**CHAPTER III – SORTING & FILTERING (to manipulate)**

**----------------------------------------------------------------------------------------------------------------**

**CHAPTER III – SUB QUERIES**

The whole idea of a subquery is to have a 2nd, 3rd or more queries nested within another query (Embed queries within other queries). Not only are subqueries helpful when it comes to getting information from multiple tables, but they're often used for adding additional criteria like a filtering criteria that's not in your current table from another table into your query.

**FROM THIS TO THIS (SUB QUERY)**

Subquery : Start with the most query (select the customer IDs from **orders table** where freight is over 100) . Most query is the main base for sub query, and going to filter down to that. This called innermost query. After that going to write the query to select my customer ID’, my company, my region to get bunch of information, And we get From **Customer table.** And we use **WHERE clause** to connect both information together.

A screenshot of a computer program

Description automatically generated A screenshot of a computer program

Description automatically generated

**Always perform the innermost SELECT portion first.** And slowly from there build and work backwards.

In that example, database performing two operations :

* Pre-selected is to Getting our customer **ID for the criteria** freight over 100.
* Going and pulling the other customer information and matching it up against the customer ID we have already pre-selected.
* There is **NO LIMIT** to the number of subquerie you can have.
* **Performance slows** when you nest too deeply.
* **Subqueris select can** only retrieve a single column.

A screenshot of a computer

Description automatically generatedA black background with white text

Description automatically generated

A black background with white text

Description automatically generated

A black background with white text

Description automatically generated

**BREAKDOWN :**

1. Starting with inner most query. It’s return a list of all order numbers that include a toothbrush.
2. Query number 2 (mid query) uses the results from step 1 (inner most). The …is the list of order numbers from query 1, the inner most query. This returns a list of Customer ID who have ordered a toothbrush.
3. Finally, the outer most query uses the results from query 2, the list of customer IDs. And this query gets the contact details of the customer who have ordered a toothbrush. And the result will show 2 columns, which are Customer name and customer details or customer contact.

Another example for subquery for TOTAL NUMBER of ORDERS placed by EVERY CUSTOMER (to get the customer name, and the region, and we want the total number of orders for those cutomers.)

**SUBQUERY RESULT (3 columns)**

A computer screen shot of a program

Description automatically generatedA screenshot of a computer

Description automatically generated

A black background with white text

Description automatically generatedA screen shot of a computer

Description automatically generated

**BREAKDOWN :**

1. Subquery counts the number of orders (**Orders**.customer\_id id) associated with each customer IN THE Customer table (**customers**.customer\_id) – (**Orders.**customer\_id = **customers**.customer\_id) . The key thing to note is HOW the subquery **correlates** with the outer query. WHERE clause has **Orders.**customer\_id = **customers**.customer\_id. This matches the orders with the customers based on the customer\_id column.
   1. For every row in example the customer being processed in the main query, the subquery fetches the count of orders from the orders tables that have the same customer\_id
   2. The results of this query is selected and labeled as orders.
   3. So, for each customer in your result, you will see the number of orders they may.
2. Main outer query operates on the customer table (FROM customers), and it retrieve two columns for every customer in the table, customer name and customer state. The results show in upper, has 3 columns, which are 2 columns from main outer query and 1 column from subquery(total count of orders associated with each customer). The list will also be sorted by customer name in ascending order (DEFAULT) due to customer name clause.

|  |  |
| --- | --- |
| **Pros (subquery)** | **Cons (subquery)** |
| 1. Flexibility (Versatile) 2. Readability (well structured-easier to understand) 3. Isolation of logic (encapsulate of logic) 4. Avoding joins (alternative to join) | 1. Performance (Especially if not optimized and a lot of nested) 2. Complexity (if you have many level nested of subquery) 3. Limited functionality 4. DBMS Support (not same level of performance) |

**JOIN..** Case if we store data in multiple table, and breaking down the data into different table is beneficial. How we break the data down into tables is often modeled **after a business problem or business process**. To get the data come together, we use keys. We are able to join those tables together **based on individually unique key.** This key serves as a link between the tables and critical piece to being able to join records and tables together.

Breaking data into tables :

1. Breaking data into tables allows us the opportunity to scale the information
2. How we break the data down into tables is often modeled after a business problem or a business process.
3. When we split out the information into separate tables, it helps us store the data more efficiently, so we're not duplicating records.
4. This is an accurate statement. A key serves as a link between the tables is critical to being able to join records and tables together. However, that is not a benefit of breaking data into tables as the question asked.
5. When we want to update a record or change something, when that data isn't touching all different levels, it's a lot easier to manipulate within tables.

WHAT IS JOIN :

1. JOINS are what associate the correct records from each table on the fly.
2. JOINS also allows data retrieval from multiple tables in one query.
3. JOINS are not physical entity (not creating permanent), they persist for the duration of the query execution.

**CARTESIAN (Cross) JOIN.. VERY SIMPLE JOINS (not matching on anything, just simply multiplying)**

Allows to take each record from the 1st table and match it with all of the records from the 2nd table. Each row from the first table joins **with all the rows of another table**. 1st table contains X rows, and the 2nd table connect Y rows. You will have the end result of X times Y. These are not frequently used, but they are helpful in specific cases. Because of multiplying for the results, it will increase the size of your data, and potential to return incorrect results, because **Cross join not matching it on anything.**

|  |  |
| --- | --- |
| A diagram of a cross-join  Description automatically generated | A black background with white text  Description automatically generated |
| A black rectangular object with a white background  Description automatically generated |

Product\_name from the product’s table

Unit price from the product’s table

Company\_name from the supplier’s table.. THEY are from separate tables (list from suppliers, and also from products). Between that JOIN TABLE FROM Suppliers (table) and then cross joined with Products (table).

**INNER JOINS(FREQUENTLY USED)**

Is used to select records that have matching values in both tables. **KEYS become really important** in tables.

Matching in Both tables (table 1 & table 2)

|  |  |
| --- | --- |
| A diagram of a table  Description automatically generated |  |
|  |

Grab it from suppliers, the type of join we are going to do inner join with products and then what you are listing is really that key. The connection between the key, the two tables, is the Supplier ID in the supplier table (**Suppliers.*supplierid***), and the supplier ID in the product table (**Products. *Supplierid***). Supplier ID equal supplier ID, but before that I have pre-qualified which supplier ID. Supplier ID from the supplier table that equals the supplier IDs from the products table. Company name is something that was listed both in products and suppliers table, it wouldn’t know which table to pull it from. So we need to prequalify it by listing out suppliers before the company name , so that I would know for sure where to pull it from.

INNER JOIN syntax : (if you don’t need a join, don’t join).

1. Join type is specified (INNER JOIN)
2. Join condition is in the FROM Clause and uses the ON Clause.
3. Joining more tables together affects overall performance.
4. Can join multiple tables, no limit.
5. List all the tables, and then define conditions.

JOINS MUST Be SPECIFIC about which table you want it to comefrom.

**SELECT orderID column from Orders Table, CompanyName column from Customers Table, and LastName from Employees Table**

(Will joins within three tables to do that)

FROM I have orders with an Inner join on Customers and that’s going to be joined by the Customer ID.

(Again I just listed out my prequalifying name with o.CustomerID and c.CustomerID). => Get *Order ID and my CompanyName*.

Another INNER JOIN for employees table and we are going to join that to the orders through the EmployeeID.

**ALIASES**

We’ve used aliases when we are aggregating a field. We can do a similar thing when we’re using tables and joining tables together. **Because it make it a lot easier to read and write.** An alias is helpful because it can help you **by just shortening names and simplifying** how we’re pre-qualifying them. It’s just ONLY stored for that duration of the query, not rewriting the name of the table or rewriting anything. We can see in the bottom that we want to get column vendor id from table vendors also in Products table. We can make it shorter as follow Vendors table as V and Products table as p. Aliases keep it in a logical fashion., with using proper abbreviation names.

A screen shot of a computer

Description automatically generated A computer screen shot of a black background

Description automatically generated

**SELF JOINS (A table is joined to itself).**

Is Join the original table to itself. For example ON THE Employee table, we want to find all the employees and their respective managers. As the employee table contains a reports to field, which is foreign key referencing the employee ID in the same table.

A computer screen shot of a black background

Description automatically generated

**LEFT JOINS, RIGHT JOINS, and FULL OUTER JOINS (ADVANCED JOINS).**

**LEFT JOINS**

Starting with Left Joins. Most database management system including

Returns all records from the left table (table 1) and the matched records from the right table (table 2).. The result is **NULL from the right side** when there is no match.

I still want everything from the customer table , I don’t’t care if they didn’t have an order but if they did and bring it also together and bring it in one order table.

**RIGHT JOINS**

Similar with left join except in this instance from the right table (table 2) and the matched record from the left table (table 1). The result is **NULL from the left side** when there is no match.

Be careful if you are using left and right joins, and which table you are listing as coming first and make sure that relates to the left or right in the type of join you use.

#Difference between right and left is the order the tables are relating. LEFT JOINS can be turned into right joins by reversing the order of the tables.

**FULL OUTER JOIN**

Return all records when there is a match in either left (table1) or right (table 2) table records.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **FROM CUSTOMERS (table)-prequalifying** | **FROM ORDERS (table)-prequalifying** | Whether there's a matching 1 or matching 2 |
| Have a customer who haven’t place an order.  -**LEFT JOIN** Order **ON** Customer- | Have a order who isn’t from Customer (table).  -**RIGHT JOIN** Customer **ON** Order- | Just return and Give me everything |

**UNIONS**

UNION Operator is Used to combine the results-set of two or more SELECT statements. Each select statement within UNION must have the same number of columns. Columns must have similar data types. The columns in each SELECT statement must be in the same order.

UNION operator is used to combine the results of two or more queries or table sets into one table and one statement. Each SELECT statement gets unioned with each other and kind of stacked on top of each other. Analogy : Want to build a building with 10 floor, within 2 people. EVERY People must or can build 5 floors. 1st Floor – 5th Floor with A, and 6th – 10 th Floor with B. So the building must be the same size and also have similar types of data.

SELECT customers from Germany, and we want what cities in Germany our customers are from. We also then want to know what cities we have suppliers in. Separate two queries , means build 2 SELECT .. FROM clause..(from customers table and also suppliers table, but same want to know what cities-SELECT CITY, from GERMANY – WHERE Country = ‘Germany’).

A computer screen with white text

Description automatically generated A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. How many albums does the artist Led Zeppelin have?

SELECT albums.Title, ARTISTS.Name

FROM albums INNER JOIN artists ON ARTISTS.ArtistID = albums.ArtistID

WHERE ARTISTS.Name = 'Led Zeppelin';

1. Create a list of album titles and the unit prices for the artist "Audioslave".

SELECT artists.Name, albums.Title, tracks.UnitPrice

FROM ( (albums INNER JOIN artists ON albums.ArtistId = artists.ArtistId)

INNER JOIN tracks ON albums.AlbumID  = tracks.AlbumID

)

WHERE artists.Name = 'Audioslave'

;

1. Find the first and last name of any customer who does not have an invoice. Are there any customers returned from the query?

NO

1. Find the total price for each album.

SELECT albums.AlbumId, albums.Title, SUM(tracks.UnitPrice)

FROM albums INNER JOIN tracks ON albums.AlbumId = tracks.AlbumId

WHERE albums.Title = 'Big Ones'

;

A diagram of a diagram

Description automatically generated with medium confidence

SQL Vs PYTHON :

<https://mode.com/blog/learning-python-sql>

A white background with black text

Description automatically generated A white paper with black text

Description automatically generated

A white text on a white background

Description automatically generated

A white background with black text

Description automatically generated

A screenshot of a computer

Description automatically generated

A white background with black text

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Using a subquery, find the **name**s of all the **tracks** for the album "Californication".

SELECT AlbumId, Name, TrackId

FROM Tracks WHERE AlbumId in (

SELECT AlbumId

FROM ALBUMS WHERE Title = 'Californication');

1. Find the total number of invoices for each customer along with the customer's full name, city and email.

SELECT  Customers.FirstName, Customers.LastName,

    Customers.City,  Customers.Email, (Invoices.Total)

FROM Invoices INNER JOIN Customers ON Customers.CustomerId = Invoices.CustomerId

WHERE Customers.City = 'Prague'

 ;

1. Retrieve the track name, album, artistID, and trackID for all the albums.

SELECT Tracks.Name, TrackId, Tracks.AlbumId, Albums.Title, Albums.ArtistId

FROM Tracks INNER JOIN Albums ON Albums.AlbumId = Tracks.AlbumId;

1. Retrieve a list with the managers last name, and the last name of the employees who report to him or her.

SELECT \* FROM Employees; -- ID 6. Michael Reports to 1…

1. Find the name and ID of the artists who do not have albums.

SELECT Artists.Name, Albums.ArtistId

FROM Artists

LEFT JOIN Albums ON Artists.ArtistId = Albums.ArtistId

WHERE Albums.ArtistId IS NULL;  --LEFT JOIN

1. Use a UNION to create a list of all the employee's and customer's first names and last names ordered by the last name in descending order.

SELECT FirstName, LastName FROM Employees

UNION

SELECT FirstName, LastName FROM Customers  ORDER BY LastName DESC;

1. See if there are any customers who have a different city listed in their billing city versus their customer city. ..**NO**

**---------------------------------------------------------------------------------------------------------------------------------------------------------------------------**

**---------------------------------------------------------------------------------------------------------------------------------------------------------------------------**