1. The aim of this exercise it to stop anyone deleting events which happened in the UK (country number 7).  To do this create a trigger which operates on the **tblEvent**table. In your trigger, create and set the value of the following two variables from the **deleted**table:

|  |  |
| --- | --- |
| **Variable** | **What it should contain** |
| **@EventId** | The number of the event someone is trying to delete |
| **@CountryId** | The country number in which this event took place. |

If this event didn't take place in country id number 7, delete it:

-- if country is not UK (7), allow deletion

IF @CountryId <> 7

BEGIN

DELETE

FROM tblEvent

WHERE EventId = @EventId

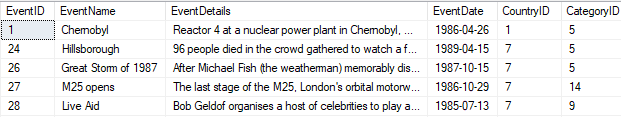
END

Test that your query works by deleting events which did and didn't happen in the UK.

1. Create a comma-delimited list variable containing all of the names of events that occurred in your decade of birth. Use **LEFT** to remove the extra comma, and **QUOTENAME** to add apostrophes.

List variable to be used with dynamic SQL

Now use this list to filter another select statement which shows all of (\*) the information about those events from the event table. You will need to use dynamic SQL.



1. The aim of this exercise is to use CTEs to answer the following questions:

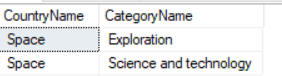
|  |  |  |
| --- | --- | --- |
| **Step** | **What to do** | **Returns** |
| 1 | Get a list of those events which contain none of the letters in the word **OWL** | 3 rows |
| 2 | Use this to get a list of all of those events which take place in the countries for the events in list 1. | 9 rows |
| 3 | Get a third list of all of the events which share the same categories as any of the events in the second list. | 116 rows |

If you display the results in date order, you should get this:



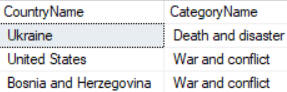
*The first few of the 116 events returned by your final query.*

1. Space: the final frontier.  But it's also one of the countries in the events database.  The aim of this exercise is to list out all the categories of events which occurred in the **Space**country:



***Space****events, as it turns out, only use two categories.*

You'll then list all of the events which didn't occur in **Space**, with their country names and categories:



*The first few of the 444 events which didn't happen in the****Space****country.*

You can then show the 8 countries which had non-Space events in the same category as one of the **Space**events.  Read on!

To start with, write a select statement to return the **CategoryName** of events which occur in the country **Space** only:

CTE Countries CategoryName

*Perhaps war should be a****CategoryName****in****Space****, since the satellite USA-193 was shot down by a long-range missile?*

Now turn this into a **CTE** so the list is stored in a table that we can join to later. The layout looks like this:

With CTE\_Name

AS

(

Your select statement goes here

)

-- Additional CTEs don't need a second WITH separate with commas

,CTE\_Name

AS

(

)

The second CTE should hold a list of all the countries and their categories excluding (<>) **Space.** You can then join the two CTEs together based on the **CategoryName**to show the final answer:



*The 8 countries in which events occurred in terrestrial (ie non-****Space****) countries, but where these events' categories were shared by****Space****events!*

1. Create two CTEs in the same query as follows:

|  |  |
| --- | --- |
| **CTE** | **What it should list out** |
| **TopCountries** | The ids of the 3 countries with the most events (use **TOP 3**) |
| **TopCategories** | The ids of the 3 categories with the most events (use **TOP 3**again) |

Use these to show all possible combinations in a final **SELECT** statement which should look like this:

-- combine these together (every possible combination)

SELECT

cy.CountryID,

cg.CategoryID

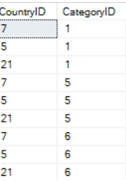
FROM

TopCountries AS cy

CROSS JOIN TopCategories AS cg

*The cross join gives all possible combinations of rows in the two CTEs - see below for what the results should look like.*

You should now be looking at something like this:



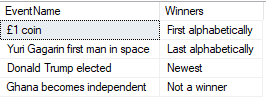
*There are 9 possible combinations.*

If you still have the energy, use this to show the number of events for each combination:



***United Kingdom****/****Politics is the****combination of country and category which has the most events.*

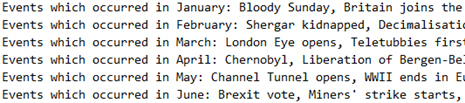
1. Time to test the events and see who receives a podium finish! Create a function to test each event and classify it as the oldest, newest, alphabetically first, alphabetically last or (in all other cases) as a loser:



*Donald Trump's election is the newest event in the database, for example, and hence a winner.*

The best way to do this is create a **CASE** statement that tests if the current parameter is **IN** the output of a select statement holding the winners:

1. Ever wondered which month is the most eventful? Or which month an event occurredloo in? Create a loop to list the events in each month, by looping from month number 1 to 12:



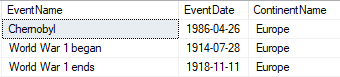
*Loops can save you time on repetitive tasks.*

The hardest part in creating this query is changing the month number into the month name, as sadly there is no in-built function.

1. The aim of this exercise is to link two stored procedures together. Start by creating a stored procedure which shows the **ContinentName** where the first event occurred.

Stored Procedure Output Parameter

Now create a second stored procedure which filters events to show only those which happened in the continent passed in via a parameter. Run this with **Europe**:



*Not a particularly good time in Europe's history.*

Using an output parameter take the **ContinentName** produced by the first stored procedure and store it in a variable. Use this variable to filter the second procedure.

1. First write a stored procedure to show which country has the most events:

Output parameters Countries Events

*Your query should show these results - we now want to capture these two bits of information, and return them to the calling query.*

Add two output parameters to your procedure:

|  |  |
| --- | --- |
| **Parameter** | **What it should take** |
| **@TopCountry** | The country name |
| **@EventCount** | The number of events for it |

Amend your procedure to capture the country name and number of events in these variables, within your **SELECT** clause:

SELECT TOP 1

--Values go into parameters just like variables

@TopCountry = CountryName

,@EventCount= COUNT(E.CountryID)

FROM ...

Open a new window to execute the procedure. Create two variables to store the values held in the output parameters.

Select the two variables to check the information has been successfully output.