ORDER BY

The optional *ORDER BY* clause is how we tell SQL to sort the results. Its position, right at the end of the select statement, is accurate.

Sorting is the *very last thing* to happen to our result set. All the filtering and calculating and manipulating has been done by the time SQL gets around to sorting the data, because sorting can be very time-consuming; imagine if SQL sorted a million rows from a table before realising that *none* of the rows meet the WHERE clause!

If we do not specify an *ORDER BY* clause, SQL does not guarantee the order in which rows will be returned.

By default, columns are sorted in ascending order (1 to 10, a to z, etc.) but it is possible to tell SQL to sort them in descending order (10 to 1, z to a and so on) using the *DESC* keyword.

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

As sorting happens right at the end, we can also order by column aliases.

SELECT DISTINCT

The SELECT DISTINCT statement is used to return only unique values.

Inside a table, a column often contains many duplicate values; and sometimes you only want to list the unique values.

SELECT TOP

The SELECT TOP clause is used to specify the number of records to return.

```
SELECT TOP number|percent column_name(s)
FROM table_name
WHERE condition;
```

Please note that this TOP syntax is native to Microsoft SQL and differs for other databases.

Exercise 1: Basic ORDER BY clauses

Northwind Traders are attempting to learn more about their product pricing. They ask you to produce a list of current products, sorted by category and then by unit price in reverse order.

In this exercise, you will use basic ORDER BY clauses.

Task 1: Create a report that only returns current products

1. Create a new query and save it with a name of "CurrentProducts.sql".
2. Write a report that uses the Northwind database and displays the ProductID, ProductName, CategoryID, Discontinued and UnitPrice columns from the dbo.Products table and only selects rows with a Discontinued value of 0.
3. Execute the query and verify that it returns 69 rows.

Task 2: Write a report that sorts current products by category

1. Modify your existing report.
2. Add an ORDER BY clause to the query that sorts the results based on the CategoryID column.
3. Execute the query and verify that it returns the 69 rows but that all the products with a CategoryID of 1 are together, all those with CategoryID 2 are together, etc.

Task 3: Write a report that sorts current products by category and unit price

1. Modify your existing report.
2. Add an additional statement to the ORDER BY clause that sorts the results in reverse order on UnitPrice.
3. Execute the query and verify that your 69 rows are still sorted by CategoryID but that within those categories the most expensive products are displayed first.

Exercise 2: Sorting on calculated columns

Northwind are still trying to get a handle on their stock levels. They are really pleased with a stock report you wrote earlier but now that they know you can sort data, they'd like you to make it easier to see which products have the highest value of future stock.

In this exercise, you will work on an earlier query and add an ORDER BY clause to it.

Task 1: Re-use an existing query

1. Create a copy of the script file "StockList.sql".
2. Rename the new file "SortedStockList.sql".

NOTE: If you can't find the query file, the SQL is reproduced below:

```sql
SELECT
        ProductID
,       ProductName
,       UnitPrice
,       UnitsInStock
,       UnitsOnOrder
,       UnitPrice * UnitsInStock AS CurrentStockValue
,       (UnitsInStock + UnitsOnOrder) * UnitPrice AS FutureStockValue
FROM Northwind.dbo.Products
```

3. Open the query in SSMS.
4. Execute the query and verify that it returns 77 rows.

Task 2: Write a report that sorts on a calculated column

1. Modify the existing report.
2. Add an order by clause that sorts on the value of the calculated "FutureStockValue" column, in reverse order.
3. NOTE: There are three different ways of achieving this.
4. Execute the query and verify that the first product returned is Cote de Blaye, product id 38.

Exercise 3: SELECT DISTINCT

Northwind now asks you to produce a list of the unique countries in which they have customers.

In this exercise, you will use the DISTINCT modifier on a SELECT statement.

Task 1: Create a report that selects the customer’s countries

1. Create a new query and save it with a name of “CustomerCountries.sql”.
2. Write a report that uses the Northwind database and selects the Country column from the dbo.Customers table.
3. Execute the query and verify that it returns 91 rows.

Task 2: Select distinct rows

1. Modify the existing report.
2. Modify the SELECT statement so that it looks for distinct values only.
3. You’re expecting 21 rows now. Notice that the countries are sorted alphabetically as well.

Exercise 4: SELECT TOP

Northwind now asks you to produce a list of all the top ten most expensive products based on unit price.

In this exercise, you will use the TOP modifier. The main task is to create a report that selects the top ten rows from the dbo.Products table sorted on reverse value of unit price.

Task 1: Create a report of the top ten most expensive products

1. Create a new query and save it with a name of "MostExpensiveProducts.sql".
2. Write a report that uses the Northwind database and selects the ProductID, ProductName and UnitPrice columns from the dbo.Products table.
3. Order the results in descending order of UnitPrice.
4. Restrict the results to the first ten results.
5. You should retrieve 10 rows. Cote de Blaye, unsurprisingly, should be the first.

(Optional) Exercise 5: More TOPping

1. Here is an additional challenge, if you feel you want it!
2. Modify a copy of the SortedStockList report from Exercise 2 Task 2 of this lab and select only the top ten products based on their current stock value.