# Top 10 SQL Interview Mistakes and How to Avoid Them

Discover the top 10 SQL mistakes that trip up candidates in interviews! From tricky queries to common misunderstandings, learn how to spot these errors and assess candidates like a pro. Includes real examples, solutions, and expert tips to elevate your interviews!

## Mistake 1: Misunderstanding GROUP BY Behaviour

#### **Question:**

```
SELECT department, COUNT(employee_id), salary
FROM employees
GROUP BY department;
```

#### **Problem:**

Candidates forget that every column in the SELECT clause must either be in the GROUP BY clause or an aggregate function.

#### **Correct Answer:**

```
SELECT department, COUNT(employee_id), AVG(salary) AS average_salary
FROM employees
GROUP BY department;
```

## **Explanation:**

Highlight that salary here should be aggregated (e.g., AVG(salary)), as it's not in the GROUP BY clause. Misunderstanding this concept can lead to errors in report generation.

## Mistake 2: Forgetting to Use NULLAware Functions

## **Question:**

```
sql

SELECT employee_id, (bonus + salary) AS total_compensation
FROM employees;
```

#### **Problem:**

Candidates fail to handle NULL values in numeric calculations, leading to incorrect results.

#### **Correct Answer:**

```
SELECT employee_id, COALESCE(bonus, 0) + salary AS total_compensation FROM employees;
```

## **Explanation:**

Use COALESCE or ISNULL to handle NULL values and ensure calculations are robust.

## Mistake 3: Confusing INNER JOIN with OUTER JOIN

## **Question:**

```
-- Retrieve all departments and their employees

SELECT d.department_name, e.employee_name

FROM departments d

INNER JOIN employees e

ON d.department_id = e.department_id;
```

## **Problem:**

Candidates use INNER JOIN when LEFT JOIN is more appropriate to include all departments, even those without employees.

#### **Correct Answer:**

```
SELECT d.department_name, e.employee_name
FROM departments d
LEFT JOIN employees e
ON d.department_id = e.department_id;
```

## **Explanation:**

Explain the difference between INNER JOIN (matches only rows with common keys) and LEFT JOIN (includes unmatched rows from the left table).

## Mistake 4: Ignoring Index Usage in WHERE Clauses

## **Question:**

```
SELECT * FROM orders WHERE YEAR(order_date) = 2023;
```

#### **Problem:**

Candidates use functions on indexed columns in the WHERE clause, which negates the index.

#### **Correct Answer:**

```
sql

SELECT * FROM orders WHERE order_date >= '2023-01-01' AND order_date < '2024-01-01'
```

## **Explanation:**

Highlight that applying functions (like YEAR) to columns prevents index usage, impacting performance.

# **Mistake 5: Writing Suboptimal NOT IN Queries**

## **Question:**

```
SELECT employee_id
FROM employees
WHERE department_id NOT IN (SELECT department_id FROM departments);
```

#### **Problem:**

If the subquery returns NULL, no rows are returned.

#### **Correct Answer:**

```
SELECT employee_id
FROM employees e
WHERE NOT EXISTS (
SELECT 1 FROM departments d WHERE e.department_id = d.department_id
);
```

## **Explanation:**

Advocate using NOT EXISTS over NOT IN to handle NULL values and improve performance.

## **Mistake 6: Overusing SELECT \* in Queries**

#### **Question:**

```
sql

SELECT * FROM employees WHERE department_id = 10;
```

#### **Problem:**

Candidates retrieve unnecessary columns, leading to inefficiency.

#### **Correct Answer:**

```
SELECT employee_id, employee_name, salary FROM employees WHERE department_id = 10;
```

## **Explanation:**

Discuss the importance of selecting only the required columns to optimize query performance.

## Mistake 7: Confusing HAVING and WHERE

## **Question:**

```
SELECT department_id, COUNT(employee_id)

FROM employees

HAVING COUNT(employee_id) > 5 AND department_id = 10;
```

## **Problem:**

Candidates use HAVING for filtering non-aggregated columns, which should be in the WHERE clause.

#### **Correct Answer:**

```
SELECT department_id, COUNT(employee_id)

FROM employees

WHERE department_id = 10

GROUP BY department_id

HAVING COUNT(employee_id) > 5;
```

# **Explanation:**

Explain that WHERE filters rows before aggregation, whereas HAVING filters after aggregation.

# Mistake 8: Missing Index on JOIN Columns

## **Question:**

```
SELECT e.employee_id, d.department_name

FROM employees e

JOIN departments d

ON e.department_id = d.department_id;
```

#### **Problem:**

Candidates fail to verify that the department\_id column is indexed, leading to slower joins.

## **Best Practice:**

Ensure both employees.department\_id and departments.department\_id are indexed for optimal join performance.

## **Mistake 9: Misusing DISTINCT**

## **Question:**

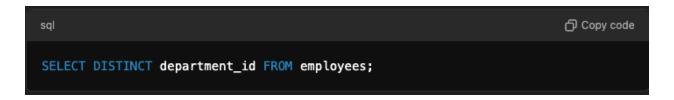
```
SELECT DISTINCT department_id, employee_name
FROM employees;
```

#### **Problem:**

Candidates misunderstand that DISTINCT applies to the entire row, not individual columns.

# **Correct Explanation:**

If only unique department\_id is required:



# Mistake 10: Lack of Understanding of Window Functions

#### **Question:**



#### **Problem:**

Some candidates confuse GROUP BY and window functions.

#### **Correct Answer:**

Explain that window functions (OVER) provide row-wise aggregates without collapsing rows, unlike GROUP BY.

This guide provides you, as an interviewer, with key concepts, practical examples, and explanations to assess and guide candidates effectively. Let me know if you need additional refinements!