

SQL TUTORIAL

Thomasina Cummings
GCS TRAINING

Your tutor will guide you through the tasks in this tutorial. The content may not be detailed enough to work though alone, so please make every effort to attend.

You are encouraged to complete all written tasks as well as the practical tasks and make notes for future reference throughout.

Contents

Tables.....	2
Databases.....	4
Relationships.....	5
Data Types	6
Setting up your SQL Environment	7
Creating a New, Blank Database	8
Creating a Table in SQL	9
Adding Content to a Table	11
Viewing the Contents of a Table.....	13
Additional Information	15
Keys and Relationships	15
Logical Operators	1
Request Queries.....	0

Tables

For ease of working with information, it is common to store data in a table. This is an example of a table with Customer information stored in **columns**.

Customer ID	FirstName	Surname	Telephone Number
DAV325	Susan	Davies	01792 555899
KEL587	Martin	Kelly	01792 365478

The same data could just as easily be **transposed**. This is the same data in a table with **rows**.

Customer ID	DAV325	KEL587
FirstName	Susan	Martin
Surname	Davies	Kelly
Telephone Number	01792 555899	01792 365478

The problem with using a flat table like this soon becomes apparent when you decide to add more information, like Order Number, Order, Date, or Order Contents.

Customer ID	DAV325	KEL587
First Name	Susan	Martin
Surname	Davies	Kelly
Telephone Number	01792 555899	01792 365478
Order Number	01	02
Order Date	08/09/2023	08/09/2023
Order Contents	Power Washer	Garden Hose

When the first order is placed, this works fine. But, eventually, the customer returns to place another order.

Customer ID	DAV325	KEL587	DAV325
First Name	Susan	Martin	Susan
Surname	Davies	Kelly	Davies
Telephone Number	01792 555899	01792 365478	01792 555899
Order Number	01	02	02
Order Date	08/09/2023	08/09/2023	10/09/2023
Order Contents	Power Washer	Garden Hose	Garden Furniture

The data entry clerk now has to retype data that already exists in the table (Customer ID, First Name, Surname, Telephone Number) and with each new order this pattern continues.

What if one day, when recording an order, they make a typo and accidentally type DAV326? The data held on the customer is now unreliable. Searching for the customer's orders will return incomplete information.

At this point, it often becomes more practical to separate the data. Instead of adding columns, the organisation adds tables: Customer Information in one table, order information in another.

Before long, the organisation adds another table: Stock information. Then another: Supplier information.

The organisation now has multiple tables, each containing related information, and that data all needs to be linked in some way. The customer table needs to have all of Susan's order numbers, so that the order can be checked in the order table. The products in the order's table have to have product codes so that they can be checked in the product table. Each product needs...Well! You get the picture, I'm sure.

Databases

Data in a database is also held in tables, although it is common to hear them called **entities**. The name of the table/entity often describes what it contains (student, tutor, course, doctor, customer, etc.)

Databases can contain one or more tables.

A database with a single table is called a **flat file database**. This can be used for things like event logs or configuration data. However, flat files cannot be used if you want to avoid data **redundancy** or **duplication** as we saw above.

When multiple tables are stored in a database, they can be linked together forming a **relationship** between the data in the various tables. We refer to this as a **relational database**.

Databases can be created using a variety of software such as **Access** or **Microsoft SQL Server Management Studio**.

Access has a user-friendly front end and can be easier for first time database users to understand while they get to grips with the notion of databases. But, behind the interface, the code used is the same.

Here is an example of a database in Access. The headings are referred to as **fields**. Each entry in the table is known as a **record**.

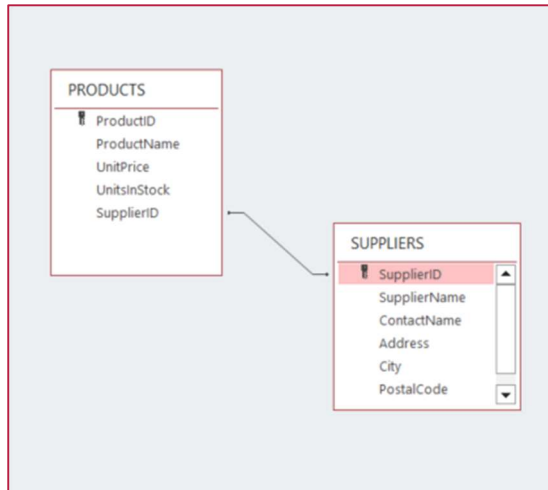
PRODUCTS		SUPPLIERS			
ProductID	ProductNam	UnitPrice	UnitsInStock	SupplierID	Click to Add
1	Milk	£2.50	75	1	
2	Bread	£1.50	67	2	
3	Butter	£2.69	58	1	
4	Sugar	£0.89	24	2	
5	Beans	£1.50	120	1	
6	Soup	£1.27	35	2	
7	Flour	£0.85	80	4	
8	Tomatoes	£0.49	120	2	
9	Syrup	£1.34	113	1	
10	Cereal	£3.60	98	2	
11	Beefburgers	£2.50	50	5	
12	Salt	£1.74	50	1	
*(New)		£0.00	0		

PRODUCTS		SUPPLIERS						
	SupplierID	SupplierName	ContactName	Address	City	PostalCode	Country	Click to Add
	1	ABC Supplies	John Smith	123 Main Street	Anytown	12345	USA	
	2	XYZ Co	Jane Doe	456 Oak Street	Othertown	67890	Canada	
	3	DEF LTD	Jack Jones	789 Elm Grove	Newcastle	45899	USA	
	4	HJK HFK CO	Janet Brown	234 Palm Street	Bristol	12489	China	
	5	PTR Supplies	Tom Jones	134 Los Angeles	Derby	123605	UK	
*	(New)							

This database has two tables/entities: PRODUCTS and SUPPLIERS.

Each product, and supplier, has a unique identifier called a **Primary Key**.

Notice how the Products table has a column for supplier ID so that it can be linked to the Suppliers table. Behind the scenes, this is the relationship that these two tables have.



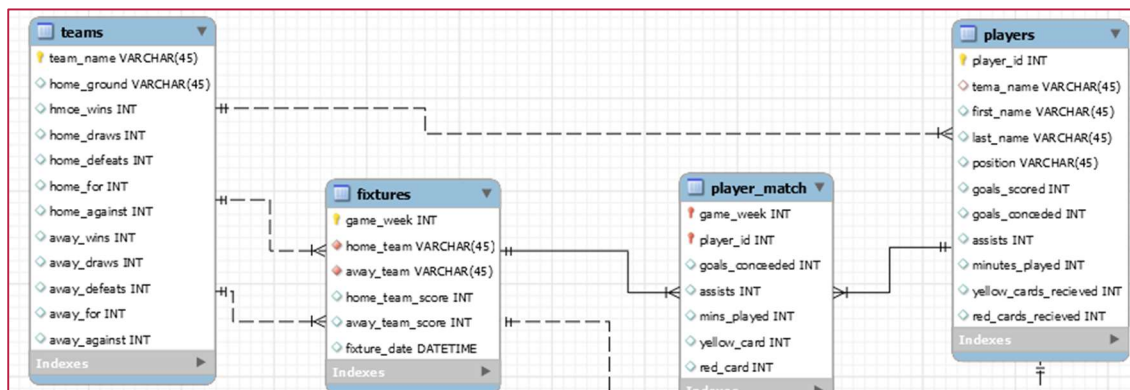
Relationships

One to one: like a person and their national insurance number (each person has one number, and each number relates to only one person).

One to many: like a car make and model (each maker has many models, but models will only have one maker).

Many - many: like product and components (a product can have many components, likewise components can be used in many products).

We view these relationships using a **relationship diagram**. These can be simplified like the version shown above, or more complex like the one below.



Can you work out which type of relationship the symbols at the ends of the lines are showing?

This relationship can be used to form the basis of **queries** such as 'all products from the same supplier'.

SupplierNam ▼	ProductNam ▼
ABC Supplies	Milk
ABC Supplies	Butter
ABC Supplies	Beans
ABC Supplies	Syrup
ABC Supplies	Salt
✱	

At the back end, these queries are made possible by a coding language called **SQL**

```
SELECT SUPPLIERS.SupplierName, PRODUCTS.ProductName,  
FROM SUPPLIERS INNER JOIN PRODUCTS ON SUPPLIERS.SupplierID =  
PRODUCTS.SupplierID  
WHERE (((PRODUCTS.SupplierID)=1);
```

Look again at the result of this query. Can you see what the code is doing to produce these results? See if you can identify the entities and fields elements of the code.

SupplierNam ▼	ProductNam ▼
ABC Supplies	Milk
ABC Supplies	Butter
ABC Supplies	Beans
ABC Supplies	Syrup
ABC Supplies	Salt
✱	

Don't be too concerned about the INNER JOIN element for now. In simple terms, it is just being used to join the two tables so that information can be extracted from both.

Data Types

Before we start getting deeper into the SQL side of it, let's take a small detour to investigate the types of data that can be held in fields in a database. Data can be text, number, date, time, even an automatically assigned value to identify a specific record.

When tables are created it is necessary to state which type of data the field will hold. In this extract, you can see the type and length of the field being specified after the field name. Varchar is basically a way of declaring a variable character length.

```
First_Name varchar(20),  
Last_Name varchar(25) Not Null,  
Email varchar(25) Not Null,  
Phone_Number varchar(20),  
Job_ID varchar(10)
```

Not Null means that this field must be supplied and cannot be empty.

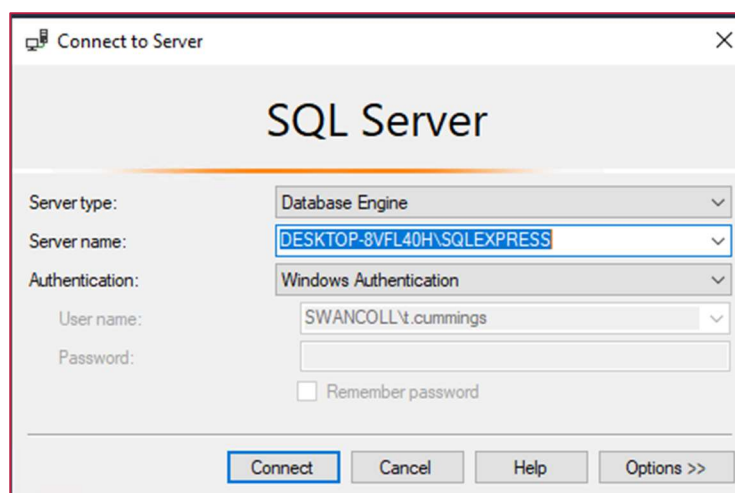
Setting up your SQL Environment

For this course you are going to use SQL Server Management Studio (SSMS). You will also need administrative privileges in order to create databases. If you do not have these, you should run your installation in a virtual machine.

<https://learn.microsoft.com/en-us/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16>

The setup and install varies from machine to machine. Follow the appropriate install instructions for your operating system and ensure you can open and run the program before continuing.

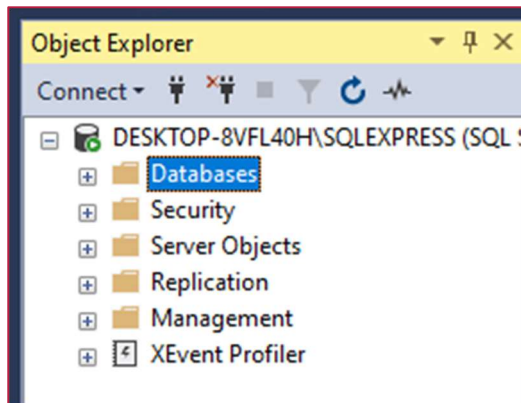
When the system opens, you will need to Connect to the Server. If you use Windows Authentication, you should see something similar to this - your server's name, and your username, should automatically be detected from your Windows environment.



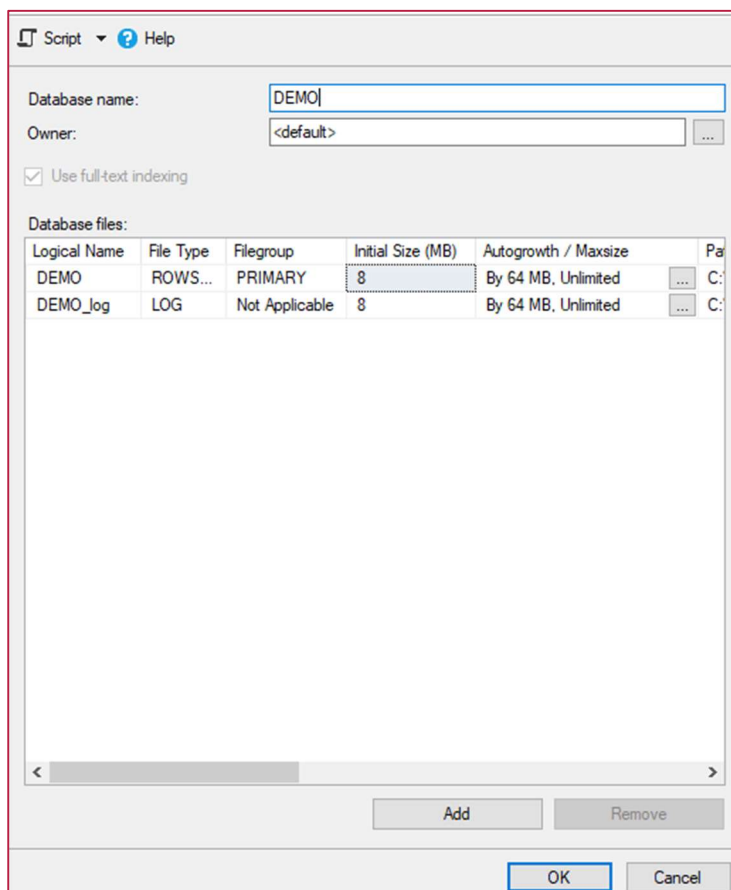
Note: If you choose to use a different program, you should still be able to achieve the same things, but you may need to explore the layout to find all of the items discussed.

Creating a New, Blank Database

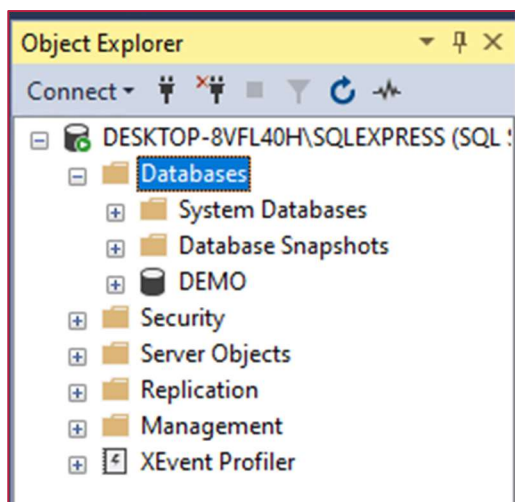
Once SSMS is running, the **Object Explorer** pane will load at the left-hand side of the window. For now, don't worry about the Tabs and Ribbons above this. Each element will be introduced as it is needed.



Right click on the Databases folder to create a **New Database** called **DEMO**. You do not need to change any of the other settings.

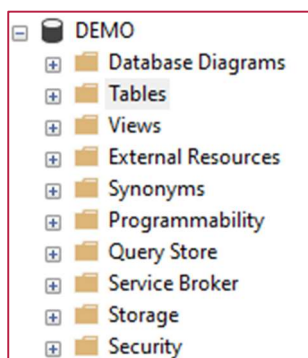


This database will now join the system databases in the object explorer.



You have now created a blank database to which you can add your tables.

If you click the plus button to the left of your DEMO database, you will see that there are several subfolders.



Creating a Table in SQL

Creating a table can be done a number of ways. The first and simplest way to do this is by right clicking on the Tables folder and selecting New Table.

A form will load, prompting you to supply the column name, the data type, and select whether you want to 'Allow Nulls'

Column Name	Data Type	Allow Nulls
		☐

Column names should not contain spaces. To make the names easy to read, you can replace space with an underscore or capitalise the first letter of each word (Camel Case).

This_is_the_underscore_method and ThisIsTheCamelCaseMethod

The Data Types can be typed in, or you can use the Column Properties tab. Pay particular attention as to whether 'Nulls' are allowed.

	Column Name	Data Type	Allow Nulls
▶	Emp_No	int	☐
	First_Name	varchar(20)	☒
	Last_Name	varchar(25)	☐
	Email	varchar(20)	☐
	Phone_Number	varchar(20)	☒
			☐

Column Properties

- (General)

(Name)	Emp_No
Allow Nulls	No
Data Type	int
Default Value or Binding	
- Table Designer

Collation	<database default>
Computed Column Specification	
Condensed Data Type	int
Description	
Deterministic	Yes
DTS-published	No
Full-text Specification	No
Has Non-SQL Server Subscriber	No
Identity Specification	No
Indexable	Yes
Is Columnset	No
Is Sparse	No
Merge-published	No
Not For Replication	No
Replicated	No
RowGUID	No
Size	4

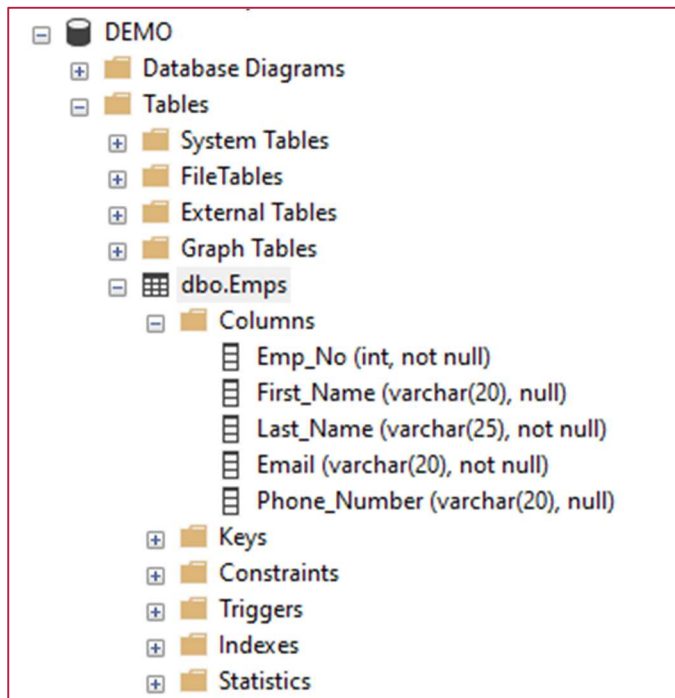
Just in case that data isn't clear in the image, here are the details you need.

```
Emp_No int Not Null Primary Key,  
First_Name varchar(20),  
Last_Name varchar(25) Not Null,  
Email varchar(25) Not Null,  
Phone Number varchar(20),
```

Use the **Save** button on the ribbon, to save the Table as **Emps**.



If you expand the folder structure, you will now see an entry called **dbo.Emps**, inside which you will see your columns.



If you were to manually code this using SQL, this is what you might see.

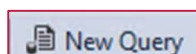
```
CREATE TABLE Emps
Emp_No int Not Null Primary Key,
First_Name varchar(20),
Last_Name varchar(25)Not Null,
Email varchar(25) Not Null,
Phone_Number varchar(20),
```

The command for creating a table is CREATE TABLE.

Try to get used to the different ways of representing the same information.

Adding Content to a Table

This table is currently empty. The next step is to add some records. Click **New Query**

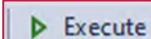


A blank document will appear in the main window to the right of Object Explorer.

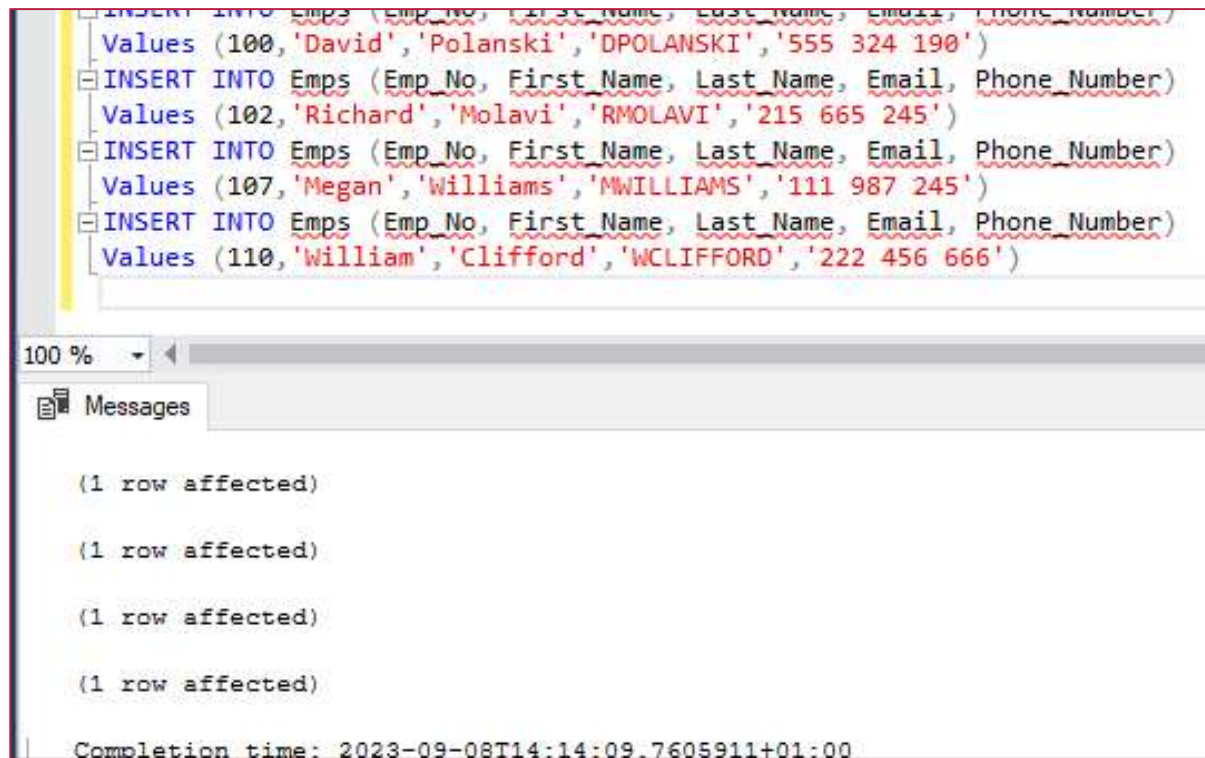
Type the following in the blank query window.

```
INSERT INTO Emps (Emp_No, First_Name, Last_Name, Email, Phone_Number)
Values (100,'David','Polanski','DPOLANSKI','555 324 190')
INSERT INTO Emps (Emp_No, First_Name, Last_Name, Email, Phone_Number)
Values (102,'Richard','Molavi','RMOLAVI','215 665 245')
INSERT INTO Emps (Emp_No, First_Name, Last_Name, Email, Phone_Number)
Values (107,'Megan','Williams','MWILLIAMS','111 987 245')
INSERT INTO Emps (Emp_No, First_Name, Last_Name, Email, Phone_Number)
Values (110,'William','Clifford','WCLIFFORD','222 456 666')
```

Click the Execute button on the ribbon.



If you've typed it correctly, you will receive feedback that 4 rows have been affected.



Note how we used INSERT INTO to add content to the table.

If you get errors, you may need to fix your syntax.

Be careful not to run successful commands twice or you may get duplicate entries.

You can add `--` before a command to stop it running again. This will turn the command into a **comment**.

If you do accidentally end up with content in your table that you do not want, you can use DELETE FROM to remove contents.

```
DELETE FROM Emps
WHERE Emp_No = '100'
```

This command will delete employee number 100 from the Emps table.

Note the use of quotes around the employee number and the use of WHERE to set a condition to be applied to the instruction.

Try adding and removing employees until you feel comfortable with the content so far.

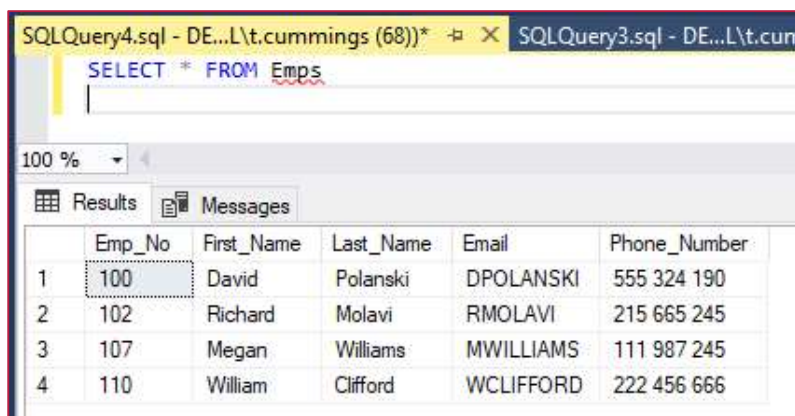
Viewing the Contents of a Table

The feedback from the system implies we have added content. But we need a way to check.

Create a new Query and type the following:

```
SELECT * FROM Emps
```

When this is executed, you will be presented with the results.



The screenshot shows a SQL query editor with two tabs: 'SQLQuery4.sql - DE...L\t.cummings (68)*' and 'SQLQuery3.sql - DE...L\t.cummings'. The active tab shows the query 'SELECT * FROM Emps'. Below the query, there is a 'Results' tab and a 'Messages' tab. The 'Results' tab displays a table with 6 columns: 'Emp_No', 'First_Name', 'Last_Name', 'Email', and 'Phone_Number'. The table contains 4 rows of data.

	Emp_No	First_Name	Last_Name	Email	Phone_Number
1	100	David	Polanski	DPOLANSKI	555 324 190
2	102	Richard	Molavi	RMOLAVI	215 665 245
3	107	Megan	Williams	MWILLIAMS	111 987 245
4	110	William	Clifford	WCLIFFORD	222 456 666

SELECT is the query that enables us to extract (or select) information from a table.

The `*` in this statement is a wildcard meaning ALL or EVERYTHING.

If you do not want to see the whole table, you can SELECT one or more columns using a comma to separate.

In this query, we are selecting just the First_Name and Last_Name.

```
SELECT First_Name, Last_Name FROM Emps
```

	First_Name	Last_Name
1	David	Polanski
2	Richard	Molavi
3	Megan	Williams
4	William	Clifford

Task 1: What command would you need to type, to extract from the database the employee whose Emp_ID is 100?

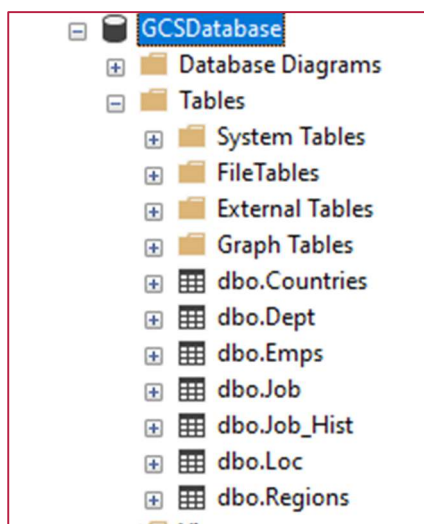
In order to practice working with more commands, we are going to need a much more complex data set. Typing data is slow and boring, so your tutor has made available a data set that you can use.

In order to use this script, you will need to create a new database called GCSDatabase.



DATABASE PRACTICE SCRIPT.txt

Once the database has been created, paste the contents of the provided text file into a new query and execute. Check all of your tables have been created.



Note: In addition to the tables, and the table content, you may notice there are a few other instructions in this file.

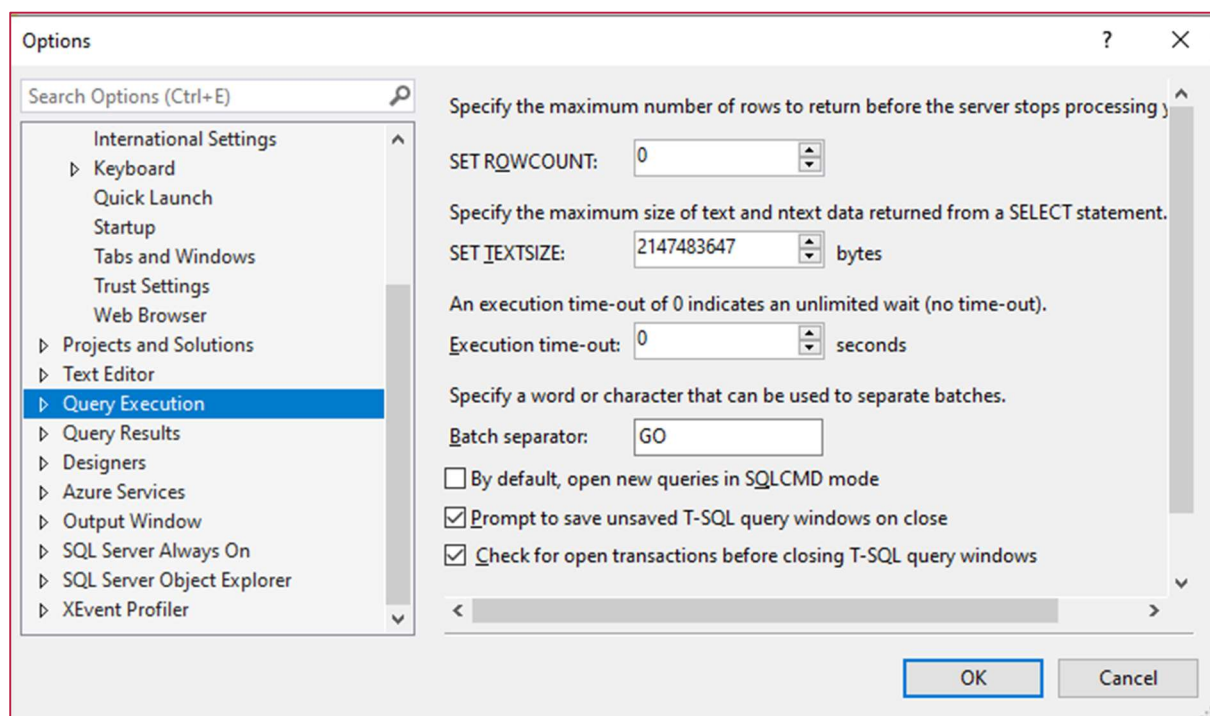
Additional Information

CREATE INDEX is used to set up an index that can make data retrieval faster.

GO is used to run a batch of commands together, before moving to the next.

The word Go is not fixed and, should you download code from online with a different batch separator, it is possible to change it to something else using Tools > Options.

Note: This is not something you need to know at this point, but should you choose to explore other code you may find it useful.



With regards the rest of the script, you should now recognise the commands that are being used to set up tables and add content to tables.

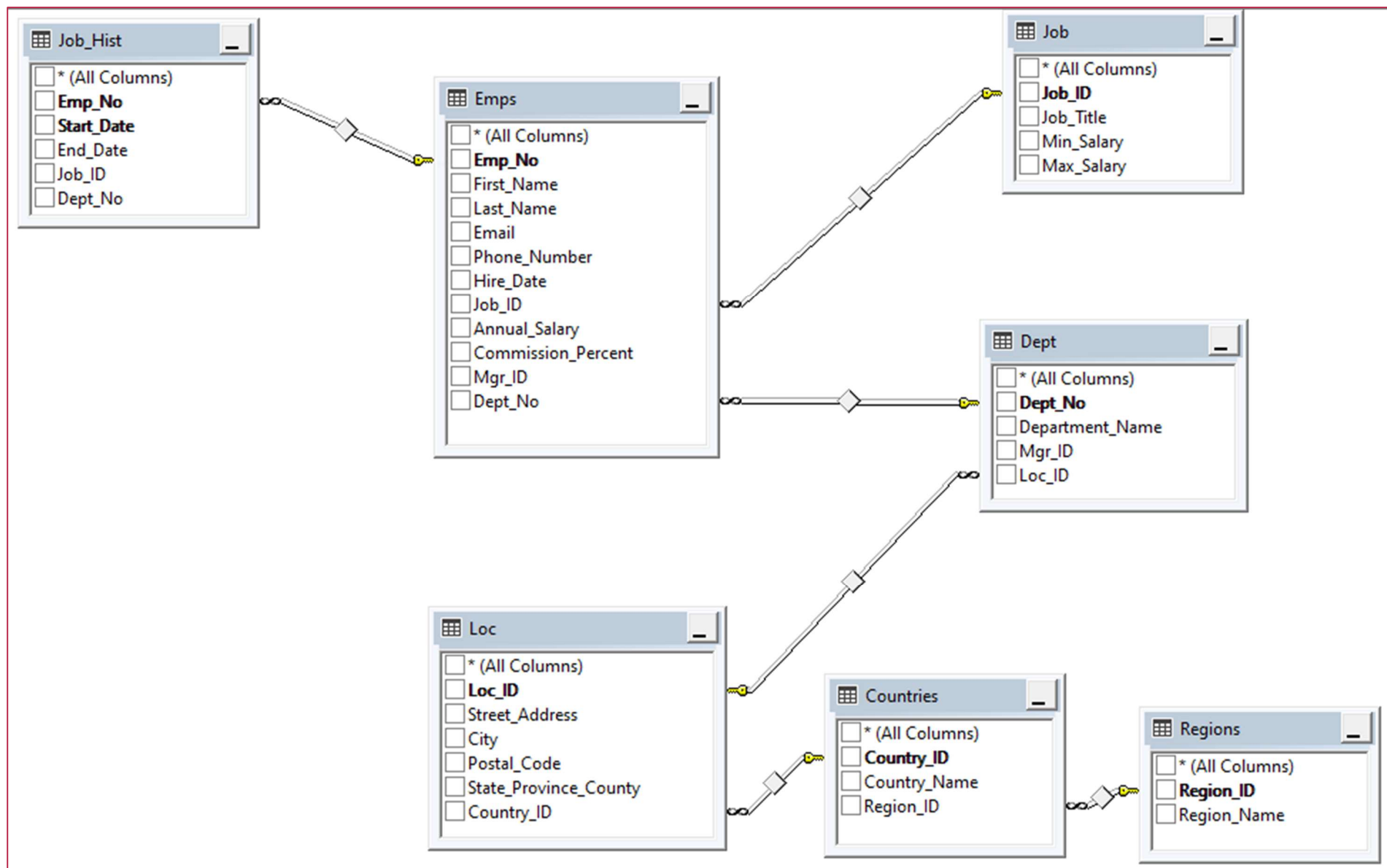
Keys and Relationships

As introduced earlier, each record in a table will usually have a unique identifier.

In the Countries table, this is the County ID. We refer to this field as the **Primary Key** field.

A Primary Key may also appear in other tables as a **Foreign Key**.

Task 2: Below is a visual representation of the tables in the GCSDatabase.
Look for the relationships between the tables. Note the shared keys.



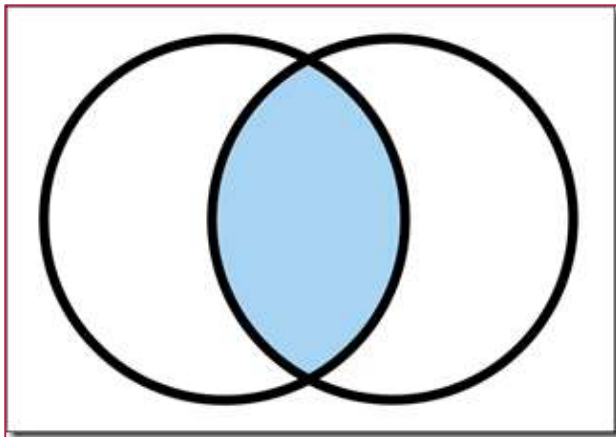
Behind the scenes, this is shown by the following SQL.

```
SELECT
FROM Emps
INNER JOIN Dept ON Emps.Dept_No = Dept.Dept_No
INNER JOIN Job ON Emps.Job_ID = Job.Job_ID
INNER JOIN Job_Hist ON Emps.Emp_No = Job_Hist.Emp_No
INNER JOIN Loc ON Dept.Loc_ID = Loc.Loc_ID
INNER JOIN Countries ON Loc.Country_ID = Countries.Country_ID
INNER JOIN Regions ON Countries.Region_ID = Regions.Region_ID
```

Notice the INNER JOIN command appearing again.

The inner join indicates that the field appears in both tables. Also notice that the common field might have different names in one table to another.

As a side note, some of you might remember doing sets (Venn Diagrams) in primary school. If so, the following visual might help you to picture how the inner join works. The two circles represent a pair of tables. The shaded area is the shared field i.e., an inner join.



In order to carry out the remainder of the course, you are going to need to load another script. Create a new query, paste the contents of the script into the query and execute.



DATABASE TASK SCRIPT.txt

This command adds headings to the output. This will help us keep the output organised as the queries are built and run.

```
SELECT '' as 'Request 1' where 1!=1
```

Below this in comments, you will see the queries you are going to build.

The output, when you click execute, will be a list of blank Request Numbers. These will populate with table data as you compile and run the queries you design in the remainder of the course.



Make sure your script has loaded and is producing this output before continuing.

Logical Operators

A few times since this course began, you may have noticed the equal sign (=) being used in the commands. This one, for example, where we deleted entries where Emp_No was equal to 100.

```
DELETE FROM Emps  
WHERE Emp_No = '100'
```

Equals is a common operation and, as such, didn't need an introduction of its own. However, as you start to build more complex SQL queries, it is often not enough to just try to carry out queries on a table using equals.

The following operators may be useful going forward. Different systems may use the alternative variation given.

<	LESS THAN
>	MORE THAN
!= <>	NOT EQUAL TO
&& AND	AND
 OR	OR
LIKE	
BETWEEN	
IN	

Read the definitions and criteria. Write a test statement that uses the operator. You may make up your own field or table names.

<	LESS THAN	Field data will be compared to a test value and only returned if they are less than it. Can also be used in conjunction with =	Members are under 18	SELECT * FROM table WHERE age<18
>	MORE THAN	Field data will be compared to the test value, and only returned if they are greater than it.	Members are taller than 6ft	
!= <>	NOT EQUAL TO	Field data will be compared to the test value, and only returned if they are not equal to it.	Member isn't Welsh	
&& AND	AND	Field data will be compared to two or more test values and returned if all criteria are met.	Member is Female and Unemployed	
 OR	OR	Field data will be compared against two or more test values and returned if any of the criteria are met.	Member is Male or Employed	
LIKE	Field data will be compared to the test value and returned if it is similar.		Member's name starts with X % can be used as a wildcard after the letter (X)	
BETWEEN	Field data will be compared to two test values and returned only if they fall between the two values.		Member is over 18 and under 60	
IN	Field data will be compared against a set of values and returned if they are common to anything in the set.		Member is in the list of countries (France, Germany, UK) - don't forget your single quotes.	

Request Queries

Think about how you might structure your query, using the examples you have already seen.

New SQL will be introduced as required to refine your queries.

For further research and guidance, you can consult the following websites.

<https://www.w3schools.com/sql/default.asp>

<https://www.sqltutorial.org/sql-cheat-sheet/>

<https://learn.microsoft.com/en-us/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver16>

<https://learn.microsoft.com/en-us/sql/ssms/ssms-utility?view=sql-server-ver16>

Use 'execute' to test your query.

Use any errors/warnings/output returned to improve your query if you don't get the correct answer first time.

If necessary, add SELECT statements to your queries, to show the result of adding, removing, or updating data.

Request 1

List the last name, first name and employee number of all programmers who were hired on or before 21 May 1991 sorted in ascending.

New SQL

ORDER BY can be used to sort data by a given field.

You can also add asc or desc to the end of the command to state whether it should be low to high, A-Z, high to low, Z-A, etc.

Note: some systems require dates to be entered as YYYY-MM-DD. others prefer dates in 'single quotes'

Request 2

List the department number, last name and salary of all employees who were hired between 16/09/87 and 12/05/96 sorted in ascending order of last name within department number.

Request 3

List all the data for all jobs where the maximum salary is greater than 15000 sorted in descending order of the maximum salary.

Request 4

List the last name, first name, job id and commission of employees who earn commission sorted in ascending order of first name within last name.

Request 5

Which jobs are found in the IT and Sales departments? You might need to use LIKE for this one.

Request 6

List the last name of all employees in departments 50 and 90 together with their monthly salaries (rounded to 2 decimal places), sorted in ascending order of last name.

New SQL

FORMAT (Annual_Salary / 12, 'N2') to calculate monthly salary and display to two decimal places.

/ is the symbol for divide. Although not needed here, * is the symbol for multiply.

AS <name> allows you to view the result of a calculation under a new, temporary, column heading.

Request 7

Show the total salaries figure for one month displayed with no decimal places.

Request 8

Show the total number of employees.

New SQL

COUNT(*) to count all members meeting a specified condition.

Request 9

List the department number, department name and the number of employees for each department that has more than 2 employees grouping by department number and department name.

New SQL

FROM <Table1> INNER JOIN <Table2>
ON <Table1.field> = <Table2.field>

GROUP BY allows you to group by given tables or fields

HAVING allows you to specify additional criteria that must be met

Request 10

List the department number, department name and the number of employees for the department that has the highest number of employees using appropriate grouping.

New SQL

TOP <number> to show a specified number of instances

Request 11

List the department number and name for all departments where no programmers work.

New SQL

DISTINCT returns values just once, even if there are multiple instances which would return the same criteria.

Request 12

Add the following new job IT_ANAL, System Analyst, 10000, 15000. Remember to add a select statement to show that it has been added. You should also comment this query out after running to stop repeat entries if you are working in the same query text for all requests.

Request 13

Update all the maximum salaries for jobs with an increase of 1000.

New SQL

SET can be used to assign new values. New values can be created using mathematical calculations such as +, -, * and /

Request 14

List all the data for jobs sorted in ascending order of job id.

Request 15

The job history for employee number 102 is no longer required. Delete this record. List all the data for job history sorted in ascending order of employee number to show that the employee's history has been removed.

Request 16

Produce a list of employees showing percentage raises, employee numbers and old and new salaries. Employees in departments 20 and 10 are given a 5% rise, employees in departments 50, 80, 90 and 110 are given a 10% rise and employees in other departments are not given a rise.

New SQL

CASE is used to outline various scenarios.

WHEN is used to specify the different scenarios.

ELSE is a catch all statement that applies if none of the WHEN scenarios apply.

END AS <name> specifies a new temporary heading for the new column and ends the CASE statement.

Request 17

Create a new view for manager's details only using all the fields from the employee table. Apply a CHECK constraint.

New SQL

CREATE VIEW [<view_name>] AS followed by the query will assign the query to a view

WITH CHECK OPTION will make sure that if the name already exists the command doesn't run.

DROP VIEW can also be used to remove a view.

Request 18

Show all the fields and all the managers, using the view for managers, sorted in ascending order of employee number.

Request 19

Grant the authority to all public users to access the view for managers for SELECT statements only.

GRANT SELECT ON [<view name>] TO [<user group>]

Request 20

Create an index named LOC_POSTAL_CODE on the Postal Code in the locations table. Provide a printout showing that the index has been created.

New SQL

CREATE INDEX <name> ON Table <field>

You can check the index has been created by viewing the indexes folder of the table in object explorer:

