Displays the results of the SELECT statement, showing multiple rows of booking IDs and their incomplete ride reasons.
- Timeline:** Shows the execution history with statements numbered 21 through 26, detailing the actions taken and their durations.

Booking_ID	Incomplete_Rides_Reason
CNR5176704322	Customer Demand
CNR912632867	Vehicle Breakdown
CNR7924302885	Customer Demand
CNR1640228587	Other Issue
CNR7623690602	Other Issue
CNR9590311980	Customer Demand
CNR5963244684	Customer Demand
CNR9526078867	Customer Demand
CNR7154043084	Customer Demand
CNR3193710797	Other Issue
CNR2073830950	Customer Demand
CNR9526078867	Customer Demand