IST 278 Test 1 Hands-On Part

Name: Michael Bacu Date: 6/12/2023

# **Test Instructions**

1. Type your name and the date in the spaces provided.
2. Use the SQL Server Management Studio and the IST278EagleCorp.
3. Complete the **4** exercises in this document (make sure to scroll down this document has several pages to it).
4. Upload and submit a completed copy of this test document before the due date.

**Exercise 1 (an Advanced Query Exercise):**

Code a query that uses the IST278EagleCorp Database to produce a result set showing South Carolina and Georgia Customer order information that consists of CustomerID, OrderID, State, City, and OrderWeight

Hints:

Approach this problem as follows:

1. State the problem to be solved in English.
2. Use English or Pseudocode to outline the query. In this step you need to identify the subqueries you will use and the data they will return. You should include the aliases you will use for any derived tables.
3. If any of the subqueries identified in step 2 are particularly involved, use English or pseudocode to outline the further details for it.
4. Code the subqueries and test them to be sure that they return the correct data.
5. Code and test the final query.

Instructions:

Paste below the work you did to solve this problem and the run results. Include all outline / Pseudocode you wrote as well as the query code and the run results you got when executing it.

**--- Paste below this line the work you did for this exercise --**

We need to retrieve the customer order information for customers located in South Carolina and Georgia. The result set should include CustomerID, OrderID, State, City, and OrderWeight.

1. Select all the necessary columns from the correct tables and get SUM for the OrderWeight
2. Join the correct tables on each other being Customer, CustOrder, CustOrderLine, InventoryPart, Shipment
3. Specify for SC and GA
4. Group by to avoid invalid error

SELECT Customer.CustomerID, CustOrder.OrderID, Customer.State, Customer.City, SUM(OrderQuantity \* InventoryPart.Weight) AS OrderWeight

FROM CustOrder

JOIN Customer ON CustOrder.CustomerID = Customer.CustomerID

JOIN CustOrderLine ON CustOrder.OrderID = CustOrderLine.OrderID

JOIN InventoryPart ON CustOrderLine.PartNumber = InventoryPart.PartNumber

JOIN Shipment ON CustOrder.OrderID = Shipment.OrderID

WHERE ShipState IN ('SC', 'GA')

GROUP BY Customer.CustomerID, CustOrder.OrderID, Customer.State, Customer.City

**---** **Paste below this line the run results from executing the script you wrote**

A screenshot of a computer

Description automatically generated

**Exercise 2 (Scripting / Flow Control Exercise):**

Create a script per the following directions:

1. Create two variables of type int. Name one @CustomerCount and the other one @EmployeeCount.
2. Use a select statement to populate the @CustomerCount with the number of rows on the Customer table.
3. Use a select statement to populate the @EmployeeCount with the number of rows on the Employee table.
4. Code a print statement that uses data type conversion and concatenation to print out a message that states the number of customers. The message should look like the following when this line of code executes:

We have 232 Customers.

Note: 232 was the number of customers at the time I ran it, but that might be different when you run it since the table is subject to insertions and deletions.

1. Code a print statement that uses data type conversion and concatenation to print out a message that states the number of employees. The message should look like the following when this line of code executes:

We have 40 Employees.

Note: 40 was the number of employees at the time I ran it, but that might be different when you run it since the table is subject to insertions and deletions.

1. Code if else logic to compare the @EmployeeCount variable to the @CustomerCount variable and do the following conditionally:

If the @EmployeeCount is greater than the @CustomerCount

Print a message that states “We have more Employees than Customers”

If the @CustomerCount is greater than the @EmployeeCount

Print a message that states “We have more Customers than Employees”

If the @CustomerCount is equal to the @EmployeeCount

Print a message that states “We have the same number of Customers as Employees”

**--- Paste below this line the Script you wrote for this exercise --**

DECLARE @CustomerCount INT, @EmployeeCount INT;

SELECT @CustomerCount = COUNT(\*) FROM Customer;

SELECT @EmployeeCount = COUNT(\*) FROM Employee;

PRINT 'We have ' + CAST(@CustomerCount AS VARCHAR(10)) + ' Customers.';

PRINT 'We have ' + CAST(@EmployeeCount AS VARCHAR(10)) + ' Employees.';

IF @EmployeeCount > @CustomerCount

PRINT 'We have more Employees than Customers.';

ELSE IF @CustomerCount > @EmployeeCount

PRINT 'We have more Customers than Employees.';

ELSE

PRINT 'We have the same number of Customers as Employees.';

**---** **Paste below this line the run results from executing the script you wrote (make sure to include the completion time when copying and pasting the messages that appeared when you ran it)--**

A screenshot of a computer

Description automatically generated with medium confidence

**Exercise 3 (a view exercise)**

Write a CREATE VIEW statement to create a view named xxNoOrders (where the xx are your initials). The view is to return the customer ID, last name and phone of each customer with no orders.

**--- Paste below this line the code you wrote for creating your view –**

CREATE VIEW MBNoOrders AS

SELECT CustomerID, CustLastName, Phone

FROM Customer

WHERE CustomerID NOT IN (SELECT DISTINCT CustomerID FROM CustOrder);

**--- Paste below this line the code you wrote for code for selecting everything from your view –**

SELECT \*

FROM MBNoOrders;

**--- Paste below this line the run results from selecting everything from your view –**

A screenshot of a computer

Description automatically generated

**Exercise 4 (a Stored Procedure exercise)**

Use the IST278EagleCorp database and write a stored procedure named spInventoryQuantityRangeXX where the XX are your initials. This stored procedure is to receive two optional parameters and use these parameters to filter to reduce the number of rows returned in the result set. The @QuantityMin and @QuantityMax are parameters to be used to specify the requested range of inventory stock levels.

Code this stored procedure to have it:

* Handle a call where only one parameter value is supplied by raising a 50001 error with a message of “Incorrect number of parameters supplied.”
* Handle a call where the value passed to the @QuantityMax is less than the value passed to the @QuantityMin by raising a 50001 error with a message of “Invalid parameter values supplied.”
* Handle a call where no parameter values are supplied by doing the following:
  + printing a message that states “No Parameter values supplied. Defaulting to selecting all rows.”
  + returning the PartNumber, PartDescription, and StockLevel for each inventory item, sorted by StockLevel with the lowest level appearing first.
* Handle a call where two valid parameter values are supplied by returning a result set consisting of PartNumber, PartDescription, and StockLevel for each inventory item that meets the filtering criteria, sorted by StockLevel with the lowest level appearing first.

**---** **Paste below this line the Stored Procedure you wrote for this exercise –**

CREATE PROCEDURE spInventoryQuantityRangeMB

@QuantityMin INT = NULL,

@QuantityMax INT = NULL

AS

BEGIN

IF (@QuantityMin IS NOT NULL AND @QuantityMax IS NULL) OR (@QuantityMin IS NULL AND @QuantityMax IS NOT NULL)

BEGIN

THROW 50001, 'Incorrect number of parameters supplied.', 1;

RETURN;

END;

IF @QuantityMin IS NOT NULL AND @QuantityMax IS NOT NULL AND @QuantityMax < @QuantityMin

BEGIN

THROW 50002, 'Invalid parameter values supplied.', 1;

RETURN;

END;

IF @QuantityMin IS NULL AND @QuantityMax IS NULL

BEGIN

PRINT 'No Parameter values supplied. Defaulting to selecting all rows.';

SELECT PartNumber, PartDescription, StockLevel

FROM InventoryPart

ORDER BY StockLevel;

RETURN;

END;

SELECT PartNumber, PartDescription, StockLevel

FROM InventoryPart

WHERE (StockLevel >= @QuantityMin OR @QuantityMin IS NULL)

AND (StockLevel <= @QuantityMax OR @QuantityMax IS NULL)

ORDER BY StockLevel;

END;

**---** **Paste below this line the completion status messages from executing your create stored procedure code-**

A screenshot of a computer

Description automatically generated with medium confidence

**--- Paste below this line the statements you wrote to test your stored procedure and the run results from those tests –**

A screenshot of a computer

Description automatically generated