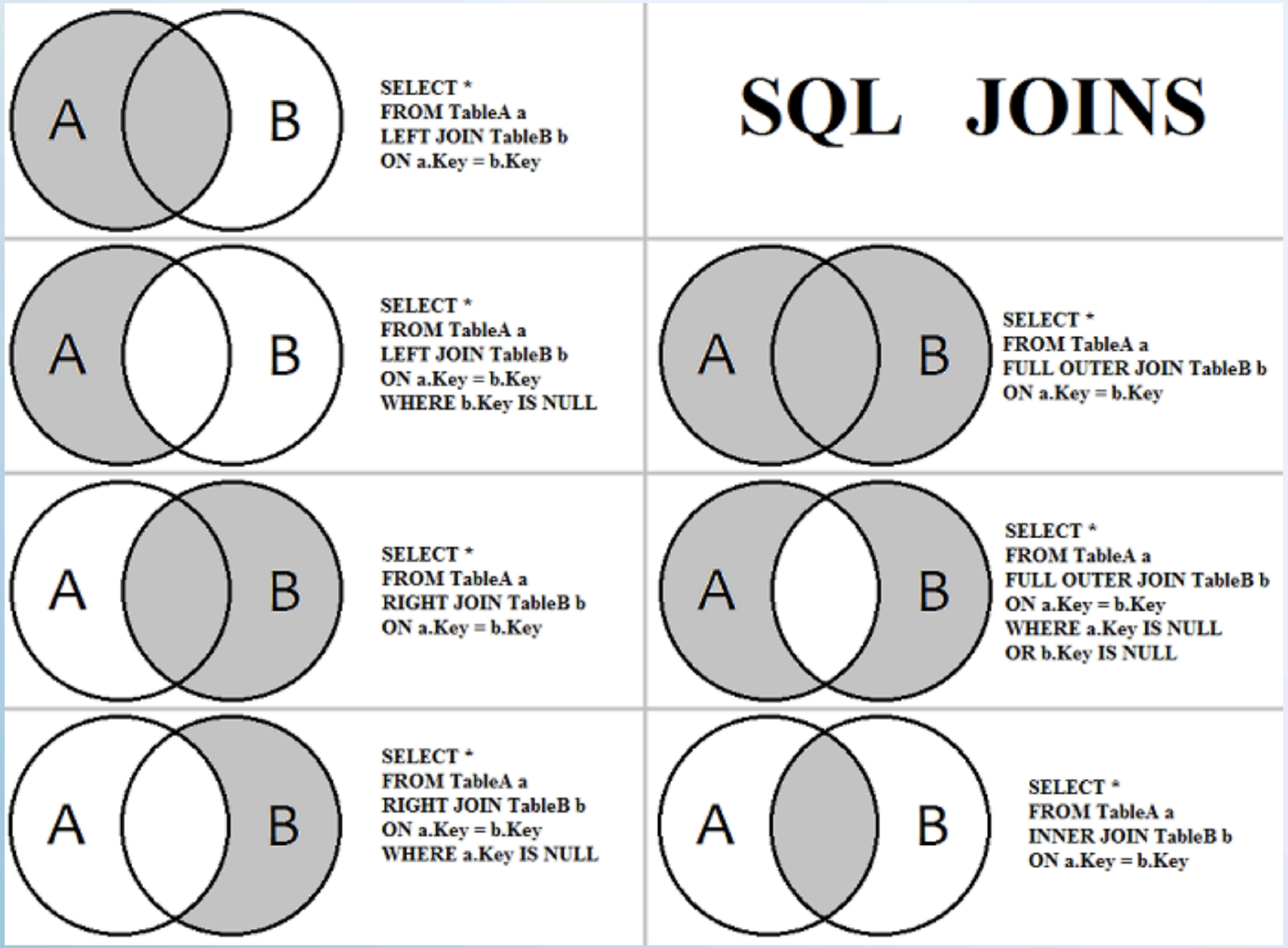
SQL Joins & CTEs Workshop – Understanding Table Relationships and Complex Queries

CODEUS_



```
SELECT *
FROM TableA a
LEFT JOIN TableB b
ON a.Key = b.Key
```

```
SELECT *
FROM TableA a
LEFT JOIN TableB b
ON a.Key = b.Key
WHERE b.Key IS NULL
```

```
SELECT *
FROM TableA a
FULL OUTER JOIN TableB b
ON a.Key = b.Key
```

```
SELECT *
FROM TableA a
RIGHT JOIN TableB b
ON a.Key = b.Key
```

```
SELECT *
FROM TableA a
FULL OUTER JOIN TableB b
ON a.Key = b.Key
WHERE a.Key IS NULL
OR b.Key IS NULL
```

```
SELECT *
FROM TableA a
RIGHT JOIN TableB b
ON a.Key = b.Key
WHERE a.Key IS NULL
```

```
SELECT *
FROM TableA a
INNER JOIN TableB b
ON a.Key = b.Key
```

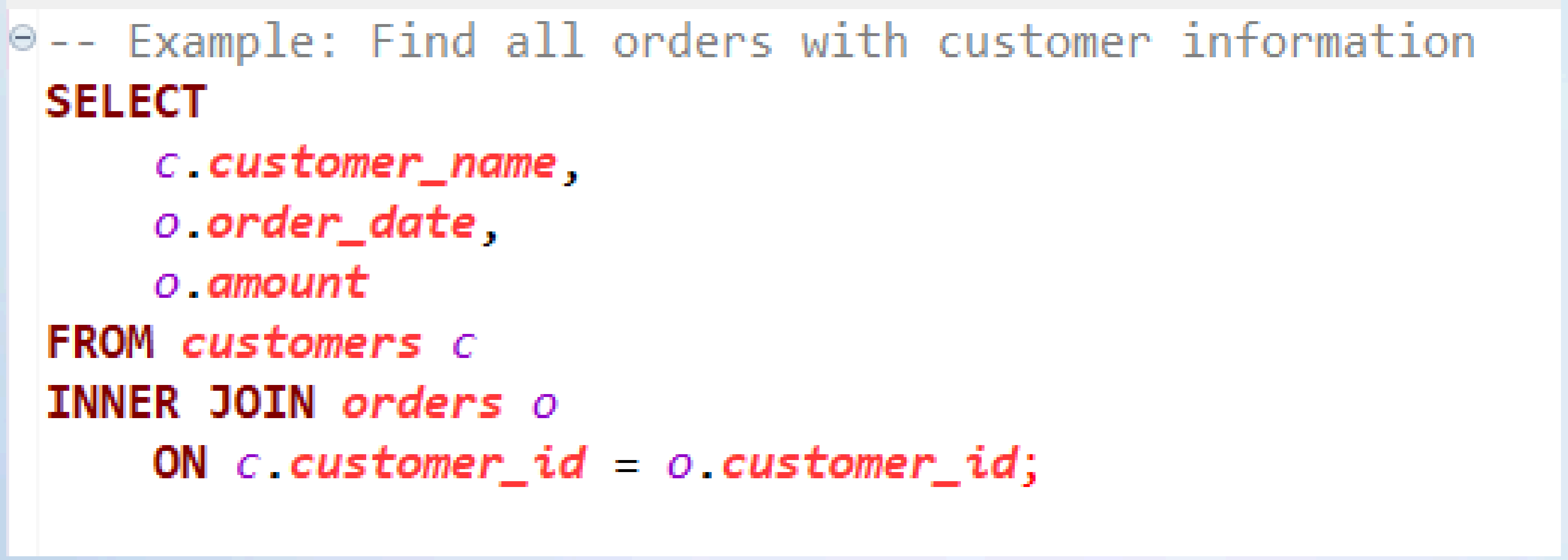
What is a JOIN?

- SQL JOIN combines rows from two or more tables
- Based on a related column between them
- Helps retrieve data that spans multiple tables
- Essential for relational database queries

CODEUS_

INNER JOIN

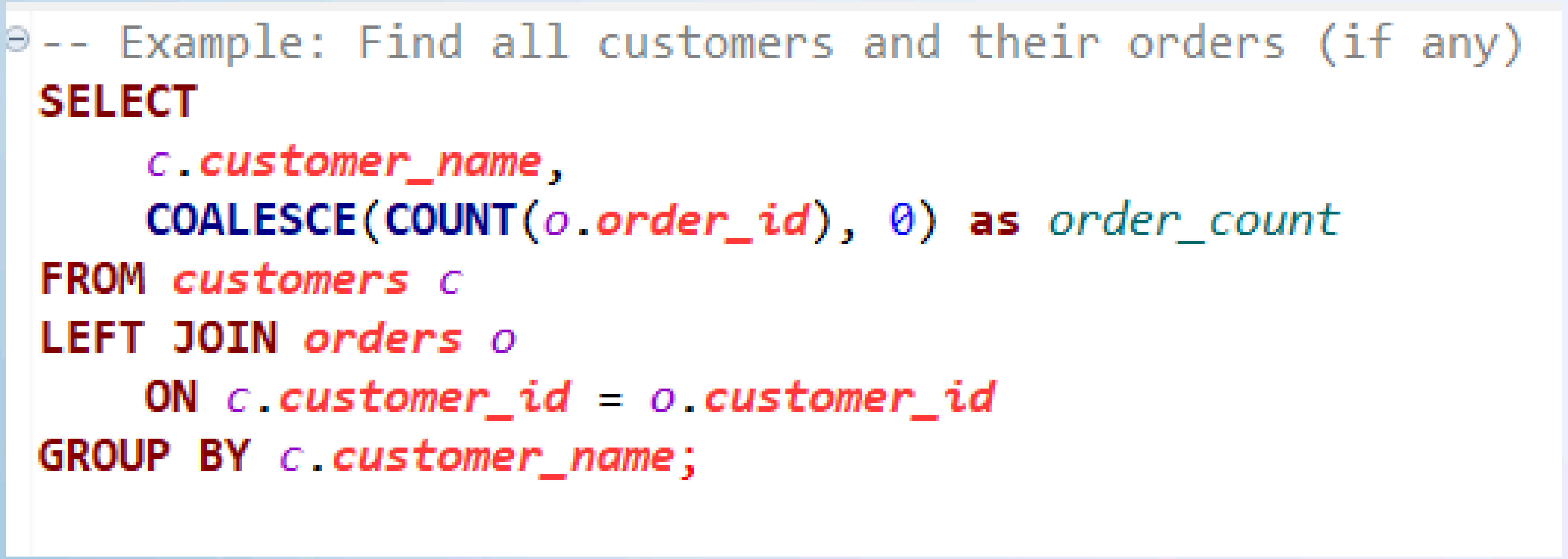
- Returns only matching rows from both tables
- Most common type of JOIN
- Like intersection in set theory
- Only returns data when there's a match in both tables



```
-- Example: Find all orders with customer information
SELECT
    c.customer_name,
    o.order_date,
    o.amount
FROM customers c
INNER JOIN orders o
    ON c.customer_id = o.customer_id;
```

CODEUS_

LEFT JOIN

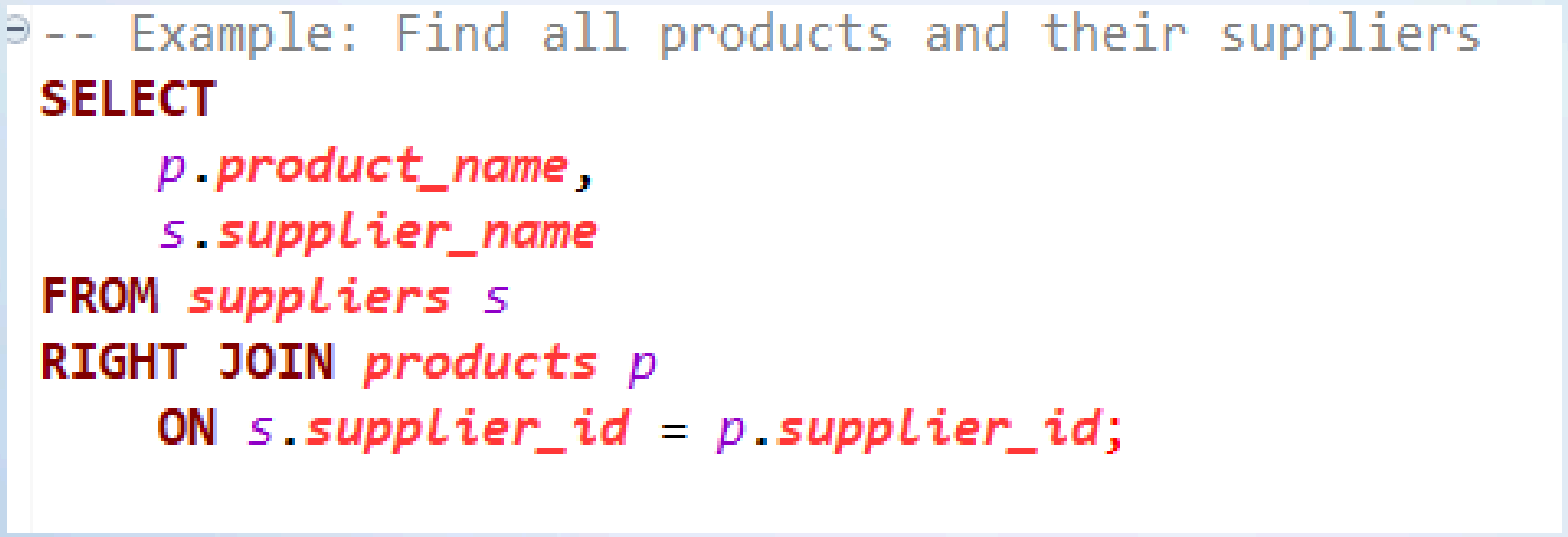


```
-- Example: Find all customers and their orders (if any)
SELECT
    c.customer_name,
    COALESCE(COUNT(o.order_id), 0) as order_count
FROM customers c
LEFT JOIN orders o
    ON c.customer_id = o.customer_id
GROUP BY c.customer_name;
```

- Returns all records from the left table
- Matching records from the right table
- NULL for non-matching right table records
- Useful for finding missing relationships

RIGHT JOIN

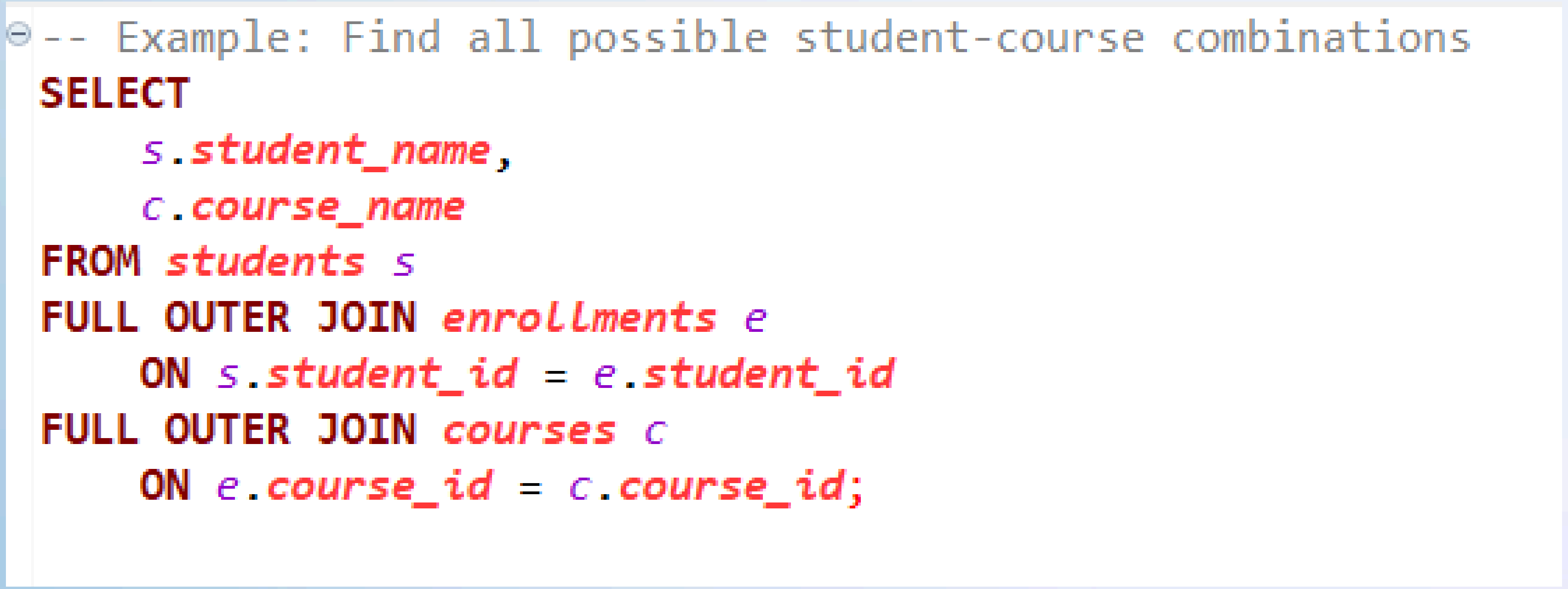
- Returns all records from the right table
- Matching records from the left table
- NULL for non-matching left table records
- Less common but useful in specific scenarios



```sql
-- Example: Find all products and their suppliers
SELECT
    p.product_name,
    s.supplier_name
FROM suppliers s
RIGHT JOIN products p
    ON s.supplier_id = p.supplier_id;
```

CODEUS_

FULL OUTER JOIN

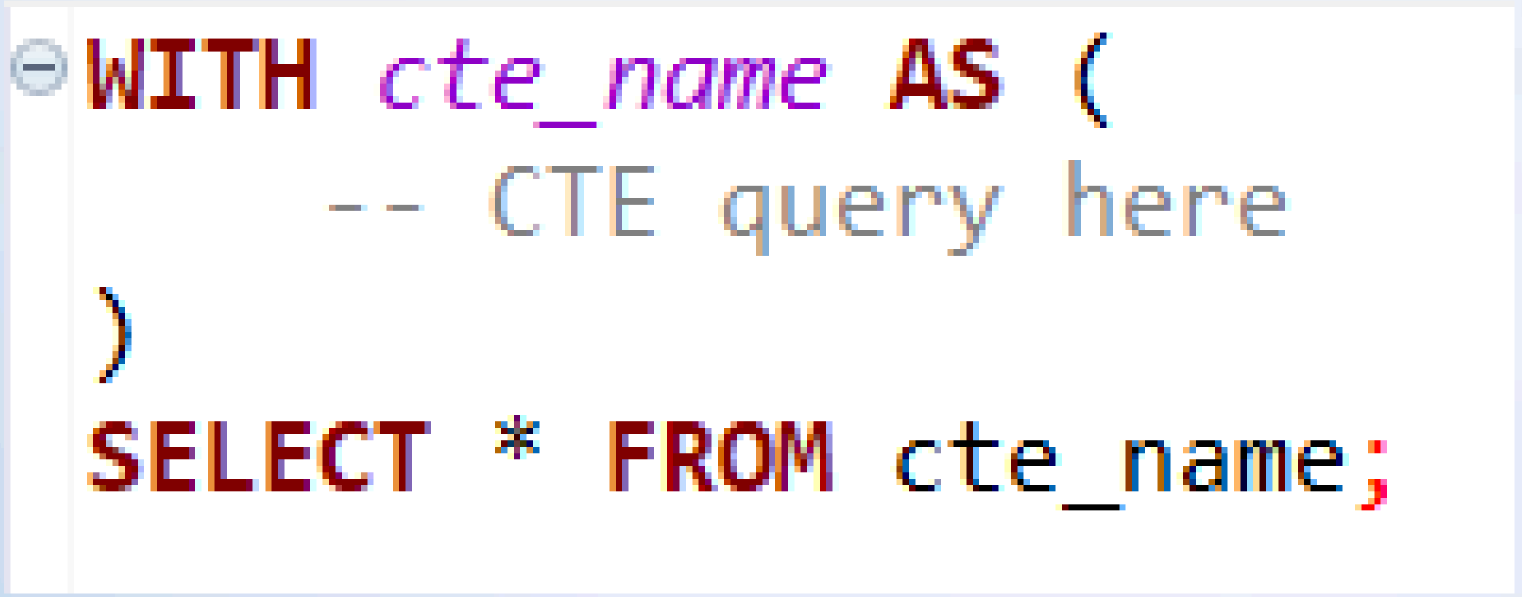


```
-- Example: Find all possible student-course combinations
SELECT
    s.student_name,
    c.course_name
FROM students s
FULL OUTER JOIN enrollments e
    ON s.student_id = e.student_id
FULL OUTER JOIN courses c
    ON e.course_id = c.course_id;
```

- Combines LEFT and RIGHT JOIN
- Returns all records from both tables
- NULL where there are no matches
- Useful for finding all missing relationships

What are CTEs?

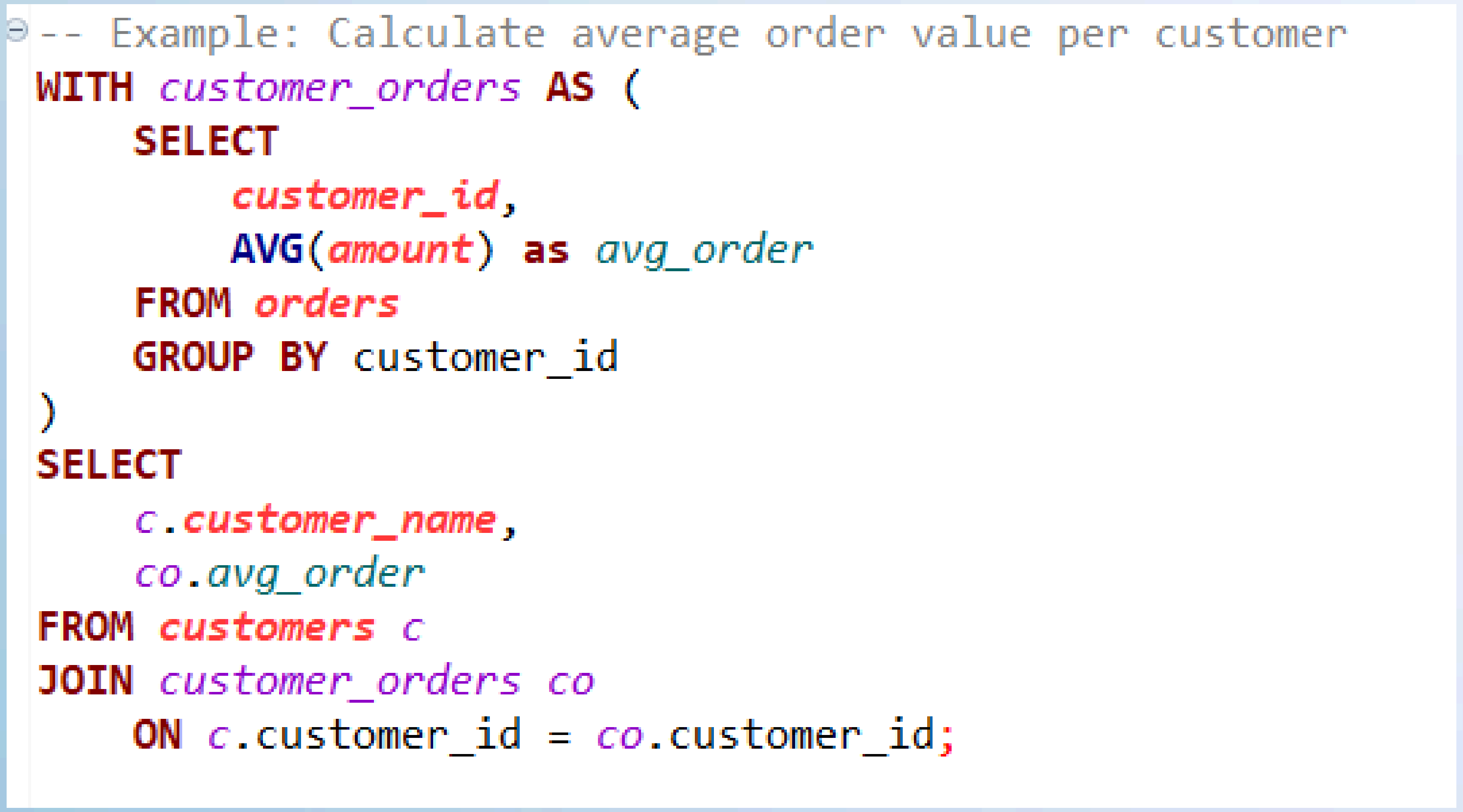
- Common Table Expressions (WITH clause)
- Temporary named result set
- Makes complex queries more readable
- Can be referenced multiple times in main query
- Helps break down complex logic



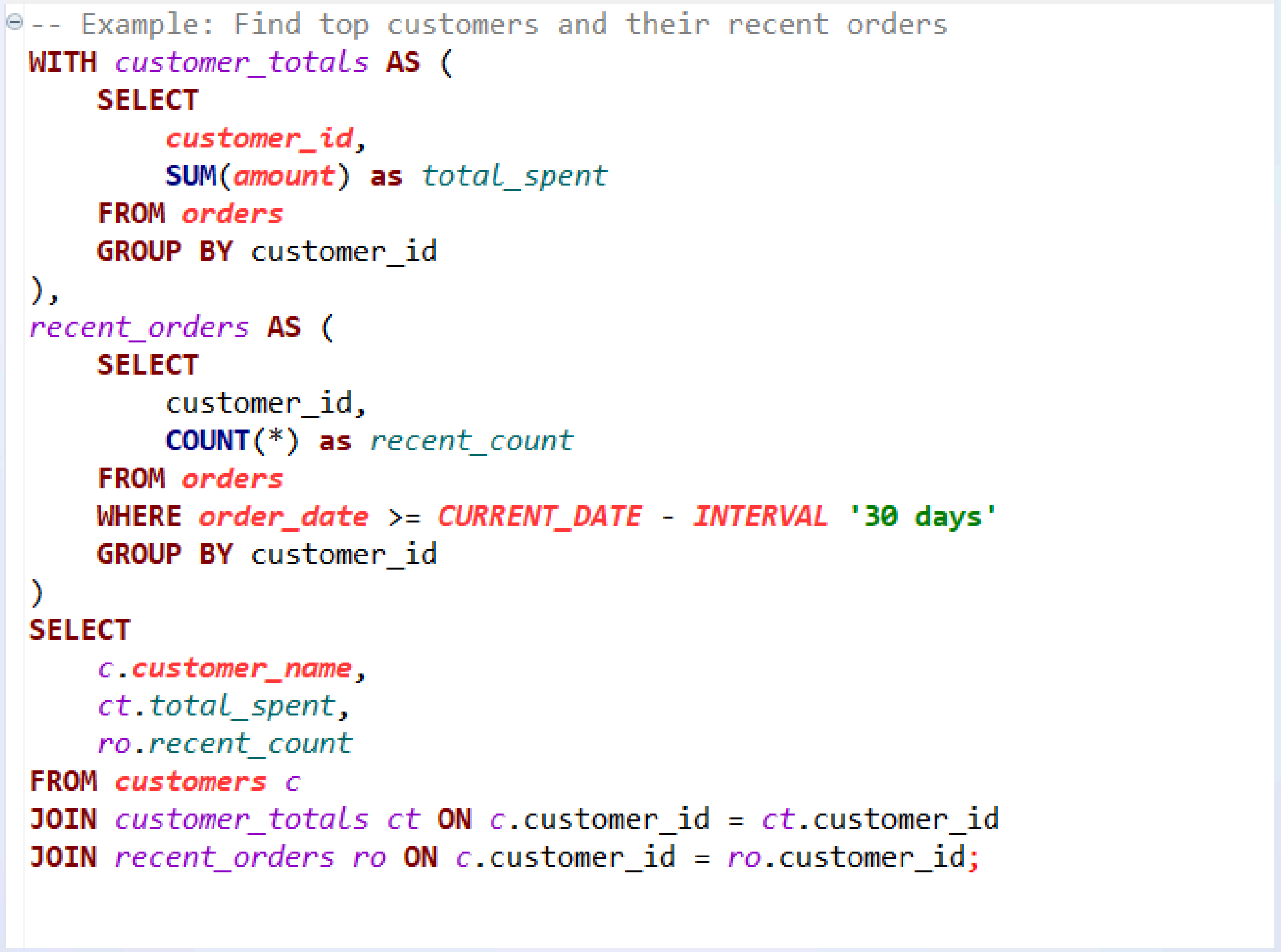
```sql
WITH cte_name AS (
    -- CTE query here
)
SELECT * FROM cte_name;
```

CODEUS_

CTE examples



```sql
-- Example: Calculate average order value per customer
WITH customer_orders AS (
    SELECT
        customer_id,
        AVG(amount) as avg_order
    FROM orders
    GROUP BY customer_id
)
SELECT
    c.customer_name,
    co.avg_order
FROM customers c
JOIN customer_orders co
    ON c.customer_id = co.customer_id;
```



```sql
-- Example: Find top customers and their recent orders
WITH customer_totals AS (
    SELECT
        customer_id,
        SUM(amount) as total_spent
    FROM orders
    GROUP BY customer_id
),
recent_orders AS (
    SELECT
        customer_id,
        COUNT(*) as recent_count
    FROM orders
    WHERE order_date >= CURRENT_DATE - INTERVAL '30 days'
    GROUP BY customer_id
)
SELECT
    c.customer_name,
    ct.total_spent,
    ro.recent_count
FROM customers c
JOIN customer_totals ct ON c.customer_id = ct.customer_id
JOIN recent_orders ro ON c.customer_id = ro.customer_id;
```

CODEUS_

Best Practices

- Always specify JOIN type explicitly
- Use meaningful table aliases
- Include JOIN conditions in ON clause
- Consider performance with large datasets
- Use CTEs for better code organization
- Comment complex queries

Common Pitfalls

- Forgetting JOIN conditions
- Using wrong JOIN type
- Not handling NULL values
- Creating cartesian products
- Writing overly complex CTEs
- Poor naming conventions

Thank you! Happy Coding!

CODEUS_