# Frequently Asked SQL Interview Questions

Swipe to next slide →

## > Q1. How to find duplicates in a given table.

```
SELECT employee_id
      ,COUNT(1) AS count_of_records
FROM employees
GROUP BY employee_id
HAVING COUNT(1) > 1
```

```
-- OUTPUT
| employee_id | count_of_records |
| ----------- | ---------------- |
| 1           | 2                |
```

Swipe to next slide

## > Q2. How to delete duplicates?

SQL

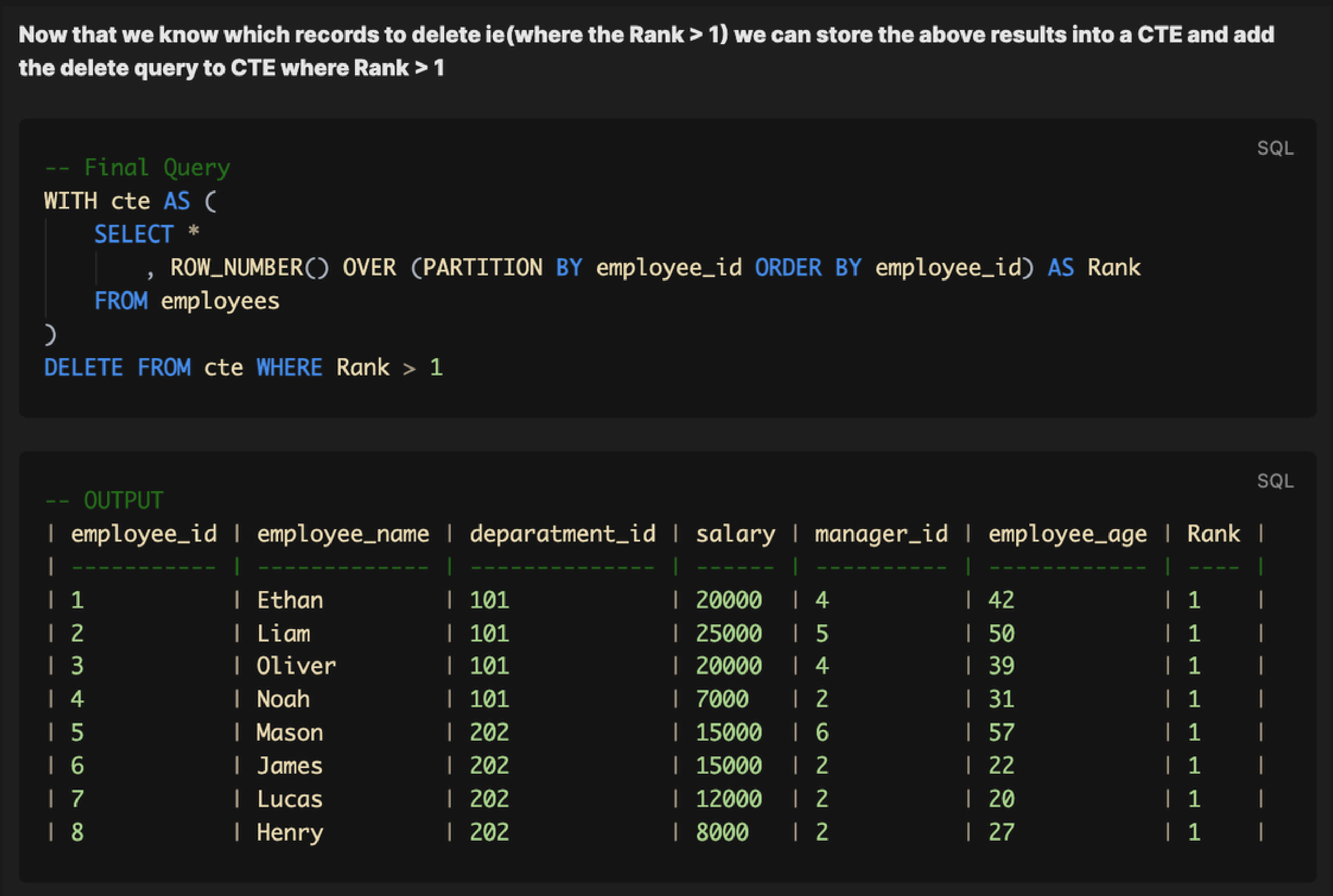
```sql
SELECT *
        , ROW_NUMBER() OVER (PARTITION BY employee_id ORDER BY employee_id) AS Rank
    FROM employees
```

SQL

```
-- OUTPUT
| employee_id | employee_name | deparatment_id | salary | manager_id | employee_age | Rank |
| ----------- | ------------- | -------------- | ------ | ---------- | ------------ | ---- |
| 1           | Ethan         | 101            | 20000  | 4          | 42           | 1    |
| 1           | William       | 303            | 18000  | 2          | 54           | 2    |
| 2           | Liam          | 101            | 25000  | 5          | 50           | 1    |
| 3           | Oliver        | 101            | 20000  | 4          | 39           | 1    |
| 4           | Noah          | 101            | 7000   | 2          | 31           | 1    |
| 5           | Mason         | 202            | 15000  | 6          | 57           | 1    |
| 6           | James         | 202            | 15000  | 2          | 22           | 1    |
| 7           | Lucas         | 202            | 12000  | 2          | 20           | 1    |
| 8           | Henry         | 202            | 8000   | 2          | 27           | 1    |
```

Swipe to next slide →

## > Q2. How to delete duplicates?



**Now that we know which records to delete ie(where the Rank > 1) we can store the above results into a CTE and add the delete query to CTE where Rank > 1**

SQL

```sql
-- Final Query
WITH cte AS (
    SELECT *
        , ROW_NUMBER() OVER (PARTITION BY employee_id ORDER BY employee_id) AS Rank
    FROM employees
)
DELETE FROM cte WHERE Rank > 1
```

SQL

```
-- OUTPUT
| employee_id | employee_name | deparatment_id | salary | manager_id | employee_age | Rank |
| ----------- | ------------- | -------------- | ------ | ---------- | ------------ | ---- |
| 1           | Ethan         | 101            | 20000  | 4          | 42           | 1    |
| 2           | Liam          | 101            | 25000  | 5          | 50           | 1    |
| 3           | Oliver        | 101            | 20000  | 4          | 39           | 1    |
| 4           | Noah          | 101            | 7000   | 2          | 31           | 1    |
| 5           | Mason         | 202            | 15000  | 6          | 57           | 1    |
| 6           | James         | 202            | 15000  | 2          | 22           | 1    |
| 7           | Lucas         | 202            | 12000  | 2          | 20           | 1    |
| 8           | Henry         | 202            | 8000   | 2          | 27           | 1    |
```

Swipe to next slide →

## > Q3. Difference between UNION and UNION ALL

`UNION ALL` fetches all records from both tables including duplicates whereas `UNION` will fetch all distinct records and eliminate the duplicate records.

SQL

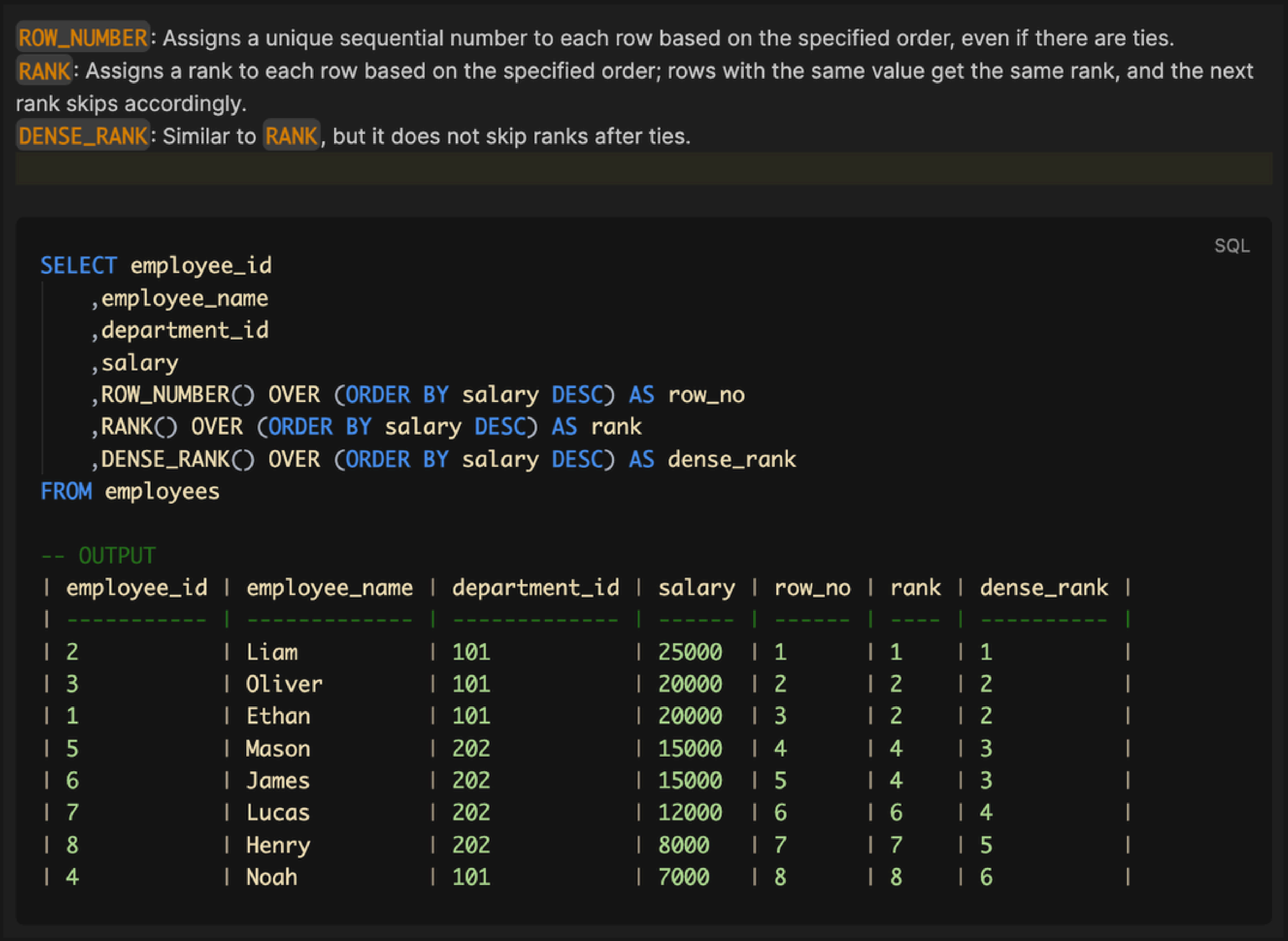
```sql
-- UNION ALL
SELECT manager_id FROM employee4
UNION ALL
SELECT manager_id FROM employees2

-- UNION
SELECT manager_id FROM employee
UNION ALL
SELECT manager_id FROM employees2
```

SQL

```
-- OUTPUT
  UNION ALL                 UNION
| manager_id |          | manager_id |
| ---------- |          | ---------- |
| 4          |          | 2          |
| 5          |          | 3          |
| 4          |          | 4          |
| 2          |          | 5          |
| 6          |          | 6          |
| 2          |
| 2          |
| 2          |
| 2          |
| 4          |
| 5          |
| 4          |
| 2          |
| 6          |
| 4          |
| 2          |
| 2          |
| 3          |
```

Swipe to next slide →

## > Q4. Difference between RANK, ROW_NUMBER, and DENSE_RANK



ROW_NUMBER: Assigns a unique sequential number to each row based on the specified order, even if there are ties.
RANK: Assigns a rank to each row based on the specified order; rows with the same value get the same rank, and the next rank skips accordingly.
DENSE_RANK: Similar to RANK, but it does not skip ranks after ties.

SQL

```sql
SELECT employee_id
    ,employee_name
    ,department_id
    ,salary
    ,ROW_NUMBER() OVER (ORDER BY salary DESC) AS row_no
    ,RANK() OVER (ORDER BY salary DESC) AS rank
    ,DENSE_RANK() OVER (ORDER BY salary DESC) AS dense_rank
FROM employees

-- OUTPUT
| employee_id | employee_name | department_id | salary | row_no | rank | dense_rank |
| ----------- | ------------- | ------------- | ------ | ------ | ---- | ---------- |
| 2           | Liam          | 101           | 25000  | 1      | 1    | 1          |
| 3           | Oliver        | 101           | 20000  | 2      | 2    | 2          |
| 1           | Ethan         | 101           | 20000  | 3      | 2    | 2          |
| 5           | Mason         | 202           | 15000  | 4      | 4    | 3          |
| 6           | James         | 202           | 15000  | 5      | 4    | 3          |
| 7           | Lucas         | 202           | 12000  | 6      | 6    | 4          |
| 8           | Henry         | 202           | 8000   | 7      | 7    | 5          |
| 4           | Noah          | 101           | 7000   | 8      | 8    | 6          |
```

Swipe to next slide →

## > Q5. Find Employees who are not present in Department Table.

```sql
-- LEFT JOIN
SELECT e.employee_id
    ,e.employee_name
    ,e.department_id
    ,e.manager_id
    ,d.department_id
FROM employees e
LEFT JOIN department d
ON e.department_id = d.department_id
WHERE d.department_id IS NULL;


-- SUBQUERY Method
SELECT employee_id
    ,employee_name
    ,department_id
    ,manager_id
FROM employees
WHERE department_id NOT IN (SELECT department_id from department);
```

```sql
-- OUTPUT
| employee_id | employee_name | department_id | manager_id | d.department_id |
| ----------- | ------------- | ------------- | ---------- | --------------- |
| 5           | Mason         | 202           | 6          | NULL            |
| 6           | James         | 202           | 2          | NULL            |
| 7           | Lucas         | 202           | 2          | NULL            |
| 8           | Henry         | 202           | 2          | NULL            |
```

Swipe to next slide →

## > Q6. Second highest salary in each Department

SQL

```sql
SELECT *
FROM
    (SELECT *
        ,DENSE_RANK() OVER (PARTITION BY department_id ORDER BY salary) AS Rank
    FROM employees) subquery
WHERE Rank = 2;
```

SQL

```
-- OUTPUT
| employee_id | employee_name | department_id | salary | manager_id | employee_age | Rank |
| ----------- | ------------- | ------------- | ------ | ---------- | ------------ | ---- |
| 1           | Ethan         | 101           | 20000  | 4          | 42           | 2    |
| 3           | Oliver        | 101           | 20000  | 4          | 39           | 2    |
| 7           | Lucas         | 202           | 12000  | 2          | 20           | 2    |
```

Swipe to next slide →

## > Q7. Find all Transactions done by Oliver

```sql
SELECT * FROM orders
WHERE LOWER(customer_name) = 'oliver'
```

```sql
-- OUTPUT
| customer_name | order_date | order_amount | customer_gender |
| ------------- | ---------- | ------------ | --------------- |
| Oliver        | 2024-01-01 | 10000        | Male            |
| oliver        | 2024-01-02 | 12000        | Male            |
| OLIVER        | 2024-01-03 | 14000        | Male            |
```

Swipe to next slide →

## > Q8. SELF JOIN
## - Find Employees and their Manager

```sql
-- Employee Table
| emp_id | emp_name | salary | manager_id |
| ------ | -------- | ------ | ---------- |
| 1      | Jennifer | 10000  | 4          |
| 2      | Liam     | 15000  | 5          |
| 3      | Oliver   | 10000  | 4          |
| 4      | Amanda   | 5000   | 2          |
| 5      | Mason    | 12000  | 6          |
| 6      | James    | 12000  | 2          |
| 7      | Lucas    | 9000   | 2          |
| 8      | Sabrina  | 5000   | 2          |
```

```sql
SELECT e.emp_name
    ,m.emp_name AS manager_name
FROM employees e
JOIN employees m
ON e.manager_id = m.emp_id

-- OUTPUT
| emp_name | manager_name|
| -------- | ------------|
| Amanda   | Liam        |
| James    | Liam        |
| Lucas    | Liam        |
| Sabrina  | Liam        |
| Jennifer | Amanda      |
| Oliver   | Amanda      |
| Liam     | Mason       |
| Mason    | James       |
```

Swipe to next slide →

## > Q8. SELF JOIN

- Find employees whose salary is greater than their Managers Salary

SQL

```sql
SELECT e.emp_id
    ,e.emp_name
    ,m.emp_name AS manager_name
    ,e.salary AS emp_salary
    ,m.salary AS manager_salary
FROM employees e
JOIN employees m
ON e.manager_id = m.emp_id
WHERE e.salary > m.salary

-- OUTPUT
| emp_id | emp_name | emp_name | emp_salary | manager_salary |
| ------ | -------- | -------- | ---------- | -------------- |
| 1      | Jennifer | Amanda   | 10000      | 5000           |
| 3      | Oliver   | Amanda   | 10000      | 5000           |
| 2      | Liam     | Mason    | 15000      | 12000          |
```

Swipe to next slide →

## > Q9. LEFT JOIN - INNER JOIN

```sql
-- Employee Table
| Employee_ID | Name    | Department_ID |
| ----------- | ------- | ------------- |
| 1           | Alice   | 101           |
| 2           | Bob     | 102           |
| 3           | Charlie | 103           |
| 4           | David   | NULL          |

-- Department Table
| Department_ID | Department_Name |
| ------------- | --------------- |
| 101           | HR              |
| 102           | Finance         |
| 104           | IT              |
```

Swipe to next slide →

# > Q9. LEFT JOIN - INNER JOIN

`INNER JOIN:` Combines rows from two tables where there is a **match** based on the specified condition.
If a row does not have a match in both tables, it will be excluded from the result.

SQL

```sql
-- INNER JOIN
SELECT e.Name
     , d.Department_Name
FROM Employees e
JOIN Departments d
ON e.Department_ID = d.Department_ID;

-- OUTPUT
| Name  | Department_Name |
| ----- | --------------- |
| Alice | HR              |
| Bob   | Finance         |
```

Swipe to next slide →

## > Q9. LEFT JOIN - INNER JOIN

LEFT JOIN: Returns all rows from the **left table** (Employees), and the matched rows from the right table (Departments).
If there is no match, the result will contain NULL for columns from the right table.

```sql
-- LEFT JOIN
SELECT e.Name
    ,d.Department_Name
FROM Employees e
LEFT JOIN Departments d
ON e.Department_ID = d.Department_ID;

-- OUTPUT
| Name    | Department_Name |
| ------- | --------------- |
| Alice   | HR              |
| Bob     | Finance         |
| Charlie | NULL            |
| David   | NULL            |
```

Swipe to next slide

## > Q10. Update Records to swap Genders

```sql
SELECT * FROM orders

-- OUTPUT
| customer_name | order_date | order_amount | customer_gender |
| ------------- | ---------- | ------------ | --------------- |
| Sabrina       | 2024-01-01 | 10000        | Male            |
| Henry         | 2024-01-02 | 12000        | Female          |
| Sabrina       | 2024-01-02 | 12000        | Male            |
| James         | 2024-01-03 | 15000        | Female          |
| Sabrina       | 2024-01-03 | 14000        | Male            |
```

```sql
-- Query
UPDATE orders
SET customer_gender =
    CASE
        WHEN customer_gender = 'Female' THEN 'Male'
        WHEN customer_gender = 'Male' THEN 'Female'
    END;

SELECT * FROM orders;

-- OUTPUT
| customer_name | order_date | order_amount | customer_gender |
| ------------- | ---------- | ------------ | --------------- |
| Sabrina       | 2024-01-01 | 10000        | Female          |
| Henry         | 2024-01-02 | 12000        | Male            |
| Sabrina       | 2024-01-02 | 12000        | Female          |
| James         | 2024-01-03 | 15000        | Male            |
| Sabrina       | 2024-01-03 | 14000        | Female          |
```

Swipe to next slide →

## 💡 Ready to Practice?

Try these queries yourself with your own datasets. Need sample data? Ask ChatGPT to generate an insert script and start exploring!

Hands on practice is the key to mastering SQL!

Let's grow together!

By Limesh Mahial