Using subqueries to solve queries

What is a subquery

- a subquery is a query within a query.
- You can create subqueries within your SQL statements.
- These subqueries can reside in the WHERE clause, the FROM clause, or the SELECT clause.
- A subquery is called *simple* or standard if does not require a join between the subquery and the query that contains it.

Advantages of subqueries

- Advantages
  - Provide an alternative way to query data that would require complex joins and unions.
  - Make the complex queries more readable.
  - Allow a complex query to be structured in a way that it is possible to isolate each part.
- Database engine evaluates the whole query in two steps:
  - First, execute the subquery.
  - Second, use the result of the subquery in the outer query.

Working with subqueries

- Most often, the subquery will be found in the WHERE clause.
- These subqueries are also called **nested subqueries.**
- A subquery can also be found in the FROM clause. These are called **inline views**.
- A subquery can also be found in the SELECT clause.

Working with subqueries

- A *correlated subquery* has a join between the subquery and the query that contains it.
- Table aliases are often required to clarify the relationship between tables in the inner and outer query - especially if they involve the *same* table.
- In terms of scope, the inner query can "see" an outer table's columns, but the inner query's columns are not visible to the outer query.

Working with subqueries

- A subquery that returns only a single row is called a scalar subquery.
- If it returns no rows, the value is NULL.
- It can be used (with some exceptions) in places where an expression is normally used.
- **Single Row Sub Query:** Sub query which returns single row output. They mark the usage of single row comparison operators, when used in WHERE conditions.
- **Multiple row sub query:** Sub query returning multiple row output. They make use of multiple row comparison operators like IN, ANY, ALL. There can be sub queries returning multiple columns also.

Working with subqueries

- Single row **subquery** : Returns zero or one row.
- Multiple row **subquery** : Returns one or more rows.
- Multiple column **subqueries** : Returns one or more columns.
- Correlated **subqueries** : Reference one or more columns in the outer **SQL** statement

Introduction to the subquery

– how to write subqueries.
– We use the bricks and colours table.
– Run the queries below to see their contents:
– select * from bricks
– select * from colours

COLOUR_NAME	MINIMUM_BRICKS_NEEDED
blue	2
green	3
red	2
orange	1
yellow	1
purple	1

BRICK_ID	COLOUR
1	blue
2	blue
3	blue
4	green
5	green
6	red
7	red
8	red
9	-

Inline Views

- An inline view replaces a table in the from clause of your query.
- In this query, "select * from bricks" is an inline view:
- select * from (select * from bricks)

BRICK_ID	COLOUR
1	blue
2	blue
3	blue
4	green
5	green
6	red
7	red
8	red
9	-

Inline Views

- You use inline views to calculate an intermediate result set.
- For example, you can count the number of bricks you have of each colour
- Sql code

```
select * from (
select colour, count(*) c
from bricks
group by colour
) brick_counts
```

COLOUR	C
green	2
red	3
-	1
blue	3

Nested Subqueries

- Nested subqueries go in your where clause.
- The query filters rows in the parent tables.
- For example, to find all the rows in colours where you have a matching brick, you could write:

```
select * from colours c
where  c.colour_name in (
select b.colour from bricks b
)
```

COLOUR_NAME	MINIMUM_BRICKS_NEEDED
blue	2
green	3
red	2

Nested Subqueries

- You can filter rows in a subquery.
- To find all the colours that have at least one brick with a brick_id less than 5, write:
- Sql code

```
select * from colours c
where c.colour_name in (
  select b.colour from bricks b
  where b.brick_id < 5
)
```

COLOUR_NAME	MINIMUM_BRICKS_NEEDED
blue	2
green	3

Correlated vs. Uncorrelated

- A subquery is correlated when it joins to a table from the parent query.
- If you don't, then it's uncorrelated.
- This leads to a difference between IN and EXISTS.
- EXISTS returns rows from the parent query, as long as the subquery finds at least one row.
- So the following uncorrelated EXISTS returns all the rows in colours:
- Sql code

```
select * from colours
where exists (
    select null from bricks
)
```

COLOUR_NAME	MINIMUM_BRICKS_NEEDED
blue	2
green	3
red	2
orange	1
yellow	1
purple	1

Scalar Subqueries

- Scalar subqueries return one column and at most one row.
- You can replace a column with a scalar subquery in most cases.
- For example, to return a count of the number of bricks matching each colour, you could write:

```
select colour_name, (
select count(*)
from bricks b
where b.colour = c.colour_name
group by b.colour
) brick_counts
from colours c
```

COLOUR_NAME	BRICK_COUNTS
blue	3
green	2
red	3
orange	-
yellow	-
purple	-

Scalar Subqueries

- Note the colours with no matching bricks return null.
- To show zero instead, you can use NVL or coalesce.
- This needs to go around the whole subquery:

```
select colour_name, nvl ( (
        select count(*)
        from bricks b
        where b.colour = c.colour_name
        group by b.colour
        ), 0 ) brick_counts
from colours c
```

COLOUR_NAME	BRICK_COUNTS
blue	3
green	2
red	3
orange	0
yellow	0
purple	0

Scalar Subqueries

- You also need to join bricks to colours in the subquery.
- If you don't, it will return four rows (one for each different value for colour in bricks).
- This leads to an error:

```
select c.colour_name, (
    select count(*)
    from bricks b
    group by colour ) brick_counts
from colours c;
```

single-row subquery returns more than one row

Questions