Advanced SQL: 4th lesson – Writing Efficient Queries

Sometimes it doesn't matter whether your query is efficient or not. For example, you might write a query you expect to run only once, and it might be working on a small dataset. In this case, anything that gives you the answer you need will do.

But what about queries that will be run many times, like a query that feeds data to a website? Those need to be efficient so you don't leave users waiting for your website to load. Or, what about queries on huge datasets? These can be slow and cost a business a lot of money if they are written poorly.

Most database systems have a query optimizer that attempts to interpret/execute your query in the most effective way possible. But several strategies can still yield huge savings in many cases.

Strategies:

* Only select the columns you want

It is tempting to start queries with SELECT \* FROM .... It's convenient because you don't need to think about which columns you need. But it can be very inefficient. This is especially important if there are text fields that you don't need, because text fields tend to be larger than other fields.

star\_query = "SELECT \* FROM `bigquery-public-data.github\_repos.contents`"

show\_amount\_of\_data\_scanned(star\_query)

basic\_query = "SELECT size, binary FROM `bigquery-public-data.github\_repos.contents`"

show\_amount\_of\_data\_scanned(basic\_query)

Data processed: 2682.118 GB

Data processed: 2.531 GB

In this case, we see a 1000X reduction in data being scanned to complete the query, because the raw data contained a text field that was 1000X larger than the fields we might need.

* Read less data

Both queries below calculate the average duration (in seconds) of one-way bike trips in the city of San Francisco.

more\_data\_query = """

SELECT MIN(start\_station\_name) AS start\_station\_name,

MIN(end\_station\_name) AS end\_station\_name,

AVG(duration\_sec) AS avg\_duration\_sec

FROM `bigquery-public-data.san\_francisco.bikeshare\_trips`

WHERE start\_station\_id != end\_station\_id

GROUP BY start\_station\_id, end\_station\_id

LIMIT 10

"""

show\_amount\_of\_data\_scanned(more\_data\_query)

less\_data\_query = """

SELECT start\_station\_name,

end\_station\_name,

AVG(duration\_sec) AS avg\_duration\_sec

FROM `bigquery-public-data.san\_francisco.bikeshare\_trips`

WHERE start\_station\_name != end\_station\_name

GROUP BY start\_station\_name, end\_station\_name

LIMIT 10

"""

show\_amount\_of\_data\_scanned(less\_data\_query)

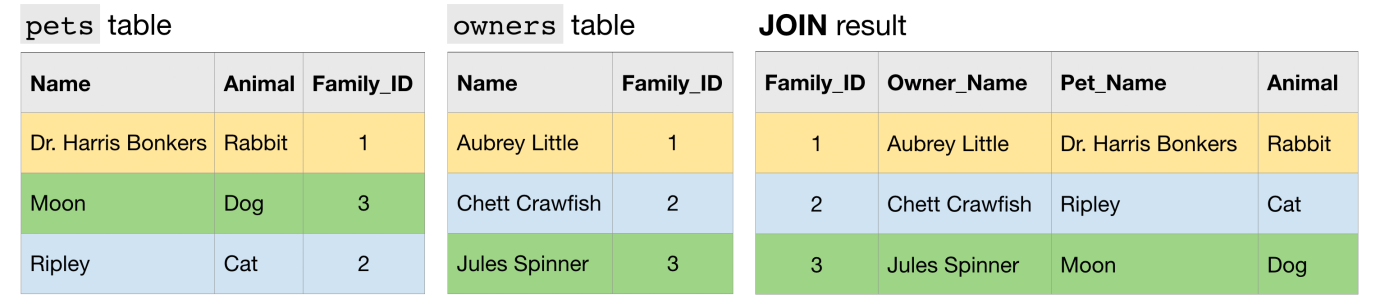
Data processed: 0.076 GB

Data processed: 0.06 GB

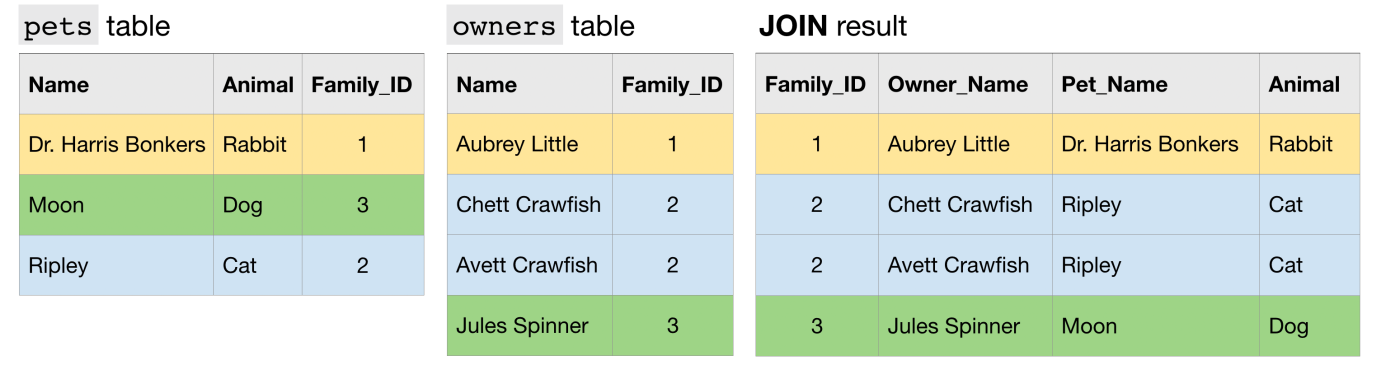
Since there is a 1:1 relationship between the station ID and the station name, we don't need to use the start\_station\_id and end\_station\_id columns in the query. By using only the columns with the station IDs, we scan less data.

* Avoid N:N JOINs

Most of the JOINs that you have executed in this course have been 1:1 JOINs. In this case, each row in each table has at most one match in the other table.



Another type of JOIN is an N:1 JOIN. Here, each row in one table matches potentially many rows in the other table.



Finally, an N:N JOIN is one where a group of rows in one table can match a group of rows in the other table. Note that in general, all other things equal, this type of JOIN produces a table with many more rows than either of the two (original) tables that are being JOINed.

