SQL NOTES – SUBQUERIES & CARDINALITY RELATIONSHIPS

SUBQUERIES (GENERAL)

- A subquery is a query inside another query
- Executed first, then the outer query uses its result
- Exists only during query execution (not stored)

SUBQUERY IN WHERE CLAUSE

```
SELECT first_name, last_name
FROM sakila.customer
WHERE address_id IN (
    SELECT address_id
    FROM sakila.address
    WHERE district = 'California'
);
```

- Subquery runs first
- Filters rows in the main query
- Used instead of JOIN sometimes

SUBQUERY IN SELECT STATEMENT

```
SELECT actor_id,
       first_name,
       last_name,
       (
         SELECT COUNT(*)
         FROM sakila.film_actor
         WHERE film_actor.actor_id = actor.actor_id
       ) AS film_count
FROM sakila.actor;
```

- Subquery adds a calculated column
- Executes once per row of outer query
- Can affect performance on large tables

DERIVED TABLES (SUBQUERY IN FROM)

```
SELECT a.actor_id, a.first_name, a.last_name, fa.film_count
FROM sakila.actor a
JOIN (
    SELECT actor_id, COUNT(film_id) AS film_count
    FROM sakila.film_actor
    GROUP BY actor_id
    HAVING COUNT(film_id) > 10
) fa
ON a.actor_id = fa.actor_id;
```

- Subquery acts like a temporary table
- Exists only during execution
- Useful for complex filtering before join

CORRELATED SUBQUERIES

```
SELECT title,
       (
         SELECT COUNT(*)
         FROM sakila.film_actor fa
         WHERE fa.film_id = f.film_id
       ) AS actor_count
FROM sakila.film f;
```

- Subquery depends on outer query
- Refers to outer table column
- Executes once per outer row
- Slower than normal subqueries

CORRELATED SUBQUERY EXAMPLE (FILTER)

```
SELECT payment_id, customer_id, amount
FROM sakila.payment p1
WHERE amount > (
    SELECT AVG(amount)
    FROM sakila.payment p2
    WHERE p2.customer_id = p1.customer_id
);
```

- Inner query runs for each customer
- Compares value row-by-row
- Cannot be replaced by simple WHERE

SUBQUERY LIMITATIONS

- Scope limited to the query only
- Code duplication required
- Hard to reuse

- Slower execution
- Errors if subquery returns multiple rows unexpectedly

WHY WE MOVE TO JOINS / CTEs / VIEWS

- Better performance
- Cleaner structure
- Easier maintenance
- Reusable logic

CARDINALITY RELATIONSHIPS (FOUNDATION)

Cardinality describes how tables are related to each other.

Total 4 types.

ONE TO ONE (1 : 1)

Example:

- user → user_profile
- One user has only one profile
- One profile belongs to only one user

```
user.user_id = user_profile.user_id
```

ONE TO MANY (1 : N)

Example:

- user → order
- One user can place many orders
- Each order belongs to one user

```
user.user_id → order.user_id
```

MANY TO ONE (N : 1)

Example:

- order → user
- Many orders belong to one user
- Same as one-to-many, just reverse view

MANY TO MANY (M : N)

Example:

- user ↔ user (friendship)
- One user can have many friends
- Each friend can have many users

Requires a bridge table.

BRIDGE TABLE (MANY TO MANY)

Example:

- friendship table

Columns:

- user_id
- friend_id

```
user ↔ friendship ↔ user
```

- Uses self join
- Stores relationships explicitly
- Same table referenced twice

WHY BRIDGE TABLE IS REQUIRED

- Relational databases cannot store M:N directly
- Bridge table breaks M:N into two 1:N relationships
- Foundation for joins

KEY DIFFERENCE (SHORT)

Many to Many	**Bridge Table**
Logical relationship	Physical table
Cannot exist directly	Required to store data
Conceptual idea	Actual implementation
Explains connection	Stores the connection