Que 1) Given a table "UserActivity" with columns "user_id," "action," and "timestamp," write an SQL query to find the number of unique users who performed each action on a specific date.

**Example:**

**UserActivity:**

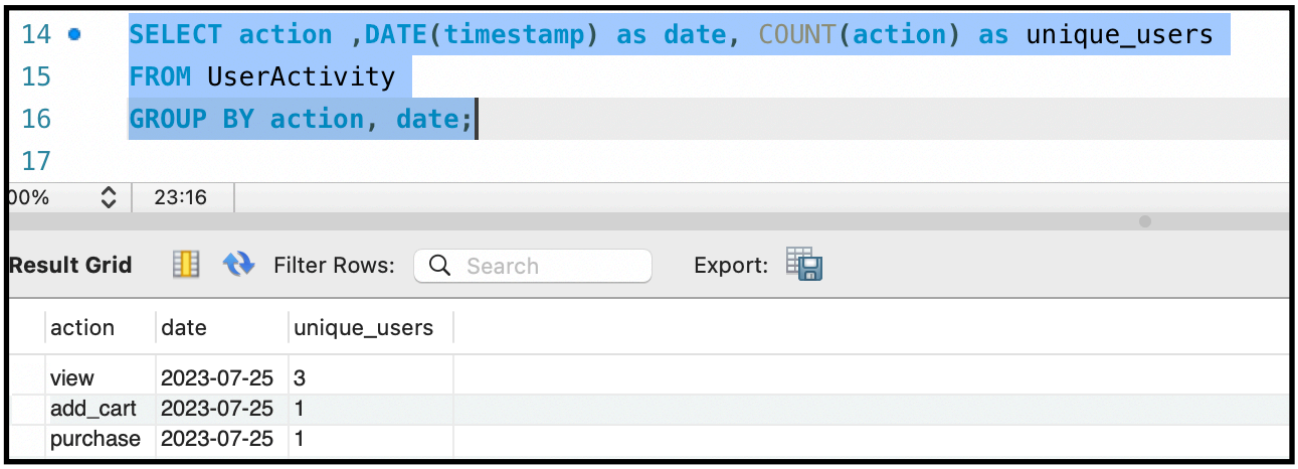
| user_id | action | timestamp |
|---|---|---|
| 1 | view | 2023-07-25 08:15:00 |
| 2 | add_cart | 2023-07-25 09:30:00 |
| 1 | purchase | 2023-07-25 10:00:00 |
| 3 | view | 2023-07-25 12:45:00 |
| 2 | view | 2023-07-25 14:20:00 |

**Result:**

| action | date | unique_users |
|---|---|---|
| view | 2023-07-25 | 3 |
| add_cart | 2023-07-25 | 1 |
| purchase | 2023-07-25 | 1 |

**Solution**

```
SELECT action ,DATE(timestamp) as date, COUNT(action) as unique_users
FROM UserActivity
GROUP BY action, date;
```




Que 2) You have a table "Sales" with columns "user_id," "product_id," "timestamp," and "purchase_status." Write an SQL query to calculate the conversion rate of users from each stage of the sales funnel (e.g., viewed the product, added to cart, completed the purchase).

**Example:**

| user_id | product_id | timestamp | purchase_status |
|---|---|---|---|
| 1 | A | 2023-07-25 08:15:00 | viewed |
| 2 | B | 2023-07-25 09:30:00 | added_to_cart |
| 1 | A | 2023-07-25 10:00:00 | completed |
| 3 | C | 2023-07-25 12:45:00 | viewed |
| 2 | B | 2023-07-25 14:20:00 | completed |

**Result:**

| purchase_status | viewed_users | added_to_cart_users | completed_users | total_users |
|---|---|---|---|---|
| viewed | 2 | 0 | 0 | 2 |
| Added_to_cart | 0 | 1 | 0 | 1 |
| completed | 1 | 0 | 2 | 3 |

**Solution:**

```
SELECT * FROM UserPurchases;
WITH
  user_summary AS
(
  SELECT
    user_id,
    product_id,
    purchase_status,
    SUM(CASE WHEN purchase_status = 'viewed' THEN 1 ELSE 0 END)
      OVER (PARTITION BY user_id, product_id)
        AS viewed_users,
    SUM(CASE WHEN purchase_status = 'added_to_cart' THEN 1 ELSE 0 END)
      OVER (PARTITION BY user_id, product_id)
        AS added_to_cart_users,
    SUM(CASE WHEN purchase_status = 'completed' THEN 1 ELSE 0 END)
      OVER (PARTITION BY user_id, product_id)
        AS completed_users,
    COUNT(*)
      OVER (PARTITION BY user_id, product_id)
        AS total_users
  FROM
    UserPurchases
)
SELECT
  purchase_status,
  SUM(viewed_users),
  SUM(added_to_cart_users),
  SUM(completed_users),
  SUM(total_users)
FROM
```

```
  user_summary
GROUP BY
  purchase_status
ORDER BY
    CASE
        WHEN purchase_status = 'viewed' THEN 1
        WHEN purchase_status = 'added_to_cart' THEN 2
        WHEN purchase_status = 'completed' THEN 3
    END;
```

Result Grid  Filter Rows: Search  Export:

| purchase_stat... | SUM(viewed_user... | SUM(added_to_cart_use... | SUM(completed_use... | SUM(total_user... |
|---|---|---|---|---|
| viewed | 2 | 0 | 1 | 3 |
| added_to_cart | 0 | 1 | 1 | 2 |
| completed | 1 | 1 | 2 | 4 |

Que 3) Write an SQL query to calculate the cumulative sum of "Revenue" for each "Category," resetting the sum when encountering a new category.

**Example:**

**Sales:**

| Category | Revenue |
|---|---|
| A | 100 |
| A | 150 |
| B | 200 |
| A | 50 |
| B | 300 |
| B | 250 |

**Result:**

| Category | Revenue | cumulative_sum |
|---|---|---|
| A | 50 | 50 |
| A | 100 | 150 |
| A | 150 | 300 |
| B | 200 | 200 |
| B | 250 | 450 |
| B | 300 | 750 |

**Solution:**

```
SELECT
    Category,
    Revenue,
    SUM(Revenue) OVER(PARTITION BY Category ORDER BY Revenue) AS
cumulative_sum
FROM
        Sal;
```

```sql
SELECT
    Category,
    Revenue,
    SUM(Revenue) OVER(PARTITION BY Category ORDER BY Revenue) AS cumulative_sum
FROM
    Sal;
```

100% 6:19

Result Grid Filter Rows: Search Export:

| Category | Revenue | cumulative_sum |
|---|---|---|
| A | 50 | 50 |
| A | 100 | 150 |
| A | 150 | 300 |
| B | 200 | 200 |
| B | 250 | 450 |
| B | 300 | 750 |

Que 4) Write an SQL query to rank "Scores" in the "Students" table. If two students have the same score, they should receive the same rank, and the next rank should be skipped.

**Example:**

| Student | Scores |
|---|---|
| Alice | 85 |
| Bob | 92 |
| Charlie | 85 |
| David | 78 |
| Eve | 92 |

**Result:**

| Student | Scores | rank |
|---|---|---|
| Bob | 92 | 1 |
| Eve | 92 | 1 |
| Alice | 85 | 3 |
| Charlie | 85 | 3 |
| David | 78 | 5 |

**Solution:**

```sql
SELECT
        StudentName,
   Score,
   RANK() OVER(ORDER BY Score DESC) AS "rank"
FROM
        Students;
```

```sql
SELECT
    StudentName,
    Score,
    RANK() OVER(ORDER BY Score DESC) AS "rank"
FROM
    Students;
```

100% 11:18

Result Grid Filter Rows: Search Export:

| StudentName | Score | rank |
|---|---|---|
| Bob | 92 | 1 |
| Eve | 92 | 1 |
| Alice | 85 | 3 |
| Charlie | 85 | 3 |
| David | 78 | 5 |

Que 5) Given a table "Transaction" with columns "transaction_id," "product_id," and "timestamp," write an SQL query to find the top 5 most frequently co-occurring product pairs in transactions.

**Example**
**Transactions:**

| transaction_id | product_id | timestamp |
|---|---|---|
| 1 | A | 2023-07-25 08:15:00 |
| 2 | B | 2023-07-25 09:30:00 |
| 3 | A | 2023-07-25 10:00:00 |
| 4 | C | 2023-07-25 12:45:00 |
| 5 | A | 2023-07-25 14:20:00 |

**Result:**

| product1 | product2 | co_occurrences |
|---|---|---|
| A | B | 3 |
| A | C | 2 |
| B | C | 1 |
| D | E | 1 |
| A | D | 1 |

```sql
SELECT
    t1.product_id AS product1,
    t2.product_id AS product2,
    COUNT(*) AS co_occurrences
FROM
    Transactions t1
JOIN
    Transactions t2
ON
    t1.product_id < t2.product_id AND t1.transaction_id != t2.transaction_id
GROUP BY
    1,2
ORDER BY
    co_occurrences DESC;
```

00% 3:22

Result Grid Filter Rows: Search Export:

| product1 | product2 | co_occurrences |
|---|---|---|
| A | B | 3 |
| A | C | 3 |
| B | C | 1 |

Que 6) Using a table "Locations" with columns "location_id," "latitude," and "longitude," write an SQL query to find the closest locations to a given set of latitude and longitude coordinates.

**Example:**

**Locations:**

| location_id | latitude | longitude |
|---|---|---|
| 1 | 40.7128 | -74.0060 |
| 2 | 34.0522 | -118.2437 |
| 3 | 41.8781 | -87.6298 |
| 4 | 37.7749 | -122.4194 |
| 5 | 29.7604 | -95.3698 |

**Example Latitude and Longitude Coordinates (target location):**

| latitude_target | longitude_target |
|---|---|
| 38.9072 | -77.0369 |

**Result:**

| location_id | latitude | longitude | distance |
|---|---|---|---|
| 1 | 40.7128 | -74.0060 | 3.602606223603 |
| 3 | 41.8781 | -87.6298 | 9.815368890392 |
| 4 | 37.7749 | -122.4194 | 38.49109332774 |
| 5 | 29.7604 | -95.3698 | 28.67409915722 |
| 2 | 34.0522 | -118.2437 | 32.64350998995 |

**Solution:**

```
SELECT
   location_id,
   latitude,
   longitude,
   (
      6371 * ACOS(
         COS(RADIANS(38.9072)) * COS(RADIANS(latitude)) *
         COS(RADIANS(longitude) - RADIANS(-77.0369)) +
         SIN(RADIANS(38.9072)) * SIN(RADIANS(latitude))
      )
   ) AS distance
FROM
   Locations
ORDER BY
   distance ASC;
```

```sql
SELECT
    location_id,
    latitude,
    longitude,
    ROUND((
        6371 * ACOS(
            COS(RADIANS(38.9072)) * COS(RADIANS(latitude)) *
            COS(RADIANS(longitude) - RADIANS(-77.0369)) +
            SIN(RADIANS(38.9072)) * SIN(RADIANS(latitude))
        )
    ),2) AS distance
FROM
    Locations
ORDER BY
    distance ASC;
```

100% 18:28

Result Grid Filter Rows: Search Export:

| location_id | latitude | longitude | distance |
|---|---|---|---|
| 1 | 40.712800 | -74.006000 | 327.58 |
| 3 | 41.878100 | -87.629800 | 955.18 |
| 5 | 29.760400 | -95.369800 | 1961.09 |
| 2 | 34.052200 | -118.243700 | 3692 |
| 4 | 37.774900 | -122.419400 | 3918.55 |

Que 7) You have two tables: "ControlGroup" with columns "user_id" and "conversion_status," and "ExperimentalGroup" with the same columns. Write an SQL query to calculate the conversion rate for both groups and determine if the experimental group has a statistically significant difference in conversion compared to the control group.

**Example:**

**Control group:**

| user_id | conversion_status |
|---|---|
| 1 | converted |
| 2 | not_converted |
| 3 | converted |
| 4 | not_converted |
| 5 | not_converted |

**Experimental group:**

| user_id | conversion_status |
|---|---|
| 6 | not_converted |
| 7 | converted |
| 8 | converted |

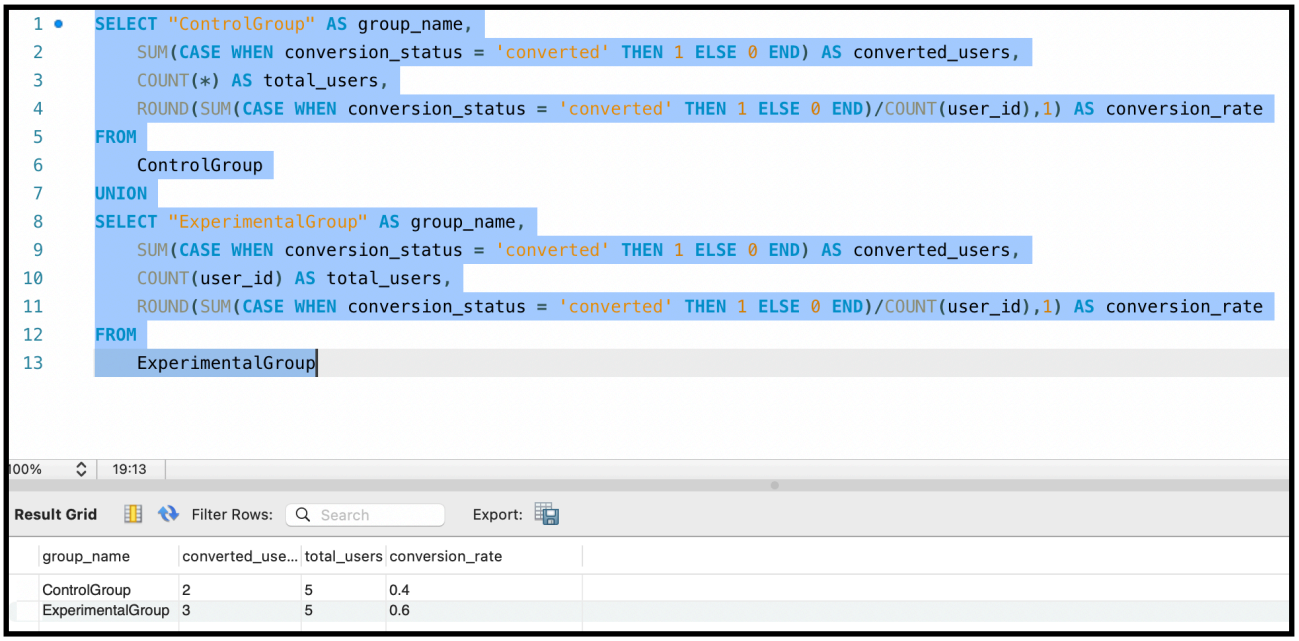
| | |
|---|---|
| 9 | not_converted |
| 10 | converted |

Result:

| group_name | converted_users | total_users | conversion_rate |
|---|---|---|---|
| ControlGroup | 2 | 5 | 0.4 |
| ExperimentalGroup | 3 | 5 | 0.6 |

**Solution:**

```
SELECT "ControlGroup" AS group_name,
        SUM(CASE WHEN conversion_status = 'converted' THEN 1 ELSE 0 END) AS
converted_users,
   COUNT(*) AS total_users,
   ROUND(SUM(CASE WHEN conversion_status = 'converted' THEN 1 ELSE 0 END)/
COUNT(user_id),1) AS conversion_rate
FROM
        ControlGroup
UNION
SELECT "ExperimentalGroup" AS group_name,
        SUM(CASE WHEN conversion_status = 'converted' THEN 1 ELSE 0 END) AS
converted_users,
   COUNT(user_id) AS total_users,
   ROUND(SUM(CASE WHEN conversion_status = 'converted' THEN 1 ELSE 0 END)/
COUNT(user_id),1) AS conversion_rate
FROM
        ExperimentalGroup
```



Que 8) Given a table "Numbers" with a single column "number," write a query to find the missing numbers in the sequence from 1 to 100 (excluding the ones present in the table).

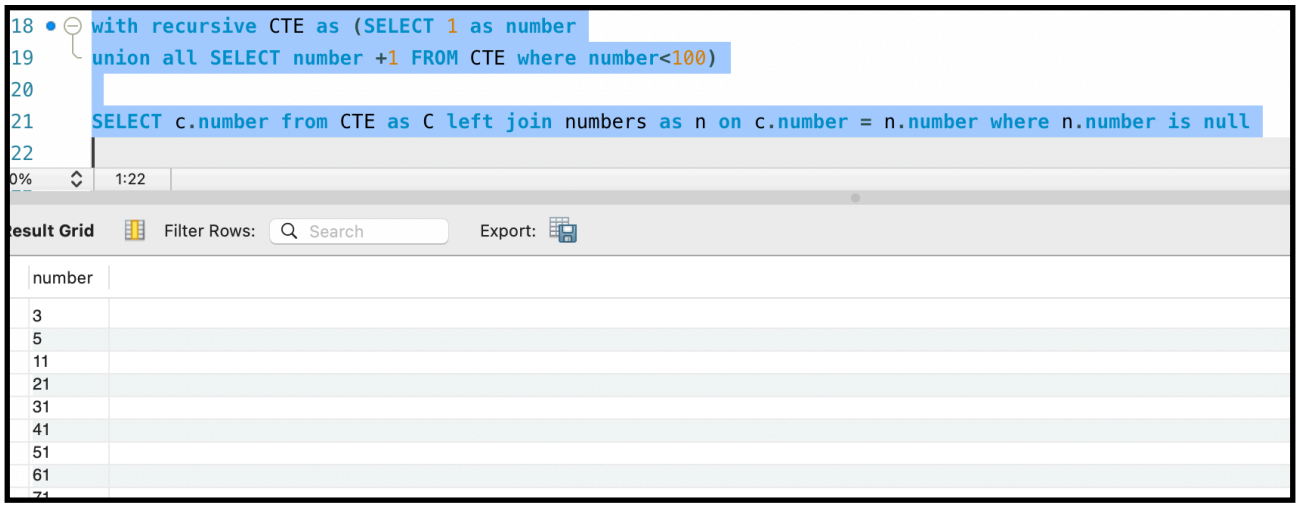
**Example:**

**Numbers:**

number
1
2
4
6
...
98
100

**Result:**

number
3
5
7
...
99




Solution:

```
WITH RECURSIVE NumberCte AS (
    SELECT 1 AS Number
    UNION ALL
    SELECT Number + 1
    FROM NumberCte
    WHERE Number < 100
)
SELECT cte.Number
FROM NumberCte cte
```

```
LEFT JOIN Numbers n ON cte.Number = n.Number
WHERE n.Number IS NULL;
```

Que 9) Using a table "CustomerData" with columns "customer_id," "age," "gender," and "purchase_amount," write an SQL query to segment customers into different age groups and genders, along with their average purchase amounts.

Example:

| customer_id | age | gender | purchase_amount |
|---|---|---|---|
| 1 | 25 | Male | 100.00 |
| 2 | 30 | Female | 50.00 |
| 3 | 22 | Male | 75.00 |
| 4 | 28 | Female | 120.00 |
| 5 | 35 | Male | 80.00 |

Result:

| age_group | gender | total_customers | avg_purchase_amount |
|---|---|---|---|
| 18-24 | Male | 10 | 75.00 |
| 18-24 | Female | 8 | 85.00 |
| 25-34 | Male | 20 | 95.00 |
| 25-34 | Female | 18 | 80.00 |
| 35-44 | Male | 15 | 110.00 |
| 35-44 | Female | 12 | 95.00 |
| 45-54 | Male | 8 | 120.00 |
| 45-54 | Female | 10 | 105.00 |
| 55+ | Male | 5 | 90.00 |
| 55+ | Female | 6 | 100.00 |

Solution:

```
SELECT
        CASE
                WHEN age BETWEEN 18 AND 24 THEN '18-24'
    WHEN age BETWEEN 25 AND 34 THEN '25-34'
    WHEN age BETWEEN 35 AND 44 THEN '35-44'
    WHEN age BETWEEN 45 AND 54 THEN '45-54'
    ELSE '55+'
        END
                AS age_group,
        gender,
  COUNT(*) AS total_customers,
  ROUND(AVG(purchase_amount),2) AS avg_purchase_amount
FROM
        CustomerData
GROUP BY
        age_group, gender
ORDER BY
        age_group, gender desc;
```

| age_group | gender | total_custom... | avg_purchase_amount |
|---|---|---|---|
| 18-24 | Male | 1 | 75.00 |
| 25-34 | Male | 1 | 100.00 |
| 25-34 | Female | 2 | 85.00 |
| 35-44 | Male | 1 | 80.00 |

Que 10) Write an SQL query to find the Nth highest salary from a "Salary" table.
**Example:**
**Salary**
50000
75000
60000
90000
80000

**Result:** The 3rd highest salary is 60000.

```sql
WITH highest AS(
    SELECT
        Salary,
        DENSE_RANK() OVER(ORDER BY Salary DESC) AS rn
    FROM
        SalaryData
)
SELECT salary as nthHighest FROM highest where rn = 3;
```

00% 55:19

Result Grid Filter Rows: Search Export:

| nthHighest |
|---|
| 75000 |