Week 2

Filtering, Sorting and Calculating

Clauses and Operators:
WHERE, BETWEEN, IN, OR, NOT, LIKE, ORDER BY, GROUP BY

WildCards:
Wildcards: allow more precise search

Math Operators.

Aggregate math functions: AVERAGE, COUNT, MAX, MIN

Filtering
Narrow the data, analysis to get specific about the data we want.

Goals
- Describe the basics of filtering,
- Use WHERE clause with common operators
- Use BETWEEN clause
- Explain concept of NULL Value

Why Filter?
- Reduce the number of data we want to work and retrieve, therefore, increase speed and performance,
- reduce strain on the client application,
- governance limitation,

To do filtering we use WHERE clause which comes after SELECT and FROM:

```
SELECT column_name, column_name
FROM table_name
WHERE column_name operator value
```

Operators:

<>	Not equal
>, >=, <=, <	
BETWEEN	Between range, inclusive
IS NULL	

```
SELECT ProductName, unitPrice, SupplierId
FROM Products
WHERE productName = ‘Tofu’
```

```
SELECT ProductName, unitPrice, SupplierId
FROM Products
WHERE unitPrice >= 75
```

```
SELECT ProductName, SupplierId
FROM Products
WHERE unitPrice >= 75
```

Another way, to look for non matches:

```
SELECT ProductName, unitPrice, supplierId
FROM Products
WHERE ProductName <> ‘Alice Mutton’
```

```
SELECT ProductName, unitPrice, supplierId, unitsInStock
FROM Products
WHERE unitsInStock BETWEEN 15 AND 80
```

```
SELECT ProductName, unitPrice, supplierId, unitsInStock
FROM Products
WHERE unitsInStock IS NULL.
```

If you want to look for something where there is just no information for that column, that's where you would want to simply use is null.

IN, OR, NOT

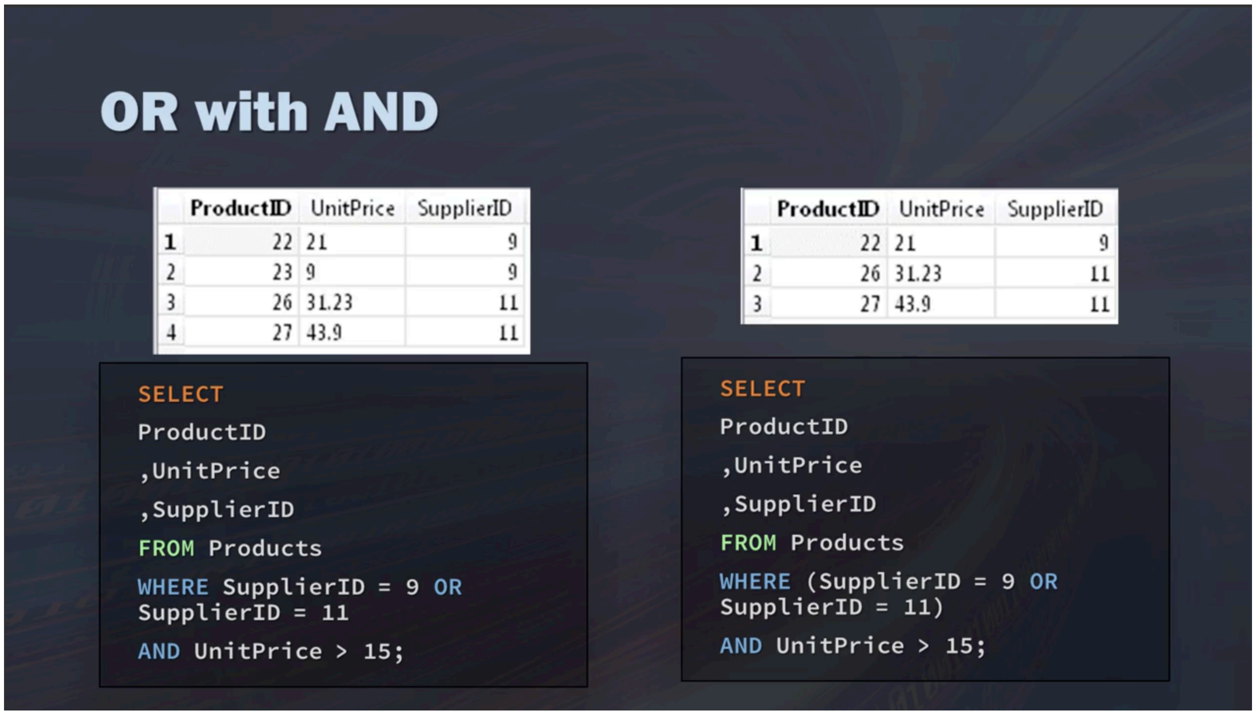
To use IN we have to specify a range of conditions. Comma delimited list of values enclosed in ().

SELECT productId, unitPrice, suppliedId
FROM Products
WHERE suppliedId IN (9, 10, 11)

OR Operator

A database management system will not evaluate the second condition if the first condition is met.
If you want both conditions to be checked you use AND.

Once it finds a product with name Tofu, it will not return anything with name Konbu !!!!
SELECT ProductName, unitPrice, supplierId, productId, unitsInStock
FROM Products
WHERE productName = 'Tofu' OR 'Konbu'



OR with AND

	ProductID	UnitPrice	SupplierID
1	22	21	9
2	23	9	9
3	26	31.23	11
4	27	43.9	11

```
SELECT
ProductID
,UnitPrice
,SupplierID
FROM Products
WHERE SupplierID = 9 OR
SupplierID = 11
AND UnitPrice > 15;
```

	ProductID	UnitPrice	SupplierID
1	22	21	9
2	26	31.23	11
3	27	43.9	11

```
SELECT
ProductID
,UnitPrice
,SupplierID
FROM Products
WHERE (SupplierID = 9 OR
SupplierID = 11)
AND UnitPrice > 15;
```

Select list of employees that are not from London or Seattle:
SELECT *
FROM Employees
WHERE NOT City='London' AND NOT City='Seattle'.

WildCards (are slow.)

wildcard	action
%Pizza	Grabs anything ending with pizza
%Pizza%	Grabs anything before and after pizza
Pizza%	Grabs anything after pizza
S%E’	Grabs anything that starts with S and ends with E
t%@gmail.com’	

These produce the same result:
WHERE size LIKE ‘%pizza’ and WHERE size LIKE ‘_pizza’

Another wild card is bracket to specify a character in a specific location within a string. [not in SQLite]

Sorting with ODER BY

Goals: Discuss the importance of sorting data, explain some rules of using ORDER BY

SELECT something
FROM database
ORDERY BY characteristic

Rules:

- Takes name of one or more columns
- Add a comma after each additional column
- Can sort by a column not retrieved
- Must always be the last clause in a SELECT statement

You can sort by column position or names of columns:

ORDER BY 2, 3
2 means 2nd column and 3 means 3rd column.

ORDER BY unitPrice, productName.

If you're using order by descending (DESC) and have unit price, it's not going to do it for all of the columns after the descending. You have to specify each individual columns for ascending (ASC) and descending, if you want it that way.

DESC or ASC only applies to the column names it directly precedes.

Math Operations

```
SELECT productId,
        UnitsOnOrder,
        unitPrice,
        UnitsOnOrder * unitPrice AS total_order_cost
FROM Products
```

```
SELECT productId,
        Quantity,
        unitPrice,
        Discount,
        (unitPrice - Discount) * Quantity AS total_cost
FROM orderDetails
```

Aggregate Functions

Aggregate functions are used to summarize data.
AVERAGE, COUNT, MIN, MAX, SUM
DISTINCT

AVG()	Averages a column of values
COUNT()	Counts the number of values
MIN()	Finds minimum values
MAX()	
SUM()	Sums the column values

```
SELECT AVG(unitPrice) AS avg_price
FROM Products
```

NOTE: Rows containing no or NULL values will be ignored by the AVG() function.

COUNT(*) counts all rows in a table containing values or NULL values.
COUNT(column) Counts all the rows in a specific column ignoring NULL Values.

```
SELECT COUNT(*) AS total_customers
FROM Customers
```

```
SELECT COUNT(CustomerId) AS total_customer
FROM Customers
```

```
SELECT MIN(unitPrice) as min_prod_price, MAX(unitPrice) as max_prod_price
FROM Products
```
Null values will be ignored by min and max functions.

```
SELECT SUM(unitPrice) AS total_prod_price
FROM Products
WHERE suppliedID = 23
```

Distinct

```
SELECT COUNT(DISTINCT CustomerID)
FROM Customers
```

- If the word DISTINCT is not specified, ALL is assumed! For example, you may have a customer who's in a table multiple times. If you're simply doing a count on your customer IDs and you don't distinguish to count just the distinct customer IDs, you may be getting duplicate records in there.
- Cannot use DISTINCT on COUNT(*)

GROUPING DATA

GROUP BY and HAVING.

We want to know the number of customers we have and we want to know this by each region.

In the following if we do not have group by, we will get an error:

```
SELECT Region, COUNT(CustomerID) AS total_customers
FROM Customers
GROUP BY Region
```

- We can group by multiple columns
- Every column in your SELECT statement must be present in a GROUP BY clause, except for aggregated calculations
- Nulls will be grouped together if your GROUP BY column contains NULLs

HAVING Clause - Filtering for Groups

- Where does not work for groups
- Where filters on rows
- Instead use HAVING clause to filter for groups.

In this example, we want the count of orders for customers. But we only want to see the total orders for the customers who've had more than two orders. So to do this, we're going to select our customer IDs, then we're going to count all of the records as orders.

```
SELECT CustomerID, COUNT(*) AS orders
FROM Orders
GROUP BY CustomerID
HAVING COUNT (*)>=2
```

- WHERE filters before the data is grouped.
- HAVING filters after data is grouped.
- Rows eliminated by WHERE clause will not be included in the groups.

Another e.g.

```
SELECT supplierID, COUNT (*) AS num_prod
FROM Products
WHERE unitPrice>=4
GROUP BY supplierID
HAVING COUNT(*)>=2
```