|  |  |
| --- | --- |
| A picture containing green  Description automatically generated  Week #11 Assignment answers  2% Individual Assignment | Triggers and 1NF  Mark Morell  Database Management – Fall 2019 |

# Assignment Type:

# Individual – Prepare and submit your results independently

# Date Due:

# Thursday, November 21st by the end of the day

# Instructions:

# Please submit your assignment electronically through eConestoga.

# Assignments should be submitted as Microsoft Word files using the course coversheet format. You **MUST** include your query as text in the Word document as well as a **FULL screenshot** of your SSMS screen (screen capture the entire application screen including the title bar through the bottom of the window). Multiple screenshots may be required.

# If you are using external sources (images, text, etc.) you must reference them as part of your assignment and not copy them as-is.

# Best practice is to research your answers and then write the response to the question in your own words.

# Please include the question number with your responses.

# Late Assignment Penalty:

|  |  |
| --- | --- |
| Days Late | Penalty % |
| 1 | 5 |
| 2 | 10 |
| 3 | 20 |
| 4 | 40 |
| 5 | 60 |
| 6 | 80 |
| 7 | 100 |

# Assignment Questions

|  |  |  |
| --- | --- | --- |
| Question # | Question | Score |
| 1 | Using the COSTUME database, we want to create a table that tracks every time we have less than 3 costumes available for rental (where the number in-stock minus the number lost is less than 3) in the CostumeInventory table. So do the following:   * Create a new table called LowCostumeInventory with the following columns:   + An ID column that’s an IDENTITY column (and PRIMARY KEY)   + The costume ID (create a duplicate index on this column)   + A column indicating the date/time that will capture when we hit low costume availability * Next create a DML trigger on the CostumeInventory table that captures all UPDATE actions to the table where if the stock of costumes less the number of lost costumes is less than 3, INSERT a new record into our new table above * After the above are in-place, write T-SQL that updates the number lost for costumes of your choice that shows both scenarios where the UPDATE **both** *does* and *does not* write a record to our new table. Write and execute a SELECT of the new table in both cases to show the results   USE Costume  GO  CREATE TABLE dbo.LowCostumeInventory (  id INT IDENTITY NOT NULL,  costumeId INT NOT NULL,  changeDate DATETIME2 NOT NULL,  CONSTRAINT PK\_id PRIMARY KEY (id),  CONSTRAINT FK\_costumeId FOREIGN KEY (costumeId)  REFERENCES dbo.Costume(id)  )  CREATE TRIGGER TR\_UPDINS\_CostumeChange  ON dbo.CostumeInventory  AFTER UPDATE, INSERT  AS  BEGIN  SET NOCOUNT ON;  INSERT INTO dbo.LowCostumeInventory  SELECT i.costumeId, GETDATE()  FROM inserted i  WHERE i.numberInStock - i.numberLost < 3;  END;  -- Now see if this works. First, check the contents of the log table.  -- It should be empty:  SELECT \* FROM dbo.LowCostumeInventory;  -- Now look at a record that we're going to update:  SELECT \* FROM dbo.CostumeInventory WHERE costumeId = 2  -- This record is nowhere near meeting our criteria so update it but still don't  -- make the trigger execute:  UPDATE dbo.CostumeInventory SET numberLost = 10 WHERE costumeId = 2  -- Check the contents of the log table. It should still be empty:  SELECT \* FROM dbo.LowCostumeInventory;  -- Now update the number lost so that our trigger fires:  UPDATE dbo.CostumeInventory SET numberLost = 18 WHERE costumeId = 2  -- Check the contents of the log table. We should have caught that update:  SELECT \* FROM dbo.LowCostumeInventory; | 10 |
| 2 | Normalize the following data to First Normal Form (1NF):   |  |  |  |  | | --- | --- | --- | --- | | Product | Description | Colour | Cost | | Volleyball | Soft-touch | White, Green, Blue/Yellow | $39.99 | | Football | 8-lace design | Brown, Blue/White, Orange | $45.99 | | Soccer Ball | Junior Size | Red/White, Red | $29.97 | | Tennis Ball | Standard | Yellow | $4.99 | | Racquetball | Blue Dot | Black, Purple | $7.97 |   Rules of 1NF:   * Tables do not have repeating groups of data * Each table must have a unique PRIMARY KEY * The table contains only **atomic** values   Product:  productName, productDescription, cost  ProductColour:  productName, colour  So the data would look like:  Product:   |  |  |  | | --- | --- | --- | | Product | Description | Cost | | Volleyball | Soft-touch | $39.99 | | Football | 8-lace design | $45.99 | | Soccer Ball | Junior Size | $29.97 | | Tennis Ball | Standard | $4.99 | | Racquetball | Blue Dot | $7.97 |   ProductColour:   |  |  | | --- | --- | | Product | Colour | | Volleyball | White | | Volleyball | Green | | Volleyball | Blue/Yellow | | Football | Brown | | Football | Blue/White | | Football | Orange | | Soccer Ball | Red/White | | Soccer Ball | Red | | Tennis Ball | Yellow | | Racquetball | Black | | Racquetball | Purple | | 5 |
|  | **Total** | **15** |