29837995—SQL Programming & Creative Writing

Task 1

Assume that you are testing the `Orders` database introduced in Watt (2014) - Appendix C

- Discuss the problems and possible consequences of using the `Orders` database. Provide examples of how you would address any problems that you identified.

> Cite any source in APA format.

The original design of the `Orders` database presented a few issues. The SQL statements in the DDL script named `OrdersAndData.sql` did not include primary keys nor foreign keys. Also, the script did not offer any statements (such as, `CONSTRAINT` statements) which would enhance data integrity by limiting the nature of values that could be entered into the tables. These factors exposed the `Orders` database to risks of losing its data in the event of a mishap. Its lack of primary keys, for instance, exposed its tables to possible duplication of rows (Rob & Coronel, 2009).

Before executing `INSERT` statements on the `Orders` database, every `CREATE TABLE` script had to be modified to ensure that it did not generate a relation with a `NULL` primary key. The original script, for instance, included the following statements to create the `tblCustomers` relation:

```
CREATE TABLE [dbo].[tblCustomers] (
  [CustomerID] nvarchar(5) NULL,
  [CompanyName] nvarchar(40) NOT NULL,
  [ContactName] nvarchar(30) NULL,
  [ContactTitle] nvarchar(30) NULL,
  [Address] nvarchar(60) NULL,
  [City] nvarchar(15) NULL,
  [Region] nvarchar(15) NULL,
  [PostalCode] nvarchar(10) NULL,
  [Country] nvarchar(15) NULL,
  [Phone] nvarchar(24) NULL,
  [Fax] nvarchar(24) NULL
)
```

This needed to be modified to read as:

```
CREATE TABLE [dbo].[tblCustomers] (
  [CustomerID] nvarchar(5) NOT NULL,
  [CompanyName] nvarchar(40) NOT NULL,
  [ContactName] nvarchar(30) NULL,
  [ContactTitle] nvarchar(30) NULL,
  [Address] nvarchar(60) NULL,
  [City] nvarchar(15) NULL,
  [Region] nvarchar(15) NULL,
  [PostalCode] nvarchar(10) NULL,
```

```
  [Country] nvarchar(15) NULL,
  [Phone] nvarchar(24) NULL,
  [Fax] nvarchar(24) NULL,
  Primary Key (CustomerID)
)
```

Also, in cases where a referential integrity needed to be established (such as in the `tblProducts` relation), the addition of a foreign-key creating statement proved remedial.

Reference

Rob, P., & Coronel, C. (2009). *Database systems: Design, implementation, and management*. Thomson/Course Technology.

Task 2

Using the `Orders` database introduced in Watt (2014) - Appendix C; write the SQL Statements for below cases

- Get all the orders placed by a specific customer. `CustomerID` for this customer is `MAGAA`

Whilst leaving out the columns `CustomerID`, `ShipName`, `ShipAddress`, `ShipCity`, `ShipRegion`, `ShipPostalCode`, and `ShipCountry` because they display repeating sets of values:

```
SELECT
  OrderID,
  EmployeeID,
  ShipVia,
  OrderDate,
  RequiredDate,
  ShippedDate,
  Freight
FROM
  tblOrders
WHERE
  CustomerID = 'MAGAA';
```

- Show customers whose `ContactTitle` is not `"Sales Associate"`. Display `CustomerID`, `CompanyName`, `ContactName`, and `ContactTitle`

```
SELECT
  CustomerID,
  CompanyName,
  ContactName,
  ContactTitle
FROM
  tblCustomers
WHERE
  ContactTitle != 'Sales Associate';
```

- Show customers who bought products where the `EnglishName` includes the string `"chocolate"`. Display `CustomerID`, `CompanyName`, `ProductID`, `ProductName`, and `EnglishName`

```sql
SELECT
  tblCustomers.CustomerID,
  tblCustomers.CompanyName,
  tblProducts.ProductID,
  tblProducts.ProductName,
  tblProducts.EnglishName
FROM
  tblCustomers
  LEFT JOIN tblOrders ON tblCustomers.CustomerID = tblOrders.CustomerID
  LEFT JOIN tblOrderDetails ON tblOrders.OrderID = tblOrderDetails.OrderID
  LEFT JOIN tblProducts ON tblOrderDetails.ProductID = tblProducts.ProductID
WHERE
  tblProducts.ProductID IN (
    SELECT
      tblProducts.ProductID
    FROM
      tblProducts
    WHERE
      EnglishName LIKE '%chocolate%'
  );
```

- Show products which were bought by customers from Italy or USA. Display `CustomerID`, `CompanyName`, `ShipCountry`, `ProductID`, `ProductName`, and `EnglishName`

```sql
SELECT
  tblCustomers.CustomerID,
  tblCustomers.CompanyName,
  tblOrders.ShipCountry,
  tblProducts.ProductID,
  tblProducts.ProductName,
  tblProducts.EnglishName
FROM
  tblCustomers
  LEFT JOIN tblOrders ON tblCustomers.CustomerID = tblOrders.CustomerID
  LEFT JOIN tblOrderDetails ON tblOrders.OrderID = tblOrderDetails.OrderID
  LEFT JOIN tblProducts ON tblOrderDetails.ProductID = tblProducts.ProductID
WHERE
  tblOrders.ShipCountry LIKE 'Italy'
  OR tblOrders.ShipCountry LIKE 'USA';
```

- Show total price of each product in each order. Note that there is not a column named as total price. You should calculate it and create a column named as `TotalPrice`. Display `OrderID`, `ProductID`, `ProductName`, `UnitPrice`, `Quantity`, `Discount`, and `TotalPrice`

```sql
SELECT
  tblOrderDetails.OrderID,
  tblOrderDetails.ProductID,
  tblProducts.ProductName,
  tblOrderDetails.UnitPrice,
  tblOrderDetails.Quantity,
  tblOrderDetails.Discount,
  (
    tblOrderDetails.UnitPrice * tblOrderDetails.Quantity * (1 -
tblOrderDetails.Discount)
  ) AS TotalPrice
FROM
  tblOrderDetails
  LEFT JOIN tblProducts ON tblOrderDetails.ProductID =
tblProducts.ProductID;
```

- Show how many products there are in each category and show the results in ascending order by the total number of products. Display `CategoryName`, and `TotalProducts`

```sql
SELECT
  CategoryID AS CategoryName,
  COUNT (*) AS TotalProducts
FROM
  tblProducts
GROUP BY
  CategoryName
ORDER BY
  TotalProducts ASC;
```

- Show the total number of customers in each `City`. Display `Country`, `City`, `TotalCustomers`

```sql
SELECT
  Country,
  City,
  COUNT (*) AS TotalCustomers
FROM
  tblCustomers
GROUP BY
  City;
```

- Show the orders which were shipped late than the actual required date. Display `OrderID`, `OrderDate`, `RequiredDate`, and `ShippedDate`

```sql
SELECT
  OrderID,
  OrderDate,
  RequiredDate,
  ShippedDate
FROM
  tblOrders
WHERE
  RequiredDate < ShippedDate;
```