SQL Queries:-

1. **select * from credentials where user_id = “166” and password = “kinjOoeJkn”;**

the "credentials" table stores login information for users of a system or application, and this query is searching for the row in the table that matches the user ID and password provided. If a row is found, it is likely that the user with ID "166" and password "kinjOoeJkn" is authorized to access the system or application.

2. **SELECT d.Driver_id FROM driver d INNER JOIN vehicle c ON d.Vehicle_id = c.vehicle_id WHERE c.type = 'SEDAN';**

This is a SQL query that is selecting the "Driver_id" column from the "driver" table, for all rows where the corresponding vehicle in the "vehicle" table has a "type" column value of "SEDAN".
The query achieves this by using an inner join between the "driver" and "vehicle" tables, where the join condition is that the "Vehicle_id" column in the "driver" table must match the "vehicle_id" column in the "vehicle" table.

3. **select count(Request_ID) from request where status = 1;**

4. **select count(Driver_id) from driver where Sex='M';**

The "count" function is being used to return the number of "Driver_id" values in the "driver" table that satisfy this criteria. The "Driver_id" column likely contains a unique identifier for each driver in the system.
Therefore, the query is essentially counting the number of male drivers in the system, where "M" likely represents "Male" or "Man".
This query can be useful for obtaining insights into the gender composition of a driver pool or workforce

5. **select r.Name, count(b.Rider_ID) as total_rides from billing b join rider r on b.Rider_ID = r.Rider_Id where b.Date_Time between '2020-01-01 00:00:00' and '2020-03-01 00:00:00' group by r.Name order by total_rides desc limit 10;**

The query achieves this by joining the "billing" table with the "rider" table on the basis of the "Rider_ID" column in the former and the "Rider_Id" column in the latter. The join ensures that each ride in the "billing" table is matched with the corresponding rider in the "rider" table.
The query then uses the "count" function to count the number of rows in the "billing" table that correspond to each rider in the "rider" table within the specified time range. This count is aliased as "total_rides".
The "group by" clause is used to group the rows by the "Name" column in the "rider" table. This ensures that each rider appears only once in the output, with the total number of rides they took during the specified time range.

The "order by" clause is used to sort the output by the "total_rides" column in descending order. This means that the rider with the most rides appears first, and so on. Finally, the "limit" clause is used to restrict the output to the top 10 riders based on the "total_rides" column.
Overall, this query can be useful for identifying the most active riders within a specific time period, which could be useful for making business decisions related to rider incentives, marketing, or service offerings.

6. **select * from wallet where amount>1500;**

7. **select * from driver where status=1 order by Rate;**

8. **select * from driver where status=1 order by Current_Rating desc;**

9. **update driver set Status=0 where Driver_id = 299;**

10. **select AVG(b.Fare) as avg_revenue from billing b Join driver d on b.Driver_id = d.Driver_Id join wallet w on d.Wallet_id = w.wallet_id where b.Date_Time between '2020-01-01 00:00:00' and '2022-01-01 00:00:00' and d.Driver_id = '206';**

This is a SQL query that calculates the average revenue generated by a driver with the "Driver_Id" of '206' between January 1, 2020 and January 1, 2022.
The query achieves this by joining the "billing" table with the "driver" table on the basis of the "Driver_id" column in both tables. It then joins the resulting table with the "wallet" table on the basis of the "Wallet_id" column in the "driver" table and the "wallet_id" column in the "wallet" table. This ensures that the query has access to the fares paid by riders for each ride, as well as the driver's wallet balance.
The query then applies two filters to the data: first, it selects only those rows where the "Date_Time" column in the "billing" table falls between January 1, 2020 and January 1, 2022. Second, it selects only those rows where the "Driver_id" column in the "driver" table has a value of '206', which corresponds to the driver of interest.
Finally, the "AVG" function is used to calculate the average of the "Fare" column in the resulting table. This gives the average revenue generated by the driver of interest during the specified time range.
This query can be useful for analyzing the financial performance of individual drivers, identifying top-performing drivers, and comparing driver performance over time.

11. **SELECT d.Driver_id FROM driver d INNER JOIN vehicle c ON d.Vehicle_id = c.vehicle_id WHERE c.type = 'SEDAN';**

This is a SQL query that selects the "Driver_id" values from the "driver" table for all drivers who are associated with a "SEDAN" vehicle in the "vehicle" table.
The query uses an "INNER JOIN" to combine the "driver" and "vehicle" tables based on the "Vehicle_id" column in the "driver" table and the "vehicle_id" column in the "vehicle"

table. This ensures that only drivers who have a corresponding record in the "vehicle" table, and whose vehicle type is "SEDAN", will be selected.
The "c.type = 'SEDAN'" condition is used to filter the results of the join, selecting only those records where the vehicle type is "SEDAN".
Finally, the query selects the "Driver_id" column from the resulting table, which gives the "Driver_id" values of all drivers associated with a "SEDAN" vehicle.
This query can be useful for identifying drivers who are authorized to drive a specific type of vehicle, such as for compliance or safety purposes, or for generating reports on the types of vehicles used by the driver pool.

12. **select distinct r.Preffered_Vehicle from billing b Inner join request r on b.Request_ID = r.Request_ID where b.Rider_ID = '101';**

This is a SQL query that selects the distinct "Preffered_Vehicle" values from the "request" table for all rides where the rider has an ID of '101'.
The query achieves this by joining the "billing" table with the "request" table based on the "Request_ID" column in both tables. It then applies a filter to the resulting table, selecting only those records where the "Rider_ID" column in the "billing" table has a value of '101', which corresponds to the rider of interest.
Finally, the "DISTINCT" keyword is used to remove any duplicates from the resulting "Preffered_Vehicle" values.
The "Preffered_Vehicle" column is likely used to store information about the type or model of vehicle that the rider has requested, which can help the ride-hailing service to match riders with appropriate drivers and vehicles.
This query can be useful for generating reports on rider preferences or demand for certain types of vehicles, or for analyzing the effectiveness of the ride-hailing service's matching algorithms.

13. **select * from billing where Date_Time between '2020-09-12 00:00:00' and '2020-09-25 00:00:00';**

This is a SQL query that selects all columns and rows from the "billing" table where the "Date_Time" column falls within a specified time range.
The query achieves this by using the "BETWEEN" keyword to filter the results based on the "Date_Time" column. Specifically, it selects only those rows where the "Date_Time" value falls between '2020-09-12 00:00:00' and '2020-09-25 00:00:00'. This is a common way to extract records for a specific period of time.
The "billing" table likely contains information about the billing or payments for the ride-hailing service, including details such as the fare amount, the ride duration, and the payment method used.
This query can be useful for generating reports on billing and payment activities during a specific time period, or for identifying any issues or discrepancies with the billing records for that period.

xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx