APPENDIX: codes used

```
/*                                                                         */
/*         COMP 1630 Project 2 - The Cus_Orders database                   */
/*                         By Yuzhe Stan Chen                              */

/* Part A - Database and Tables */

--A 1.
/* To create the new database, we need to make sure Master is selected first*/
USE MASTER;
GO

/* We also need to check if the database exists, if it does delete it */
/* since IF statements can only take a single SQL statement,
we can have a Begin-End block to include more statements. */

IF EXISTS (SELECT * FROM sysdatabases WHERE name='Cus_Orders')
begin
   raiserror('Dropping existing Cus_Orders database ....',0,1)
   DROP DATABASE Cus_Orders;
end;
GO

/* Now we can create the Cus_Orders DB */
print 'Creating Cus_Orders database....';
CREATE DATABASE Cus_Orders;
GO


-- A 2.
/* Set the newly created database as the current database before creating tables */
USE Cus_Orders;

/* The following will make sure we have choosen the right database, Cus_Orders */
if db_name() <> 'Cus_Orders'
   raiserror('Errors in Creating or Selecting Cus_Orders, please STOP now.'
            ,22,127) with log
else print 'Checked: Cus_Orders in USE!'
GO

/* Check existence of old data type objects,
and create new user defined data types */
DROP TYPE IF EXISTS dbo.csid_ch5;
DROP TYPE IF EXISTS dbo.csid_int;
CREATE TYPE csid_ch5 FROM char(5) NOT NULL;
CREATE TYPE csid_int FROM int NOT NULL;
GO

-- A 3.
/* Check the existence of tables before creating them */
DROP TABLE IF EXISTS dbo.customers;
DROP TABLE IF EXISTS dbo.orders;
DROP TABLE IF EXISTS dbo.order_details;
DROP TABLE IF EXISTS dbo.products;
DROP TABLE IF EXISTS dbo.shippers;
DROP TABLE IF EXISTS dbo.suppliers;
DROP TABLE IF EXISTS dbo.titles;
GO
```

```
/* Now we can create the tables with desired data types */
print 'Creating tables... ';
CREATE TABLE customers(
customer_id csid_ch5,
name varchar(50) NOT NULL,
contact_name varchar(30),
title_id char(3) NOT NULL,
address varchar(50),
city varchar(20),
region varchar(15),
country_code varchar(10),
country varchar(15),
phone varchar(20),
fax varchar(20)
);
GO

CREATE TABLE orders(
order_id csid_int,
customer_id csid_ch5,
employee_id int NOT NULL,
shipping_name varchar(50),
shipping_address varchar(50),
shipping_city varchar(20),
shipping_region varchar(15),
shipping_country_code varchar(10),
shipping_country varchar(15),
shipper_id int NOT NULL,
order_date datetime,
required_date datetime,
shipped_date datetime,
freight_charge money
);
GO

CREATE TABLE order_details(
order_id csid_int,
product_id int NOT NULL,
quantity int NOT NULL,
discount float NOT NULL
);
GO

CREATE TABLE products(
product_id  csid_int,
supplier_id int NOT NULL,
name varchar(40) NOT NULL,
alternate_name varchar(40),
quantity_per_unit varchar(25),
unit_price money,
quantity_in_stock int,
units_on_order int,
reorder_level int,
);
GO

CREATE TABLE shippers(
shipper_id int IDENTITY(1,1),
name varchar(20) NOT NULL
);
GO
```

```
CREATE TABLE suppliers(
supplier_id int   IDENTITY(1,1) NOT NULL,
name varchar(40) NOT NULL,
address varchar(30),
city varchar(20),
province char(2)
);
GO

CREATE TABLE titles(
title_id char(3) NOT NULL,
description varchar(35) NOT NULL
);
GO


-- A 4.
/* We can add PKs, FKs, and other constraints by altering tables */
/* Let's start with adding the PKs */
ALTER TABLE customers
ADD PRIMARY KEY ( customer_id );

ALTER TABLE orders
ADD PRIMARY KEY ( order_id );

ALTER TABLE order_details
ADD PRIMARY KEY ( order_id, product_id );

ALTER TABLE titles
ADD PRIMARY KEY ( title_id );

ALTER TABLE shippers
ADD PRIMARY KEY ( shipper_id );

ALTER TABLE suppliers
ADD PRIMARY KEY ( supplier_id );

ALTER TABLE products
ADD PRIMARY KEY ( product_id );

Go


/*Then the FKs */
ALTER TABLE customers
ADD CONSTRAINT FK_customer_title FOREIGN KEY (title_id)
REFERENCES titles (title_id);

ALTER TABLE orders
ADD CONSTRAINT FK_orders_customers FOREIGN KEY (customer_id)
REFERENCES customers (customer_id);

ALTER TABLE orders
ADD CONSTRAINT FK_orders_shippers FOREIGN KEY (shipper_id)
REFERENCES shippers (shipper_id);

ALTER TABLE order_details
ADD CONSTRAINT FK_order_details_orders FOREIGN KEY (order_id)
REFERENCES orders (order_id);

ALTER TABLE order_details
```

```
ADD CONSTRAINT FK_order_details_products FOREIGN KEY (product_id)
REFERENCES products (product_id);

ALTER TABLE products
ADD CONSTRAINT FK_product_supplier FOREIGN KEY (supplier_id)
REFERENCES suppliers (supplier_id);

GO


/* Now the other constraints */
ALTER TABLE customers
ADD CONSTRAINT default_country
DEFAULT ( 'Canada' ) FOR country;

ALTER TABLE orders
ADD CONSTRAINT default_required_date
DEFAULT (DATEADD (DAY, 10, GETDATE())) FOR required_date;

ALTER TABLE order_details
ADD CONSTRAINT ch_min_qty
CHECK (quantity >= 1);

ALTER TABLE products
ADD CONSTRAINT ch_max_qty_stock
CHECK (quantity_in_stock <= 150);

ALTER TABLE products
ADD CONSTRAINT ch_min_reorder_lv
CHECK (reorder_level >= 1);

ALTER TABLE suppliers
ADD CONSTRAINT default_province
DEFAULT ( 'BC' ) FOR province;
GO


print 'Cus_Orders database has been created....';
Go


/********************************************************************************/
/* The following is the INSERT data codes provided */

BULK INSERT titles
FROM 'C:\TextFiles\titles.txt'
WITH (
              CODEPAGE=1252,
            DATAFILETYPE = 'char',
            FIELDTERMINATOR = '\t',
            KEEPNULLS,
            ROWTERMINATOR = '\n'
       )

BULK INSERT suppliers
FROM 'C:\TextFiles\suppliers.txt'
WITH (
              CODEPAGE=1252,
            DATAFILETYPE = 'char',
            FIELDTERMINATOR = '\t',
            KEEPNULLS,
            ROWTERMINATOR = '\n'
        )

BULK INSERT shippers
FROM 'C:\TextFiles\shippers.txt'
WITH (
               CODEPAGE=1252,
            DATAFILETYPE = 'char',
            FIELDTERMINATOR = '\t',
            KEEPNULLS,
            ROWTERMINATOR = '\n'
        )

BULK INSERT customers
FROM 'C:\TextFiles\customers.txt'
WITH (
            CODEPAGE=1252,
            DATAFILETYPE = 'char',
            FIELDTERMINATOR = '\t',
            KEEPNULLS,
            ROWTERMINATOR = '\n'
        )

BULK INSERT products
FROM 'C:\TextFiles\products.txt'
WITH (
            CODEPAGE=1252,
            DATAFILETYPE = 'char',
            FIELDTERMINATOR = '\t',
            KEEPNULLS,
            ROWTERMINATOR = '\n'
        )

BULK INSERT order_details
FROM 'C:\TextFiles\order_details.txt'
WITH (
            CODEPAGE=1252,
            DATAFILETYPE = 'char',
            FIELDTERMINATOR = '\t',
            KEEPNULLS,
            ROWTERMINATOR = '\n'
        )

BULK INSERT orders
FROM 'C:\TextFiles\orders.txt'
WITH (
            CODEPAGE=1252,
            DATAFILETYPE = 'char',
            FIELDTERMINATOR = '\t',
            KEEPNULLS,
            ROWTERMINATOR = '\n'
        )


/***************************************************************/

/* Part B. SQL Statements */
-- B 1.
SELECT customer_id,
       name,
       city,
       country
FROM customers
ORDER BY customer_id;
GO

-- B 2.
ALTER TABLE customers
ADD active bit;

ALTER TABLE customers
ADD CONSTRAINT default_active
DEFAULT ( '1' ) FOR active;
GO

-- B 3.
SELECT orders.order_id,
       'Product Name' = products.name,
       'Customer Name' = customers.name,
       order_date = CONVERT(char(11), orders.order_date, 109),
       'new_shipped_date'= CONVERT(char(11),
       DATEADD(DAY, 7, orders.shipped_date), 109),
       'order_cost' = order_details.quantity*products.unit_price
FROM orders
INNER JOIN customers ON orders.customer_id = customers.customer_id
INNER JOIN order_details ON orders.order_id = order_details.order_id
INNER JOIN products ON order_details.product_id = products.product_id
WHERE ( DATENAME(MONTH, orders.order_date)='January' AND DATENAME(YEAR,
orders.order_date)='2004')
OR ( DATENAME(MONTH, orders.order_date)='February' AND DATENAME(YEAR,
orders.order_date)='2004');
GO

/*
Alt. way of WHERE clause:
... (continue from the above, but replace the WHERE clause)
WHERE ( DATENAME(MONTH, orders.order_date)='January' AND DATENAME(YEAR,
orders.order_date)='2004')
OR ( DATENAME(MONTH, orders.order_date)='February' AND DATENAME(YEAR,
orders.order_date)='2004')
*/



-- B 4.
SELECT 'Cus_Id' = customers.customer_id,
       'Cus_Name' = customers.name,
       'Cus_Phone' = customers.phone,
       'Order_No' = orders.order_id,
       'Order Date' = CONVERT(char(11), orders.order_date, 109)
FROM customers
INNER JOIN orders ON customers.customer_id = orders.customer_id
WHERE orders.shipped_date IS NULL
ORDER BY orders.order_date;
GO

-- B 5.
SELECT customers.customer_id,
       customers.name,
       customers.city,
       titles.description
FROM customers
```

```
INNER JOIN titles ON customers.title_id = titles.title_id
WHERE customers.region IS NULL;
GO

-- B 6.
SELECT 'supplier_name' = suppliers.name,
       'product_name' = products.name,
       products.reorder_level,
       products.quantity_in_stock
FROM suppliers
INNER JOIN products ON suppliers.supplier_id = products.supplier_id
WHERE products.reorder_level > products.quantity_in_stock
ORDER BY suppliers.name;
GO

-- B 7.
SELECT orders.order_id,
       'Customer Name' = customers.name,
       customers.contact_name,
       'shipped_date' = CONVERT(char(11), orders.shipped_date, 109),
       'elapsed' = DATEDIFF(YEAR, orders.shipped_date, 'Jan 01 2008')
FROM orders
INNER JOIN customers ON orders.customer_id = customers.customer_id
WHERE orders.shipped_date IS NOT NULL
ORDER BY orders.order_id, 'elapsed';
GO

-- B 8.
SELECT 'First Letter of Customer''s Name' = SUBSTRING(name, 1,1),
       'Total Count' = COUNT(*)
FROM customers
GROUP by SUBSTRING(name, 1,1)
HAVING SUBSTRING(name, 1,1) != 'F' AND SUBSTRING(name, 1,1) != 'G' AND COUNT(*) >= 6;
GO


-- B 9.
SELECT order_details.order_id,
       order_details.quantity,
       products.product_id,
       products.reorder_level,
       suppliers.supplier_id
FROM order_details
INNER JOIN products ON order_details.product_id = products.product_id
INNER JOIN suppliers ON products.supplier_id = suppliers.supplier_id
WHERE order_details.quantity > 100
ORDER BY order_details.order_id;
GO

-- B 10.
SELECT product_id,
       name,
       quantity_per_unit,
       unit_price
FROM products
WHERE name LIKE '%tofu%' OR name LIKE '%chef%';
GO


/********************************************************************************/
```

```
/* Part C */
-- C 1.
/* Check the existence of table employee */
DROP TABLE IF EXISTS dbo.employee;
GO

/* Now create the table */
CREATE TABLE employee
(
employee_id int NOT NULL,
last_name varchar(30) NOT NULL,
first_name varchar(15)  NOT NULL,
address varchar(30),
city varchar(20),
province char(2),
postal_code varchar(7),
phone varchar(10),
birth_date datetime NOT NULL
);

print 'Employee table created...';
GO

-- C 2.
/* Now add the constrains, such as the PK to employee table */
ALTER TABLE employee
ADD PRIMARY KEY ( employee_id );
GO

-- C 3.
/* Load the data for employee table, and check the results */
BULK INSERT employee
FROM 'C:\TextFiles\employee.txt'
WITH (          CODEPAGE=1252,
            DATAFILETYPE = 'char',
            FIELDTERMINATOR = '\t',
            KEEPNULLS,
            ROWTERMINATOR = '\n'
       );

SELECT *
FROM employee;
GO

/* And add the FKs to orders table to reference the employee_id */
ALTER TABLE orders
ADD CONSTRAINT FK_orders_employee FOREIGN KEY (employee_id)
REFERENCES employee (employee_id);
GO


-- C 4.
/* Check the shippers table before INSERT the new shipper */
SELECT *
FROM shippers;
GO

/* Now we are ready to do the INSERT */
INSERT INTO shippers (name)
VALUES ('Quick Express ');
```

```
SELECT *
FROM shippers;
GO


-- C 5.
/* Check the products table before UPDATE the rows */
SELECT *
FROM products
WHERE unit_price >= 5 AND unit_price <= 10
ORDER BY unit_price;
GO

/* Now do the update with unit_price */
UPDATE products
SET unit_price = products.unit_price*1.05
WHERE unit_price >= 5 AND unit_price <= 10;
GO

/* And remember to check what have been done */
SELECT *
FROM products
WHERE unit_price >= 5.25 AND unit_price <= 10.5
ORDER BY unit_price;
GO


-- C 6.
/* Check the customers table before UPDATE the rows */
SELECT *
FROM customers
WHERE fax IS NULL;
GO

/* Now do the update with fax */
UPDATE customers
SET fax = 'Unknown'
WHERE fax IS NULL;
GO

/* Remeber to check what have been done */
SELECT *
FROM customers
WHERE fax = 'Unknown';
GO


-- C 7.
/* Check existence of vw_order_cost, drop if already exists */
DROP VIEW IF EXISTS vw_order_cost;
GO

/*Create the view vw_order_cost */
CREATE VIEW vw_order_cost
AS
SELECT  orders.order_id,
        orders.order_date,
      products.product_id,
      customers.name,
      'order_cost' = order_details.quantity * products.unit_price
FROM orders
```

```
INNER JOIN order_details ON order_details.order_id = orders.order_id
INNER JOIN products ON order_details.product_id = products.product_id
INNER JOIN customers ON orders.customer_id = customers.customer_id;
GO
print 'view vw_order_cost created...';
GO

/* Now we can run the view vw_order_cost */
SELECT *
FROM vw_order_cost
WHERE order_id >= 10000 AND order_id <= 10200;
GO


-- C 8.
/* Check existence of vw_list_employees, drop if already exists */
DROP VIEW IF EXISTS vw_list_employees;
GO

/*Create the view vw_list_employees */
CREATE VIEW vw_list_employees
AS
SELECT     employee.employee_id,
           employee.last_name,
         employee.first_name,
         employee.address,
         employee.city,
         employee.province,
         employee.postal_code,
         employee.phone,
         employee.birth_date
FROM employee;
GO
print 'view vw_order_cost created...';
GO

/* Now we can run the view vw_list_employees */
SELECT employee_id,
       'name' = last_name +', '+ first_name,
         'birth_date' = CONVERT(CHAR(10),birth_date,102)
FROM vw_list_employees
WHERE employee_id in (5, 7, 9);
GO


-- C 9.
/* Check existence of vw_all_orders, drop if already exists */
DROP VIEW IF EXISTS vw_all_orders;
GO

/*Create the view vw_all_orders */
CREATE VIEW vw_all_orders
AS
SELECT orders.order_id,
       orders.shipped_date,
       customers.customer_id,
       customers.name,
       customers.city,
       customers.country
FROM orders
INNER JOIN customers ON orders.customer_id = customers.customer_id;
```

```
GO
print 'view vw_all_orders created...';
GO

/*Now we can run the view vw_all_orders */
SELECT order_id,
       customer_id,
       'customer_name' = name,
       city,
       country,
      'shipped_date' =  CONVERT(CHAR(10),shipped_date,107)
FROM vw_all_orders
WHERE shipped_date >= 'Aug 1 2002' AND shipped_date <= 'Sep 30 2002'
ORDER BY name, country;
GO



-- C 10.
/* Check existence of vw_supplier_product, drop if already exists */
DROP VIEW IF EXISTS vw_supplier_product;
GO

/*Create the view vw_supplier_product */
CREATE VIEW vw_supplier_product
AS
SELECT suppliers.supplier_id,
       'supplier_name' = suppliers.name,
       products.product_id,
       'product_name' = products.name
FROM suppliers
INNER JOIN products ON products.supplier_id = suppliers.supplier_id;
GO
print 'vw_supplier_product created...';
GO

/*Now we can run the view vw_supplier_products */
SELECT *
FROM vw_supplier_product
ORDER BY supplier_id;
GO



/* Part D */
-- D 1.
/* Check existence of sp_customer_city, drop if already exists */
DROP PROCEDURE IF EXISTS sp_customer_city;
GO

/*Now build the procedures */
CREATE PROCEDURE sp_customer_city
(
            @city varchar(30)
)
AS
SELECT  customers.customer_id,
        customers.name,
      customers.address,
      customers.city,
      customers.phone
```

```
FROM customers
WHERE @city = customers.city;
GO

/* For execution */
EXECUTE sp_customer_city 'London';
GO


-- D 2.
/* Check existence of sp_orders_by_dates, drop if already exists */
DROP PROCEDURE IF EXISTS sp_orders_by_dates;
GO

/*Now build the procedures */
CREATE PROCEDURE sp_orders_by_dates
(
            @start_date datetime,
            @end_date datetime
)
AS
SELECT orders.order_id,
       orders.customer_id,
       'customer_name' = customers.name,
       'shipper_name' = shippers.name,
       orders.shipped_date
FROM orders
INNER JOIN customers ON orders.customer_id = customers.customer_id
INNER JOIN shippers ON orders.shipper_id = shippers.shipper_id
WHERE @start_date <= orders.shipped_date AND @end_date >= orders.shipped_date;
GO

/* For execution */
EXECUTE sp_orders_by_dates 'January 1 2003', 'June 30 2003';
GO


-- D 3.
/* Check xistence of sp_product_listing, drop if already exists */
DROP PROCEDURE IF EXISTS sp_product_listing;
GO

/*Now build the procedures */
CREATE PROCEDURE sp_product_listing
(
            @prodt varchar(30) = '%',
            @month varchar(30) = '%',
            @year varchar(30) = '%'
)
AS
SELECT 'product_name' = products.name,
        products.unit_price,
      products.quantity_in_stock,
      'supplier_name'= suppliers.name
FROM products
INNER JOIN suppliers ON products.supplier_id = suppliers.supplier_id
INNER JOIN order_details ON order_details.product_id = products.product_id
INNER JOIN orders ON order_details.order_id = orders.order_id
WHERE products.name LIKE @prodt
AND DATENAME(MONTH, orders.order_date)= @month
AND DATENAME(YEAR, orders.order_date)= @year;
```

```
GO

/* For execution */
EXECUTE sp_product_listing '%Jack%', 'June', '2001';
GO


-- D 4.
/* Check trigger existence, drop if already exists */
DROP TRIGGER IF EXISTS [dbo].[Deletion_order];
GO

/* Create the trigger */
CREATE TRIGGER Deletion_order
ON order_details
FOR DELETE
AS
DECLARE @orderid varchar(30), @prodid varchar(30), @qty int
SELECT @orderid = order_id,
       @prodid = product_id,
       @qty = quantity
FROM deleted

UPDATE products
SET products.quantity_in_stock = quantity_in_stock + @qty
WHERE products.product_id = @prodid
SELECT 'Product_ID' = products.product_id,
       'Product Name' = products.name,
       'Quantity being deleted from the Order' = @qty,
       'In stock Quantity after Deletion' = products.quantity_in_stock
FROM products
WHERE product_id = @prodid;
GO

/* For issuing the Delete command */
DELETE order_details
WHERE order_id=10001
AND product_id=25;
GO



-- D 5.
/* Check trigger existence, drop if already exists */
DROP TRIGGER IF EXISTS [dbo].[tr_check_qty];
GO

/* Create the trigger */
CREATE TRIGGER tr_check_qty
ON order_details
FOR INSERT, UPDATE
AS
DECLARE @prodid varchar(30), @qty int
SELECT @prodid = product_id,
       @qty = quantity
FROM inserted
IF @qty > (SELECT products.quantity_in_stock FROM products WHERE @prodid =
products.product_id)
      BEGIN
            PRINT 'No Higher Qty in Orders Than the Stock'
            ROLLBACK TRANSACTION
      END;
GO

/* Test this trigger by the following */
UPDATE order_details
SET quantity = 30
WHERE order_id = '10044'
AND product_id = 7;
GO


-- D 6.
/* To start with, let's do a OUTER JOIN to check if there is any customer that has no
orders */
SELECT customers.customer_id, orders.order_id
FROM customers
LEFT OUTER JOIN orders ON customers.customer_id = orders.customer_id
WHERE orders.order_id IS NULL;

/* Check existence of sp_del_inactive_cust, drop if already exists*/
DROP PROCEDURE IF EXISTS sp_del_inactive_cust;
GO
/*Now build the procedures */
CREATE PROCEDURE sp_del_inactive_cust
AS
DECLARE @cust_id varchar(30)

SELECT @cust_id = customers.customer_id
FROM customers
LEFT OUTER JOIN orders ON customers.customer_id = orders.customer_id
WHERE order_id IS NULL
print @cust_id + ' is being deleted...'
DELETE FROM customers
WHERE customer_id = @cust_id;
GO

/* For execution */
EXECUTE sp_del_inactive_cust;
GO


-- D 7.
/* Check existence of sp_employee_information, drop if already exists*/
DROP PROCEDURE IF EXISTS sp_employee_information;
GO

/*Now build the procedures */
CREATE PROCEDURE sp_employee_information
(
            @emp_id int
)
AS
SELECT last_name,
       first_name,
       address,
       city,
       province,
       postal_code,
       'DATE OF BIRTH' = CONVERT(CHAR(11),birth_date,109)
FROM employee
WHERE @emp_id = employee_id;
```

```
GO

/* For execution */
EXECUTE sp_employee_information '7';
GO


-- D 8.
/* Check existence of sp_reorder_qty, drop if already exists*/
DROP PROCEDURE IF EXISTS sp_reorder_qty;
GO

/*Now build the procedures */
CREATE PROCEDURE sp_reorder_qty
(
            @unit_val varchar(30)
)
AS
SELECT products.product_id,
       'Supplier Name' = suppliers.name,
       suppliers.address,
       suppliers.city,
       suppliers.province,
       products.quantity_in_stock,
       products.reorder_level
FROM products
INNER JOIN suppliers ON products.supplier_id = suppliers.supplier_id
WHERE @unit_val > products.quantity_in_stock - products.reorder_level;
GO

/* For execution */
EXECUTE sp_reorder_qty '5';
GO


-- D 9.
/* Check existence of sp_unit_prices, drop if already exists*/
DROP PROCEDURE IF EXISTS sp_unit_prices;
GO

/*Now build the procedures */
CREATE PROCEDURE sp_unit_prices
(
            @unit_pr1 money,
            @unit_pr2 money
)
AS
SELECT product_id,
       name,
       alternate_name,
       unit_price
FROM products
WHERE unit_price >= @unit_pr1 AND unit_price <= @unit_pr2;
GO

/* For execution */
EXECUTE sp_unit_prices 5, 10;
GO
```