SUBQUERIES

VS

TEMP

TABLES

VS

CTE

WHAT'S THE DIFFERENCE?

WHY YOU SHOULD KNOW THIS...

Complex queries means multiple steps. **Subqueries**, **temporary tables**, and common table expressions (**CTEs**) can help. But what are they?

SUBQUERY
WHAT IT IS

- A query *inside* a query
- In the **FROM**, **WHERE**, or **HAVING** clause

SUBQUERY

HOW IT WORKS

- Subquery runs
- Results of the subquery *replace* the subquery
- Outer query executes

SUBQUERY

AN EXAMPLE

```sql
SELECT
    p.product_id
    , p.product_name
    , quantity.total_quantity
    , quantity.total_quantity * p.price AS total_sales_amount
FROM products AS p
INNER JOIN (
  SELECT
      product_id
      , SUM(quantity) AS total_quantity
   FROM sales
   GROUP BY product_id) AS quantity
  ON p.product_id = quantity.product_id;
```

Subquery in parenthesis

Subquery needs an alias

SUBQUERY
PROS & CONS

- **PROs**
  - Simplest option for filtering or performing calculations on data within a query
  - Often have good performance
- **CONs**
  - Can make complex queries harder to read and understand
  - Not reusable in the same query

TEMP TABLE
WHAT IT IS

- Stores a temporary result set that you can reuse
- Exists within a single MySQL session
- Can only be used by the person who created it

TEMP TABLE

HOW IT WORKS

- CREATE TEMPORARY **TABLE** statement
- Write query with results you want stored
- Use the temp table like you would any table

TEMP TABLE

AN EXAMPLE

```sql
CREATE TEMPORARY TABLE quantity
SELECT
      product_id
    , SUM(quantity) AS total_quantity
  FROM sales
  GROUP BY product_id;

SELECT
    p.product_id
    , p.product_name
    , q.total_quantity
    , q.total_quantity * p.price AS total_sales_amount
FROM products AS p
INNER JOIN quantity AS q
  ON p.product_id = q.product_id;
```

Create temp table first

Temp table needs a ;

Use the temp table in following queries

TEMP TABLE

ANOTHER EXAMPLE

```sql
CREATE TEMPORARY TABLE quantity AS
SELECT
    product_id,
    SUM(quantity) AS total_quantity
FROM sales
GROUP BY product_id;

CREATE TEMPORARY TABLE product_sales AS
SELECT
    p.product_id,
    p.product_name,
    q.total_quantity,
    q.total_quantity * p.price AS total_sales_amount
FROM products AS p
INNER JOIN quantity AS q
  ON p.product_id = q.product_id;

SELECT
    ps.product_id,
    ps.product_name,
    ps.total_quantity,
    ps.total_sales_amount,
    CASE
        WHEN ps.total_sales_amount > 1000 THEN 'High'
        ELSE 'Low'
    END AS sales_category
FROM product_sales AS ps;
```

Create temp table first

Same syntax for each temp table

Use the temp table in following queries

TEMP TABLE
PROS & CONS

- **PROs**
  - Act like regular tables for storing intermediate results
  - Can be accessed by multiple queries within the same session
  - Useful for complex data manipulation with multiple steps
- **CONs**
  - Disappear after the session ends
  - Data is physically stored

- A named temporary result set
- Used in SELECT, INSERT, UPDATE, or DELETE
- Exists for the duration of a single query

CTE

HOW IT WORKS

- Open with WITH ( ) clause
- Write query whose result set you want to use
- Select data from the CTE in a following query

CTE

AN EXAMPLE

Create CTE using WITH clause

```sql
WITH quantity AS (
SELECT
      product_id
      , SUM(quantity) AS total_quantity
   FROM sales
   GROUP BY product_id
 )

SELECT
    p.product_id
    , p.product_name
    , quantity.total_quantity
    , quantity.total_quantity * p.price AS total_sales_amount
FROM products AS p
INNER JOIN quantity AS q
  ON p.product_id = quantity.product_id;
```

No ; after CTE, closed with parenthesis

Use the CTE in following queries

CTE
ANOTHER EXAMPLE

Create *first* CTE using WITH clause

Following CTEs are separated by a comma

```sql
WITH quantity AS (
  SELECT
      product_id
      , SUM(quantity) AS total_quantity
  FROM sales
  GROUP BY product_id
)

, product_sales AS (
  SELECT
    p.product_id
    , p.product_name
    , q.total_quantity
    , q.total_quantity * p.price AS total_sales_amount
  FROM products AS p
  INNER JOIN quantity AS q
    ON p.product_id = q.product_id
)

SELECT
    ps.product_id
    , ps.product_name
    , ps.total_quantity
    , ps.total_sales_amount
    , CASE
        WHEN ps.total_sales_amount > 1000 THEN 'High'
        ELSE 'Low'
    END AS sales_category
FROM product_sales AS ps;
```

- **PROs**
  - Improve readability by breaking complex queries into named result sets
  - Can be reused multiple times within a single query
  - Support recursive queries
- **CONs**
  - Limited to a single query (not accessible by other queries in the session)

WHAT IT ALL MEANS

- Subqueries, temp tables, and CTEs all have similar functionality
- You can often use whichever one you prefer
- Each has pros and cons