```
/*
--6).
Name: Christopher Singleton                                 Date: 02/12/2017
Class: PROG 140
Project: Module 04

About this Original Script: Drops Database IF EXISTS / Creates Database / Inserts
three rows into three tables with constraints and indexes (Note: There is also a
linking table).

Project Description: Use an already existing database script to demonstrate the
progress with printing out information (success/fail) as the script runs, logic
process using variables and includes error handling with a try catch statement.

--============================= Purpose ======================================--
--============ Demonstrate Variables, Logic, and Error Handling =================--
--==========================================================================--
*/

/* DIRECTIONS: (Note: I made the directions smaller, when I made the font larger, formatting
                      messed up because of the underlined words.)
6.  through 10.

       Create a SQL script and in this script place all your statements from the last exercise
    (Module 3 #1-6) for creating and loading your database.
       This script MUST include the following elements:

       6). Your name, date, and a brief description of the contents and purpose of your script
           SQL comments at the top bracketed by /* */.

       7). A check near the beginning of your script to determine whether the database you're
           creating exists and if so to drop it.

       8). At least 3-4 "print" statements within the script that include the date and time and
           that will be executed as the script progresses to demonstrate its progress.

       9). Appropriate commenting with organized and neat statements.

       10). Simple Try-Catch error handling - similar to what was demonstrated and used in the
            CreateandLoadTestingDB.sql file included in this module's files.
*/
/* Write your sql statement in a separate file and then copy/paste as usual into a Word doc
   and submit. You will be submitting two Word files:  One for questions 1-5 and the other for
   this script, ie., questions 6-10.
   DO NOT place your statements here! */

USE master

  /* Note: There are no batch separators (GO) in this script. BEGIN TRY END CATCH
           runs as a complete script. */

--=================IF The Database Exists Then Select It=================--

BEGIN TRY --Begin with a try statement (Error Handling).
```

```
--7).
     IF EXISTS (SELECT name FROM sys.databases WHERE name = N'CustomerTransact')
     ALTER DATABASE [CustomerTransact] SET SINGLE_USER WITH ROLLBACK IMMEDIATE

--==================Drop This Database And End Function=================--

          --If it already exists so we can start fresh.
          DROP DATABASE [CustomerTransact]

          /*Print out that the table was dropped */
          /*Convert to sysdatetime and then cast to varchar. */
          PRINT 'CustomerTransact Database: Dropped Database Sucessfully.'
             + CAST(CONVERT(varchar, SYSDATETIME(), 121) AS varchar (20))

--===================Create The DataBase With Settings===================--

   --It’s always a good idea to plan the size of the database for growth.
   --Plenty of size...
   CREATE DATABASE [CustomerTransact] ON PRIMARY
   (NAME = N'CustomerTransact'
   , FILENAME = N'D:\PROG 140\CustomerTransactDB\CustomerTransact.mdf'
    --Store Database Here
   , SIZE = 10MB
   , MAXSIZE = 1GB
   , FILEGROWTH = 10MB)
   LOG ON
   (NAME = N'CustomerTransct_log'
   --Store Log File Here
   , FILENAME = N'D:\PROG 140\CustomerTransactDB\CustomerTransact_log.LDF'
       , SIZE = 1MB
   , MAXSIZE = 1GB
   , FILEGROWTH = 10MB)

   EXEC [CustomerTransact].dbo.sp_changedbowner @loginame = N'SA', @map=false
   -- Log In Owner Database Name = SA --Note: SA Means "System Administrator"

--================Set's the Recovery Record Log Settings=================--

/*Note: Only use this mode "BULK_LOGGED" when there are no other users, otherwise
        data loss can happen. */

   ALTER DATABASE [CustomerTransact] SET RECOVERY BULK_LOGGED

   -- Below: Prints out the system Date Time with a message.
   --Print out that the database was created with Date/Time, CAST to varchar.
   PRINT 'CustomerTransact Database: Database was Created Sucessfully.'
         + CAST(CONVERT(varchar, SYSDATETIME(), 121) AS varchar (20))

--===========================Use This Database===========================--
```

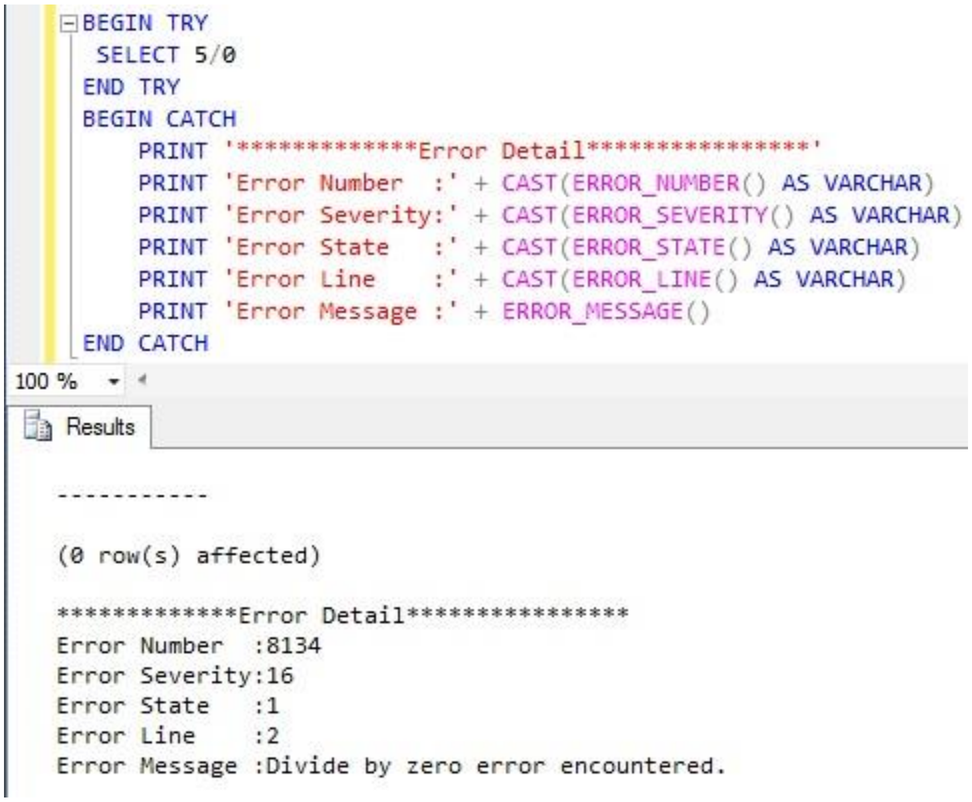
```
   USE CustomerTransact

--===============Create The Customer Table With Data Types=================--
   --1).
   CREATE TABLE Customer
                --PK-Set to row 1, increment 1.
               ([CustomerID] int NOT NULL PRIMARY KEY Identity(1,1)
               ,[FirstName] nvarchar(25) NOT NULL --DEFAULT 'unknown'
               ,[LastName] nvarchar(25) NOT NULL
               --Index Email (used often for identity of customer.)
               ,[Email] nvarchar(25) --UNIQUE (Unclustered)
               ,[Phone] nvarchar(14)--UNIQUE (Unclustered)
                /* Note: Phone Index often used for identity of customer. */
               )

--================Create The SalesOrder Table With Data Types==============--

   CREATE TABLE SalesOrder
   --Identity set to row 1, increment 1. Set's the identity to row one with PK.
               ([SalesOrderID] int NOT NULL PRIMARY KEY Identity(1,1)
             ,[CustomerID] int NOT NULL --Foreign Key
               ,[SalesOrderDate] datetime NOT NULL
               ,[SaleAmount] decimal(8,2) NULL --, CHECK SaleAmount > 0
               )

--=====Create The SalesOrderProduct (Linking Table) With Data Types======--

   CREATE TABLE SalesOrderProduct
               ([SalesOrderID] int NOT NULL --Foreign Key (Non-Clustered Index)
               ,[ProductID] int NOT NULL --Foreign Key (Non-Clustered Index)
               )

--=================Create The Product Table with Data Types================--

   CREATE TABLE Product
   --Identity set to row 1, increment 1. Set's the identity to row one with PK.
                --PK-Set to row 1.
               ([ProductID] int NOT NULL PRIMARY KEY Identity(1,1)
               ,[ProductName] nvarchar(25) NOT NULL --Unique (Unclustered)
               ,[Description] nvarchar(25) NOT NULL
                -- DEFAULT 0 --Cost can be 0 because of damaged goods.
               ,[Cost] decimal(8,2) NOT NULL
--Check Constraint, Price needs to be greater than zero. (Not giving it away.)
               ,[Price] decimal(8,2)  NOT NULL
               )

      --Print out that all tables were created with Date/Time, CAST to varchar.
      PRINT 'CustomerTransact Database: All Tables Created Sucessfully.'
            + CAST(CONVERT(varchar, SYSDATETIME(), 121) AS varchar (20))
```

```sql
--======================Create The Constraints For Tables====================--
     --2).

     ALTER TABLE SalesOrder
     ADD CONSTRAINT fk_Customer
     FOREIGN KEY (CustomerID)
     REFERENCES Customer (CustomerID) --Constraint in the SalesOrder Table.

     ALTER TABLE SalesOrderProduct
     ADD CONSTRAINT fk_SaleOrderProduct_SalesOrderID
     FOREIGN KEY (SalesOrderID)
     REFERENCES SalesOrder (SalesOrderID)
/* Constraing in the SalesOrderProduct Table. */

     ALTER TABLE SalesOrderProduct
     ADD CONSTRAINT fk_SaleOrderProduct_ProductID
     FOREIGN KEY (ProductID)
     REFERENCES Product (ProductID) --Constraing in the SalesOrderProduct Table.

     --DEFAULT Constraints:
     ALTER TABLE Customer
     ADD CONSTRAINT Default_Customer_FirstName
     DEFAULT 'Unknown' FOR FirstName

     ALTER TABLE Product
     ADD CONSTRAINT Default_Cost
     DEFAULT 0 FOR Cost
/* Damaged goods can cost 0 (returned to vendor), so default made sense. */

--------------------------------------------------------------------------------
     --CHECK Constraints:
     ALTER TABLE Product
     ADD CONSTRAINT chk_Price
     CHECK (Price > 0) --Prices are always above 0.

     ALTER TABLE SalesOrder
     ADD CONSTRAINT chk_SalesAmount
     CHECK (SaleAmount > 0) --SalesAmounts are always above 0.

--======================Create The Indexes for Tables====================--
    --3).
     --Customer index
      --Unique Composite Index (Already has a PK Index)
    CREATE UNIQUE NONCLUSTERED INDEX IX_Customer_Email_Phone
    ON Customer (Email, PHone)
```

```sql
	--SalesOrder index:
	--Simple Index Frequently used. (Already has a PK Index)
	CREATE NONCLUSTERED INDEX IX_SalesOrder_FKCustomerID
	ON SalesOrder (CustomerID)
	WITH (FILLFACTOR = 65, PAD_INDEX = ON) --This Index changes often.

	--SalesOrderProduct Table Indexes:
	--Simple Index frequently used table for joins.
	CREATE NONCLUSTERED INDEX IX_SalesOrderProduct_FKSalesOrderID
	ON SalesOrderProduct (SalesOrderID)

	--Simple Index frequently used table for joins.
	CREATE NONCLUSTERED INDEX IX_SalesOrderProduct_FKProductID
	ON SalesOrderProduct (ProductID)

	--Product Index:
	--Frequently queried Product name frequently queried.(Already has a PK Index)
	CREATE NONCLUSTERED INDEX IX_Product_ProductName
	ON Product (ProductName)
	WITH (FILLFACTOR = 65, PAD_INDEX = ON ) --This Index changes often.

--Print out Constraints/Indexes were created with Date/Time, CAST to varchar.
	PRINT 'CustomerTransact: All Constraint Relationships and Indexes Created'
	      + 'Sucessfully.'
	      + CAST(CONVERT(varchar, SYSDATETIME(), 121) AS varchar (20))

--======================Populate tables with three rows=====================--

	--4).
	--Insert into the Customer Table for testing...
	INSERT INTO Customer (FirstName, LastName, Email, Phone)
	     VALUES('Jimmy', 'Branson', 'JB@gmail.com', '206-555-1212'),
	           ('Mike', 'Smith', 'mikesmith@yahoo.com', '206-865-2531'),
	           ('Harry', 'Johnson', 'HR@outlook.com', '206-486-1315');

	--Insert into the SalesOrder Table for testing...
	INSERT INTO SalesOrder (CustomerID, SalesOrderDate, SaleAmount)
	         VALUES(1, '1982-01-01 00:00:00', 25.99),
	               (3, '1993-05-05 00:00:00', 30.99),
	               (2, '1982-02-08 00:00:00', 53.99);

	--Insert into the Product Table...
	INSERT INTO Product (ProductName, [Description], Cost, Price)
	          VALUES('GT-5900', 'Video Card', 98.99, 125.99),
	                ('XL-2588', 'LCD Monitor', 54.99, 76.99),
	                ('MXT-5690', 'Hard Drive', 89.99, 159.99);
```

```
--8).
     --Print out that three rows were populated into tables with Date/Time.
    --Converted first, then cast to ANSI standard as varchar 20.
     PRINT 'CustomerTransact Database: Three Rows Inserted into Each Table'
            +'Sucessfully.'
            + CAST(CONVERT(varchar, SYSDATETIME(), 121) AS varchar (20))

--======================Select Statements to Print Tables=====================--
     --5).
     SELECT * FROM Customer
     SELECT * FROM SalesOrder
     SELECT * FROM Product

--===========================sp_help to Print Tables==========================--

     --6). Give information about each table.
     EXEC sp_help Customer
     EXEC sp_help SalesOrder
     EXEC sp_help SalesOrderProduct
     EXEC sp_help Product

END TRY --End Try if not successful and then run the catch statement.
--9).
/*Note: It only makes sense to use the ERROR_MESSAGE() and ERROR_SEVERITY() */
BEGIN CATCH
     PRINT 'Problem found!!! '
     -- Whoops, there was an error
     -- Raise an error with the details of the exception
     DECLARE @ErrMsg nvarchar(4000), @ErrSeverity int
     SELECT @ErrMsg = ERROR_MESSAGE(), @ErrSeverity = ERROR_SEVERITY()

     RAISERROR(@ErrMsg, @ErrSeverity, 1)
END CATCH --End Catch.

--10).

/*Just copy/paste the above script, I tried it and it works fine.
NOTE: I put all the block quotes on the same line also, because I noticed that
when you copied them over into SQL Server, they created errors, which was very
odd because the end of the block quote were not seen.

The ERROR_MESSAGE() (shows the error message) and ERROR_SEVERITY() (returns
severity number of the error); both together are enough to give you an idea of
where the error is and/or what is going on.

Converted to date time (hour, min, seconds) for the print message and then cast
directly into varchar 20 (CAST is ANSI). */
```

```
/*Note: I also did some research and thought this was awesome!
Another great example in showing output, but without using variables!

Note: You could just print the errors directly (below) by using a variable with
value a value. Just did not want to forget this way also (very direct). */
```



```
BEGIN TRY
 SELECT 5/0
END TRY
BEGIN CATCH
    PRINT '*************Error Detail****************'
    PRINT 'Error Number  :' + CAST(ERROR_NUMBER() AS VARCHAR)
    PRINT 'Error Severity:' + CAST(ERROR_SEVERITY() AS VARCHAR)
    PRINT 'Error State   :' + CAST(ERROR_STATE() AS VARCHAR)
    PRINT 'Error Line    :' + CAST(ERROR_LINE() AS VARCHAR)
    PRINT 'Error Message :' + ERROR_MESSAGE()
END CATCH
```

```
-----------

(0 row(s) affected)

*************Error Detail****************
Error Number  :8134
Error Severity:16
Error State   :1
Error Line    :2
Error Message :Divide by zero error encountered.
```