1. How to select UNIQUE records from a table using a SQL Query?

Consider below EMPLOYEE table as the source data

```
CREATE TABLE EMPLOYEE (
        EMPLOYEE_ID NUMBER(6,0),
        NAME VARCHAR2(20),
        SALARY NUMBER(8,2)
);

INSERT INTO EMPLOYEE(EMPLOYEE_ID,NAME,SALARY) VALUES(100,'Jennifer',4400);
INSERT INTO EMPLOYEE(EMPLOYEE_ID,NAME,SALARY) VALUES(100,'Jennifer',4400);
INSERT INTO EMPLOYEE(EMPLOYEE_ID,NAME,SALARY) VALUES(101,'Michael',13000);
INSERT INTO EMPLOYEE(EMPLOYEE_ID,NAME,SALARY) VALUES(101,'Michael',13000);
INSERT INTO EMPLOYEE(EMPLOYEE_ID,NAME,SALARY) VALUES(101,'Michael',13000);
INSERT INTO EMPLOYEE(EMPLOYEE_ID,NAME,SALARY) VALUES(102,'Pat',6000);
INSERT INTO EMPLOYEE(EMPLOYEE_ID,NAME,SALARY) VALUES(102,'Pat',6000);
INSERT INTO EMPLOYEE(EMPLOYEE_ID,NAME,SALARY) VALUES(103,'Den',11000);


SELECT * FROM EMPLOYEE;
```

EMPLOYEE_ID	NAME	SALARY
100	Jennifer	4400
100	Jennifer	4400
101	Michael	13000
101	Michael	13000
101	Michael	13000
102	Pat	6000
102	Pat	6000
103	Den	11000

Using GROUP BY Function

Query:

```
SELECT EMPLOYEE_ID,
       NAME,
       SALARY
FROM EMPLOYEE
GROUP BY EMPLOYEE_ID, NAME, SALARY;
```

Result:

EMPLOYEE_ID	NAME	SALARY
100	Jennifer	4400
101	Michael	13000
102	Pat	6000
103	Den	11000

2. How to delete DUPLICATE records from a table using a SQL Query?

Consider the same EMPLOYEE table as source discussed in previous question

Using ROWID and ROW_NUMBER Analytic Function

STEP-1: Using ROW_NUMBER Analytic function, assign row numbers to each unique set of records. Select ROWID of the rows along with the source columns

Query:

```
SELECT ROWID,
       EMPLOYEE_ID,
       NAME,SALARY,
       ROW_NUMBER() OVER(PARTITION BY EMPLOYEE_ID,NAME,SALARY ORDER BY EMPLOYEE_ID) AS ROW_NUMBER
FROM EMPLOYEE;
```

Result:

ROWID	EMPLOYEE_ID	NAME	SALARY	ROW_NUMBER
AAASnBAAEAAACrWAAA	100	Jennifer	4400	1
AAASnBAAEAAACrWAAB	100	Jennifer	4400	2
AAASnBAAEAAACrWAAC	101	Michael	13000	1
AAASnBAAEAAACrWAAD	101	Michael	13000	2
AAASnBAAEAAACrWAAE	101	Michael	13000	3
AAASnBAAEAAACrWAAF	102	Pat	6000	1
AAASnBAAEAAACrWAAG	102	Pat	6000	2
AAASnBAAEAAACrWAAH	103	Den	11000	1

STEP-2: Select ROWID of records with ROW_NUMBER > 1

Query:

```
SELECT ROWID FROM(
      SELECT ROWID,
             EMPLOYEE_ID,
             NAME,
             SALARY,
             ROW_NUMBER() OVER(PARTITION BY EMPLOYEE_ID,NAME,SALARY ORDER BY EMPLOYEE_ID) AS
ROW_NUMBER
      FROM EMPLOYEE)
WHERE ROW_NUMBER > 1;
```

Result:

ROWID
AAASnBAAEAAACrWAAB
AAASnBAAEAAACrWAAD
AAASnBAAEAAACrWAAE
AAASnBAAEAAACrWAAG

STEP-3: Delete the records from the source table using the ROWID values fetched in previous step

Query:

```
DELETE FROM EMP WHERE ROWID IN (
    SELECT ROWID FROM(
        SELECT ROWID,
        ROW_NUMBER() OVER(PARTITION BY EMPLOYEE_ID,NAME,SALARY ORDER BY EMPLOYEE_ID) AS
ROW_NUMBER
    FROM EMPLOYEE)
WHERE ROW_NUMBER > 1);
```

Result:

The table EMPLOYEE will have below records after deleting the duplicates

ROWID	EMPLOYEE_ID	NAME	SALARY
AAASnBAAEAAACrWAAA	100	Jennifer	4400
AAASnBAAEAAACrWAAC	101	Michael	13000
AAASnBAAEAAACrWAAF	102	Pat	6000
AAASnBAAEAAACrWAAH	103	Den	11000

3. How to read TOP 5 records from a table using a SQL query?

Consider below table DEPARTMENTS as the source data

```
CREATE TABLE Departments(
   Department_ID number,
   Department_Name varchar(50)
);

INSERT INTO DEPARTMENTS VALUES('10','Administration');
INSERT INTO DEPARTMENTS VALUES('20','Marketing');
INSERT INTO DEPARTMENTS VALUES('30','Purchasing');
INSERT INTO DEPARTMENTS VALUES('40','Human Resources');
INSERT INTO DEPARTMENTS VALUES('50','Shipping');
INSERT INTO DEPARTMENTS VALUES('60','IT');
INSERT INTO DEPARTMENTS VALUES('70','Public Relations');
INSERT INTO DEPARTMENTS VALUES('80','Sales');

SELECT * FROM Departments;
```

DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
30	Purchasing
40	Human Resources
50	Shipping
60	IT
70	Public Relations
80	Sales

ROWNUM is a “Pseudocolumn” that assigns a number to each row returned by a query indicating the order in which Oracle selects the row from a table. The first row selected has a ROWNUM of 1, the second has 2, and so on.

Query:

```
SELECT * FROM Departments WHERE ROWNUM <= 5;
```

Result:

DEPARTMENT_ID	DEPARTMENT_NAME
10	Administration
20	Marketing
30	Purchasing
40	Human Resources
50	Shipping

4. How to read LAST 5 records from a table using a SQL query?

Consider the same DEPARTMENTS table as source discussed in previous question.

In order to select the last 5 records we need to find (count of total number of records – 5) which gives the count of records from first to last but 5 records.Using the MINUS function we can compare all records from DEPARTMENTS table with records from first to last but 5 from DEPARTMENTS table which give the last 5 records of the table as result.

MINUS operator is used to return all rows in the first SELECT statement that are not present in the second SELECT statement.

Query:

```
SELECT * FROM Departments

MINUS

SELECT * FROM Departments WHERE ROWNUM <= (SELECT COUNT(*)-5 FROM Departments);
```

Result:

DEPARTMENT_ID	DEPARTMENT_NAME
40	Human Resources
50	Shipping
60	IT
70	Public Relations
80	Sales

5. How to find the employee with second MAX Salary using a SQL

Consider below EMPLOYEES table as the source data

```
CREATE TABLE Employees(
        EMPLOYEE_ID NUMBER(6,0),
        NAME VARCHAR2(20 BYTE),
        SALARY NUMBER(8,2)
        );

INSERT INTO EMPLOYEES(EMPLOYEE_ID,NAME,SALARY) VALUES(100,'Jennifer',4400);
INSERT INTO EMPLOYEES(EMPLOYEE_ID,NAME,SALARY) VALUES(101,'Michael',13000);
INSERT INTO EMPLOYEES(EMPLOYEE_ID,NAME,SALARY) VALUES(102,'Pat',6000);
INSERT INTO EMPLOYEES(EMPLOYEE_ID,NAME,SALARY) VALUES(103,'Den', 11000);
INSERT INTO EMPLOYEES(EMPLOYEE_ID,NAME,SALARY) VALUES(104,'Alexander',3100);
INSERT INTO EMPLOYEES(EMPLOYEE_ID,NAME,SALARY) VALUES(105,'Shelli',2900);
INSERT INTO EMPLOYEES(EMPLOYEE_ID,NAME,SALARY) VALUES(106,'Sigal',2800);
INSERT INTO EMPLOYEES(EMPLOYEE_ID,NAME,SALARY) VALUES(107,'Guy',2600);
INSERT INTO EMPLOYEES(EMPLOYEE_ID,NAME,SALARY) VALUES(108,'Karen',2500);

SELECT * FROM Employees;
```

EMPLOYEE_ID	NAME	SALARY
100	Jennifer	4400
101	Michael	13000
102	Pat	6000
103	Den	11000
104	Alexander	3100
105	Shelli	2900
106	Sigel	2800
107	Guy	2600
108	Karen	2500

Without using SQL Analytic Functions

In order to find the second MAX salary, employee record with MAX salary needs to be eliminated. It can be achieved by using below SQL query.

Query:

```
SELECT MAX(salary) AS salary FROM Employees WHERE salary NOT IN (
SELECT MAX(salary) AS salary FROM Employees);
```

Result:

SALARY
11000

The above query only gives the second MAX salary value. In order to fetch the entire employee record with second MAX salary we need to do a self-join on Employee table based on Salary value.

Query:

```
WITH
TEMP AS(
  SELECT MAX(salary) AS salary FROM Employees WHERE salary NOT IN (
  SELECT MAX(salary) AS salary FROM Employees)
)
SELECT a.* FROM Employees a JOIN TEMP b on a.salary = b.salary
```

Result:

EMPLOYEE_ID	NAME	SALARY
103	Den	11000

6. How to find the employee with third MAX Salary using a SQL query without using Analytic Functions?

Consider the same EMPLOYEES table as source discussed in previous question

In order to find the third MAX salary, we need to eliminate the top 2 salary records. But we cannot use the same method we used for finding second MAX salary (not a best practice). Imagine if we have to find the fifth MAX salary. We should not be writing a query with four nested sub queries.

STEP-1:

The approach here is to first list all the records based on Salary in the descending order with MAX salary on top and MIN salary at bottom. Next, using ROWNUM select the top 2 records.

Query:

```
SELECT salary FROM(
SELECT salary FROM Employees ORDER BY salary DESC)
WHERE ROWNUM < 3;
```

Result:

Salary
13000
11000

STEP-2:

Next find the MAX salary from EMPLOYEE table which is not one of top two salary values fetched in the earlier step.

Query:

```
SELECT MAX(salary) as salary FROM Employees WHERE salary NOT IN (
    SELECT salary FROM(
    SELECT salary FROM Employees ORDER BY salary DESC)
    WHERE ROWNUM < 3
);
```

Result:

SALARY
6000

STEP-3:

In order to fetch the entire employee record with third MAX salary we need to do a self-join on Employee table based on Salary value.

Query:

```
WITH
TEMP AS(
  SELECT MAX(salary) as salary FROM Employees WHERE salary NOT IN (
      SELECT salary FROM(
      SELECT salary FROM Employees ORDER BY salary DESC)
      WHERE ROWNUM < 3)
)
SELECT a.* FROM Employees a join TEMP b on a.salary = b.salary
```

Result:

EMPLOYEE_ID	NAME	SALARY
102	Pat	6000

7. Assume you are given the below table on transactions from users. Write a query to get the number of users and total products bought per latest transaction date where each user is bucketed into their latest transaction date.

First, we need to get the latest transaction date for each user, along with the number of products they have purchased. This can be done in a subquery where we GROUP BY user_id and take a COUNT(DISTINCT product_id) to get the number of products they have purchased, and a MAX(transaction_date) to get the latest transaction date (while casting to a date). Then, using this subquery, we can simply do an aggregation by the transaction date column in the previous subquery, while doing a COUNT() on the number of users, and a SUM() on the number products:

```
WITH latest_date AS (
    SELECT user_id,
        COUNT(DISTINCT product_id) AS num_products,
        MAX(transaction_date::DATE) AS curr_date
    FROM user_transactions
    GROUP BY )

SELECT curr_date,
    COUNT(user_id) AS num_users,
    SUM(num_products) AS total_products
FROM
latest_date
GROUP BY 1
```

8. Write SQL Query to display the current date?

SQL has built-in function called GetDate() which returns the current timestamp.

```
SELECT GetDate();
```

9. Write an SQL Query to print the name of the distinct employee whose DOB is between 01/01/1960 to 31/12/1975.

```
SELECT DISTINCT EmpName
FROM Employees
WHERE DOB BETWEEN '01/01/1960' AND '31/12/1975';
```

10. Write an SQL Query to find an employee whose salary is equal to or greater than 10000.

```
SELECT EmpName FROM  Employees WHERE  Salary>=10000;
```

11. Write SQL Query to find duplicate rows in a database? and then write SQL query to delete them?

```
SELECT * FROM emp a
WHERE rowid = (SELECT MAX(rowid)
FROM EMP b
WHERE a.empno=b.empno)
```

to Delete:

```
DELETE FROM emp a
WHERE rowid != (SELECT MAX(rowid) FROM emp b WHERE a.empno=b.empno);
```

12. The Trips table holds all taxi trips. Each trip has a unique Id, while Client_Id and Driver_Id are both foreign keys to the Users_Id at the Users table. Status is an ENUM type of ('completed', 'cancelled_by_driver', 'cancelled_by_client').

Id	Client_Id	Driver_Id	City_Id	Status	Request_at
1	1	10	1	completed	2013-10-01
2	2	11	1	cancelled_by_driver	2013-10-01
3	3	12	6	completed	2013-10-01
4	4	13	6	cancelled_by_client	2013-10-01
5	1	10	1	completed	2013-10-02
6	2	11	6	completed	2013-10-02
7	3	12	6	completed	2013-10-02
8	2	12	12	completed	2013-10-03
9	3	10	12	completed	2013-10-03
10	4	13	12	cancelled_by_driver	2013-10-03

The `Users` table holds all users. Each user has an unique Users_Id, and Role is an ENUM type of ('client', 'driver', 'partner').

Users_Id	Banned	Role
1	No	client
2	Yes	client
3	No	client
4	No	client
10	No	driver
11	No	driver
12	No	driver
13	No	driver

Write a SQL query to find the cancellation rate of requests made by unbanned users between Oct 1, 2013 and Oct 3, 2013. For the above

tables, your SQL query should return the following rows with the cancellation rate being rounded to two decimal places.

```
+------------+-------------------+
|    Day     | Cancellation Rate |
+------------+-------------------+
| 2013-10-01 |       0.33        |
| 2013-10-02 |       0.00        |
| 2013-10-03 |       0.50        |
+------------+-------------------+
```

The solution looks like that:

```
select
        result.Request_at as "Day",
        round(sum(case when result.Status = 'completed' then 0 else 1 end)
/ count(*), 2) as "Cancellation Rate"
from (
    select
                Driver_Id,
                Status,
                Request_at
    from trips left join users on trips.client_id=users.users_id
    where users.banned = 'NO'
     ) result
        left join users on result.driver_id=users.users_id
where
        users.Banned ='NO'
        and result.Request_at between '2013-10-01' and '2013-10-03'
group by
        result.Request_at
```

13. Write a SQL query to find all duplicate emails in a table named Person.

Table: `Customers`.

```
+----+---------+
| Id | Email   |
+----+---------+
| 1  | a@b.com |
| 2  | c@d.com |
| 3  | a@b.com |
+----+---------+
```

For example, your query should return the following for the above table:

```
+---------+
| Email   |
+---------+
| a@b.com |
+---------+
```

Solution:

Since all email are in lowercase we can simply groupby email and print those that have a count >1.

```
SELECT EMAIL
FROM PERSON
GROUP BY EMAIL
HAVING COUNT(*)>1
```

14. Given a Weather table, write a SQL query to find all dates' Ids with higher temperature compared to its previous (yesterday's) dates.

```
+---------+------------------+------------------+
| Id(INT) | RecordDate(DATE) | Temperature(INT) |
+---------+------------------+------------------+
|       1 |       2015-01-01 |               10 |
|       2 |       2015-01-02 |               25 |
|       3 |       2015-01-03 |               20 |
|       4 |       2015-01-04 |               30 |
+---------+------------------+------------------+
```

For example, return the following Ids for the above `Weather` table:

```
+----+
| Id |
+----+
|  2 |
|  4 |
+----+
```

The solution is to join the table to itself when the dates differ by one day (`DATEDIFF()` function) and make sure that the temperature is higher than the previous date.

```
SELECT W1.ID
FROM WEATHER W1 INNER JOIN WEATHER W2 ON DATEDIFF(W1.RecordDate, W2.RecordDate) = 1
WHERE W1.Temperature > W2.Temperature
```

15. The Employee table holds all employees including their managers. Every employee has an Id, and there is also a column for the manager Id.

```
+----+-------+--------+-----------+
| Id | Name  | Salary | ManagerId |
+----+-------+--------+-----------+
| 1  | Joe   | 70000  | 3         |
| 2  | Henry | 80000  | 4         |
| 3  | Sam   | 60000  | NULL      |
| 4  | Max   | 90000  | NULL      |
+----+-------+--------+-----------+
```

Given the Employee table, write a SQL query that finds out employees who earn more than their managers. For the above table, Joe is the only employee who earns more than his manager.

```
+----------+
| Employee |
+----------+
| Joe      |
+----------+
```

The solution is to join again the table to itself as shown below:

```
SELECT E1.NAME AS EMPLOYEE
FROM EMPLOYEE E1 INNER JOIN EMPLOYEE E2 ON E1.MANAGERID=E2.ID
WHERE E1.SALARY>E2.SALARY
```

16. The Employee table holds all employees. Every employee has an Id, a salary, and there is also a column for the department Id.

```
+----+-------+--------+--------------+
| Id | Name  | Salary | DepartmentId |
+----+-------+--------+--------------+
| 1  | Joe   | 70000  | 1            |
| 2  | Jim   | 90000  | 1            |
| 3  | Henry | 80000  | 2            |
| 4  | Sam   | 60000  | 2            |
| 5  | Max   | 90000  | 1            |
+----+-------+--------+--------------+
```

```
+------------+----------+--------+
| Department | Employee | Salary |
+------------+----------+--------+
| IT         | Max      | 90000  |
| IT         | Jim      | 90000  |
| Sales      | Henry    | 80000  |
+------------+----------+--------+
```

The `Department` table holds all departments of the company.

```
+----+----------+
| Id | Name     |
+----+----------+
| 1  | IT       |
| 2  | Sales    |
+----+----------+
```

Write a SQL query to find employees who have the highest salary in each of the departments. For the above tables, your SQL query should return the following rows (order of rows does not matter).

The solution looks like that:

```
select
    department.name as Department,
        result.Name as Employee,
        result.Salary as Salary
from
    (select
                e1.DepartmentId,
                e1.Name,e1.Salary
    from
                Employee e1 left join Employee e2 on e1.Salary <= e2.Salary
        and e1.DepartmentId = e2.DepartmentId
    group by
        e1.DepartmentId,
        e1.Salary,
        e1.Name
    having count(distinct e2.Salary) = 1)
   result join department on result.DepartmentId=department.id
```

The solution looks pretty similar with the one presented above for the Department Highest Salary problem:

```
CREATE FUNCTION getNthHighestSalary(N INT) RETURNS INT
BEGIN
  RETURN (
      SELECT E1.SALARY
      FROM EMPLOYEE E1 LEFT JOIN EMPLOYEE E2 ON E1.SALARY <= E2.SALARY
      GROUP BY E1.SALARY
      HAVING COUNT(DISTINCT E2.SALARY) = N

  );
END
```

17. From the following table of user IDs, actions, and dates, write a query to return the publication and cancellation rate for each user.

users

user_id	action	date
1	start	1-1-20
1	cancel	1-2-20
2	start	1-3-20
2	publish	1-4-20
3	start	1-5-20
3	cancel	1-6-20
4	start	1-7-20

Desired output

user_id	publish_rate	cancel_rate
1	0.5	0.5
2	1.0	0.0
3	0.0	1.0

```
WITH users (user_id, action, date)
AS (VALUES
(1,'start', CAST('01-01-20' AS date)),
(1,'cancel', CAST('01-02-20' AS date)),
(2,'start', CAST('01-03-20' AS date)),
(2,'publish', CAST('01-04-20' AS date)),
(3,'start', CAST('01-05-20' AS date)),
(3,'cancel', CAST('01-06-20' AS date)),
(1,'start', CAST('01-07-20' AS date)),
(1,'publish', CAST('01-08-20' AS date))),

-- retrieve count of starts, cancels, and publishes for each user

t1 AS (
SELECT user_id,
sum(CASE WHEN action = 'start' THEN 1 ELSE 0 END) AS starts,
sum(CASE WHEN action = 'cancel' THEN 1 ELSE 0 END) AS cancels,
sum(CASE WHEN action = 'publish' THEN 1 ELSE 0 END) AS publishes
FROM users
GROUP BY 1
ORDER BY 1)
```

```
-- calculate publication, cancelation rate for each user by dividing
by number of starts, casting as float by multiplying by 1.0

SELECT user_id, 1.0*publishes/starts AS publish_rate,
1.0*cancels/starts AS cancel_rate
FROM t1
```

18. From the following table of transactions between two users, write a query to return the change in net worth for each user, ordered by decreasing net change.

transactions

sender	receiver	amount	transaction_date
5	2	10	2-12-20
1	3	15	2-13-20
2	1	20	2-13-20
2	3	25	2-14-20
3	1	20	2-15-20
3	2	15	2-15-20
1	4	5	2-16-20

Desired output

user	net_change
1	20
3	5
4	5
5	-10
2	-20

```
WITH transactions (sender, receiver, amount, transaction_date)
AS (VALUES
(5, 2, 10, CAST('2-12-20' AS date)),
(1, 3, 15, CAST('2-13-20' AS date)),
(2, 1, 20, CAST('2-13-20' AS date)),
(2, 3, 25, CAST('2-14-20' AS date)),
(3, 1, 20, CAST('2-15-20' AS date)),
(3, 2, 15, CAST('2-15-20' AS date)),
(1, 4, 5, CAST('2-16-20' AS date))),

-- sum amounts for each sender (debits) and receiver (credits)

debits AS (
SELECT sender, sum(amount) AS debited
FROM transactions
GROUP BY sender ),

credits AS (
SELECT receiver, sum(amount) AS credited
FROM transactions
GROUP BY receiver )
```

```
-- full (outer) join debits and credits tables on user id, taking net
change as difference between credits and debits, coercing nulls to
zeros with coalesce()


SELECT coalesce(sender, receiver) AS user,
coalesce(credited, 0) - coalesce(debited, 0) AS net_change
FROM debits d
FULL JOIN credits c
ON d.sender = c.receiver
ORDER BY 2 DESC
```

```
-- grouping by days since signup, count (non-null) user IDs as active
users, total users, and the quotient as retention rate


SELECT day_no, count(*) AS n_total,
       count(DISTINCT user_id) AS n_active,
       round(1.0*count(DISTINCT user_id)/count(*), 2) AS retention
FROM t1
GROUP BY 1
```

19. From the given trips and users tables for a taxi service, write a query to return the cancellation rate in the first two days in October, rounded to two decimal places, for trips not involving banned riders or drivers.

trips

trip_id	rider_id	driver_id	status	request_date
1	1	10	completed	2020-10-01
2	2	11	cancelled_by_driver	2020-10-01
3	3	12	completed	2020-10-01
4	4	10	cancelled_by_rider	2020-10-02
5	1	11	completed	2020-10-02
6	2	12	completed	2020-10-02
7	3	11	completed	2020-10-03

users

user_id	banned	type
1	no	rider
2	yes	rider
3	no	rider
4	no	rider
10	no	driver
11	no	driver
12	no	driver

Desired output

request_date	cancel_rate
2020-10-01	0.50
2020-10-02	0.33

```
WITH trips (trip_id, rider_id, driver_id, status, request_date)
AS (VALUES
(1, 1, 10, 'completed', CAST('2020-10-01' AS date)),
(2, 2, 11, 'cancelled_by_driver', CAST('2020-10-01' AS date)),
(3, 3, 12, 'completed', CAST('2020-10-01' AS date)),
(4, 4, 10, 'cancelled_by_rider', CAST('2020-10-02' AS date)),
(5, 1, 11, 'completed', CAST('2020-10-02' AS date)),
(6, 2, 12, 'completed', CAST('2020-10-02' AS date)),
(7, 3, 11, 'completed', CAST('2020-10-03' AS date))),


users (user_id, banned, type)
AS (VALUES
(1, 'no', 'rider'),
(2, 'yes', 'rider'),
(3, 'no', 'rider'),
(4, 'no', 'rider'),
(10, 'no', 'driver'),
(11, 'no', 'driver'),
(12, 'no', 'driver'))
```

```
-- filter trips table to exclude banned riders and drivers, then
calculate cancellation rate as 1 - fraction of trips completed,
rounding as requested and filtering to first two days of the month


SELECT request_date, round(1 - 1.0*sum(CASE WHEN status = 'completed'
THEN 1 ELSE 0 END)/count(*), 2) AS cancel_rate
FROM trips
WHERE rider_id NOT IN (SELECT user_id
                       FROM users
                       WHERE banned = 'yes' )
AND driver_id NOT IN (SELECT user_id
                      FROM users
                      WHERE banned = 'yes' )
GROUP BY request_date
HAVING extract(DAY FROM request_date) <= 2
```