SQL Query Optimization

Anish Chakravorty
Follow me on LinkedIn

What is Query Optimization?

Query Optimization involves rewriting SQL queries to run faster and use fewer resources.

Key Benefits:

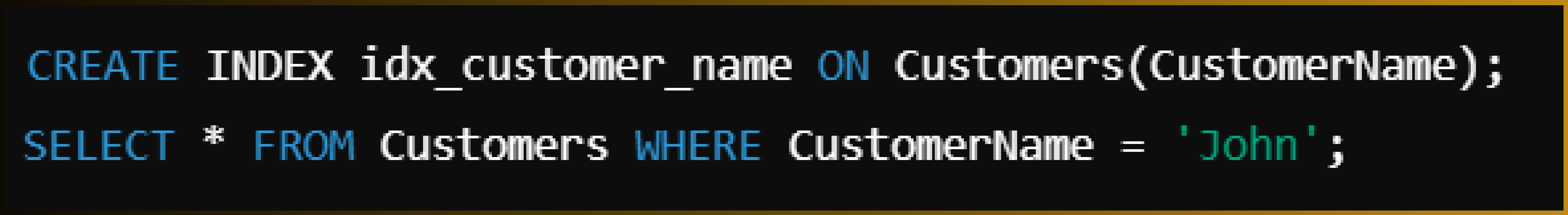
- ⏱️ Faster Execution
- 🔄 Reduced Resource Usage
- 📈 Improved Scalability

→

Use Indexes Wisely

- Use indexes on frequently searched or filtered columns.
- Avoid over-indexing, as it can slow down write operations.

Example:



```sql
CREATE INDEX idx_customer_name ON Customers(CustomerName);
SELECT * FROM Customers WHERE CustomerName = 'John';
```

Minimize SELECT * Usage

- Fetching unnecessary columns wastes resources.
- Instead, specify only the required columns.

Example:

Bad:

```sql
SELECT * FROM Orders;
```

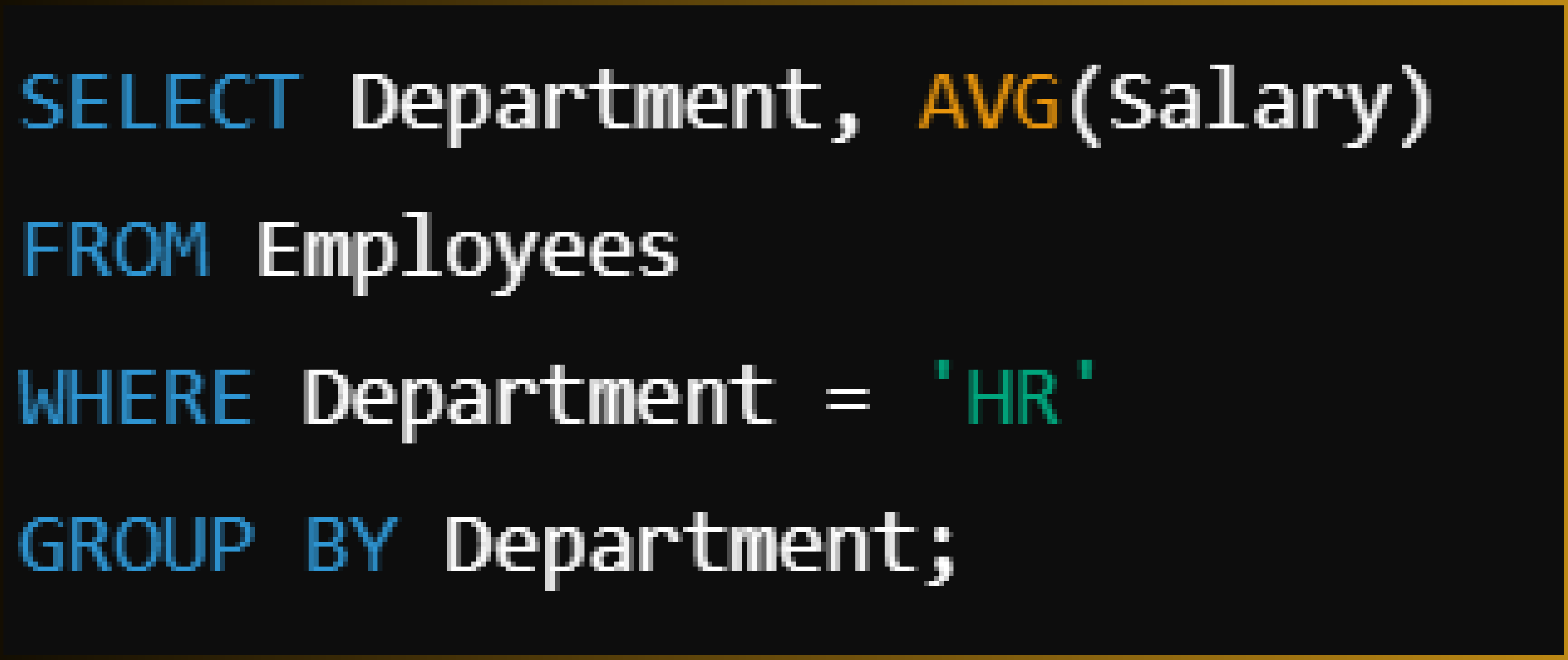
Good:

```sql
SELECT OrderID, OrderDate FROM Orders;
```

Pro Tip: This reduces memory usage and improves query speed.

Use WHERE Instead of HAVING

- Use WHERE to filter rows before aggregation.
- Reserve HAVING for filtering aggregated results.

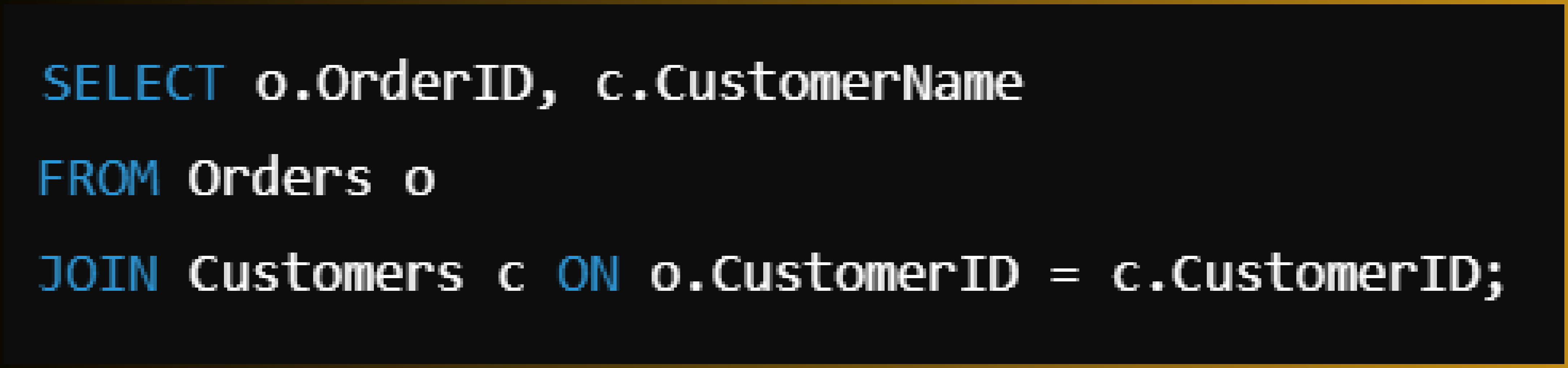


```sql
SELECT Department, AVG(Salary)
FROM Employees
WHERE Department = 'HR'
GROUP BY Department;
```

Pro Tip: Filtering early reduces the dataset size and improves performance.

Leverage Joins Effectively

- Use INNER JOIN for specific matching rows.
- Avoid CROSS JOIN unless needed.
- Index join columns for faster lookups.



```sql
SELECT o.OrderID, c.CustomerName
FROM Orders o
JOIN Customers c ON o.CustomerID = c.CustomerID;
```

Pro Tip: Always use ON conditions in joins to avoid Cartesian products.

Avoid Subqueries When Possible

Subqueries can slow down queries; use joins when possible for better performance.

Example:

Bad:

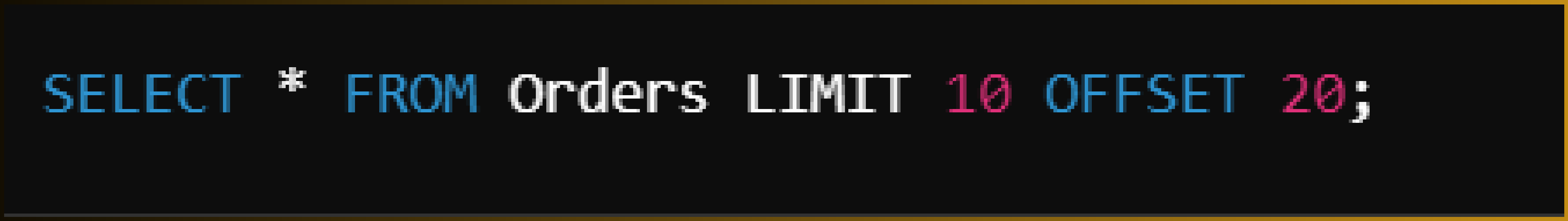
```sql
SELECT Name FROM Employees WHERE Salary > (SELECT AVG(Salary) FROM Employees);
```

Good:

```sql
WITH AvgSalary AS (SELECT AVG(Salary) AS AvgSal FROM Employees)
SELECT Name FROM Employees, AvgSalary WHERE Employees.Salary > AvgSalary.AvgSal;
```

Limit Results with Pagination

- Fetching millions of rows is resource-intensive.
- Instead, paginate results using LIMIT and OFFSET.



```
SELECT * FROM Orders LIMIT 10 OFFSET 20;
```

Pro Tip: Always fetch data in manageable chunks.

Analyze Query Performance

- Execution plans show how the database executes your query.
- Identify slow parts of your query.
- Adjust indexing, joins, or filters based on insights.

```
EXPLAIN SELECT * FROM Orders WHERE OrderDate > '2024-01-01';
```

Pro Tip: Tools like MySQL Workbench or SQL Server Management Studio make analyzing plans easier.

Key Takeaways

Optimize Your Queries Like a Pro

- Use indexes strategically.
- Avoid fetching unnecessary data.
- Analyze performance using execution plans.
- Write clean, structured queries.

Follow me to get more **Information** and content like this.

Anish Chakravorty

Follow me on LinkedIn