Optimize Your SELECT Statements for Faster Queries!

SELECT statements retrieve data from databases, but poorly optimized queries can slow things down.

1 Avoid SELECT * in Queries

- Be specific about the columns you need.
- Because SELECT * loads all columns, slowing down performance.

```
SELECT OrderID , OrderDate, CustomerName
             , Sales, Profit
FROM orders
```

2 Use Indexes on Required Columns

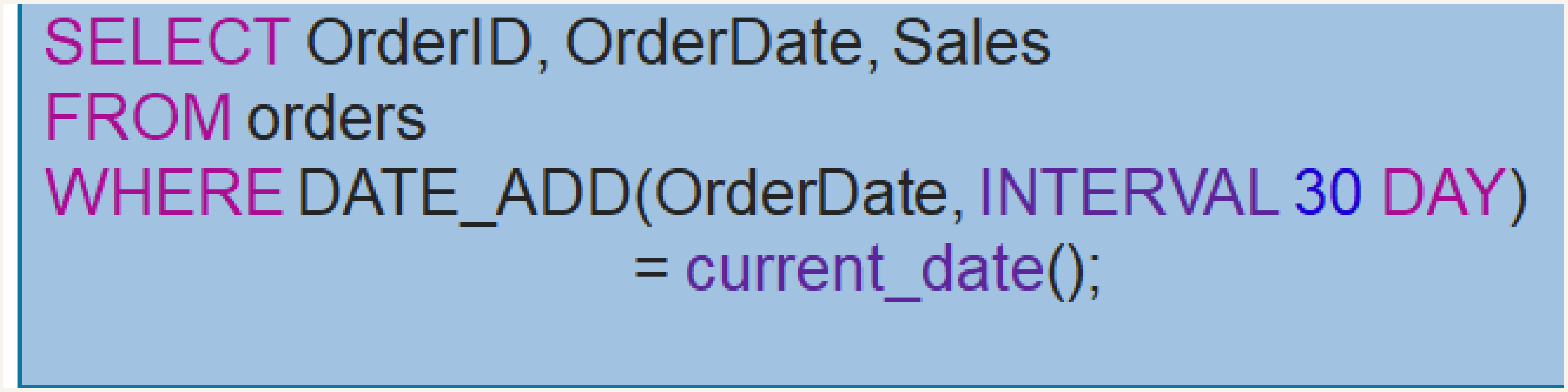
- Indexes speed up data retrieval by limiting rows scanned.
- Focus on columns frequently used in WHERE, JOIN, or ORDER BY.

```
CREATE INDEX IdxCustomerName ON
orders(CustomerName);
```

3 Avoid Functions in WHERE Clause

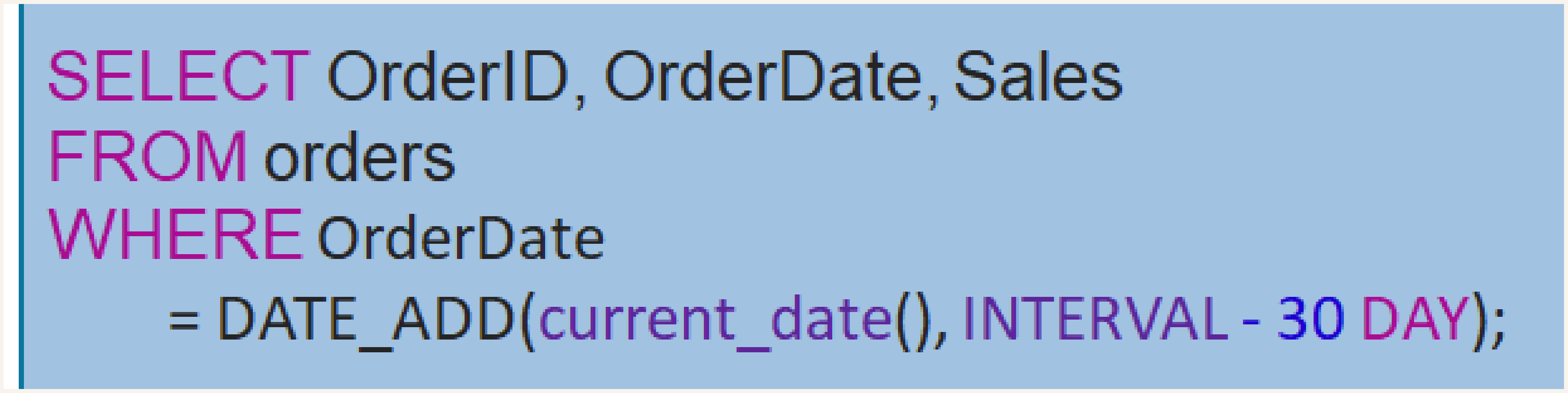
Problem: Functions in WHERE clauses prevent indexes from being used.

Non-optimized Query:



```
SELECT OrderID, OrderDate, Sales
FROM orders
WHERE DATE_ADD(OrderDate, INTERVAL 30 DAY)
                        = current_date();
```

Optimized Query:



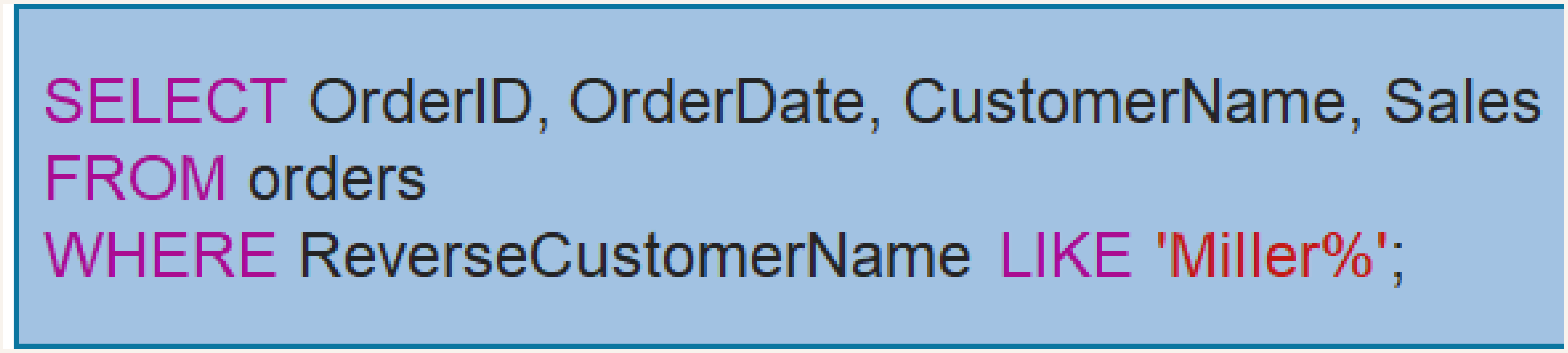
```
SELECT OrderID, OrderDate, Sales
FROM orders
WHERE OrderDate
      = DATE_ADD(current_date(), INTERVAL - 30 DAY);
```

@DATASAPIENT

4

Avoid Leading Wildcards in LIKE

- Problem: % at the start of a string forces a full scan.
- Solution: Use trailing wildcards or reverse column design.

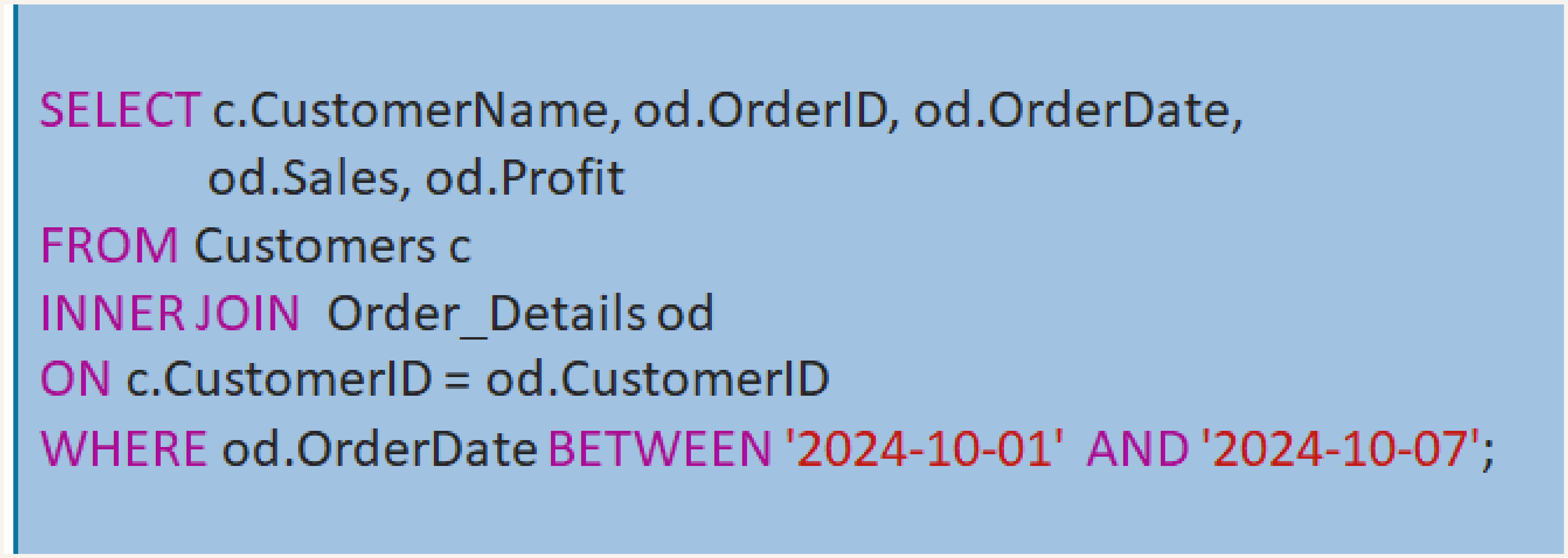


```
SELECT OrderID, OrderDate, CustomerName, Sales
FROM orders
WHERE ReverseCustomerName  LIKE 'Miller%';
```

5

Use INNER JOIN for Efficiency

- INNER JOIN is faster than OUTER JOIN because it only returns matching rows.



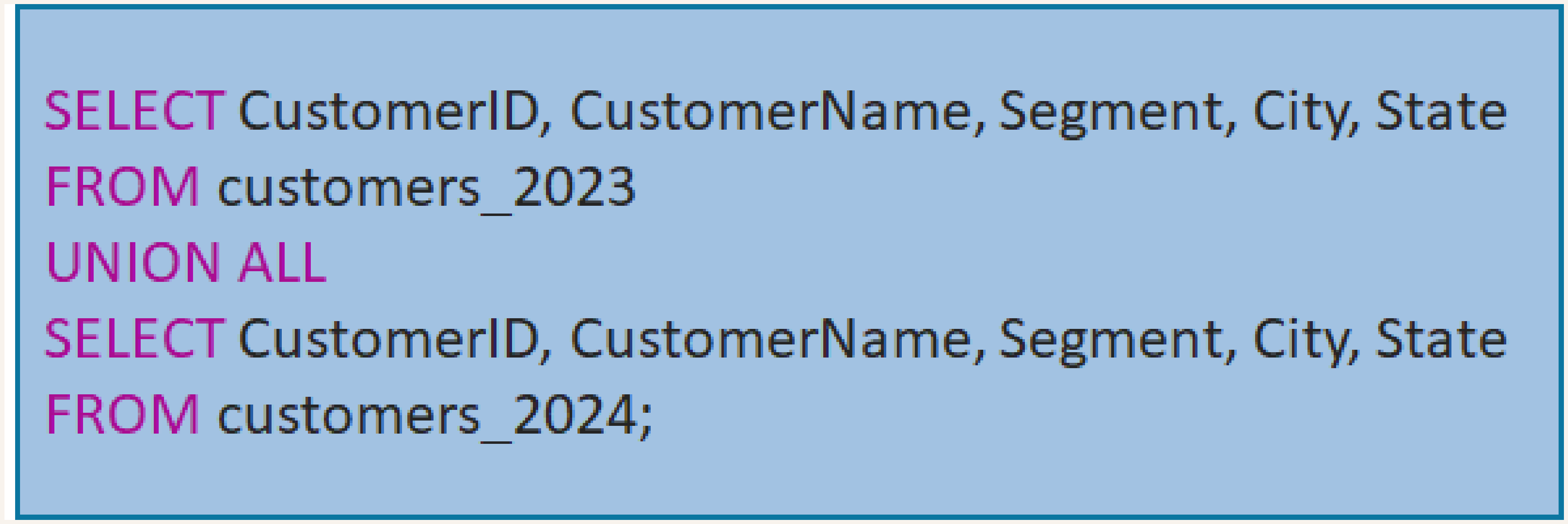
```
SELECT c.CustomerName, od.OrderID, od.OrderDate,
       od.Sales, od.Profit
FROM Customers c
INNER JOIN  Order_Details od
ON c.CustomerID = od.CustomerID
WHERE od.OrderDate BETWEEN '2024-10-01'  AND '2024-10-07';
```

@DATASAPIENT

6

Use UNION ALL instead of UNION

- Why?: UNION sorts and removes duplicates, slowing performance.
- Solution: Use UNION ALL to avoid this.

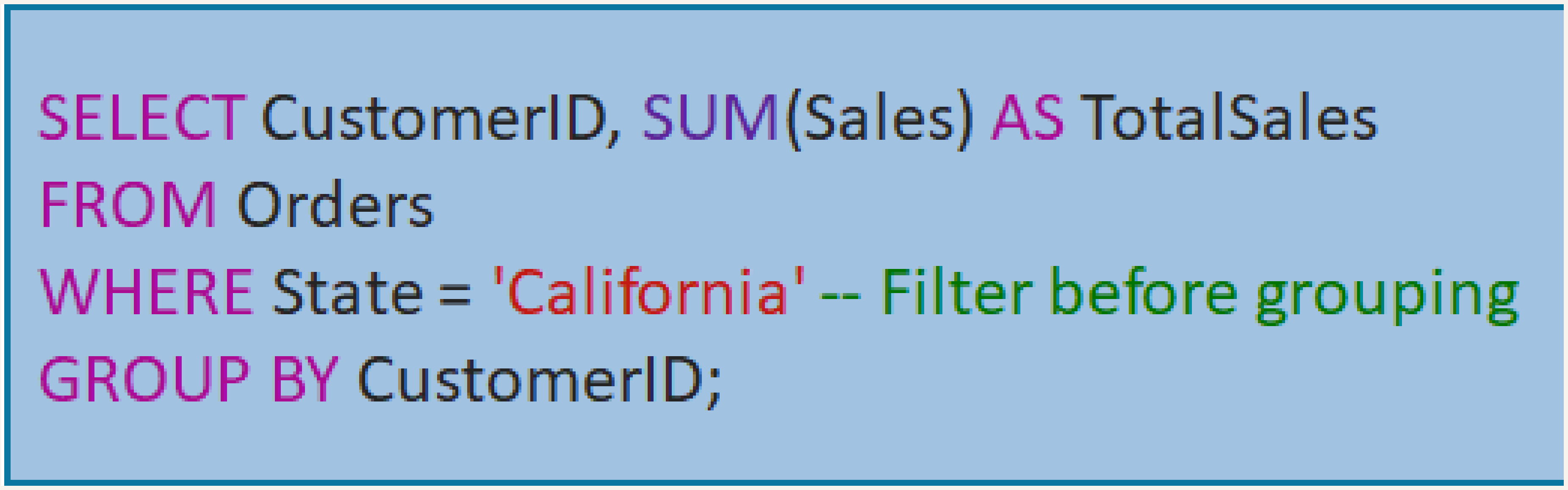


```
SELECT CustomerID, CustomerName, Segment, City, State
FROM customers_2023
UNION ALL
SELECT CustomerID, CustomerName, Segment, City, State
FROM customers_2024;
```

@DATASAPIENT

7 Apply Filters Early in Queries

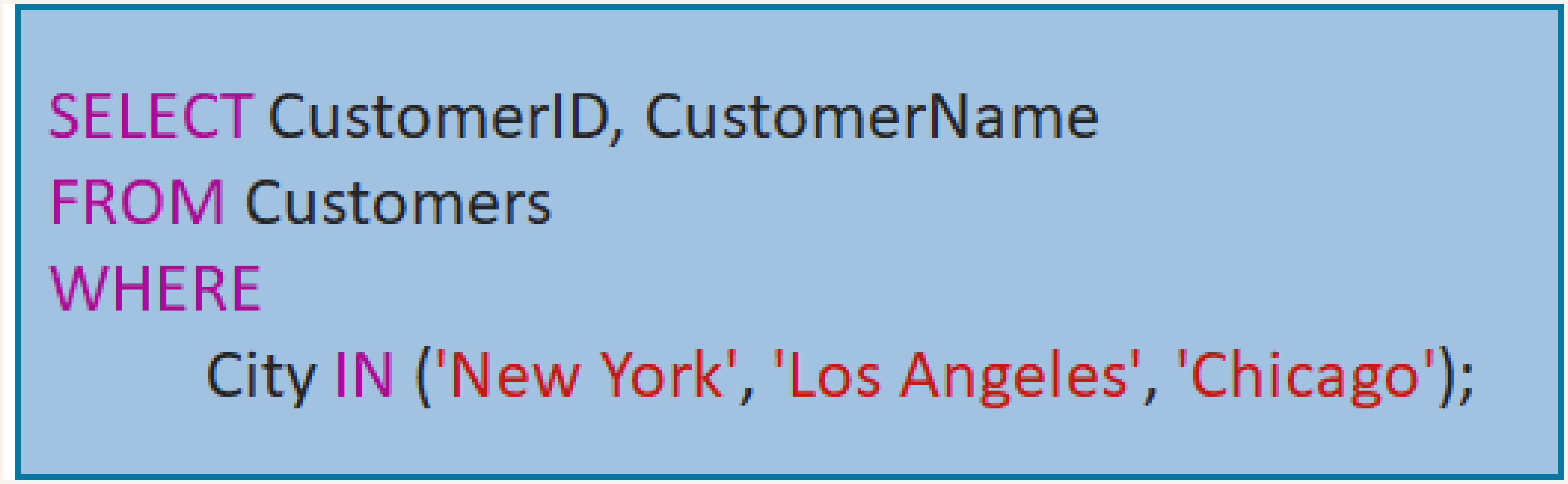
- Filter records before grouping to reduce the workload.



```
SELECT CustomerID, SUM(Sales) AS TotalSales
FROM Orders
WHERE State = 'California' -- Filter before grouping
GROUP BY CustomerID;
```

8 Replace OR with IN for Better Performance

- Problem: Multiple OR clauses slow down query execution.
- Solution: Use the IN operator for faster performance.



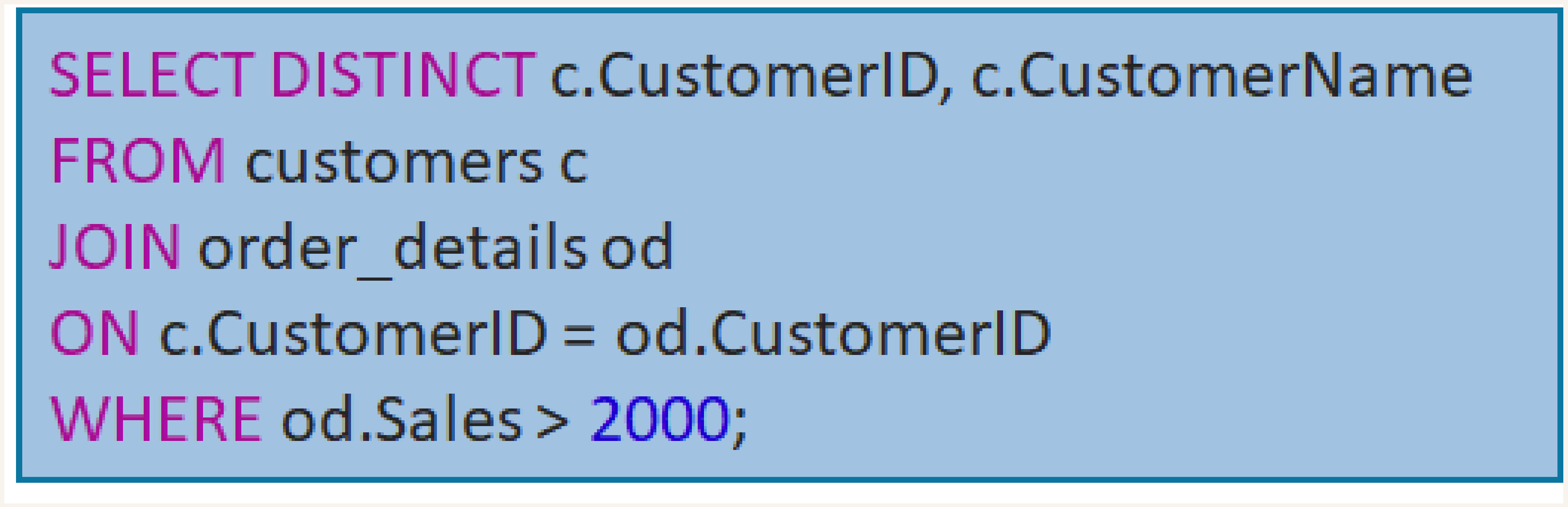
```
SELECT CustomerID, CustomerName
FROM Customers
WHERE
        City IN ('New York', 'Los Angeles', 'Chicago');
```

@DATASAPIENT

9

Avoid Subqueries for Performance

- Subqueries can cause multiple table scans, leading to inefficiency.
- Use JOIN instead for better performance.

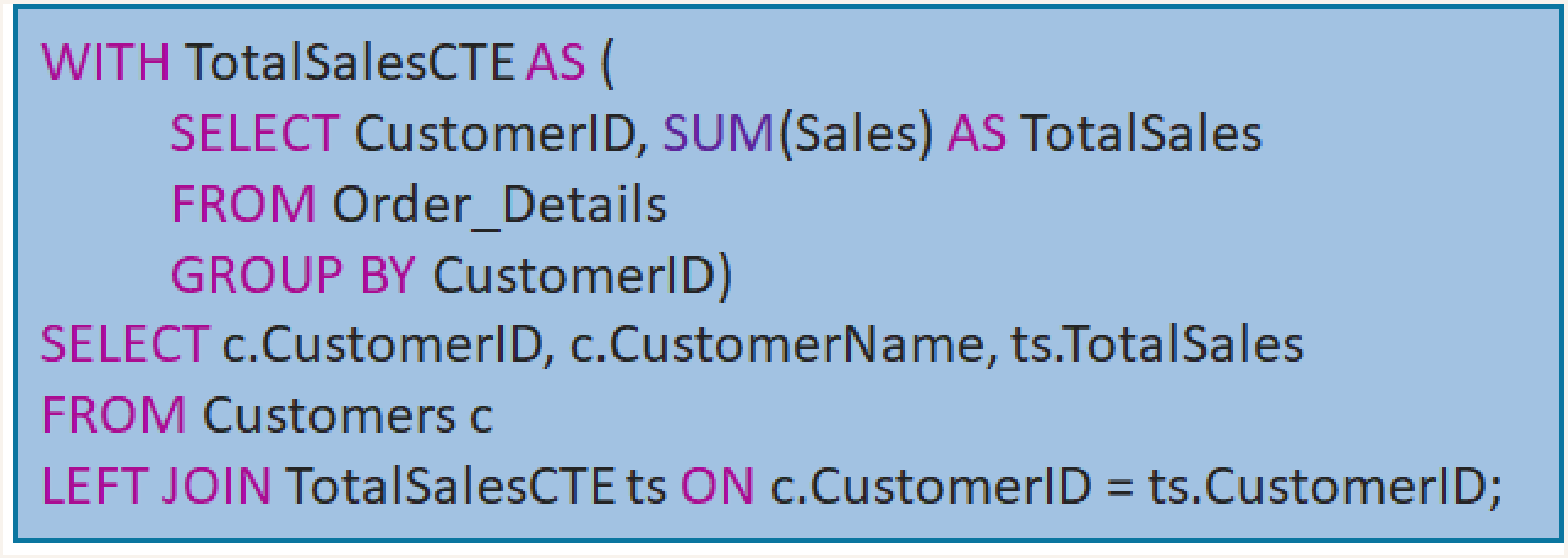


```sql
SELECT DISTINCT c.CustomerID, c.CustomerName
FROM customers c
JOIN order_details od
ON c.CustomerID = od.CustomerID
WHERE od.Sales > 2000;
```

10

Use Common Table Expressions (CTEs)

- CTEs break down complex queries into simpler parts.
- They improve both readability and efficiency.



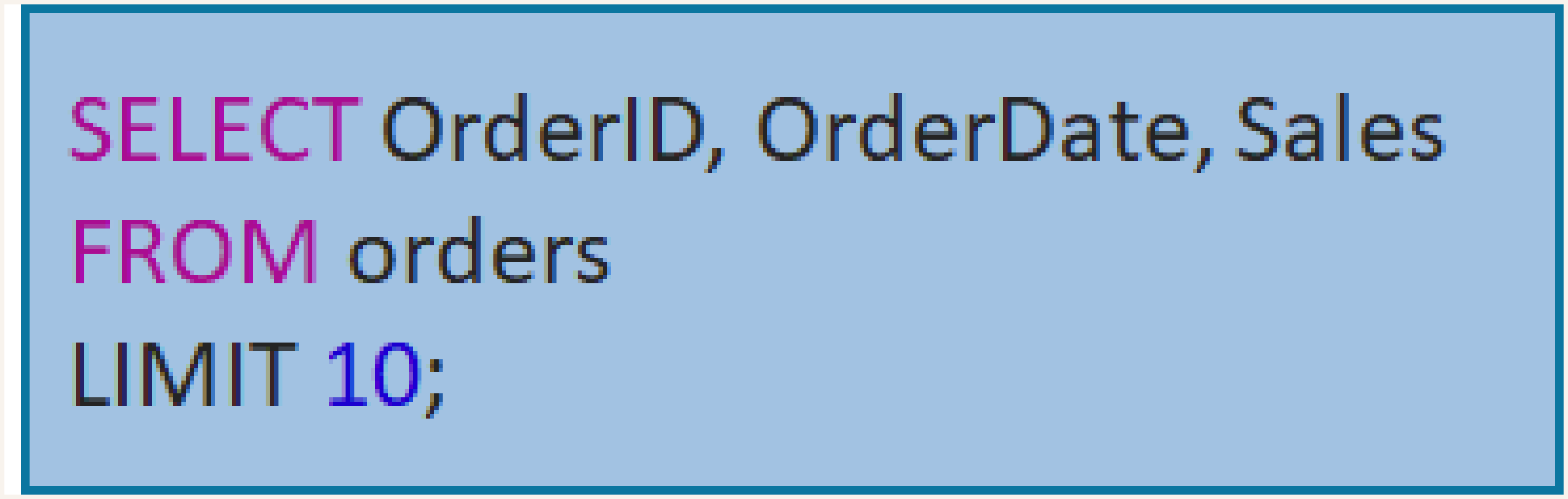
```
WITH TotalSalesCTE AS (
        SELECT CustomerID, SUM(Sales) AS TotalSales
        FROM Order_Details
        GROUP BY CustomerID)
SELECT c.CustomerID, c.CustomerName, ts.TotalSales
FROM Customers c
LEFT JOIN TotalSalesCTE ts ON c.CustomerID = ts.CustomerID;
```

@DATASAPIENT

11

Use LIMIT for Large Datasets

- Limit the number of rows returned to reduce database load.



```
SELECT OrderID, OrderDate, Sales
FROM orders
LIMIT 10;
```

Summary of Key Techniques

- Efficient SELECT queries lead to faster response times and a better user experience.
- Implement best practices like using indexes, avoiding unnecessary functions, and filtering early!
- Start optimizing your queries today!

FOLLOW FOR MORE

@DATASAPIENT

LIKE OUR CONTENT

SHARE WITH OTHERS

REPOST

LEAVE A COMMENT