Short notes on sub queries.

If you read a few different notes and practice you become much better at the topic.

**SQL Topic: Subqueries**

You might have noticed that even with a complete query, there are many questions that we cannot answer about our data without additional post, or pre, processing. In these cases, you can either make multiple queries and process the data yourself, or you can build a more complex query using SQL subqueries.

Example: General subquery

Your company has a list of all Sales Associates(the table is called sales\_associates) and contains data on the revenue that each Associate brings in, and their individual salary. Times are tight, and you now want to find out which of your Associates are costing the company more than the average revenue brought per Associate.

First, you would need to calculate the average revenue all the Associates are generating:

SELECT AVG(revenue\_generated)

FROM sales\_associates;

And then using that result, we can then compare the costs of each of the Associates against that value. To use it as a subquery, we can just write it straight into the **WHERE** clause of the query:

SELECT \*

Checking to see if salary paid is bigger than the revenue that the sales associate generates. Company is paying more in salary than it gets in revenue

FROM sales\_associates

WHERE salary >

**(SELECT AVG(revenue\_generated)**

**FROM sales\_associates)**;

As the constraint is executed, each Associate's salary will be tested against the value queried from the inner subquery.

A subquery can be referenced anywhere a normal table can be referenced. Inside a **FROM** clause, you can **JOIN** subqueries with other tables, inside a **WHERE** or **HAVING** constraint, you can test expressions against the results of the subquery, and even in expressions in the **SELECT** clause, which allow you to return data directly from the subquery. They are generally executed in the same logical order as the part of the query that they appear in, as described in the last lesson.

Because subqueries can be nested, each subquery must be fully enclosed in parentheses in order to establish proper hierarchy. Subqueries can otherwise reference any tables in the database and make use of the constructs of a normal query (though some implementations do not allow subqueries to use **LIMIT** or **OFFSET**).

**Correlated subqueries**

A more powerful type of subquery is the *correlated subquery* in which the inner query references, and is dependent on, a column or alias from the outer query. Unlike the subqueries above, each of these inner queries need to be run for each of the rows in the outer query, since the inner query is dependent on the current outer query row.

Example: Correlated subquery

Instead of the list of just Sales Associates above, imagine if you have a general list of Employees, their departments (engineering, sales, etc.), revenue, and salary. This time, you are now looking across the company to find the employees who perform worse than average in their department.

For each employee, you would need to calculate their cost relative to the average revenue generated by all people in their department. To take the average for the department, the subquery will need to know what department each employee is in:

SELECT \*

FROM employees

WHERE salary >

(SELECT AVG(revenue\_generated)

FROM employees AS dept\_employees

**WHERE dept\_employees.department = employees.department**);

These kinds of complex queries can be powerful, but also difficult to read and understand, so you should take care using them. If possible, try and give meaningful aliases to the temporary values and tables. In addition, correlated subqueries can be difficult to optimize, so performance characteristics may vary across different databases.