COMP 1630 (FRI)

A00701896 : NGAMPIT SUTTHASILP

PART B - SQL Statements

B1. List the order details where the quantity is between 65 and 70. Display the order id and quantity from the OrderDetails table, the product id and reorder level from the Products table, and the supplier id from the Suppliers table. Order the result set by the order id. The query should produce the result set listed below.

```
-- PART B.1
SELECT o.OrderID,
       OD.Quantity,
       P.ProductID,
       P.ReorderLevel,
       S.SupplierID
FROM Orders O
INNER JOIN OrderDetails OD ON O.OrderID    = OD.OrderID
INNER JOIN Products P      ON OD.ProductID = P.ProductID
INNER JOIN Suppliers S     ON S.SupplierID = P.SupplierID
WHERE OD.Quantity BETWEEN 65 AND 70
ORDER BY O.OrderID
```

B2. List the product id, product name, English name, and unit price from the Products table where the unit price is less than $8.00. Order the result set by the product id. The query should produce the result set listed below.

```
--Part B.2
SELECT   ProductID,
         ProductName,
         EnglishName,
         UnitPrice
FROM     Products
WHERE    UnitPrice < 8
ORDER BY ProductID
```

B3. List the customer id, company name, country, and phone from the Customers table where the country is equal to Canada or USA. Order the result set by the customer id. The query should produce the result set listed below.

```
-- Part B.3
SELECT   CustomerID,
         CompanyName,
         Country,
         Phone
FROM     Customers
WHERE    Country IN ('USA','Canada')
ORDER BY CustomerID
```

B4. List the products where the reorder level is equal to the units in stock. Display the supplier id and supplier name from the Suppliers table, the product name, reorder level, and units in stock from the Products table. Order the result set by the supplier id. The query should produce the result set listed below.

```
--Part B.4
SELECT    S.SupplierID,
          S.Name,
          P.ProductName,
          P.ReorderLevel,
          P.UnitsInStock
FROM Suppliers S
INNER JOIN Products P ON P.SupplierID = S.SupplierID
WHERE    P.ReorderLevel = P.UnitsInStock
ORDER BY S.SupplierID
```

B5. List the orders where the shipped date is greater than or equal to 1st of Jan 1994, and calculate the length in years from the shipped date to 1st of Jan 2009. Display the order id, and the shipped date from the Orders table, the company name, and the contact name from the Customers table, and the calculated length in years for each order. Display the shipped date in the format MMM DD YYYY. Order the result set by order id and the calculated years. The query should produce the result set listed below.

```
--Part B.5
SELECT O.OrderID,
       C.CompanyName,
       C.ContactName,
          CONVERT(CHAR(12),O.ShippedDate, 109)      AS  ShippedDate,
          DATEDIFF(YEAR,O.ShippedDate,'1 Jan 2009') AS  ElapsedYears
FROM Orders O
INNER JOIN Customers C ON C.CustomerID = O.CustomerID
INNER JOIN Shippers S  ON S.ShipperID  = O.ShipperID
WHERE O.ShippedDate >= '1 Jan1994'
ORDER BY O.OrderID
```

B6. List all the orders where the order date is between 1st of Jan and 30th of Mar 1992, and the cost of the order is greater than or equal to $1500.00. Display the order id, order date, and a new shipped date calculated by adding 10 days to the shipped date from the Orders table, the product name from the Products table, the company name from the Customer table, and the cost of the order. Format the date order date and the shipped date as MON DD YYYY. Use the formula (OrderDetails.Quantity * Products.UnitPrice) to calculate the cost of the order. Order the result set by order id. The query should produce the result set listed below.

```
-- Part B.6
SELECT o.OrderID,
       p.ProductName,
          c.CompanyName,
          CONVERT(CHAR(12),o.OrderDate,109)                   AS  OrderDate,
          CONVERT(CHAR(12),DATEADD(DAY,10,o.ShippedDate),109) AS  NewShippedDate,
       od.Quantity*p.UnitPrice                                AS  OrderCost
FROM Customers c
INNER JOIN Orders o          ON  o.CustomerID   = c.CustomerID
INNER JOIN OrderDetails od ON  o.OrderID      = od.OrderID
INNER JOIN Products p        ON  p.ProductID    = od.ProductID
WHERE o.OrderDate BETWEEN '1 Jan 1992' AND '30 Mar 1992'
AND (od.Quantity*p.UnitPrice)  >= 1500
ORDER BY  o.OrderID
```

B7. List all the orders with a shipping city of Vancouver. Display the order from the Orders table, and the unit price and quantity from the OrderDetails table. Order the result set by the order id. The query should produce the result set listed below.

```
-- PartB.7
SELECT o.OrderID,
       od.UnitPrice,
       od.Quantity
FROM   Orders o
INNER JOIN OrderDetails od ON o.OrderID = od.OrderID
WHERE o.ShipCity    = 'Vancouver'
AND   o.ShipCountry = 'Canada'
ORDER BY O.OrderID
```

B8. List all the orders that have not been shipped (shipped date is null). Display the customer id, company name and fax number from the Customers table, and the order id and order date from the Orders table. Order the result set by the customer id and order date. The query should produce the result set listed below.

```
-- PartB.8
SELECT c.CustomerID,
       c.CompanyName,
       c.Fax,
       o.OrderID,
       o.OrderDate
FROM   Customers c
INNER JOIN Orders o ON  c.CustomerID = o.CustomerID
WHERE  o.ShippedDate IS NULL
ORDER BY c.CustomerID
```

B9. List the products which contain choc or tofu in their name. Display the product id, product name, quantity per unit and unit price from the Products table. Order the result set by product id. The query should produce the result set listed below.

```
--Part B.9
SELECT    ProductID,
          ProductName,
          QuantityPerUnit,
          UnitPrice
FROM      Products
WHERE     (ProductName LIKE  '%choc%' or ProductName LIKE '%tofu%')
ORDER BY ProductID
```

B10. List the number of products and their names beginning with each letter of the alphabet. Only display the letter and count if there are at least three product names begin with the letter. The query should produce the result set listed below.

```
-- PartB.10
SELECT    SUBSTRING(ProductName,1,1)       AS  ProductName,
          COUNT(ProductName)               AS  Total
FROM      Products
GROUP BY SUBSTRING(ProductName,1,1)
HAVING   COUNT(ProductName) > = 3
ORDER BY ProductName
```

COMP 1630 (FRI)

A00701896: NGAMPIT SUTTHASILP

PART C - INSERT, UPDATE, DELETE and VIEWS

C1. Create a view called vw_supplier_items listing the distinct suppliers and the items they have shipped. Display the supplier id and name from the Suppliers table, and the product id and product name from the Products table. Use the following query to test your view to produce the result set listed below.

C1.1

```
CREATE VIEW vw_supplier_items
AS
SELECT      s.SupplierID,
            s.Name,
            p.ProductID,
            p.ProductName
FROM Suppliers s
INNER JOIN Products p ON p.SupplierID = s.SupplierID
```

```
C1.2 (verify)
```

```
SELECT       *
FROM        vw_supplier_items
ORDER BY    Name,ProductID
```

C2. Create a view called vw_employee_info to list all the employees in the Employee table. Display the employee id, last name, first name, and birth date. Format the name as first name followed by a space followed by the last name. Use the following query to test your view to produce the result set listed below.

C2.1

```
CREATE VIEW vw_employee_info
AS
SELECT      e.EmployeeID,
            (e.FirstName + ' ' + e.LastName)    AS  Name,
            e.BirthDate
FROM        Employees e
```

C2.2

```
SELECT *
FROM   vw_employee_info
WHERE  EmployeeID IN (3,6,9)
```

C3. Using the UPDATE statement, change the fax value to Unknown for all rows in the Customers table where the current fax value is null (22 rows affected)

```
UPDATE     Customers
SET        Fax  = 'UNKNOWN'
WHERE      Fax IS NULL
```

C4. Create a view called vw_order_cost to list the cost of orders. Display the order id and order_date from the Orders table, the product id from the Products table, the company name from the Customers table, and the order cost. To calculate the cost of the orders, use the formula: (OrderDetails.Quantity * OrderDetails.UnitPrice). Use the following query to test your view to produce the result set listed below.

--C4.1

```
CREATE VIEW       vw_order_cost
AS
SELECT            o.OrderID,
                  o.OrderDate,
                  p.ProductID,
                  c.CompanyName,
                  (od.Quantity * od.UnitPrice)         AS   OrderCost
FROM              Orders o
INNER JOIN        Customers c         ON      o.CustomerID = c.CustomerID
INNER JOIN        OrderDetails  od    ON     od.OrderID    = o.OrderID
INNER JOIN        Products p          ON      p.ProductID  = od.ProductID
```

--C4.2

```
SELECT       *
FROM         vw_order_cost
WHERE        orderID BETWEEN 10100 AND 10200
ORDER BY     ProductID
```

C5. Using the INSERT statement, add a row to the Suppliers table with a supplier id of 16 and a name of ‘Supplier P’.

--C5.1

```
INSERT INTO   Suppliers
(       SupplierID,
        Name                )
VALUES
(       '16',
        'Supplier P'        )
```

--C5.2

```
SELECT    *
FROM     Suppliers
WHERE    SupplierID = 16
```

C6. Using the UPDATE statement, increate the unit price in the Products table by 15% for rows with a current unit price less than $5.00 (2 rows affected).

```
UPDATE      Products
SET         UnitPrice = (UnitPrice * 1.15)
WHERE       UnitPrice < 5
```

C7. Create a view called vw_orders to list orders. Display the order id and shipped date from the Orders table, and the customer id, company name, city, and country from the Customers table. Use the following query to test your view to produce the result set listed below.

--C7.1

```
CREATE VIEW      vw_orders
AS
SELECT           o.OrderID,
                 c.CustomerID,
                             c.CompanyName,
                             c.City,
                             c.Country,
                             o.ShippedDate
FROM             Orders o
INNER JOIN       Customers c    ON    c.CustomerID = o.CustomerID
```

--C7.2

```
SELECT    *
FROM      vw_orders
WHERE     ShippedDate  BETWEEN  '1993-01-01' AND '1993-01-31'
ORDER BY  CompanyName,Country
```

COMP 1630 (FRI)

A00701896: NGAMPIT SUTTHASILP

PART D - Stored Procedures and Triggers

D1. Create a stored procedure called sp_emp_info to display the employee id, last name, first name, and phone number from the Employees table for a particular employee. The employee id will be an input parameter for the stored procedure. Use the following query to test your stored procedure to produce the result set listed below.

```
 --PartD.1
CREATE PROCEDURE sp_emp_info
(         @emp_id     varchar(16))
AS
SELECT    EmployeeID,
          LastName,
                FirstName,
                Phone
FROM      Employees
WHERE     EmployeeID = @emp_id
GO
```

--Part D.2

EXEC sp_emp_info 7

D2. Create a stored procedure called sp_orders_by_dates displaying the orders shipped between particular dates. The start and end date will be input parameters for the stored procedure. Display the order id, customer id, and shipped date from the Orders table, the company name from the Customer table, and the shipper name from the Shippers table. Use the following query to test your stored procedure to produce the result set listed below.

```
--Part D.2
CREATE PROCEDURE sp_orders_by_dates
(        @start_date     varchar(11),
        @end_date       varchar(11))
AS
SELECT         o.OrderID,
               c.CustomerID,
               c.CompanyName,
               s.CompanyName   AS  ShipperName,
               o.ShippedDate
FROM         Orders o
INNER JOIN  Customers c  ON  c.CustomerID = o.CustomerID
INNER JOIN  Shippers s   ON  s.ShipperID  = o.ShipperID
WHERE    o.ShippedDate BETWEEN @start_date AND @end_date
GO

--Part D.2

EXEC sp_orders_by_dates '1991-01-01', '1991-12-31'
```

D3. Create a stored procedure called sp_products listing a specified product ordered during a specified month and year. The product name, month, and year will be input parameters for the stored procedure. Display the product name, unit price, and units in stock from the Products table, and the supplier name from the Suppliers table. Use the following query to test your stored procedure to produce the result set listed below.

```
--Part D.3
CREATE PROCEDURE sp_products
(       @pro_name       varchar(11),
        @month          varchar(7)
        @year           varchar(7) )
AS
SELECT         p.ProductName,
               p.UnitPrice,
               p.UnitsInStock,
               s.Name
FROM           Suppliers  s
INNER JOIN     Products   p   ON  s.SupplierID = p.SupplierID
WHERE    p.ProductName = @pro_name,

GO
```

D4. Create a stored procedure called sp_unit_prices listing the products where the unit price is between particular values. The two unit prices will be input parameters for the stored procedure. Display the product id, product name, English name, and unit price from the Products table. Use the following query to test your stored procedure to produce the result set listed below.

```
--Part D.3.1
CREATE PROCEDURE sp_unit_prices
(       @first_price      money,
        @second_price     money    )
AS
SELECT  ProductID,
        ProductName,
              EnglishName,
              UnitPrice
FROM    Products
WHERE   UnitPrice BETWEEN @first_price AND @second_price
GO
```

--PartD.3.2

EXEC sp_unit_prices 5.50, 8.00

D5. Create a stored procedure called sp_customer_city displaying the customers living in a particular city. The city will be an input parameter for the stored procedure. Display the customer id, company name, address, city and phone from the Customers table. Use the following query to test your stored procedure to produce the result set listed below.

PartD5.1

```
--Part D.3
CREATE PROCEDURE sp_customer_city
(          @city      varchar(10)   )
AS
SELECT   CustomerID,
         CompanyName,
         Address,
         City,
         Phone
FROM     Customers
WHERE    City = @city
GO
```

PartD5.2

```
EXEC sp_customer_city 'Paris'
```

D6. Create a stored procedure called sp_reorder_qty to show when the reorder level subtracted from the units in stock is less than a specified value. The unit value will be an input parameter for the stored procedure. Display the product id, product name, units in stock, and reorder level from the Products table, and the supplier name from the Suppliers table. Use the following query to test your stored procedure to produce the result set listed below.

```
--Part D.6
CREATE PROCEDURE sp_reorder_qty
(       @unit      varchar(10)   )
AS
SELECT          p.ProductID,
                p.ProductName,
                s.Name,
                p.UnitsInStock,
                p.ReorderLevel
FROM            Products p
INNER JOIN    Suppliers s ON  p.SupplierID = s.SupplierID
WHERE   @unit > p.UnitsInStock-p.ReorderLevel
GO



--PartD6.2
EXEC sp_reorder_qty 9
```

D7. Create a stored procedure called sp_shipping_date where the shipped date is equal to the order date plus 10 days. The shipped date will be an input parameter for the stored procedure. Display the order id, order date and shipped date from the Orders table, the company name from the Customers table, and the company name from the Shippers table. Use the following query to test your stored procedure to produce the result set listed below.

```
 --Part D.7.1
CREATE PROCEDURE sp_shipping_date
(       @shipped_date      varchar(10)   )
AS
SELECT          o.OrderID,
                c.CompanyName    AS  CustomerName,
                s.CompanyName    AS  ShipperName,
              o.OrderDate,
              o.ShippedDate
FROM         Orders o
INNER JOIN   Customers c ON  o.CustomerID = c.CustomerID
INNER JOIN   Shippers  s ON  o.ShipperID  = s.ShipperID
WHERE   @shipped_date = DATEADD(DAY,10,o.OrderDate)
GO
```

--Part D.7.2

```
EXEC sp_shipping_date '1993-11-29'
```

D8. Create a stored procedure called sp_del_inactive_cust to delete customers that have no orders. Use the following query to test your procedure. The stored procedure should delete 1 row.

--PartD.8

```
CREATE PROCEDURE        sp_del_inactive_cust
AS
DELETE                  customers
FROM                    customers c
LEFT OUTER JOIN         orders o ON c.CustomerID = o.CustomerID
WHERE                   o.CustomerID IS NULL
GO
```

--PartD.8.2

```
EXEC sp_del_inactive_cust
```

D9. Create an UPDATE trigger called tr_check_qty on the OrderDetails table to prevent the updating of orders of products that have units-in-stock less than the quantity ordered. Use the following query to test your trigger.

```
CREATE TRIGGER tr_check_qty
ON      OrderDetails
FOR     UPDATE
AS
SELECT Quantity
FROM   inserted i
INNER JOIN  Products p ON i.ProductID = p.ProductID
WHERE       p.UnitsInStock < i.Quantity
    BEGIN
          PRINT 'Quantity units in stock less than quantity ordered, can not update'
          ROLLBACK TRANSACTION
       END
GO
```

PartD9.2

```
UPDATE OrderDetails SET Quantity = 40 WHERE OrderID = 10044 AND ProductID = 77
```

D10. Create an INSTEAD OF INSERT trigger called tr_insert_shippers on the Shippers table preventing inserting a row with a company name which already exists. Use the following query to test your trigger.

```
CREATE TRIGGER tr_insert_shippers
ON Shippers
INSTEAD OF INSERT
AS
IF EXISTS (SELECT CompanyName FROM inserted i
    WHERE i.CompanyName = CompanyName)
   BEGIN
       PRINT 'Company Already Exist'
   END
ELSE
   BEGIN
     INSERT INTO Shippers
        SELECT *
        FROM   inserted
   END
GO
```

PartD10.2

```
INSERT Shippers VALUES ( 4, 'Federal Shipping' )
```