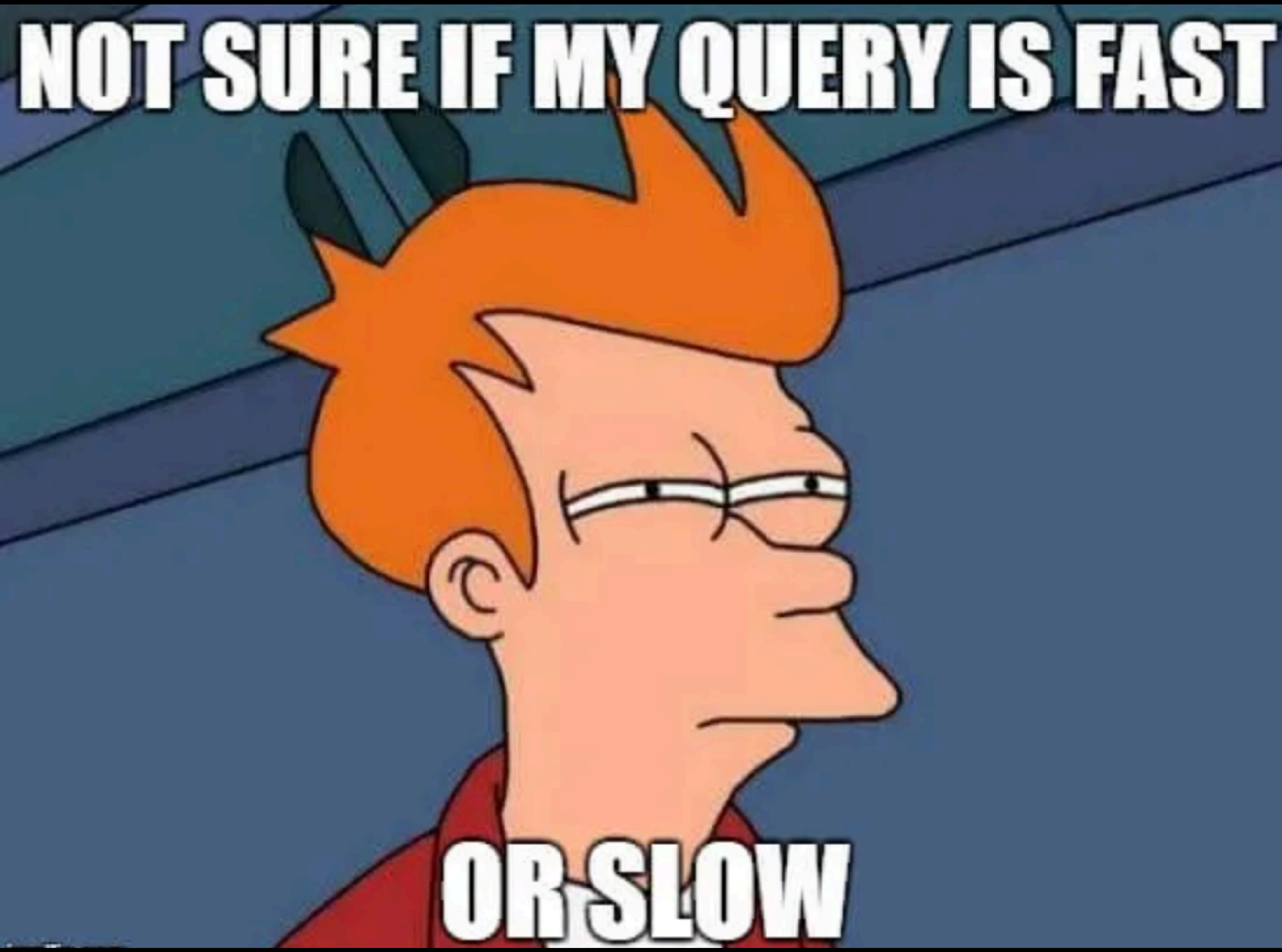
Varun Sagar
Theegala

SQL OPTIMIZATION



CAN BE HELPFUL!

LEARN HOW →

Efficient SQL querying is key to streamlined data processing in organizations.

As a result, there are many **simple optimization practice**s that have been adopted, **vital for enhancing database performance**, **reducing resource load**, and delivering quicker insights.

LET’S DIVE DEEP INTO 10 SIMPLE OPTIMIZATION PRACTICES

→

#1

INDEXING

WHAT IT IS ?

Using indexes to improve the database query performance.

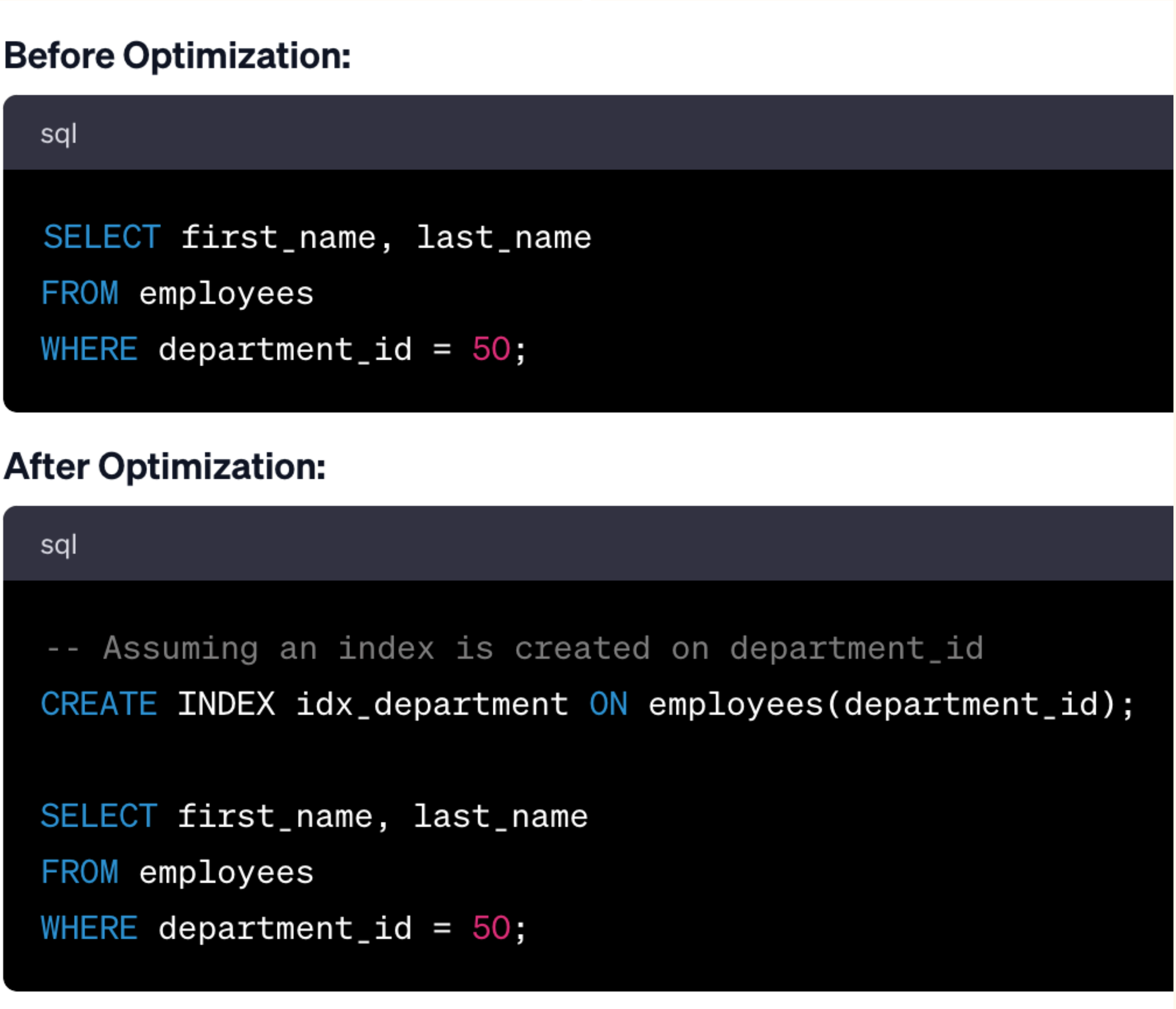
WHY IT MATTERS ?

Indexes speed up the retrieval of rows from a table, making queries more efficient.

HOW TO OPTIMIZE ?

Identify frequently queried columns and create indexes on those columns.

LET'S WORK WITH AN EXAMPLE



Before Optimization:

```sql
SELECT first_name, last_name
FROM employees
WHERE department_id = 50;
```

After Optimization:

```sql
-- Assuming an index is created on department_id
CREATE INDEX idx_department ON employees(department_id);

SELECT first_name, last_name
FROM employees
WHERE department_id = 50;
```

Making department_id an index column will impact query speed

#2

Avoiding SELECT *

WHAT IT IS ?

Specifying needed columns instead of using SELECT *

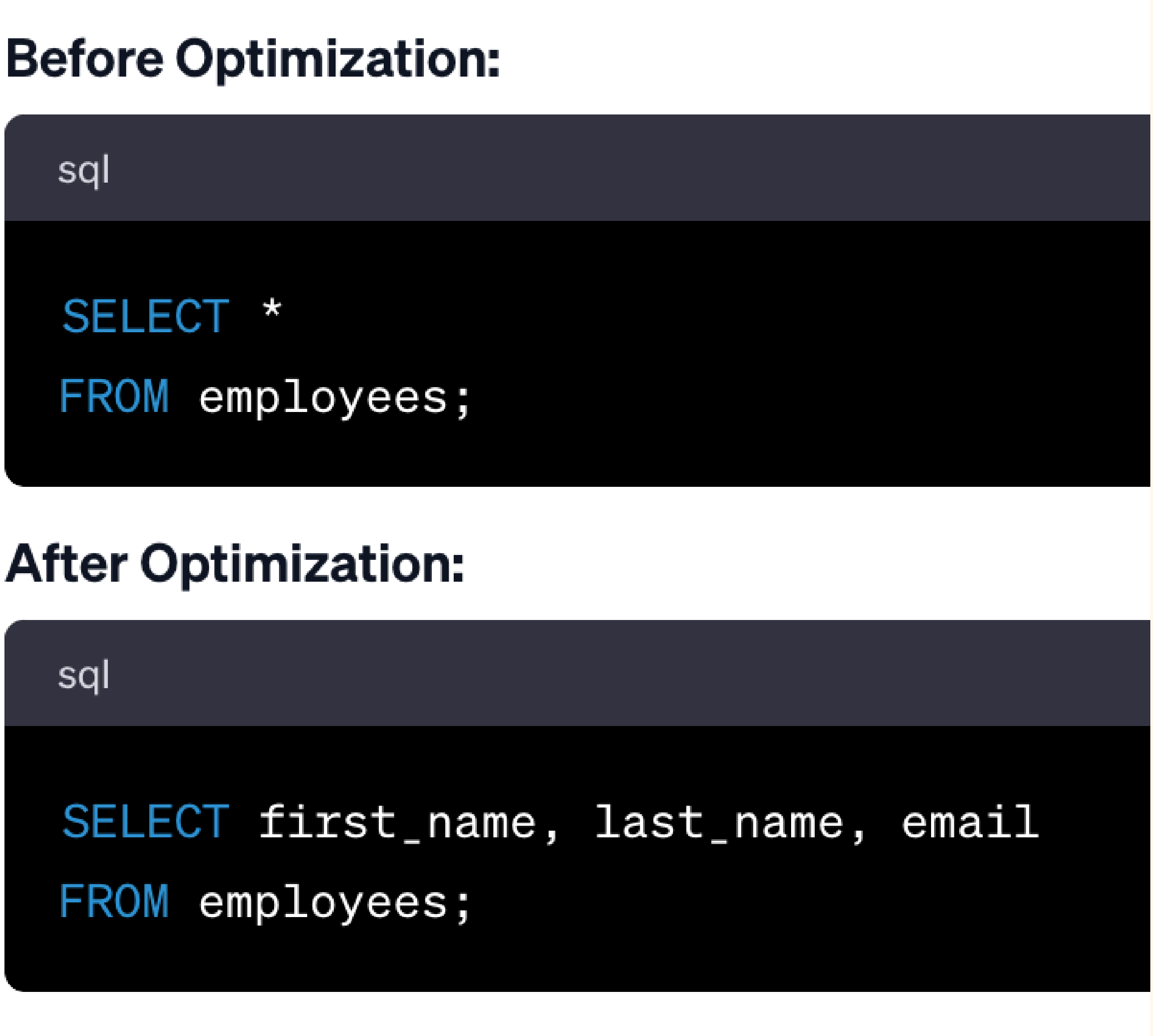
WHY IT MATTERS ?

Increases query efficiency by reducing data load and processing time.

HOW TO OPTIMIZE ?

Analyze the data requirements and explicitly list only the necessary columns in your SELECT statement.

LET'S WORK WITH AN EXAMPLE



Before Optimization:

sql

```sql
SELECT *
FROM employees;
```

After Optimization:

sql

```sql
SELECT first_name, last_name, email
FROM employees;
```

#3

Use Joins Instead of Sub-queries

WHAT IT IS ?

Replacing sub-queries with joins to enhance performance.

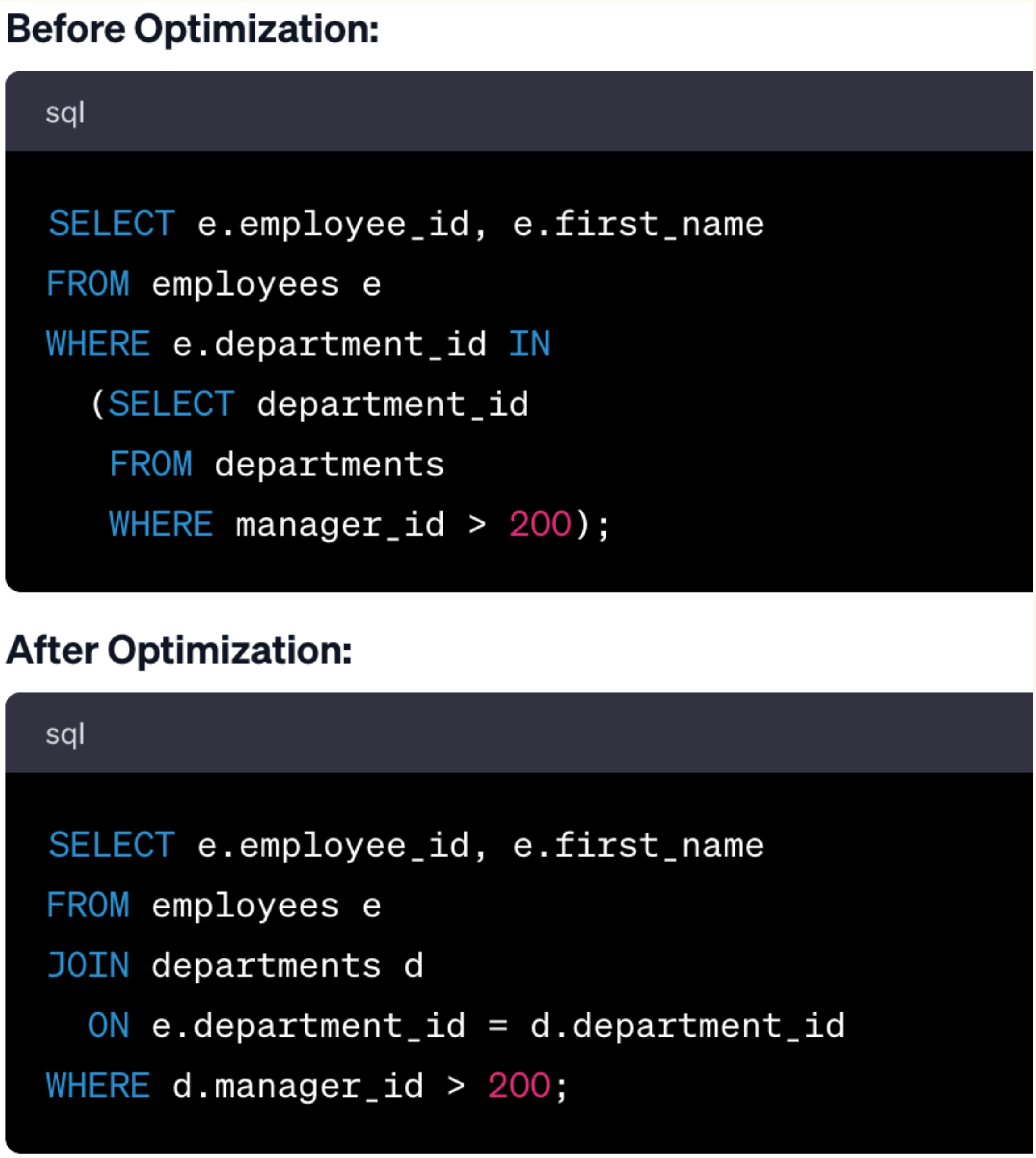
WHY IT MATTERS ?

Joins are generally more efficient and faster than nested sub-queries.

HOW TO OPTIMIZE ?

Convert sub-queries into JOIN operations where possible to ensure the most efficient type of JOIN for your data.

LET’S WORK WITH AN EXAMPLE



Before Optimization:

sql

```sql
SELECT e.employee_id, e.first_name
FROM employees e
WHERE e.department_id IN
  (SELECT department_id
   FROM departments
   WHERE manager_id > 200);
```

After Optimization:

sql

```sql
SELECT e.employee_id, e.first_name
FROM employees e
JOIN departments d
  ON e.department_id = d.department_id
WHERE d.manager_id > 200;
```

#4

Proper Use Of WHERE Clauses

WHAT IT IS ?

Efficiently filtering data using WHERE clauses.

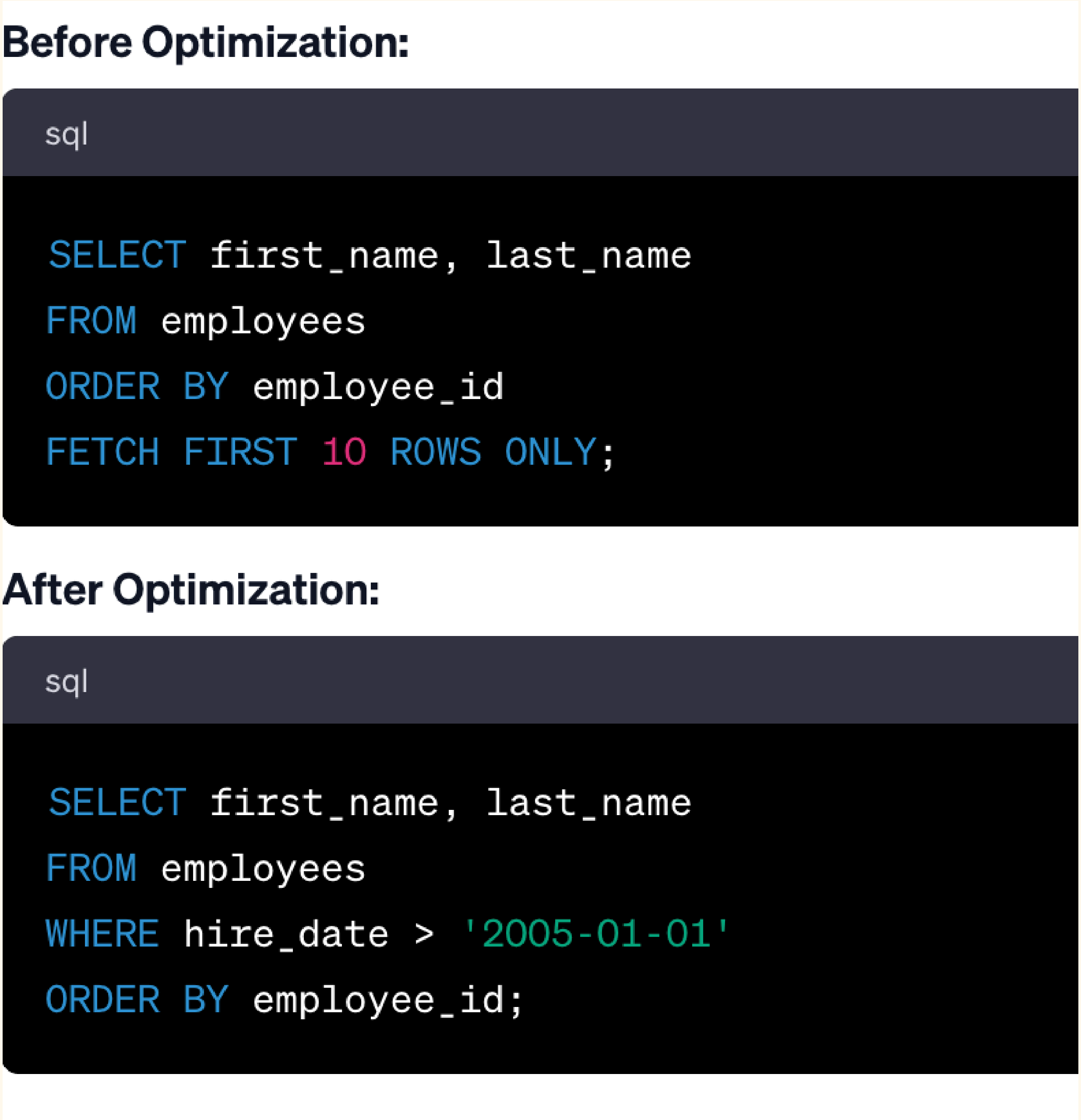
WHY IT MATTERS ?

Reduces the amount of data processed, improving query speed.

HOW TO OPTIMIZE ?

Utilize WHERE clauses to filter data as early as possible in the query process.

LET’S WORK WITH AN EXAMPLE



Before Optimization:

sql

```sql
SELECT first_name, last_name
FROM employees
ORDER BY employee_id
FETCH FIRST 10 ROWS ONLY;
```

After Optimization:

sql

```sql
SELECT first_name, last_name
FROM employees
WHERE hire_date > '2005-01-01'
ORDER BY employee_id;
```

NEXT

#5

Limiting Result Set

WHAT IT IS ?

Using clauses like LIMIT or FETCH to restrict the number of rows returned.

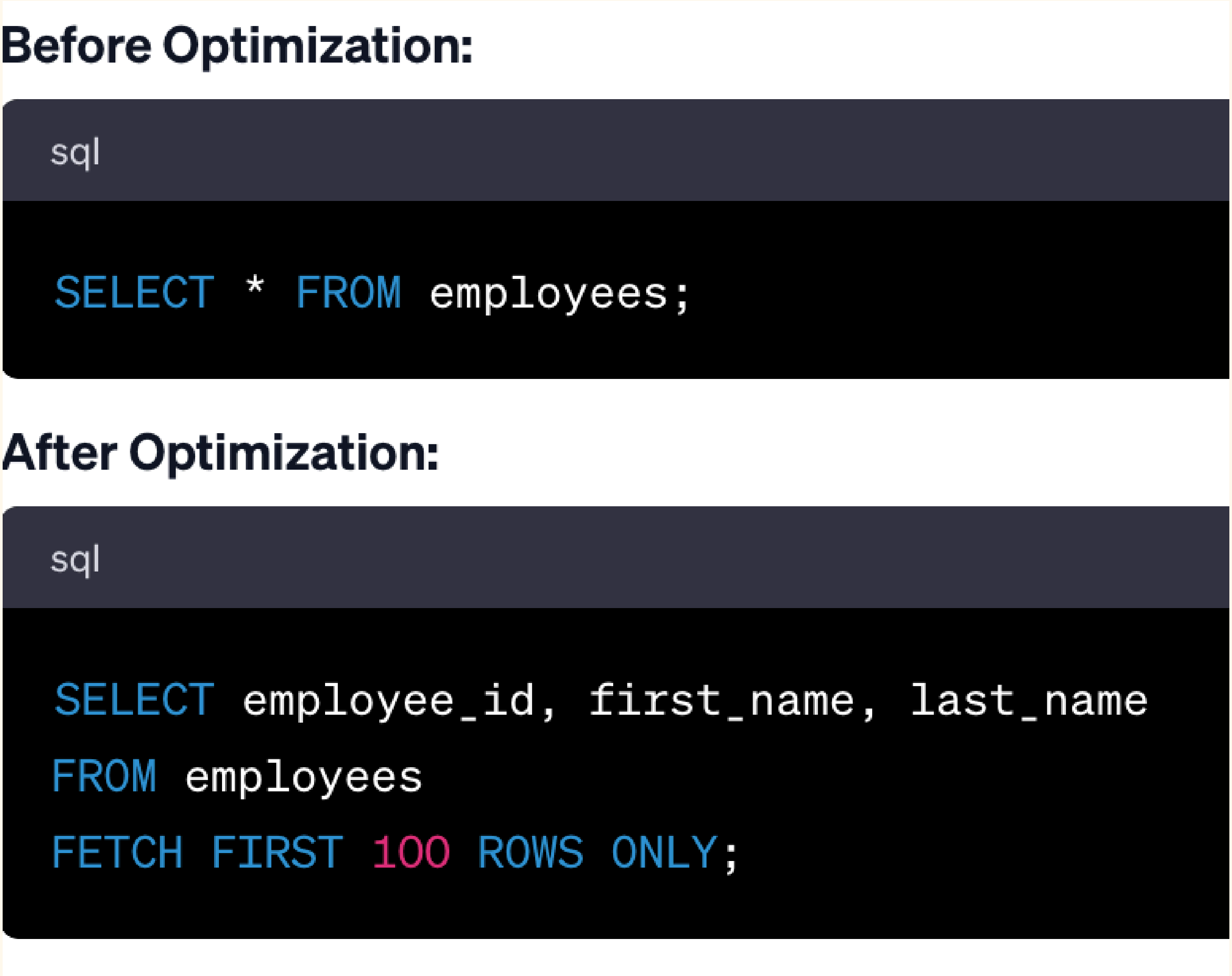
WHY IT MATTERS ?

Prevents over-fetching of data, saving resources and time.

HOW TO OPTIMISE ?

Implement LIMIT or FETCH FIRST clauses in queries where the full dataset is not required.

LET’S WORK WITH AN EXAMPLE



Before Optimization:

sql

```sql
SELECT * FROM employees;
```

After Optimization:

sql

```sql
SELECT employee_id, first_name, last_name
FROM employees
FETCH FIRST 100 ROWS ONLY;
```

As shown, entering column names to pull would reduce the response time

Varun Sagar
Theegala

NEXT ➡

#6

Optimize GROUP BY & HAVING Clauses

WHAT IT IS ?

Efficiently grouping data and filtering large groups.

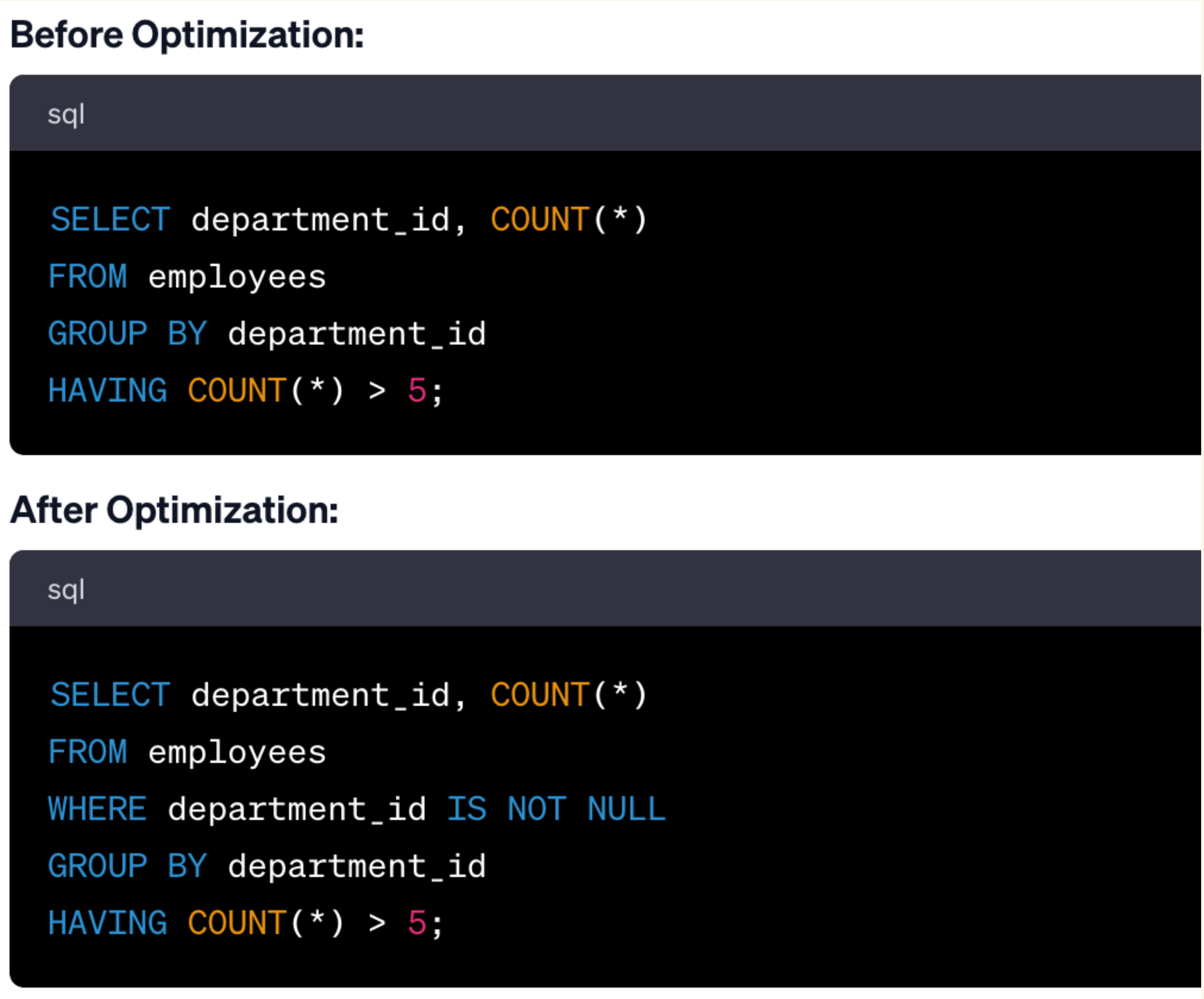
WHY IT MATTERS ?

Improves performance, especially in large datasets.

HOW TO OPTIMIZE ?

Ensure the use of GROUP BY and HAVING clauses is done in a way that minimizes the amount of data being grouped.

LET'S WORK WITH AN EXAMPLE



Before Optimization:

sql

```sql
SELECT department_id, COUNT(*)
FROM employees
GROUP BY department_id
HAVING COUNT(*) > 5;
```

After Optimization:

sql

```sql
SELECT department_id, COUNT(*)
FROM employees
WHERE department_id IS NOT NULL
GROUP BY department_id
HAVING COUNT(*) > 5;
```

As shown, removing records with no department_id reduce the data size

Varun Sagar Theegala

NEXT

#7

Use Temporary Tables Wisely

WHAT IT IS ?

Strategic use of temporary tables for complex queries.

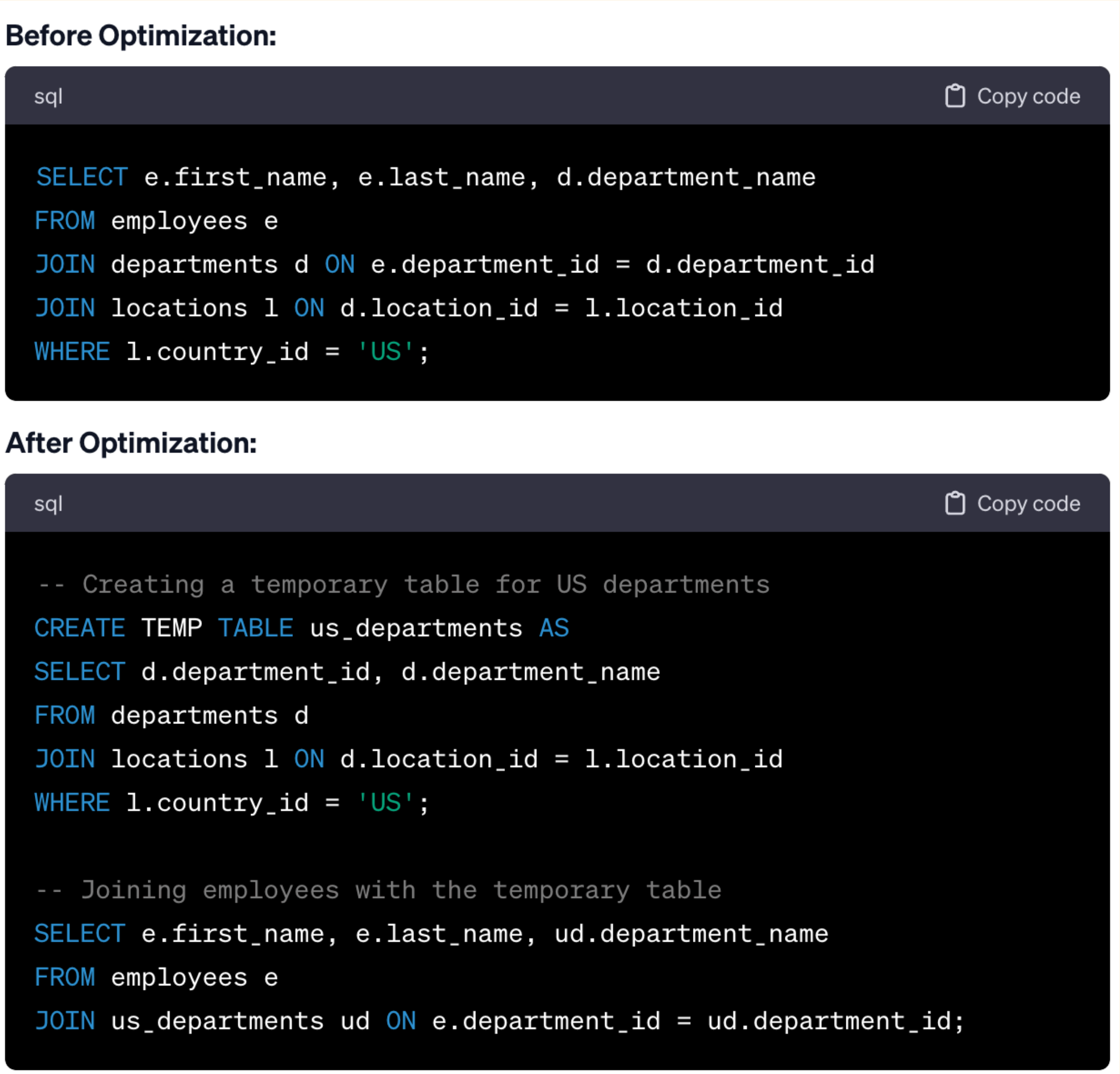
WHY IT MATTERS ?

Can simplify queries and improve performance in multi-step processes.

HOW TO OPTIMIZE ?

Use temporary tables to store intermediate results, especially when dealing with multiple complex joins or subqueries.

LET'S WORK WITH AN EXAMPLE



Before Optimization:

```sql
SELECT e.first_name, e.last_name, d.department_name
FROM employees e
JOIN departments d ON e.department_id = d.department_id
JOIN locations l ON d.location_id = l.location_id
WHERE l.country_id = 'US';
```

After Optimization:

```sql
-- Creating a temporary table for US departments
CREATE TEMP TABLE us_departments AS
SELECT d.department_id, d.department_name
FROM departments d
JOIN locations l ON d.location_id = l.location_id
WHERE l.country_id = 'US';

-- Joining employees with the temporary table
SELECT e.first_name, e.last_name, ud.department_name
FROM employees e
JOIN us_departments ud ON e.department_id = ud.department_id;
```

#8

Avoid or Optimize OR Clauses

WHAT IT IS ?

Replacing or optimizing OR clauses for efficiency.

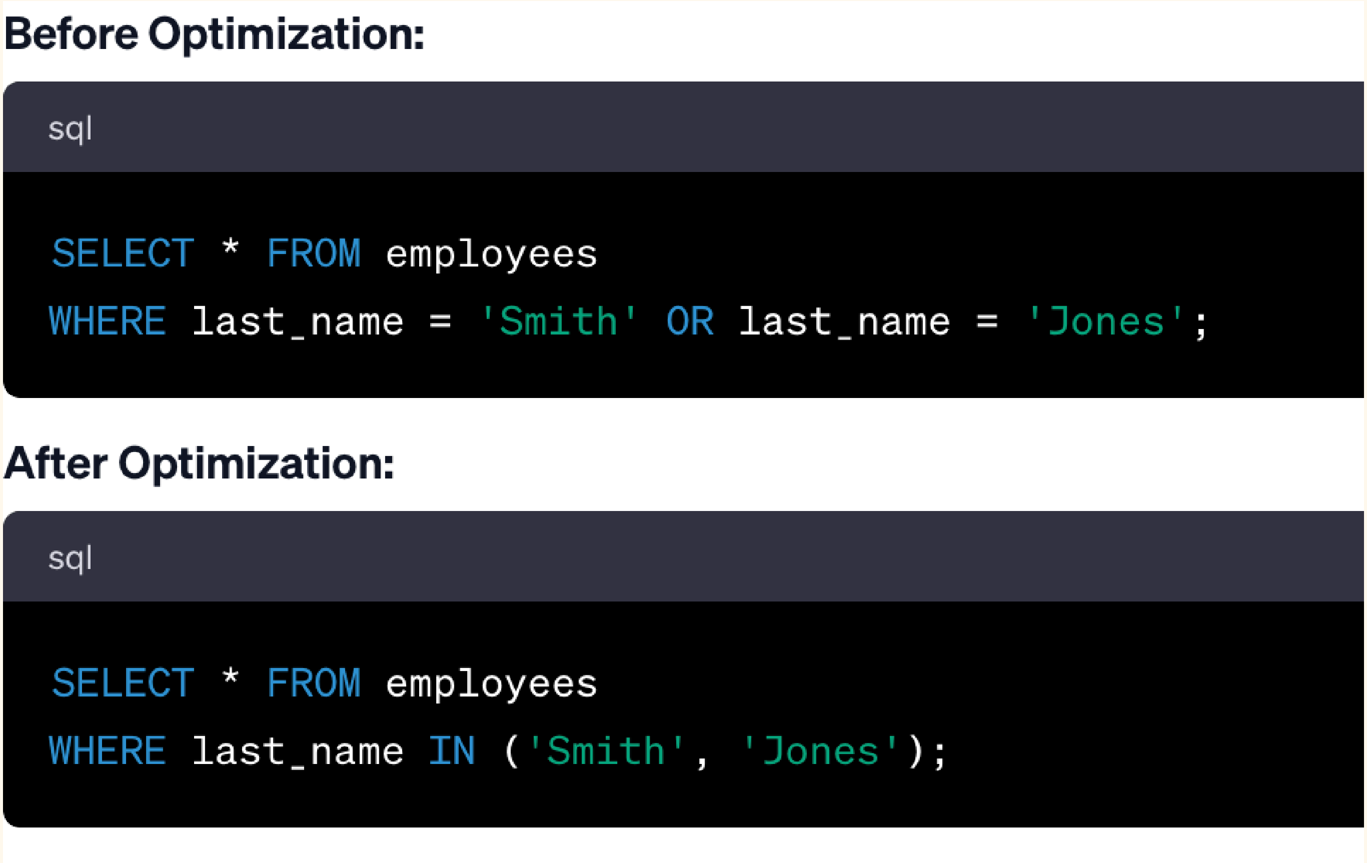
WHY IT MATTERS ?

OR clauses can slow down queries; alternatives can improve performance.

HOW TO OPTIMIZE ?

Replace OR clauses with IN statements where possible, or break the query into multiple UNIONed queries.

LET'S WORK WITH AN EXAMPLE



Before Optimization:

sql

```sql
SELECT * FROM employees
WHERE last_name = 'Smith' OR last_name = 'Jones';
```

After Optimization:

sql

```sql
SELECT * FROM employees
WHERE last_name IN ('Smith', 'Jones');
```

As shown, replacing the OR Clause with IN would impact query speed.

#9

Use EXISTS Instead of IN

WHAT IT IS ?

Using EXISTS for subquery checks instead of IN.

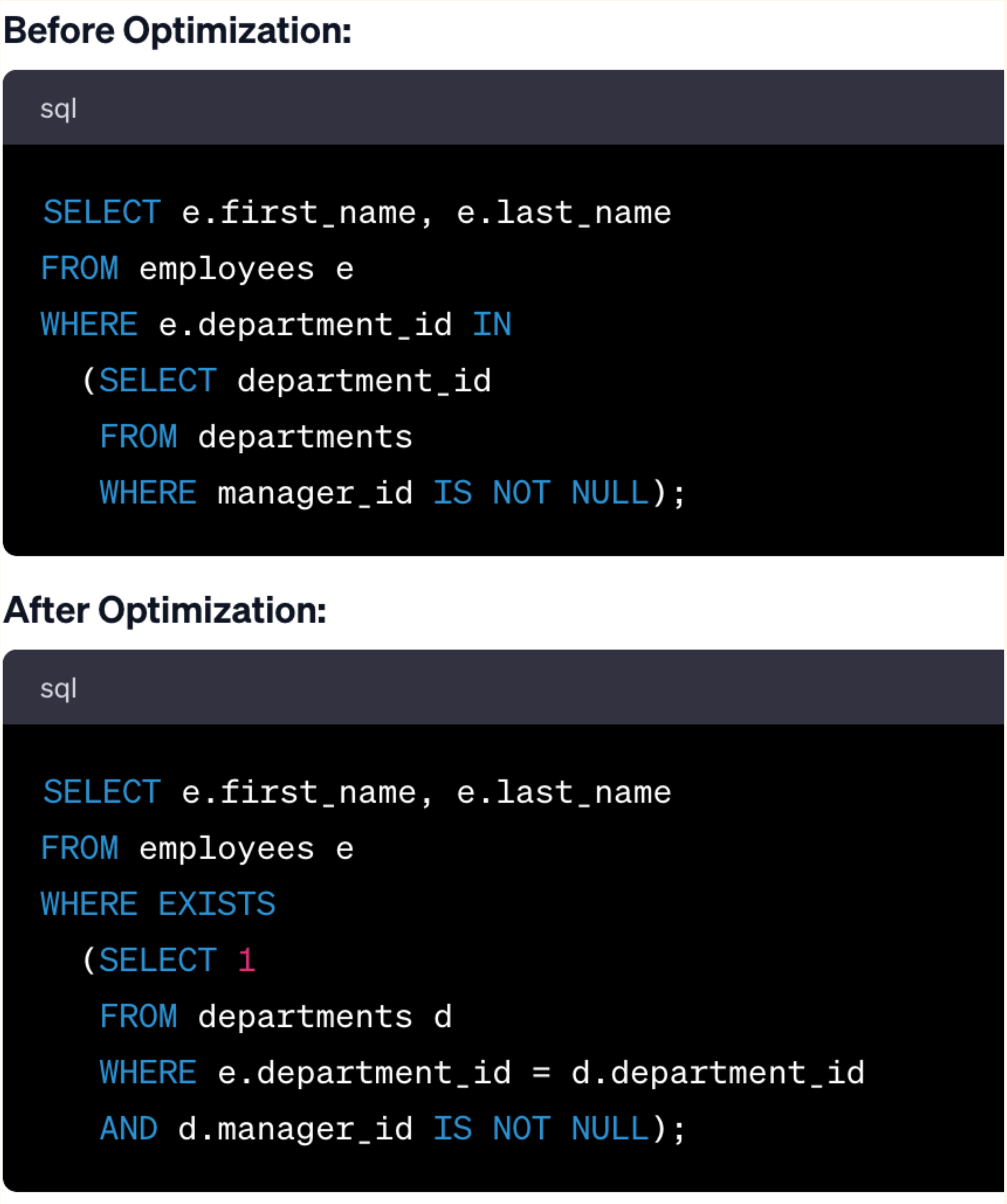
WHY IT MATTERS ?

EXISTS can be faster, especially with large subquery results.

HOW TO OPTIMIZE ?

Replace IN clauses with EXISTS when checking for the existence of a row in a subquery.

LET'S WORK WITH AN EXAMPLE



Before Optimization:

```sql
SELECT e.first_name, e.last_name
FROM employees e
WHERE e.department_id IN
  (SELECT department_id
   FROM departments
   WHERE manager_id IS NOT NULL);
```

After Optimization:

```sql
SELECT e.first_name, e.last_name
FROM employees e
WHERE EXISTS
  (SELECT 1
   FROM departments d
   WHERE e.department_id = d.department_id
   AND d.manager_id IS NOT NULL);
```

#10
Optimize Join Orders

WHAT IT IS ?

Ordering joins to process smaller datasets first.

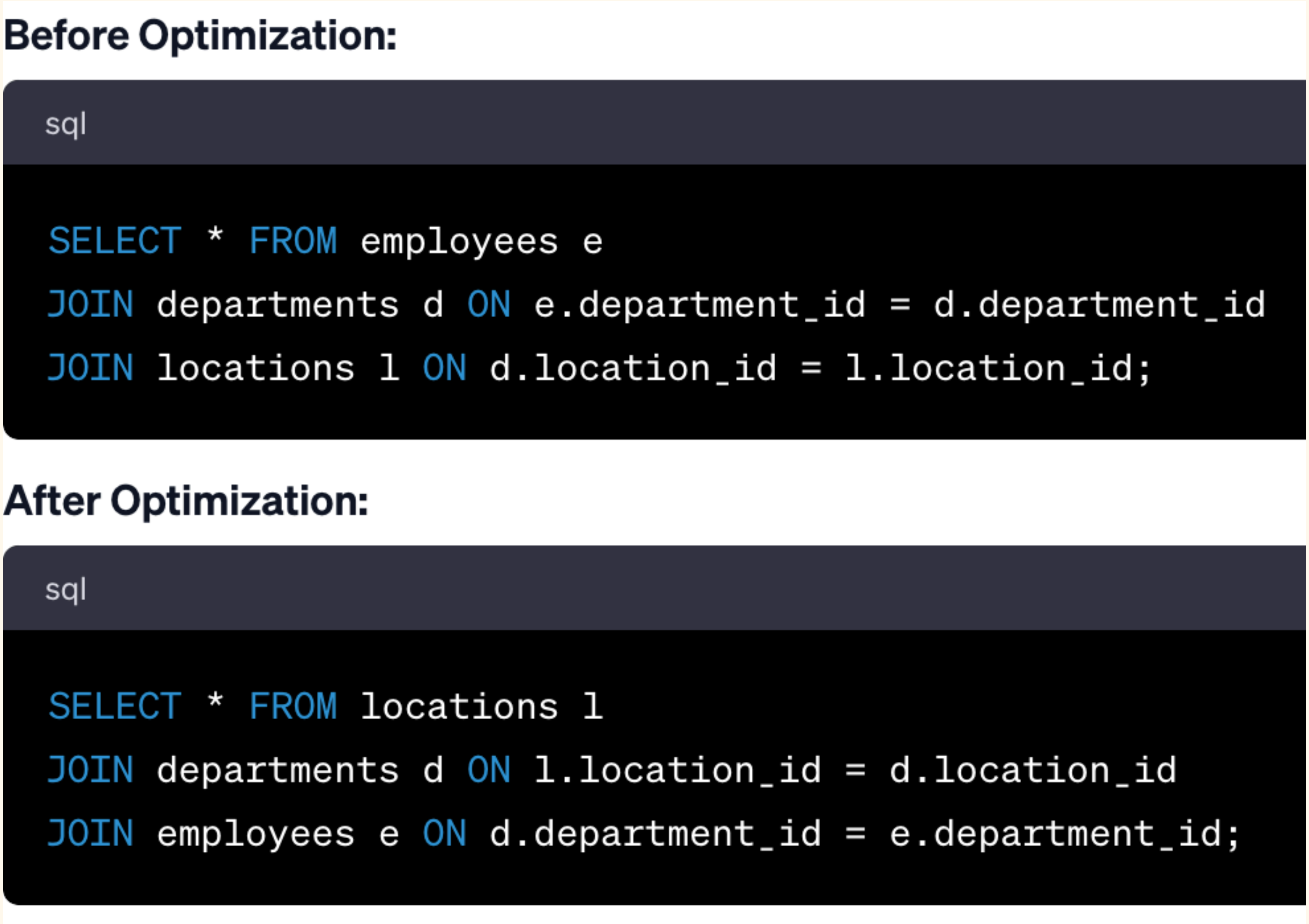
WHY IT MATTERS ?

Reduces overall query processing time by minimizing intermediate result size.

HOW TO OPTIMIZE ?

Analyze the size of the tables involved in joins and structure the query to start with the smallest table, gradually joining larger tables.

LET'S WORK WITH AN EXAMPLE



Before Optimization:

sql

```sql
SELECT * FROM employees e
JOIN departments d ON e.department_id = d.department_id
JOIN locations l ON d.location_id = l.location_id;
```

After Optimization:

sql

```sql
SELECT * FROM locations l
JOIN departments d ON l.location_id = d.location_id
JOIN employees e ON d.department_id = e.department_id;
```

By replacing employees with a smaller dataset like locations as the base, we reduce intermediate table sizes

REMEMBER

Mastering these SQL optimization techniques is essential for data professionals aiming to **elevate data handling efficiency**.

Implementing these practices ensures your organization's data workflows are both **robust and responsive**.

Varun Sagar
Theegala

SHARE THIS

If you think **your network** would find this **valuable**

FOLLOW ME

I help you **GROW & SUSTAIN** as a **Data Analyst**