Subquery Performance

by Sophia

WHAT'S COVERED

This tutorial explores the use of the EXPLAIN statement in two parts:

1. EXPLAIN explained
2. Subqueries vs JOIN

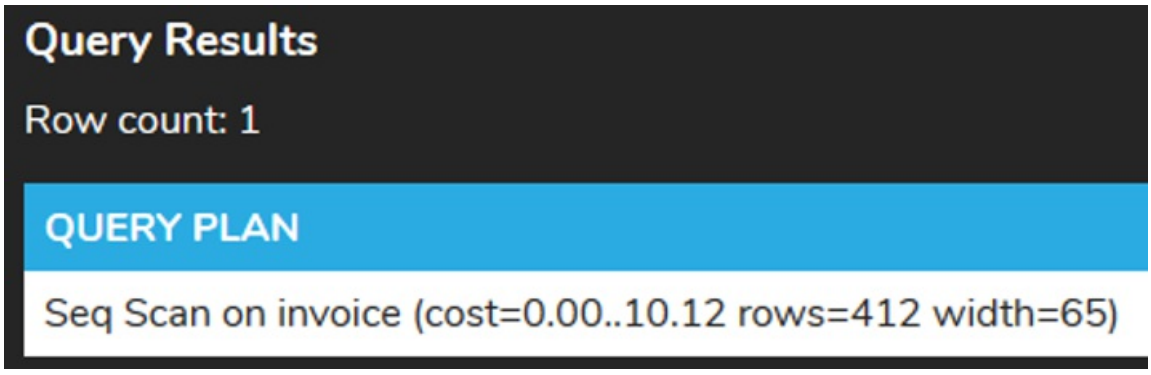
1. EXPLAIN explained

When using subqueries, you do have to consider performance as it relates to each individual part of the query. This can and will be different each time you run a query. However, you can get a general idea of how efficient a query will be in PostgreSQL, because a query plan is created for every query that is attempted to run in the database. The database tries to look at the query structure and the properties of the data and tables to create a good plan before it is executed.

Understanding all of the details of a plan can be quite complex, so our focus is on the basics around what to look for within a plan. We can use the EXPLAIN command on a query to display some of those details.

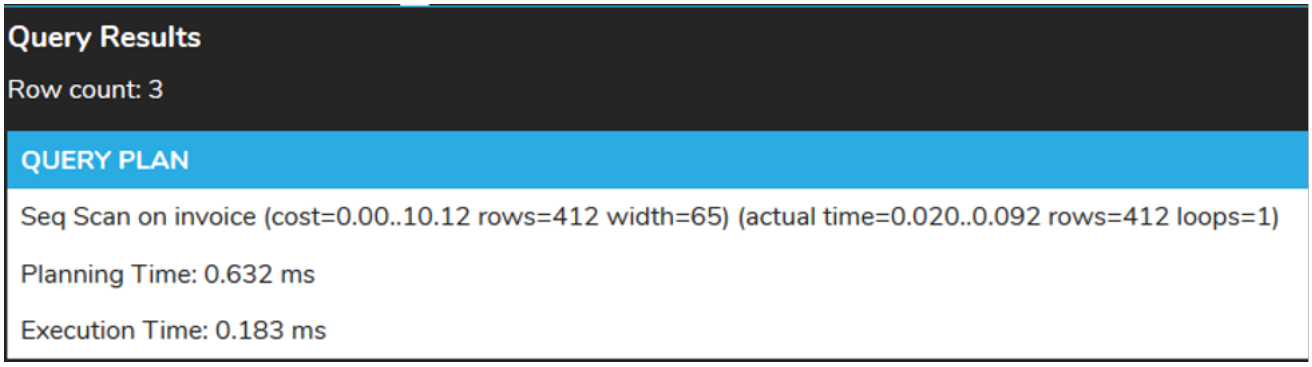
Let's query the invoice table and see what the results tell us:

```
EXPLAIN
SELECT *
FROM invoice;
```

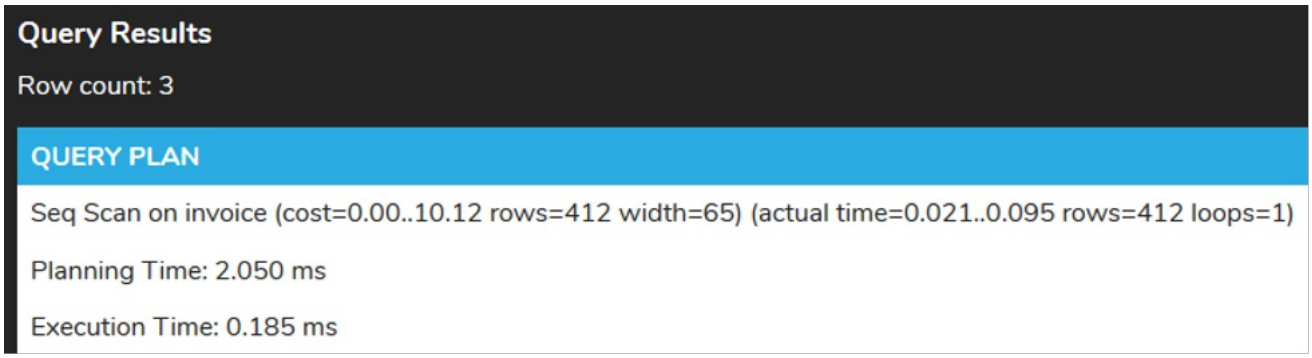


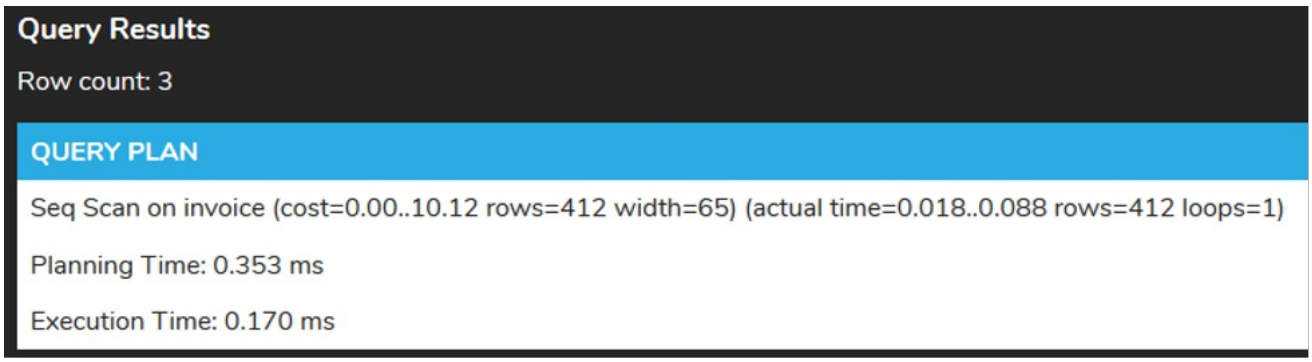
Using just EXPLAIN, the database will not run the query. It will create an estimation of the execution plan based on the statistics that it has. This means that the plan can differ a bit from reality. In PostgreSQL though, we can execute the query as well. To do so, we can use the EXPLAIN ANALYZE at the beginning of the query instead of just EXPLAIN.

```
EXPLAIN ANALYZE
SELECT *
FROM invoice;
```



Note that this is an actual run process, so the timing most likely will be different each time you run it. For example, running it two more times yields the following results:





As such, we cannot simply depend on the planning and execution time when comparing various statements. Each query plan consists of nodes, and they can be nested and executed from the inside out. This means that the innermost node is executed before an outer node. Each of the nodes has a set of associated statistics, like the cost, the number of rows that are returned, the number of loops that are performed (if needed), and some other choices. From the last execution above, we can see that the cost shows 0.00 .. 10.12 and we estimate that there are 412 rows returned. The width also shows the estimated width of each row in bytes, which in our case is 65. The cost field shows how expensive the node was. This can be a bit complex as it is measured using different settings that start to become quite technical. The type of node is an important item on the left of the first line. In this case, we have a “Seq Scan”.

There are quite a few nodes, but the most common ones are as follows:

- Seq Scan – This is a sequential scan that is performed over a table within the database. It can be very

slow if we're retrieving many rows in a table, so it is best to avoid these types of nodes for large tables.

- Index Scan – A scan over an index can be much faster to find data. Think of it like searching through the table of contents or glossary in a book to point you to a specific page. This is much faster than looking at each page one at a time. Index scans tend to be the fastest means to search for data.
- Bitmap Index Scan and Bitmap Heap Scan – In terms of performance, the bitmap scans fall in between the sequential scan and the index scan. These can occur if we are reading too much data from an index scan, but too little from a sequential scan.

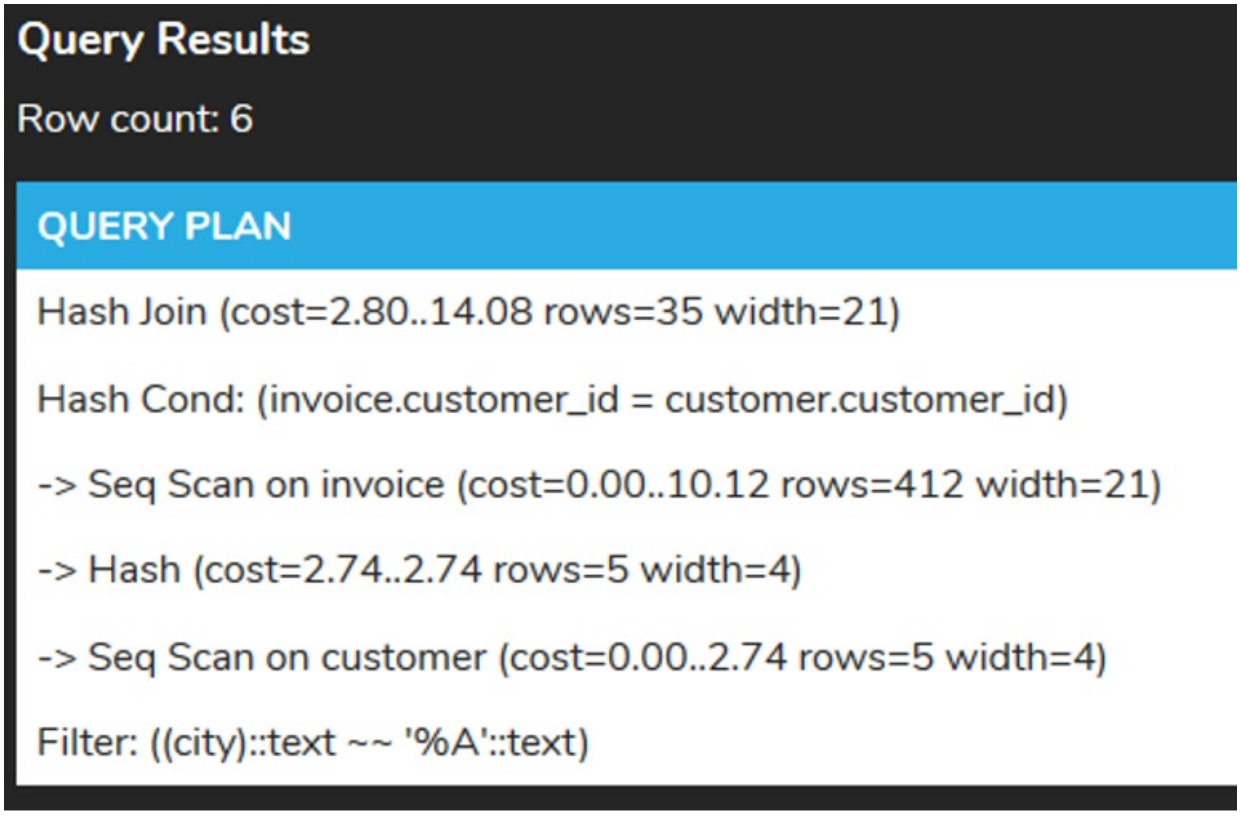
Anytime that we are querying data from a primary key, a unique key, or another index field, it should show as an index scan or a bitmap scan. As such, the results will generally be much faster than a sequential scan.

---

2. Subqueries vs JOIN

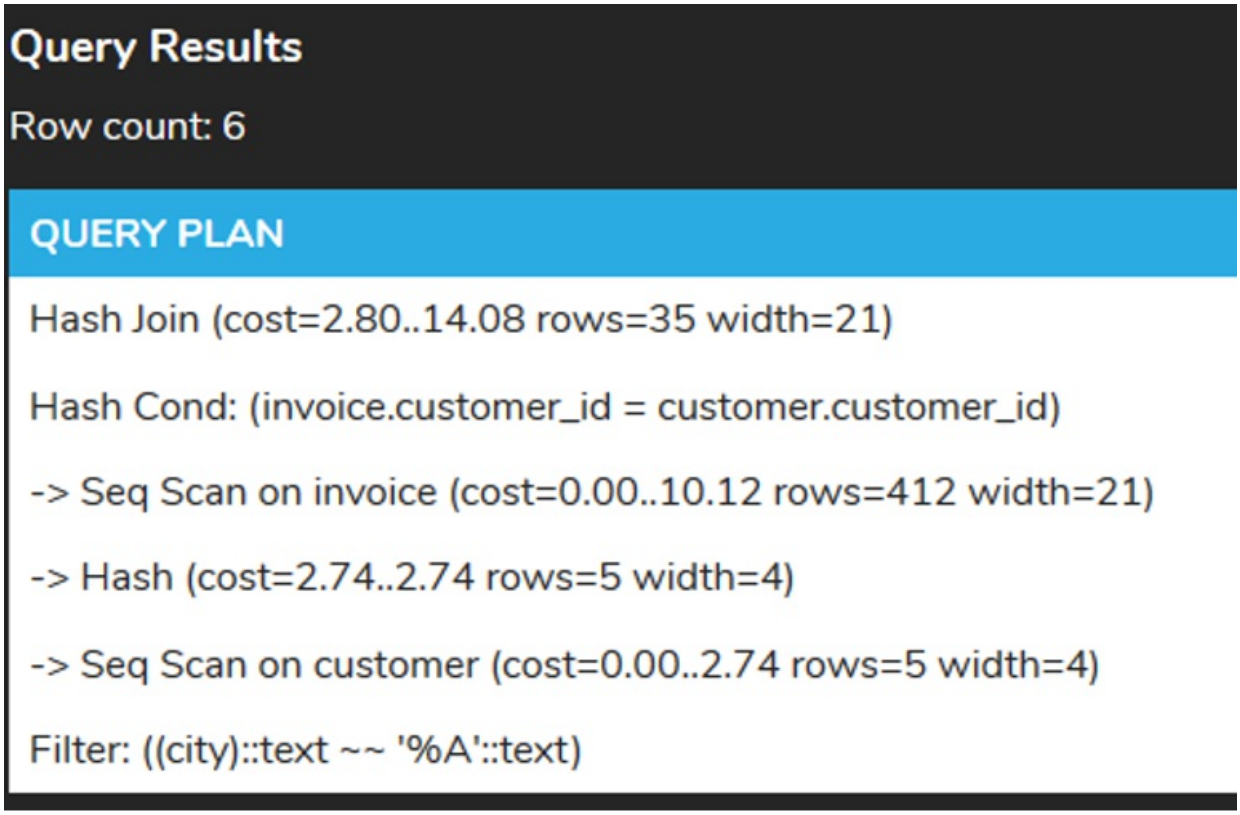
Subqueries are more complex because you are using nested queries. For example, here is a subquery similar to what we used in the prior tutorial:

```
EXPLAIN
SELECT invoice_id, invoice_date,customer_id, total
FROM invoice
WHERE  customer_id IN
    (SELECT customer_id FROM customer
     WHERE city LIKE '%A');
```



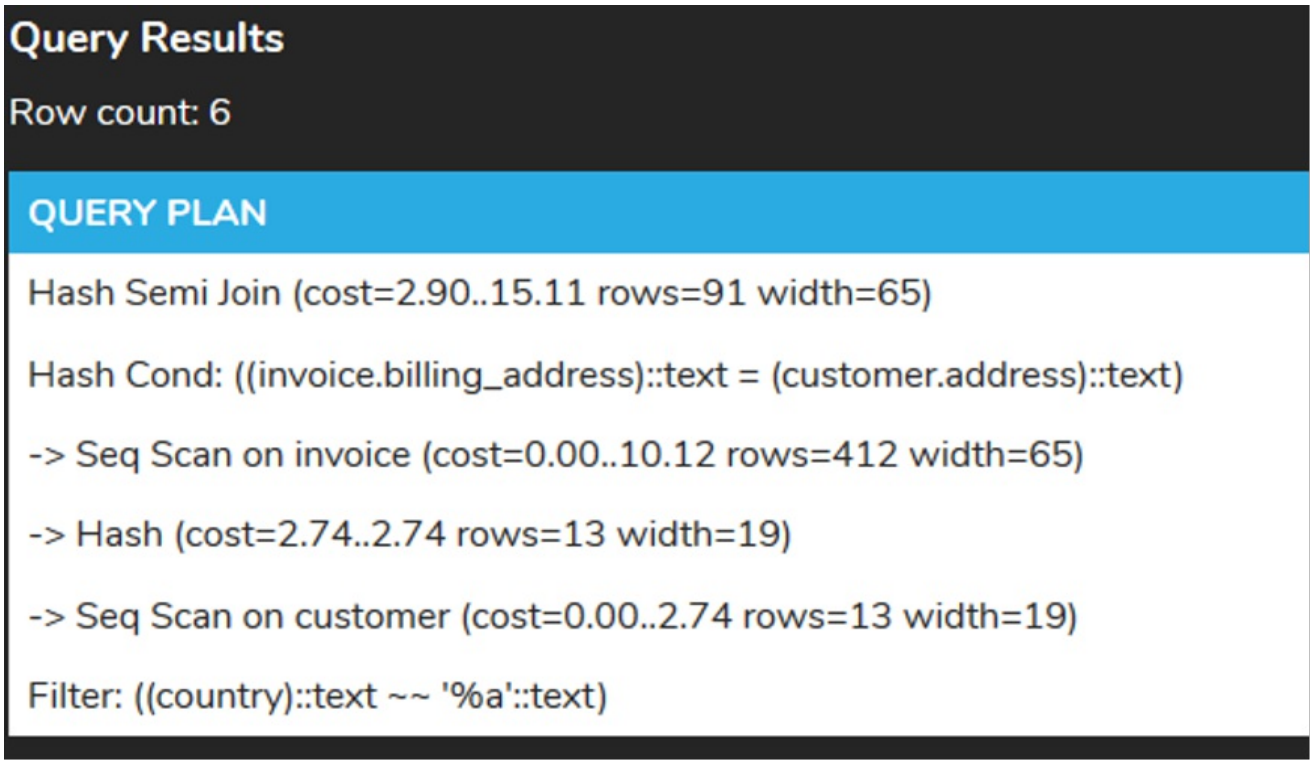
Here, we have the sequential scan on the customer table in the inner query. Then, another sequential scan on the outer query. However, this may not always result in the fastest execution. In most cases, a join will be faster if it is using primary key, foreign key or other indexed columns. For example:

```
EXPLAIN
SELECT invoice_id, invoice_date,invoice.customer_id, total
FROM invoice
INNER JOIN customer
ON invoice.customer_id = customer.customer_id
WHERE city LIKE '%A';
```

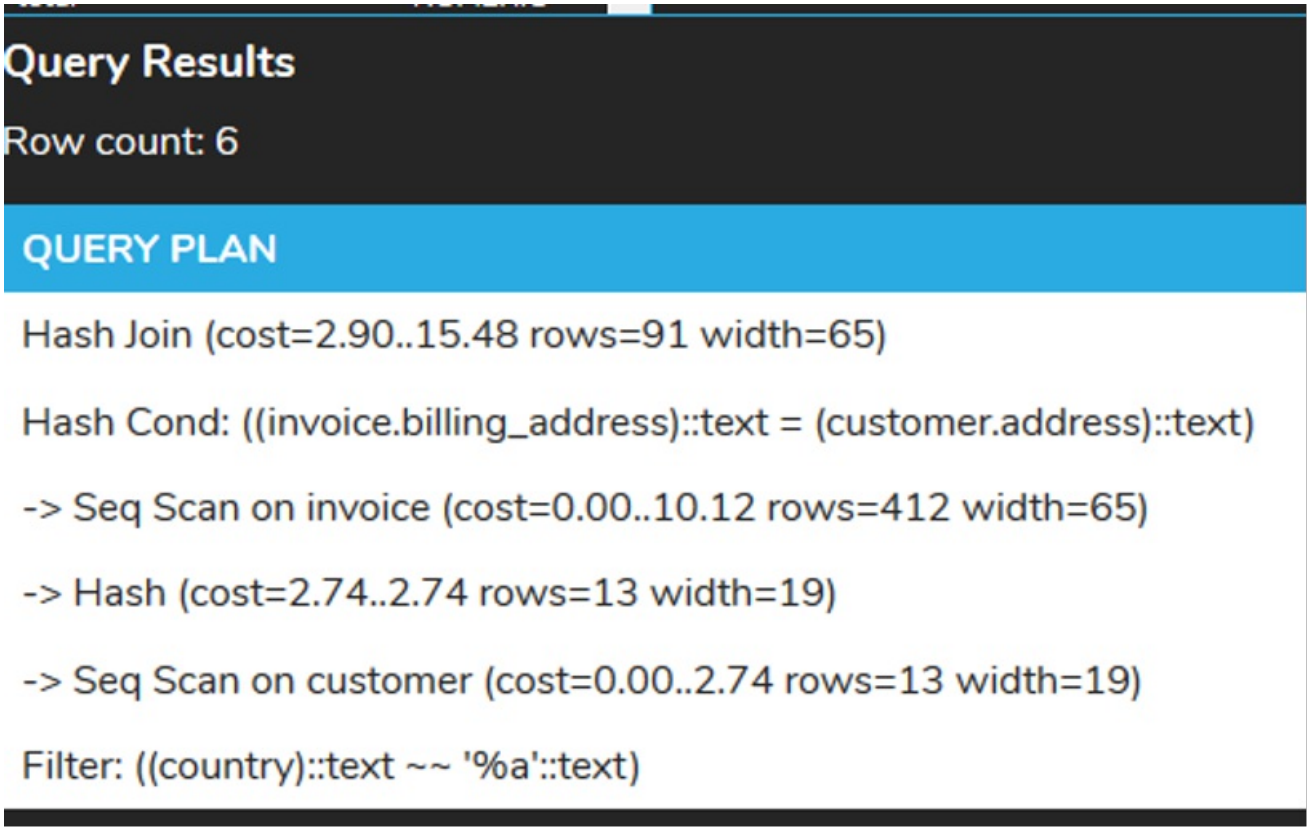


The results are basically identical, with the type of join and the cost associated with each. As the data sets get larger, and the result sets differ, it will become easier to distinguish them, especially being that the customer_id was used to join them together. What if we joined it on the billing address instead:

```
EXPLAIN
SELECT *
FROM invoice
WHERE billing_address IN
(SELECT address FROM customer
WHERE COUNTRY like '%a');
```




```
EXPLAIN
SELECT invoice.*
FROM invoice
INNER JOIN customer
ON customer.address = invoice.billing_address
WHERE COUNTRY like '%a';
```



Conceptually, the subquery would end up being slightly faster on the cost of the hash join, as the subquery

resulted in 15.11, while the inner join resulted in 15.48. The higher the values, the slower the queries would run. When we compare the explain plans of queries to gauge their performance, we want to look at the type of scan they are using and then cost. Generally, in order of slowest to fastest, sequential scan is the slowest, then index, then hashmap, and then bitmap. If the scans are the same across both types of queries, we want to compare the cost of each join. The higher the value of the cost, the slower it would take, whereas the lower the cost, the faster the join would take.

Video Transcription

[MUSIC PLAYING] Here, you'll see an example of an EXPLAIN PLAN. So EXPLAIN PLAN, where we have just the EXPLAIN and then the query, they'll provide us with the QUERY PLAN that's going to be involved in order to query the information. They'll tell us a little bit of information, such as the cost, number of rows associated with it, as well as the size of the data associated with it.

It won't actually run this and test this. We can do so by doing and EXPLAIN ANALYZE. Once we do so, we'll be able to see the actual time that it actually took to run it, as well as what the planning time was and execution. Note that when it comes to the actual time, the planning time and the execution will differ every time you run it. So if I try to run it again, you'll see that the time actually differs in this case here. Run it again-- it changes and so forth.

So it's not reliable to take a look at the planning execution type all the time. It's important to make note of what the cost is, amount of tows, as well as the width. But we also want to be able to identify some additional details like if it's a sequential scan. Sequential scans will search through the entire set of data one at a time. Index scans will utilize an index, kind of like a table of contents or glossary in the back of the book, to be able to pinpoint things faster. Bitmap scans or hash scans will be also much faster than a sequential scan.

[MUSIC PLAYING]

Your turn! Open the SQL tool by clicking on the LAUNCH DATABASE button below. Then enter in one of the examples above and see how it works. Next, try your own choices for which columns you want the query to provide.

The performance of queries can be explored further using the EXPLAIN statement.

Source: Authored by Vincent Tran