UNIVERSITY OF EXETER
Q-STEP CENTRE

Q-Step
A step-change in quantitative social science skills

Funded by the Nuffield Foundation, ESRC and HEFCE

Introduction to SQL for Data Science

Q-Step Workshop – 20/03/2019

Lewys Brace
l.brace@Exeter.ac.uk

Introduction

- The role of a data scientist is to turn raw data into actionable insights.
- Much of the world's raw data, such as electronic medical records and customer transaction histories, lives in organized collections of tables called *relational databases*.
- Therefore, to be an effective data scientist, you must know how to wrangle and extract data from these databases using a domain-specific language called SQL (Structured Query Language).

Relational databases

- You can think of a relational database as a collection of tables.
- A table is just a set of rows and columns, like a spreadsheet, which represents exactly one type of entity; i.e. a table might represent employees in a company or purchases made, but not both.
- Each row, or *record*, of a table contains information about a single entity; i.e. in a table representing employees, each row represents a single person.
- Each column, or *field*, of a table contains a single attribute for all rows in the table; i.e. in a table representing employees, we might have a column containing first and last names for all employees.

id	name	age	nationality
1	Jessica	22	Ireland
2	Gabriel	48	France
3	Laura	36	USA

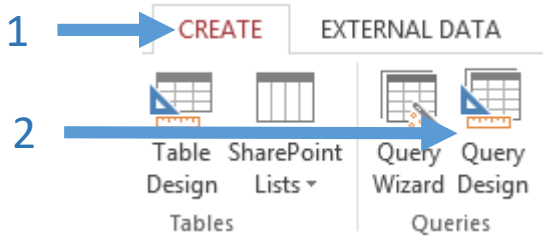
The practice database

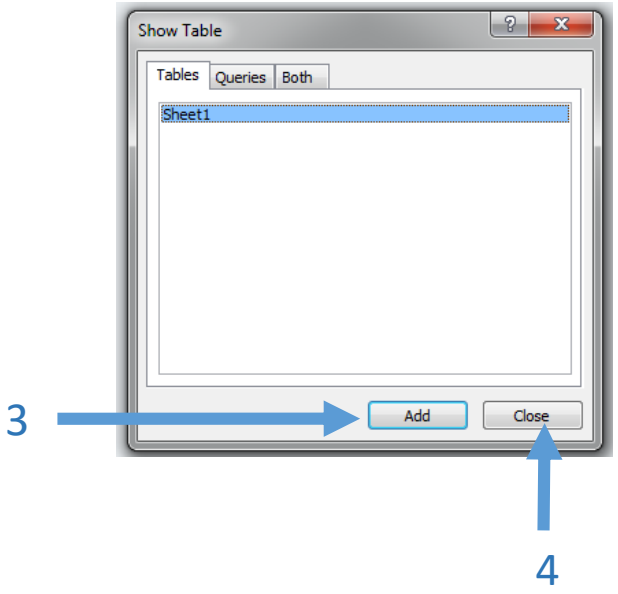
- For this course, we are going to be writing and implementing our SQL code within Microsoft Access.
- We'll be using a database that details various aspects of different dinosaur species.
- This database can be downloaded from https://github.com/LewBrace/Q-Step_SQL_workshop.
- Download and open this database

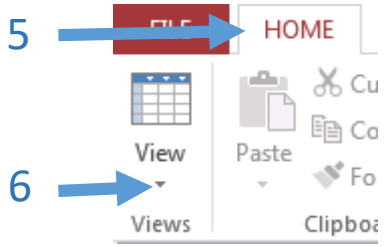
Selecting a single column

- While SQL can be used to create and modify databases, the focus of this course will be *querying* databases.
- A *query* is a request for data from a database table, or combination of tables.
- Querying is an essential skill for a data scientist, since the data you need for your analyses will often live in databases.

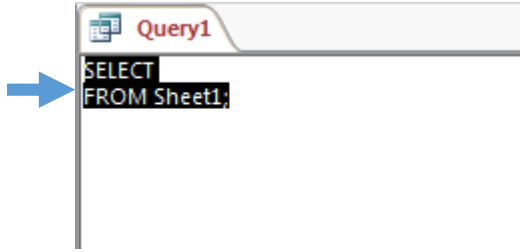
Opening the SQL editor





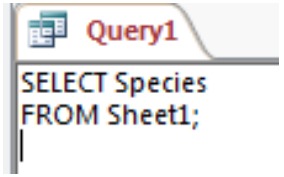


The SQL View Object tab has made the (very rational) assumption that you want to retrieve some information from the Sheet1 table, so it has written the first part for you. It doesn't know exactly what you want to retrieve, so it displays only the part it feels confident about.

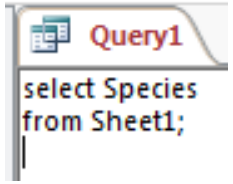


Selecting a single column

- In SQL, you can select data from a table using a SELECT statement; i.e. the following query selects the Species column from the Sheet1 table.
- The semi-colon tells SQL where the end of your query is.

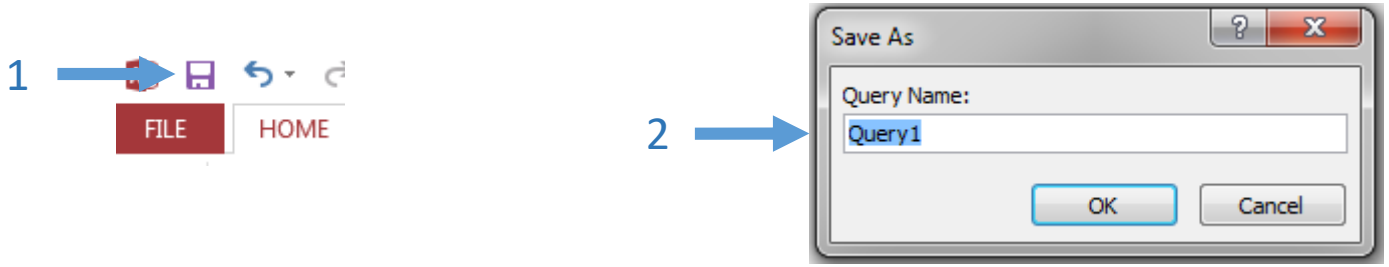


- In SQL, SELECT and FROM are keywords.
- Keywords in SQL are not case-sensitive, which means that the following would also work.

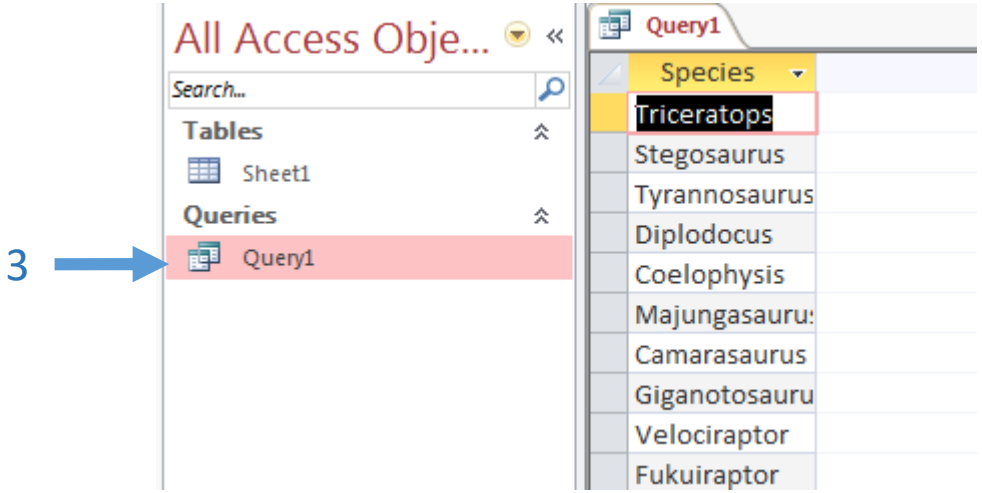


- However, convention dictates that writing keywords in uppercase is `best practice'.

- Once you're done coding your query, save it.



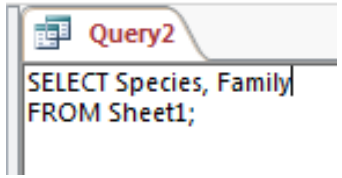
- Your query can then be executed by clicking on the corresponding query tab in the left-hand panel.



Selecting multiple columns

- Selecting multiple columns is easy enough.
- Just add the extra column names to the code, separated by commas.

In:



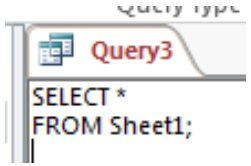
Out:

Query2

Species	Family
Triceratops	Ceratopsian
Stegosaurus	Ankylosaurid
Tyrannosaurus	Large Theropo
Diplodocus	Sauropod
Coelophysis	Small Theropo
Majungasauru:	Large Theropo
Camarasaurus	Sauropod
Giganotosauru	Large Theropo
Velociraptor	Small Theropo

- You can select all columns in a table by using *.

In:



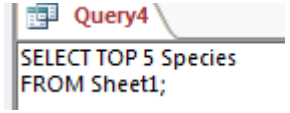
Out:

ID	Species	Family	Existence pe	Existed fron	Existed unti	Diet	Average we	Average len	Area 1	Area 2	Area 3
1	Triceratops	Ceratopsian	Late Cretaceou	68	66	Herbivore	5500	9	USA		
2	Stegosaurus	Ankylosaurid	Late Jurassic	155	145	Herbivore	2000	9	USA		
3	Tyrannosaurus	Large Theropo	Late Cretaceou	68	66	Carnivore	7000	12	Canada	USA	

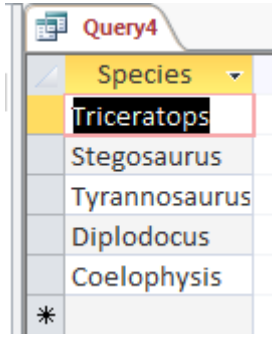
Retrieving a range of records

- You can run a query for the top *x* rows of data by using the SELECT TOP keyword.

In:

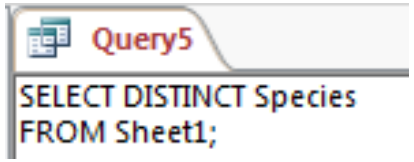


Out:



The DISTINCT keyword

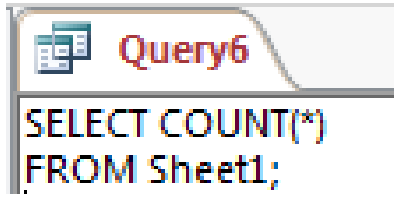
- If your data includes duplicate values and you only want to return all of the unique values from a column, you can use the DISTINCT keyword.



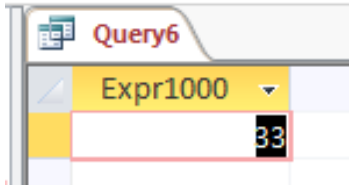
Counting

- You can count the number of rows in your table by using the COUNT keyword; i.e. count the number of species.

In:

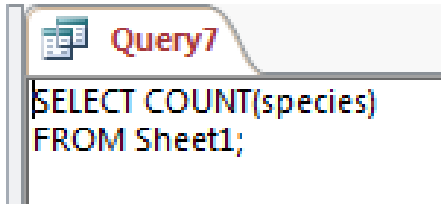


Out:

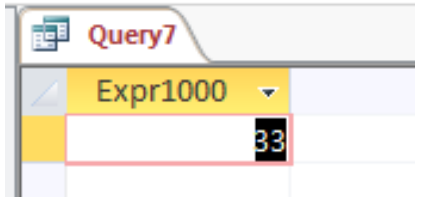


- While COUNT(*) tells you the number of rows in a table, if you want to know the number of non-missing values in a specific column, you can use COUNT.
- This is useful if you have missing values in one or more of your columns.

In:

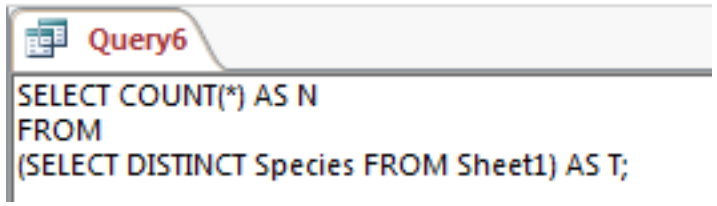


Out:

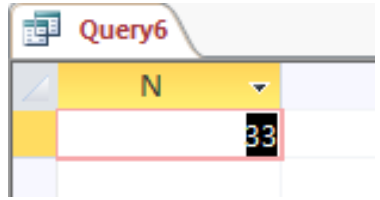


- It's also common to combine COUNT with DISTINCT to count the number of distinct values in a column.

In:



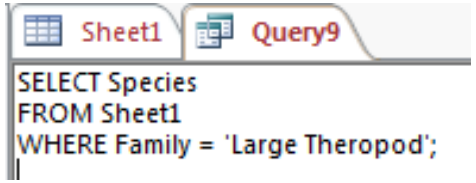
Out:



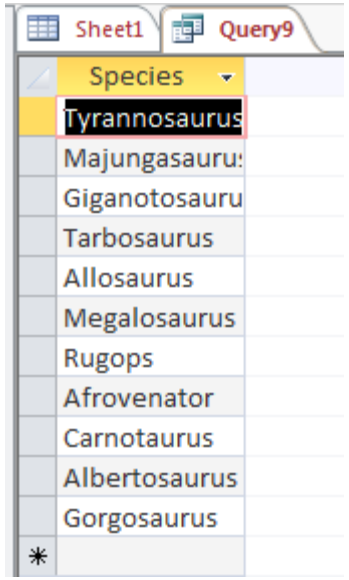
Filtering results

- The WHERE keyword allows you to filter your results based on their values.

In:



Out:



Operators:

=	Equal to
<>	Not equal
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

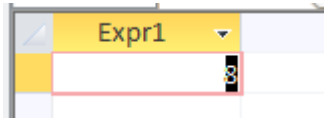
Filtering by numerical values

- Using the COUNT keyword, it is also possible to count the number of records that fulfil a specific criteria.

In:

```
SELECT Count(*) AS Expr1
FROM Sheet1
WHERE Average_weight_kg > 4000;
```

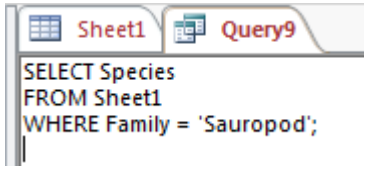
Out:



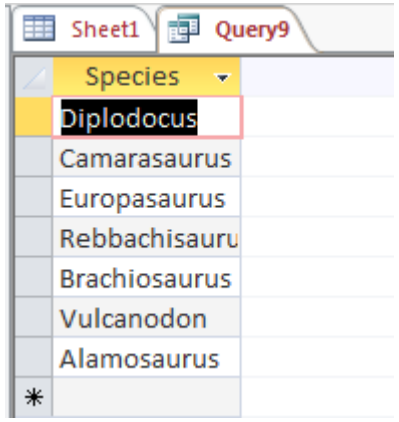
Filtering by text

- The WHERE keyword also enables you to filter by text values.

In:



Out:



Selecting data based on multiple conditions

- You may need to select data based on multiple conditions.
- You can do this by combining your WHERE queries with the AND keyword.

In:

```
SELECT *
FROM Sheet1
WHERE Family='Large Theropod'
and Diet='Carnivore';
```

Out:

ID	Species	Family	Existence_period	Existed fron	Existed unti	Diet	Average_
3	Tyrannosaurus	Large Theropo	Late Cretaceous	68	66	Carnivore	
6	Majungasauru	Large Theropo	Late Cretaceous	84	71	Carnivore	
8	Giganotosauru	Large Theropo	Early Cretaceous	112	90	Carnivore	
19	Tarbosaurus	Large Theropo	Late Cretaceous	74	70	Carnivore	
21	Allosaurus	Large Theropo	Late Jurassic	156	144	Carnivore	
22	Megalosaurus	Large Theropo	Mid Jurassic	170	155	Carnivore	
27	Rugops	Large Theropo	Late Cretaceous	96	94	Carnivore	
28	Afrovenator	Large Theropo	Early Cretaceous	132	121	Carnivore	
30	Carnotaurus	Large Theropo	Late Cretaceous	72	70	Carnivore	
31	Albertosaurus	Large Theropo	Late Cretaceous	76	74	Carnivore	
33	Gorgosaurus	Large Theropo	Late Cretaceous	80	73	Carnivore	

The OR keyword

- If you wanted to select rows based on multiple conditions where some but not all of the conditions need to be bet, you can use the OR keyword.

In:

```
SELECT *
FROM Sheet1
WHERE Area_1 = 'South Africa'
OR Area_1 = 'Madagascar';
```

Out:

ID	Species	Family	Existence_period	Existed fron
5	Coelophysis	Small Theropo	Late Triassic	225
6	Majungasauru	Large Theropo	Late Cretaceous	84

- When using AND and OR, ensure that you enclose the individual clauses in parentheses.

In:

```
SELECT *
FROM Sheet1
WHERE (Area_1 = 'USA' OR Area_1 = 'Canada')
AND (Existence_period = 'Late Jurassic' or Existence_period = 'Late Cretaceous');
```

Out:

ID	Species	Family	Existence_period	Existed fron	Existed unti	Diet	Average_we	Average len	Area_1
1	Triceratops	Ceratopsian	Late Cretaceous	68	66	Herbivore	5500	9	USA
2	Stegosaurus	Ankylosaurid	Late Jurassic	155	145	Herbivore	2000	9	USA
3	Tyrannosaurus	Large Theropo	Late Cretaceous	68	66	Carnivore	7000	12	Canada
4	Diplodocus	Sauropod	Late Jurassic	155	145	Herbivore	20000	26	USA
7	Camarasaurus	Sauropod	Late Jurassic	150	140	Herbivore	20000	23	USA
12	Styracosaurus	Ceratopsian	Late Cretaceous	76	70	Herbivore	2700	5.5	USA
16	Parasauroloph	Euornithopod	Late Cretaceous	76	74	Herbivore	3500	11	Canada
24	Einiosaurus	Ceratopsian	Late Cretaceous	76	74	Herbivore	1300	6	USA
31	Albertosaurus	Large Theropo	Late Cretaceous	76	74	Carnivore	2500	9	Canada
32	Alamosaurus	Sauropod	Late Cretaceous	70	65	Herbivore	30000	21	USA
33	Gorgosaurus	Large Theropo	Late Cretaceous	80	73	Carnivore	2500	8.6	Canada

The BETWEEN keyword

- If you wanted to get the records where the average weight is between two values, you don't have to use < and >.
- Instead, you can use BETWEEN.

In:

```
SELECT *
FROM Sheet1
WHERE Average_weight_kg
BETWEEN 4000 AND 8000;
```

Out:

ID	Species	Family	Existence_period	Existed fron	Existed unti	Diet	Average_we	Average len	Area_1
1	Triceratops	Ceratopsian	Late Cretaceous	68	66	Herbivore	5500	9	USA
3	Tyrannosaurus	Large Theropo	Late Cretaceous	68	66	Carnivore	7000	12	Canada
8	Giganotosauru	Large Theropo	Early Cretaceous	112	90	Carnivore	8000	12.5	Argentina
13	Iguanodon	Euornithopod	Early Cretaceous	140	110	Herbivore	4000	10	England
19	Tarbosaurus	Large Theropo	Late Cretaceous	74	70	Carnivore	4000	10	China
20	Rebbachisauru	Sauropod	Early Cretaceous	112	99	Herbivore	7000	20	Morocco

- You can use the BETWEEN keyword with multiple clauses in the same way you use the WHERE keyword.

In:

```
SELECT *
FROM Sheet1
WHERE Average_weight_kg BETWEEN 4000 AND 8000
AND Area_1 = 'England';
```

Out:

ID	Species	Family	Existence_period	Existed fron	Existed unti	Diet	Average_we	Average len	Area_1	Area 2	Area 3
13	Iguanodon	Euornithopod	Early Cretaceous	140	110	Herbivore	4000	10	England	Germany	Spain

The IN keyword

- If you want to select rows based upon three or more different values from a single column, the WHERE keyword can start to become unwieldly.
- This is where the IN keyword comes in useful.

In:

```
SELECT *
FROM Sheet1
WHERE Average_weight_kg IN (200, 500, 7000, 40000);
```

Out:

ID	Species	Family	Existence_period	Existed fron	Existed unti	Diet	Average_we	Average len	Area_1	Area 2	Area 3
3	Tyrannosaurus	Large Theropo	Late Cretaceous	68	66	Carnivore	7000	12	Canada	USA	
10	Fukuiraptor	Small Theropo	Early Cretaceous	121	99	Carnivore	200	4.2	Japan		
11	Gallimimus	Ornithomimos	Late Cretaceous	74	70	Omnivore	200	6	Mongolia		
17	Europasaurus	Sauropod	Late Jurassic	154	151	Herbivore	500	6.2	Germany		
20	Rebbachisauru	Sauropod	Early Cretaceous	112	99	Herbivore	7000	20	Morocco		
23	Brachiosaurus	Sauropod	Late Jurassic	155	140	Herbivore	40000	30	Algeria	Tanzania	USA

NULL and IS NULL

- NULL represents a missing or unknown value.
- You can check values using the expression IS NULL.
- The IS NULL is useful when combined with the WHERE keyword to figure out what data you're missing.
- If you want to filter out missing values so that you only get results which are not NULL. To do this, you can use the IS NOT NULL keyword.

In:

```
SELECT *
FROM Sheet1
WHERE Existed_from_million_years_ago IS NULL;
```

Out:

ID	Species	Family	Existence_period	Existed_from_million_years_ago	Existed unti	Diet
1	Triceratops	Ceratopsian	Late Cretaceous		66	Herbivore
(New)						

The LIKE and NOT LIKE keywords

- When filtering by text, the WHERE command only allows you to filter by text that matches your search criteria exactly.
- However, in the real world, you often want to search for a pattern rather than a specific match.
- This is where the LIKE keyword comes in.
- LIKE allows you to search for a pattern in a column.
- The LIKE command requires you to use a wildcard placeholder for some other values. There are two of these you can use with the LIKE command.

- The % wildcard will match zero, one, or many characters in text; i.e. the following would return ‘Data’, ‘DataC’, ‘DataCamp’, ‘DataMind’, and so on.

```
SELECT name
FROM companies
WHERE name LIKE 'Data%';
```

- The _ wildcard will match a single character; i.e. the following query matches companies like ‘DataCamp’, ‘DataComp’, and so on.

```
SELECT name
FROM companies
WHERE name LIKE 'DataC_mp';
```

- You can also use the NOT LIKE operator to find records that don’t match the pattern you specify.

Aggregate function

- You can perform some calculation on the data contained within a database.
- You can use SQL's in-built aggregate functions in order to do this.
- A few examples are:

Calculate the average value:

In:
```
SELECT AVG(Average_weight_kg)
FROM Sheet1
```

Out:

Expr1000
5400.15151

Calculate the maximum value:

In:
```
SELECT MAX(Average_weight_kg)
FROM Sheet1
```

Out:

Expr1000
40000

Calculate the summed value:

In:
```
SELECT SUM(Average_weight_kg)
FROM Sheet1;
```

Out:

Expr1000
178205

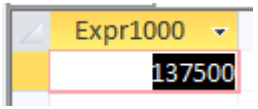
Using aggregate functions with the WHERE command

- Aggregate functions can be combined with the WHERE clause in order to gain further insights from your data.

In:

```
SELECT SUM(Average_weight_kg)
FROM Sheet1
WHERE Average_weight_kg > 4000;
```

Out:



A note on arithmetic

- In addition to aggregate functions, you can also perform basic arithmetic using the standard symbols; +, -, *, /.

In:

```
SELECT (4*3);
```

Out:

Expr1000
12

- Be careful when dividing. While the SQL editor in Access handles division correctly; i.e:

In:

```
SELECT (4/3);
```

Out:

Expr1000
1.33333333

- However, some other editors assume that, if you feed in an integer, you want an integer as output. So you'd get 1 as a result to the above.
- If you want to get the proper result when using one of these editors, you can use:

```
SELECT (4.0 / 3.0) AS result;
```

Aliases

- When using aggregate functions, such as ANG() and MAX(), SQL automatically creates an alias name; i.e:

In:

```
SELECT AVG(Average_weight_kg),
AVG(Existed_from_million_years_ago)
FROM Sheet1;
```

Out:

Expr1000	Expr1001
5400.15151515152	114.212121212121

- You can use the AS keyword to create an alias that specifies the name given to the result column.

In:

```
SELECT AVG(Average_weight_kg) AS Mean_weight,
AVG(Existed_from_million_years_ago) AS Mean_existed_from
FROM Sheet1;
```

Out:

Mean_weight	Mean_existed_from
5400.15151515152	114.212121212121

Sorting results

- The ORDER BY keywords sorts the values of a column in either ascending or descending order.

In:
```
SELECT Species
FROM Sheet1
ORDER BY Average_weight_kg;
```

Out:

Species
Microraptor
Velociraptor
Coelophysis
Khaan
Prenocephale

- By default, it will sort in ascending order. You use the DESC keyword to sort in descending order.

In:
```
SELECT Species
FROM Sheet1
ORDER BY Average_weight_kg DESC;
```

Out:

Species
Brachiosaurus
Alamosaurus
Diplodocus
Camarasaurus
Giganotosauru
Tyrannosaurus

Sorting multiple columns

- The ORDER BY keyword can also be used to sort multiple columns.
- When doing this, SQL will first sort by the first specified column, then the second, and so on.

In:

```
SELECT Family, Species
FROM Sheet1
ORDER BY Average_weight_kg, Existed_from_million_years_ago;
```

Out:

Family	Species
Small Theropo	Microraptor
Small Theropo	Velociraptor
Small Theropo	Coelophysis
Small Theropo	Khaan
Euornithopod	Prenocephale
Ornithomimos	Gallimimus
Small Theropo	Fukuiraptor
Sauropod	Europasaurus

Sorting by multiple columns

- The ORDER BY command can also be sued to sort multiple columns.
- SQL will sort by the first specific column, and then by the second specified column, and so on.

In:

```
SELECT Species, Family
FROM Sheet1
ORDER BY Average_weight_kg, Existed_until_millions_years_ago;
```

Out:

Species	Family
Microraptor	Small Theropo
Velociraptor	Small Theropo
Coelophysis	Small Theropo
Khaan	Small Theropo
Prenocephale	Euornithopod
Fukuiraptor	Small Theropo
Gallimimus	Ornithomimos

The GROUP BY keyword

- You may often want to aggregate your sorted results; i.e. if you have a data base of UK house holds, you may want to count the number of males and number of females.
- You can use the GROUP BY keyword to do this.

In:

```
SELECT Family, count(*)
FROM Sheet1
GROUP BY Family;
```

Out:

Family	Expr1001
Ankylosaurid	2
Ceratopsian	3
Euornithopod	3
Large Theropo	11
Ornithomimos	1
Sauropod	7
Small Theropo	6

Filtering results of aggregate functions

- In SQL, aggregate functions cannot be used in WHERE clauses.
- Therefore, this means that, if you want to filter based on the result of an aggregate function, you have to use the HAVING clause.

In:

```
SELECT Diet, COUNT(*)
FROM Sheet1
GROUP BY Diet
HAVING COUNT (Average_length_m) > 12;
```

Out:

Diet	Expr1001
Carnivore	16
Herbivore	15

Any questions?