Ironhack Student Portal

Lesson Goals

- Understand the need for subqueries and temp tables.
- Learn about subqueries and how they work.
- Learn how to join subqueries to tables inside a main query.
- Learn about temp tables and their benefits over subqueries.
- Learn how to swap out subqueries and temp tables from queries.
- Learn how to write efficient SQL queries.

Introduction

When working with data in databases, there will often be times when you will need to perform a series of steps to transform the data into a form that answers the exact question you want to answer. These steps often require you to create a series of queries. There may be one query that returns part of the result you are looking for, a second query that returns another part, and a third query that provides the remaining information you need to put a report together. In cases like these, knowing how to properly use subqueries and temp tables can help you construct an efficient and repeatable analytical process for retrieving the information you need.

In this lesson, we are going to learn about subqueries and temp tables, including the differences between them, how to write SQL code that utilizes them, and when one is preferable over the other. We will continue using the familiar publications database from previous lessons to explore examples of how these concepts can be applied. Finally, we will provide you with some useful tips on how to write efficient SQL queries.

Subqueries

Subqueries are queries nested within other queries. A subquery returns results that can be used by the main query it is embedded in, transformed further, or joined with other tables or subqueries. Let's look at an example using our publications data set.

Suppose we had built a query to summarize store sales, including the number of orders, the number of items, and the total quantity of units sold.

```
SELECT stores.stor_name AS Store, COUNT(DISTINCT(ord_num)) AS Orders, COUNT(title_id) AS Items,
SUM(qty) AS Qty
FROM publications.sales sales
INNER JOIN publications.stores stores ON stores.stor_id = sales.stor_id
GROUP BY Store;
```

Store	Orders	Items	Qty
Barnum's	2	2	125
Bookbeat	4	4	80
Doc-U-Mat: Quality Laundry and Books	3	6	130
Eric the Read Books	2	2	8
Fricative Bookshop	3	3	60
News & Brews	2	4	90

If we wanted to then show the average number of items per order and average quantity per item for each store, we could do that by embedding this query as a subquery into another query that performs those calculations.

```
SELECT Store, Items/Orders AS AvgItems, Qty/Items AS AvgQty
FROM (
    SELECT stores.stor_name AS Store, COUNT(DISTINCT(ord_num)) AS Orders, COUNT(title_id) AS
Items, SUM(qty) AS Qty
    FROM publications.sales sales
    INNER JOIN publications.stores stores ON stores.stor_id = sales.stor_id
    GROUP BY Store
) summary;
```

Store	AvgItems	AvgQty
Barnum's	1	62.5
Bookbeat	1	20
Doc-U-Mat: Quality Laundry and Books	2	21.6667
Eric the Read Books	1	4
Fricative Bookshop	1	20
News & Brews	2	22.5

The main query uses the results of the subquery and allows us to perform the calculations using the field names we assigned in the subquery. In order to access the results from the subquery, we also needed to enclose it in parentheses and give the subquery a name (summary).

A main query can contain multiple subqueries, sometimes with other subqueries nested within them as well. From the perspective of the main query, the results returned from a subquery serve the same purpose as a table would. They are a source of data for the main query. In fact, you can even join tables and subquery results inside the main query.

Suppose we wanted to see sales by title for the two stores that averaged more than one item per order. We can obtain these results by adding the `stor_id` field to our subquery, joining the subquery results to the sales table, creating another join between the sales table and the title table, and then adding a `WHERE` clause to the main query so that it returns only results where the average items per order were greater than 1.

```
SELECT Store, ord_num AS OrderNumber, ord_date AS OrderDate, title AS Title, sales.qty AS Qty,
price AS Price, type as Type
FROM (
 SELECT stores.stor_id AS StoreID, stores.stor_name AS Store, COUNT(DISTINCT(ord_num)) AS Orders,
COUNT(title_id) AS Items, SUM(qty) AS Qty
 FROM publications.sales sales
 INNER JOIN publications.stores stores ON stores.stor_id = sales.stor_id
 GROUP BY StoreID, Store
) summary
INNER JOIN publications.sales sales ON summary.StoreID = sales.stor_id
INNER JOIN publications.titles ON sales.title_id = titles.title_id
WHERE Items / Orders > 1;
```

Store	OrderNumber	OrderDate	Title	Qty	Price	Type
News & Brews	D4482	9/14/94 0:00	Is Anger the Enemy?	10	10.95	psychology
News & Brews	P2121	6/15/92 0:00	Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranean	40	20.95	trad_cook
News & Brews	P2121	6/15/92 0:00	Fifty Years in Buckingham Palace Kitchens	20	11.95	trad_cook
News & Brews	P2121	6/15/92 0:00	Sushi, Anyone?	20	14.99	trad_cook
Doc-U-Mat: Quality Laundry and Books	N914008	9/14/94 0:00	Is Anger the Enemy?	20	10.95	psychology

Store	OrderNumber	OrderDate	Title	Qty	Price	Type
Doc-U-Mat: Quality Laundry and Books	N914014	9/14/94 0:00	The Gourmet Microwave	25	2.99	mod_cook
Doc-U-Mat: Quality Laundry and Books	P3087a	5/29/93 0:00	Computer Phobic AND Non-Phobic Individuals: Behavior Variations	20	21.59	psychology
Doc-U-Mat: Quality Laundry and Books	P3087a	5/29/93 0:00	Life Without Fear	25	7	psychology
Doc-U-Mat: Quality Laundry and Books	P3087a	5/29/93 0:00	Prolonged Data Deprivation: Four Case Studies	15	19.99	psychology
Doc-U-Mat: Quality Laundry and Books	P3087a	5/29/93 0:00	Emotional Security: A New Algorithm	25	7.99	psychology

As you can imagine, this has the potential to get quite complicated, especially when there gets to be multiple tables and subqueries all joined together. When code gets complicated like this, the likelihood of there being errors increases. Additionally, it is important to note that subqueries are executed on-the-fly within their specific main query. That means that if you want to use the same subquery for a different main query, you would need to rewrite it (or copy it). This redundancy exposes you to potential inconsistencies in the way you pull your data together, which is something we would like to avoid if possible. One way to prevent your queries from getting overly complex and avoid redundancy is to use temp tables.

Temp Tables

Temp tables are temporary tables that are built from query results and deleted once your database session is over. They serve the purpose of storing query results for use later in the session. The way we would avoid having to write the same subquery multiple times is by saving the query results to a temp table and then querying the temp table every time we need to access those results.

Let's see how we could have swapped out the subquery in the previous section for a temp table. We can use the `CREATE TEMPORARY TABLE` command to do this, followed by the name we want to give the new table and then the query.

```
CREATE TEMPORARY TABLE publications.store_sales_summary
SELECT stores.stor_id AS StoreID, stores.stor_name AS Store, COUNT(DISTINCT(ord_num)) AS Orders,
COUNT(title_id) AS Items, SUM(qty) AS Qty
FROM publications.sales sales
INNER JOIN publications.stores stores ON stores.stor_id = sales.stor_id
GROUP BY StoreID, Store;
```

This created a new temp table in the publications database that has the results of our query stored in it. Now that our temp table has been created, we can query it just like any other table.

```
SELECT * FROM publications.store_sales_summary;
```

StoreID	Store	Orders	Items	Qty
6380	Eric the Read Books	2	2	8
7066	Barnum's	2	2	125
7067	News & Brews	2	4	90
7131	Doc-U-Mat: Quality Laundry and Books	3	6	130
7896	Fricative Bookshop	3	3	60
8042	Bookbeat	4	4	80

We can now replace our subquery with this table in the main query where we retrieved title sales details for stores with average items per order greater than 1.

```
SELECT Store, ord_num AS OrderNumber, ord_date AS OrderDate, title AS Title, sales.qty AS Qty,
price AS Price, type as Type
FROM publications.store_sales_summary summary
INNER JOIN publications.sales sales ON summary.StoreID = sales.stor_id
INNER JOIN publications.titles ON sales.title_id = titles.title_id
WHERE Items / Orders > 1
```

This is much easier to read than the example with the subquery in it. Additionally, if we need to use the data stored in the temp table again, we can just query it or add a join to that table.

Efficient SQL Tips

As you learn to use increasingly more complex SQL queries, it's a good idea to provide you with some useful tips on how to write efficient SQL queries.

Retrieve only the dataset you need

If you only need 100 rows from your database, add `LIMIT 100` to your query instead of selecting all the records.

Also, don't always use `SELECT *` on tables with lot of columns. If you need the data from only a few columns, specify which columns to select e.g.:

```
SELECT column_1, column_2, column_3, ... FROM table_name
```

Optimize `WHERE`

In `WHERE` clause, avoid using unnecessary parentheses and unnecessary conditions.

Example 1

Avoid:

```
((a AND b) AND c OR (((a AND b) AND (c AND d))))
```

Use:

```
(a AND b AND c) OR (a AND b AND c AND d)
```

Example 2

Avoid:

```
(B>=5 AND B=5) OR (B=6 AND 5=5) OR (B=7 AND 5=6)
```

Use:

```
  B=5 OR B=6
```

Optimize `JOIN`

If all columns in ORDER BY and GROUP BY clauses come from the same table, that table is preferred first when joining. For example, instead of using:

```
SELECT c.year, c.color, c.name
FROM sales s
INNER JOIN cars c
ON s.car_id = c.car_id
WHERE s.date = '01-01-2019';
```

Use:

```
SELECT c.year, c.color, c.name
FROM cars c
INNER JOIN sales s
ON c.car_id = s.car_id
WHERE s.date = '01-01-2019';
```

For the full documentation on MySQL optimization, refer to the MySQL Reference Manual.

Summary

In this lesson, we have introduced subqueries and temp tables. While both are useful for putting together more complex, multi-step data analyses, an important difference between them is that you can store temp table results for use later in your SQL session. As you continue to build your analytical abilities, you will find opportunities to use these concepts to keep your processes intuitive and efficient.

Also, we provided some helpful tips on how to write efficient SQL queries. You will find these tips useful when you write increasingly complex queries with massive databases.