WEEK 3

Subqueries and Joins

Subqueries

- Subqueries are queries embedded into other queries.
- Subqueries merge data from multiple sources together.
- Helps with adding other filtering criteria.

Example: Need to know the region each customer is from who has had an order with freight over 100.

- Start with the innermost query.
- Subquery's SELECT can only select **one** column. You cannot select multiple columns and bring them all to the overall query.

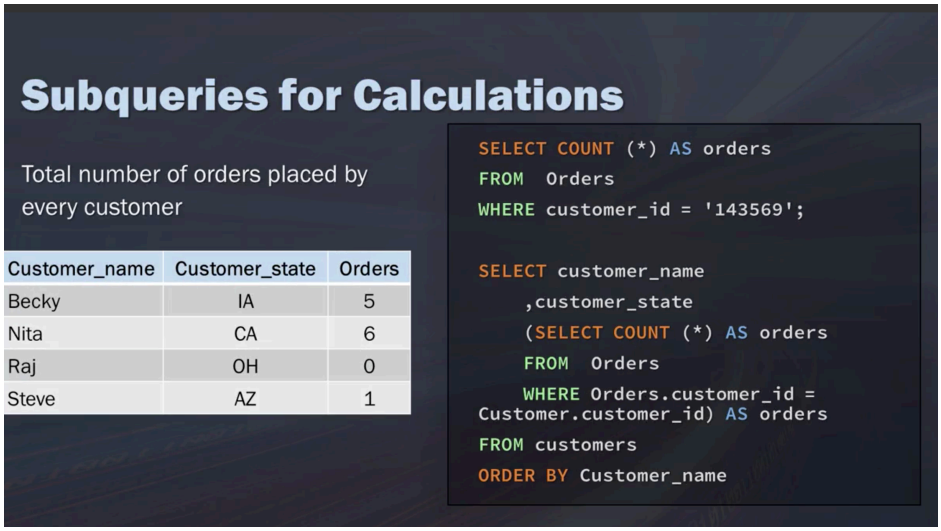
```
SELECT customerID, CompanyName, Region
FROM Customers
WHERE customerID IN (SELECT CustomerID
                     FROM Orders
                     WHERE Freight>100)
```

The innermost query is used to filter for customers that we are getting back.

- Having too many nested subqueries makes the process slow.

Subqueries for Calculation

In this example, we want to get the company name and the region, and we want a total number of orders for these customers.



Joins

Benefits of breaking data into pieces

- Efficient Storage
- Easier manipulation
- Greater scalability
- Logically models a process
- Tables are related through common values(keys)

The first thing we want to go over with joins is that these joins are what associate the correct records from each table on the fly.(what does this mean?)
To explain, we'll go back to our example of you have a table that list the different customer demographics but you want to know all of the orders that the customer has made. So, a join will allow you to associate the correct customer information with the correct order, quickly and on the fly with your query.

- Joins are not permanent. They persist for the duration of the query execution.

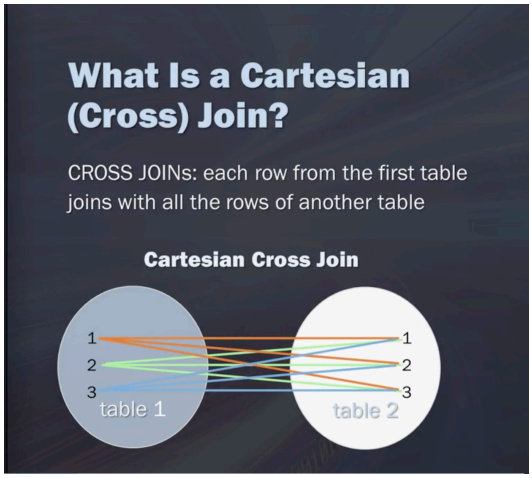
Joins allow data retrieval from multiple tables in one query.

Cartesian (Cross) Joins

Take each record from first record and match it with all the records from the second rows.
Loot at this link.

```
SELECT prod_name,
       unit_price,
       company_name
FROM Suppliers CROSS JOIN Products
```

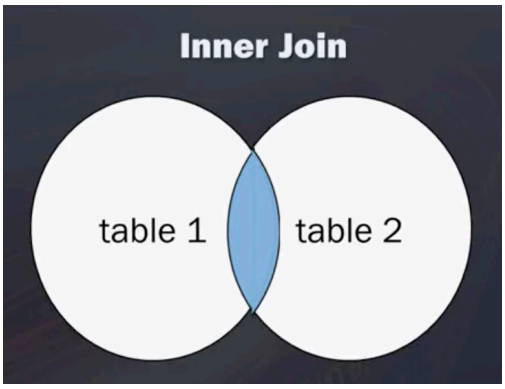
```
SELECT Vendor_name, prod_name, prod_price
FROM   Vendors, Products
WHERE  Vendors.vendor_id = Products.vendor_id
```



If table one has n1 rows and second row has n2 rows, your join will have n1*n2 rows.

Inner Join

- Define inner join
- Explain when and how to use inner join
- Pre-qualify column names to make your SQL code that much cleaner and efficient



The inner join is used to select records that have matching values in both tables.

How to do this?
So again, we're still listing out the columns that we want from both tables.

```
SELECT suppliers.company_name,
       product_name,
       unit_price
FROM Suppliers INNER JOIN Products
ON Suppliers.supplier_id = Products.supplier_id
```

- Join condition is in the FROM clause and uses ON clause.
- Joining more tables together will effect performance.
- There is no limit on the number of tables to join.
- List all tables then define condition.

tableName.Column_Name is called pre-qualification. We are telling which table to use to grab the given column.

Example:

```
SELECT o.order_id, c.company_name, e.last_name
FROM   (orders o INNER JOIN Customers c
         ON o.customer_id = c.Customer_id )
INNER JOIN Employees e ON o.Employee_id = e.Employee_id
```

Aliases and Self Joins

What is an alias?
- SQL aliases give a table or a column a temporary name.
- Make column names more readable
- An alias exist only for duration of a query.

```
SELECT column_name
FROM table_name AS alias_name
```

```
SELECT Vendor_name, prod_name, prod_price
FROM   Vendors, Products
WHERE  Vendors.vendor_id = Products.vendor_id
```

If we use aliases we can write the above query as follows:

```
SELECT Vendor_name, prod_name, prod_price
FROM   Vendors AS v, Products AS p
WHERE  v.vendor_id = p.vendor_id
```

Self Joins

```
SELECT column_name(s)
FROM   table T1, table T2
WHERE  condition
```

Example: Match customers from the same city
Take the table and treat it like two separate tables
Join the original table to itself:

```
SELECT A.customer_name AS customer_name1,
          B.customer_name AS customer_name2
          A.city
FROM   Customers A, Customers B
WHERE A.customer_id =  B.customer_id  AND A.city = B.city
ORDER BY A.city
```

Look at this example.

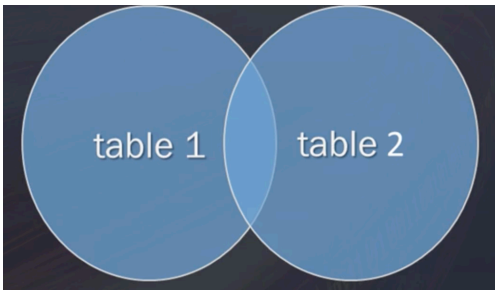
Left, Right and Full Outer Join

SQLite only does left join. Right and full outer joins are used in other databases.

Left join returns all records from the left table (table 1) and the matched records from the right table (table2). The result is NULL from the right side, when there is no match.

If we do inner join on the tables given in the image on the right here, we will be missing the customers who have not placed an order yet. The left join will look at the customer table, and bring in all the orders placed by the customers. It tells to SQL, hey, I do not care in a customer placed an order or not, I want all the customers, and if they did place an order, then bring also all together and bring in the order table.

Right join is the same as left join except the order of tables are switched.
It returns all the records from the right table, and matched records from the left table. The result is NULL from the left side, when there is no match.



Full Outer Join: Return all the records when there is a match either in left table, or right table records.

The following query will select all customers and any orders they might have. We want the list of all customer names, whether they have an order or not, but if they do have an order, we want that information as well.

```
SELECT c.customer_name, o.order_id
FROM customer c LEFT JOIN Orders o ON c.customer_id = o.customer_id
ORDER BY c.customer_name
```

The following will select all the customers and all the orders. (Full Join)

```
SELECT customers.customer_name, orders.order_id
FROM  Customers FULL OUTER JOIN Orders
ON customers.customer_id = orders.customer_id order_id
ORDER BY customers.customer_name
```

Union (distinct. It will not give you duplicates)

- The UNION operator is used to combine the result-set of two or more SELECT statements (queries) into one table.
- 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.

```
SELECT column_names FROM table1
UNION
SELECT column names FROM table2
```

Example:

```
SELECT city, country FROM customers
WHERE Country = 'Germany'
UNION
SELECT city, country FROM  suppliers
WHERE  Country = 'Germany'
ORDER BY city
```

Take a look here: Union Example.