SELECT Statements

The SELECT statement

- Used to query the database
- Retrieves information from the database
- Does not change anything in the database

- Made up of two parts (clauses) the SELECT and the FROM

```
SELECT information to retrieve
FROM  tablename;
```

SELECT what… FROM where

- We can SELECT column values… FROM a tablename

Show the product name and lines from the products table

```
SELECT productName, productLine
FROM products;
```

- We can SELECT all values…FROM a tablename

Show all the information in the products table.

```
SELECT *
FROM products;
```

Both the SELECT and FROM are mandatory in a basic select statement.

Filtering the information

- WHERE – specifies a condition
- The WHERE clause limits the change to the areas that we specifically wanted.
- It acts as a filter
- It is optional (without it all values will be selected or update or deleted)
- It is all about true conditions

Show the product name and line for all products that are motorcycles

```
SELECT productName, productLine
FROM products
WHERE productLine = 'Motorcycles';
```

Comparison Operators

=	means equal to
<>	means not equal to
<	means less than
>	means greater than
<=	means less than or equal to
>=	means greater than or equal to

Select all the information for products that are not motorcycles

```
SELECT *
FROM products
WHERE productLine <> 'Motorcycles';
```

Select everything that costs less than 40 dollars

```
SELECT *
FROM products
WHERE MSRP < 40;
```

What about strings?

Select all the customer information where the last name starts with the letter a and b

```
SELECT *
FROM customers
WHERE contactLastName < 'C';
```

- For numbers it is the standard numeric sequence, for strings it is alphabetical.

LIKE operator

- The LIKE operator implements pattern matching

WILDCARDS

- The % looks for one or more characters before
- The _ looks for one character

Select all the customer information for customers who's last name starts with the word Ash

```
SELECT *
FROM customers
WHERE contactLastName LIKE 'Ash%';
```

BETWEEN

- Tests a range

Get all the information from the products table for items that can be bought for 100 – 200 dollars

```
SELECT *
FROM products
WHERE buyPrice BETWEEN 100 AND 200;
```

COMPOUND CONDITIONS

- The AND … OR.. Conditions

Select all information where the vendor is Second Gear Diecast for Vintage Cars

```
SELECT *
FROM products
WHERE productVendor = 'Second Gear Diecast' AND productLine  =
  'Vintage Cars';
```

returns results only where both conditions are met in the same record

Select all the information for products that are classic cars and vintage cars

```
SELECT *
FROM products
WHERE productLine  = 'Classic Cars' OR productLine  = 'Vintage Cars';
```

Returns results where ***either*** of the conditions are met

Compounding things a little further…

- Combining AND and OR… AND takes precedence over OR

```
SELECT *
FROM products
WHERE productLine = 'Classic Cars' OR productLine = 'Planes' AND MSRP < 100;
```

This statement is evaluated like:

```
SELECT *
FROM products
WHERE productLine = 'Classic Cars' OR (productLine = 'Planes' AND MSRP < 100);
```

- Use parenthesis to get the results that you want

```
SELECT *
FROM products
WHERE (productLine = 'Classic Cars' OR productLine = 'Planes') AND MSRP < 100;
```

IN condition

- Used to check a list of values

```
SELECT *
FROM products
WHERE productLine IN ('Planes', 'Trains', 'Ships');
```

- Or you can see what is NOT in a list of values

```
SELECT *
FROM products
WHERE productLine NOT IN ('Planes', 'Trains', 'Ships');
```

Creating Order

ORDER BY

- We have talked about the records in a table being unordered – their position in the table does not signify importance or order
- We can control the output – the order that information is displayed upon retrieval
- The ORDER BY clause helps to ensure that the result set is returned in a specified sequence

ORDER BY

- It is **ALWAYS** the last clause in a SELECT statement
- Consists of the ORDER BY clause followed by the sort key

```
SELECT columns
FROM table
ORDER BY sort_column [ASC|DESC];
```

- The sort_column is the sort key – the column that information will be ordered by
- ASC means ascending (lowest to highest)
- DESC means descending (highest to lowest)
- If no sort direction is listed, ascending is the default

ORDER BY

```
SELECT productName, productLine, productVendor, quantityInStock
FROM products
WHERE productLine = 'Trains';
```

productName	productLine	productVendor	quantityInStock
Collectable Wooden Train	Trains	Carousel DieCast Legends	6450
1950's Chicago Surface Lines Streetcar	Trains	Gearbox Collectibles	8601
1962 City of Detroit Streetcar	Trains	Classic Metal Creations	1645

ORDER BY

Show the trains we have – show the product name, the line, the vendor and the quantity in stock from the least to most (ASC is the default)

```
SELECT productName, productLine, productVendor, quantityInStock
FROM products
WHERE productLine = 'Trains'
ORDER BY quantityInStock;
```

productName	productLine	productVendor	quantityInStock
1962 City of Detroit Streetcar	Trains	Classic Metal Creations	1645
Collectable Wooden Train	Trains	Carousel DieCast Legends	6450
1950's Chicago Surface Lines Streetcar	Trains	Gearbox Collectibles	8601

ORDER BY

Show the trains we have – show the product name, the line, the vendor and the quantity in stock from the most to the least

```
SELECT productName, productLine, productVendor, quantityInStock
FROM products
WHERE productLine = 'Trains'
ORDER BY quantityInStock DESC;
```

productName	productLine	productVendor	quantityInStock
1950's Chicago Surface Lines Streetcar	Trains	Gearbox Collectibles	8601
Collectable Wooden Train	Trains	Carousel DieCast Legends	6450
1962 City of Detroit Streetcar	Trains	Classic Metal Creations	1645

ORDER BY – multiple columns

Show the products we have – show the product name, the line, the vendor and the quantity in stock from the least to most and show them by category in alphabetical order

```
SELECT productName, productLine, productVendor, quantityInStock
FROM products
ORDER BY productLine ASC, quantityInStock DESC;
```

1928 British Royal Navy Airplane	Planes	Classic Metal Creations	3627
1900s Vintage Tri-Plane	Planes	Unimax Art Galleries	2756
P-51-D Mustang	Planes	Gearbox Collectibles	992
F/A 18 Hornet 1/72	Planes	Motor City Art Classics	551
The USS Constitution Ship	Ships	Red Start Diecast	7083
The Queen Mary	Ships	Welly Diecast Productions	5088
1999 Yamaha Speed Boat	Ships	Min Lin Diecast	4259
HMS Bounty	Ships	Unimax Art Galleries	3501

ORDER BY

Insert some records in a table – empty string, word, Word, 2 and 123

```
SELECT *
FROM order_test
ORDER BY order_string;
```

4	
1	123
5	2
2	word
3	Word

- Numerical data sorts from smaller to larger
- String data sorts alphabetically
- Temporal data sorts chronologically

Sorting

- Three(3) factors affect sorting speed
  1. The number of rows selected (more rows, slower sorting)
  2. Number of columns in the ORDER BY clause (more columns, slower sorting)
  3. Length of columns (longer the columns, slower the sorting speed)

- Recommend restricting the sort to the minimum number of rows required.
- The sort column does not have to appear in the list of columns, but without it there is no reference for the order

Dealing with NULLS

Testing for Nulls

- Remember that Nulls represent missing or unknown values
- Nulls do not satisfy specific conditions
- LIKE, BETWEEN, IN and WHERE can’t find NULLS
- IS NULL will check for Null conditions

Show all information where there is no state value

```
SELECT *
FROM offices
WHERE state IS NULL;
```

Show all information where there is a state value

```
SELECT *
FROM offices
WHERE state IS NOT NULL;
```

Checking for nulls

- The COALESCE() function returns the first non-null expression
- Same as checking with a Case expression if the value is not null and if it is, then apply the else statement

```
SELECT city, phone, addressLine1, COALESCE(state, 'N/A'), country, postalCode
FROM offices;
```

Matching Patterns

Matching Patterns

- Like is useful if you don't know the exact value – you retrieve information based on partial information
- LIKE only works with character strings – not numbers or datetimes

Show all the employees with a first name that has any two letters followed by rr and any other characters.

```
SELECT firstName, lastName
FROM employees
WHERE firstName LIKE '__rry%';
```

Using an Alias

ALIASES

```
SELECT firstName AS First, lastName AS Last
FROM employees;
```

First	Last
Diane	Murphy
Mary	Patterson
Jeff	Firrelli
William	Patterson
Gerard	Bondur
Anthony	Bow

ALIASES

```
SELECT city, phone, addressLine1, COALESCE(state, 'N/A'), country, postalCode
FROM offices;
```

city	phone	addressLine1	COALESCE(state, 'N/A')	country	postalCode
San Francisco	+1 650 219 4782	100 Market Street	CA	USA	94080
Boston	+1 215 837 0825	1550 Court Place	MA	USA	02107
NYC	+1 212 555 3000	523 East 53rd Street	NY	USA	10022

```
SELECT city, phone, addressLine1 AS address, COALESCE(state, 'N/A') AS state ,
country, postalCode AS postal
FROM offices;
```

city	phone	address	state	country	postal
San Francisco	+1 650 219 4782	100 Market Street	CA	USA	94080
Boston	+1 215 837 0825	1550 Court Place	MA	USA	02107

Removing Duplicates

Eliminating Duplicate Rows

- DISTINCT allows you to eliminate duplicate values

Show all the products

```
SELECT productLine
FROM products;
```

productLine
Classic Cars
Classic Cars
Classic Cars
Classic Cars
Classic Cars

```
SELECT DISTINCT productLine
FROM products;
```

productLine
Classic Cars
Motorcycles
Planes
Ships
Trains
Trucks and Buses
Vintage Cars

All nulls are considered duplicates and only one null will be returned