COMP 1630 - Project Two

Instructor: Mark Bacchus
Date: 26 November 2019

Matthew Simpson
Student Number: A00820997

Table of Contents

Introduction

This is the second project for COMP1630 in Fall Semester of 2019 at BCIT. This project is designed to set our knowledge of SQL database creation and maintenance, from creating tables and populating them, to listing information from tables (including from multiple sources), to complex prepared statements and triggers to follow to allow for manipulating the data contained in said tables.

For each problem I will be including the original question (omitting any data that cannot be directly referenced in this document), the code I created to solve the question, and a screenshot from SQL Server Database Studio 2017.

Each screenshot will contain either:

A. The query window and the results/messages window beneath it, for all questions where there are no listed results beyond “command completed successfully”
B. The results / messages window for queries that have results.

This document may be a little on the long side as in putting in the effort to make sure it’s nicely formatted, each question will start a new page (or series of pages).

I did experience one challenge which is outlined in the challenges section at the end of the document.

I have also included two database diagrams as appendices, one after the initial setup and a second after the *employee* table is added in part C.

Lastly, I have appended my entire script as an appendix, also.

Part A

Concerning the creation of the database and the loading of data.

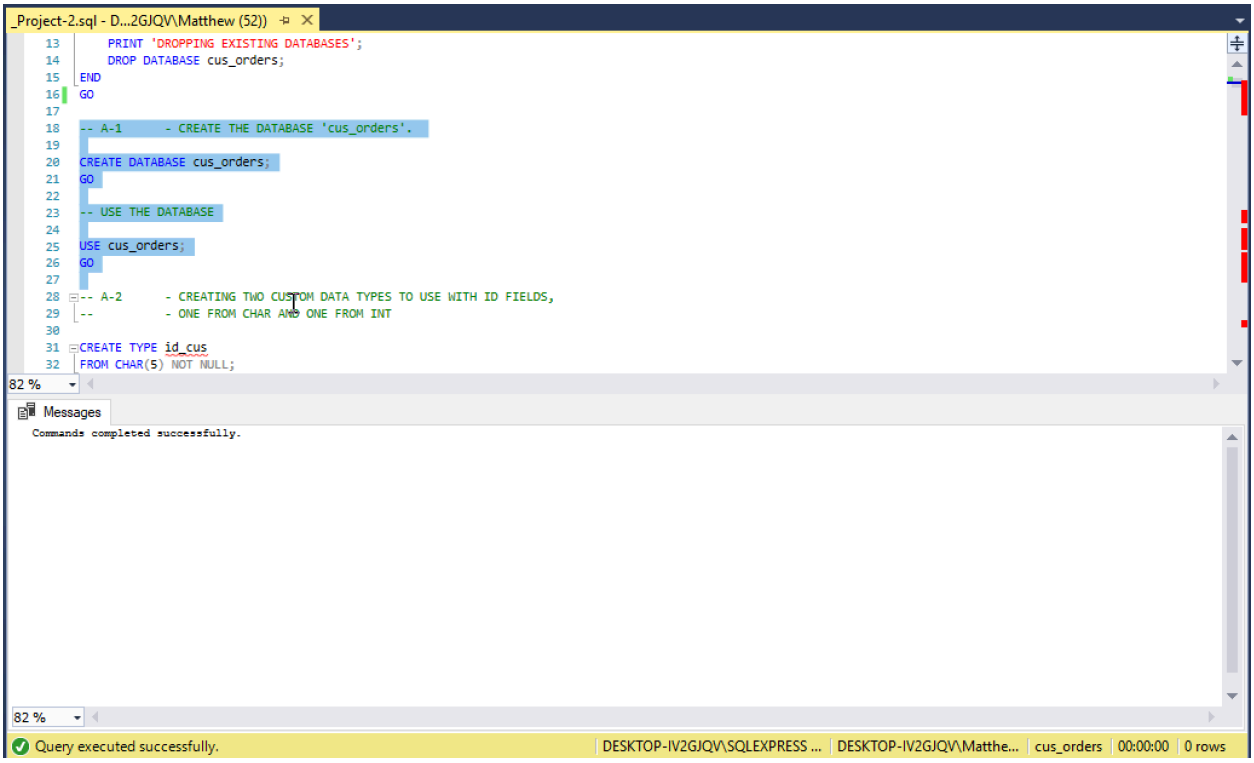
Problem A-1

Create a database called **cus_orders**

Code:

```
CREATE DATABASE cus_orders;
GO

USE cus_orders;
GO
```

Completed Successfully:



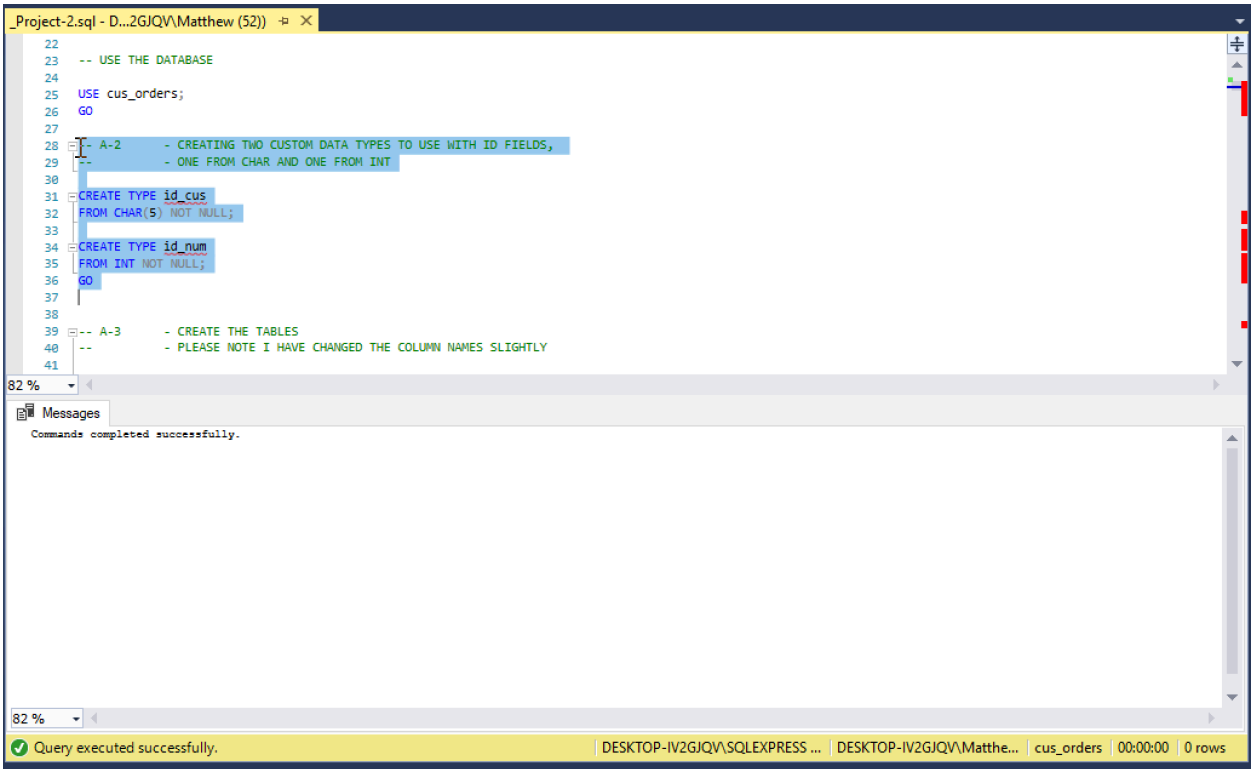
Problem A-2

Create a user defined data types for all similar Primary Key attribute columns (e.g. order_id, product_id, title_id**),** to ensure the same data type, length and null ability.

Code:

```
CREATE TYPE id_cus
FROM CHAR(5) NOT NULL;

CREATE TYPE id_num
FROM INT NOT NULL;
GO
```

Completed Successfully:



Problem A-3

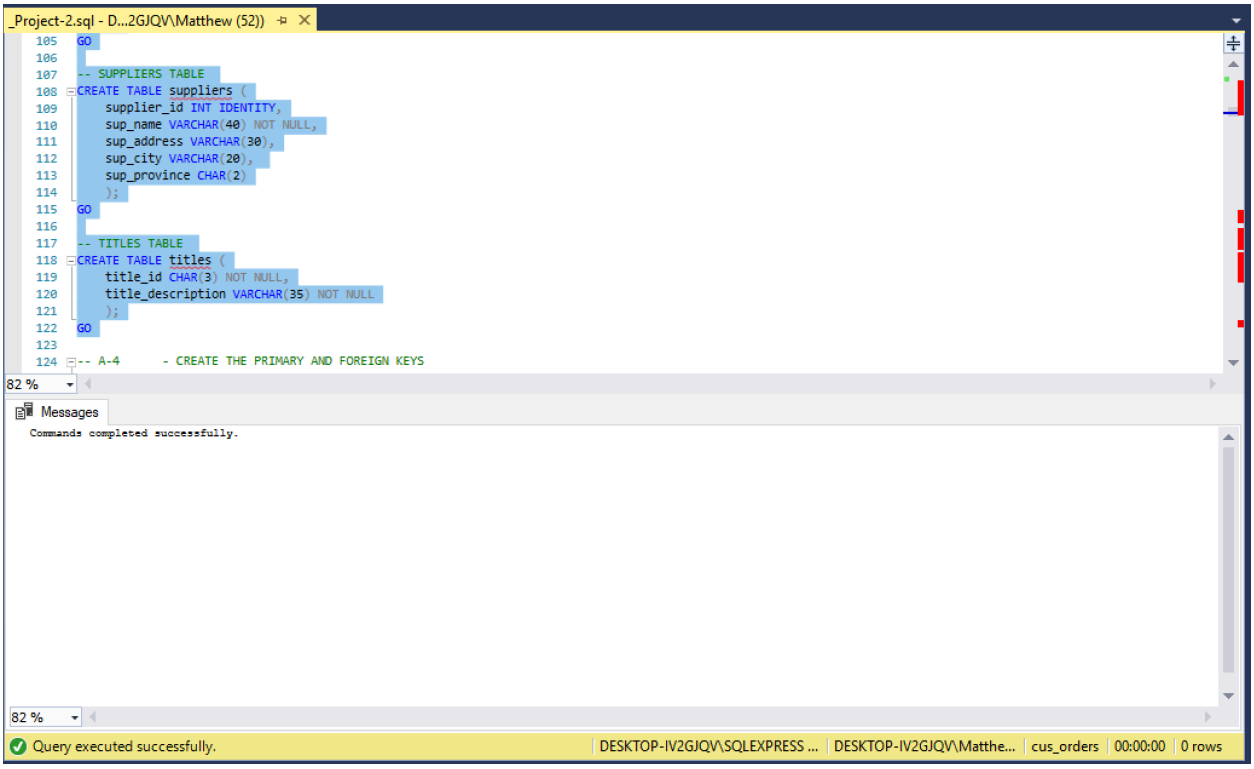
Create the following tables: *customers, orders, order_details, products, shippers, suppliers,* and *titles*

Code:

```
-- CUSTOMERS TABLE
CREATE TABLE customers (
  customer_id id_cus,
  customer_name VARCHAR(50) NOT NULL,
  customer_contact_name VARCHAR(30),
  title_id CHAR(3),
  customer_address VARCHAR(50) NOT NULL,
  customer_city VARCHAR(20),
  customer_region VARCHAR(15),
  customer_country_code VARCHAR(10),
  customer_country VARCHAR(15),
  customer_phone VARCHAR(20),
  customer_fax VARCHAR(20)
  );
GO

-- ORDERS TABLE
CREATE TABLE orders (
  order_id id_num,
  customer_id id_cus,
  employee_id INT NOT NULL,
  order_shipping_name VARCHAR(50),
  order_shipping_address VARCHAR(50),
  order_shipping_city VARCHAR(20),
  order_shipping_region VARCHAR(15),
  order_shipping_country_code VARCHAR(10),
  order_shipping_country VARCHAR(15),
  shipper_id INT NOT NULL,
  order_date DATETIME,
  order_required_date DATETIME,
  order_shipped_date DATETIME,
  order_freight_charge MONEY
  );
GO

-- ORDER_DETAILS TABLE
CREATE TABLE order_details (
  order_id id_num,
  product_id id_num,
  quantity INT NOT NULL,
  discount FLOAT NOT NULL
  );
GO
```

```
-- PRODUCTS TABLE
CREATE TABLE products (
  product_id id_num,
  supplier_id INT NOT NULL,
  prod_name VARCHAR(40),
  prod_alt_name VARCHAR(40),
  prod_qty_per_unit VARCHAR(25),
  prod_unit_price MONEY,
  prod_qty_in_stock INT,
  prod_units_on_order INT,
  prod_reorder_level INT
  );
GO

-- SHIPPERS TABLE
CREATE TABLE shippers (
  shipper_id INT IDENTITY,
  ship_name VARCHAR(20) NOT NULL
  );
GO

-- SUPPLIERS TABLE
CREATE TABLE suppliers (
  supplier_id INT IDENTITY,
  sup_name VARCHAR(40) NOT NULL,
  sup_address VARCHAR(30),
  sup_city VARCHAR(20),
  sup_province CHAR(2)
  );
GO

-- TITLES TABLE
CREATE TABLE titles (
  title_id CHAR(3) NOT NULL,
  title_description VARCHAR(35) NOT NULL
  );
GO
```

Completed Successfully:



Problem A-4

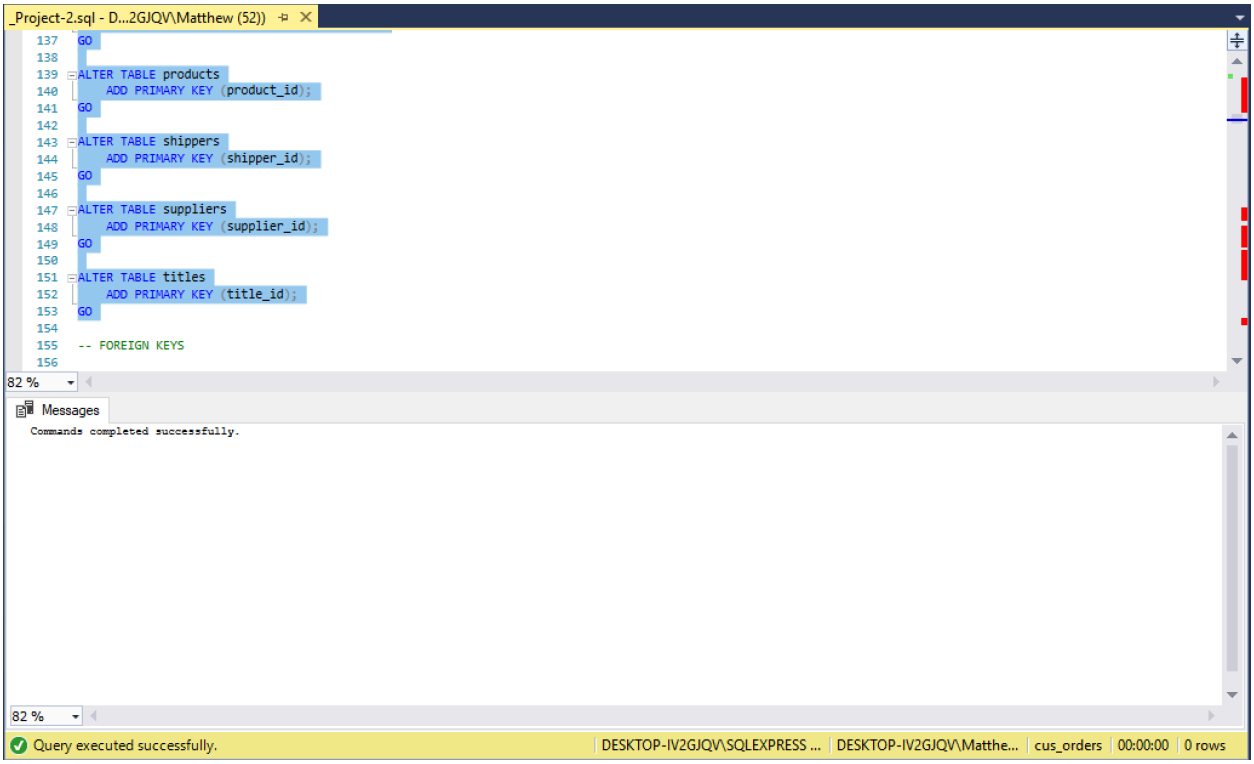
Set the **primary keys** and **foreign keys** for the tables

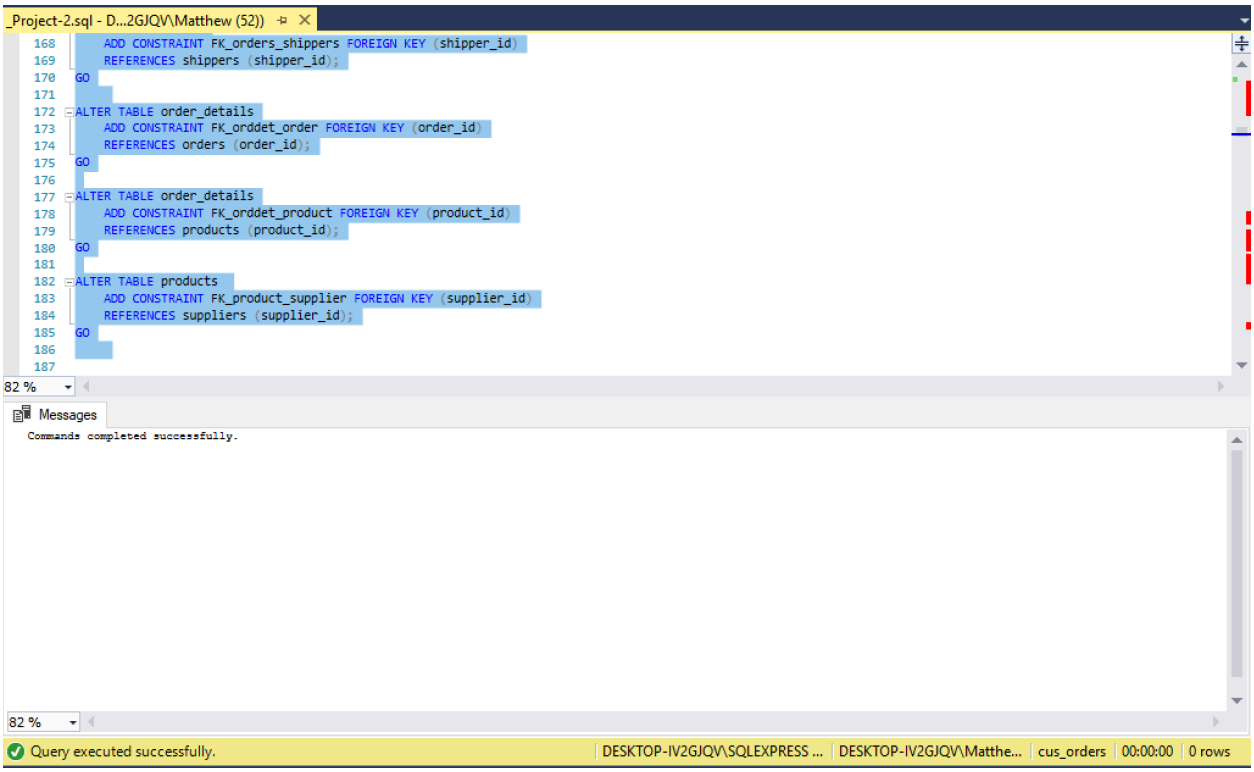
Code:

```
-- PRIMARY KEYS

ALTER TABLE customers
  ADD PRIMARY KEY (customer_id);
GO

ALTER TABLE orders
  ADD PRIMARY KEY (order_id);
GO

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

ALTER TABLE products
  ADD PRIMARY KEY (product_id);
GO

ALTER TABLE shippers
  ADD PRIMARY KEY (shipper_id);
GO

ALTER TABLE suppliers
  ADD PRIMARY KEY (supplier_id);
GO

ALTER TABLE titles
  ADD PRIMARY KEY (title_id);
GO

-- FOREIGN KEYS

ALTER TABLE customers
  ADD CONSTRAINT FK_customer_title FOREIGN KEY (title_id)
  REFERENCES titles (title_id);
GO

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

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

```
ALTER TABLE order_details
  ADD CONSTRAINT FK_orddet_order FOREIGN KEY (order_id)
  REFERENCES orders (order_id);
GO

ALTER TABLE order_details
  ADD CONSTRAINT FK_orddet_product FOREIGN KEY (product_id)
  REFERENCES products (product_id);
GO

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

Completed in two steps, both successful:





Problem A-5

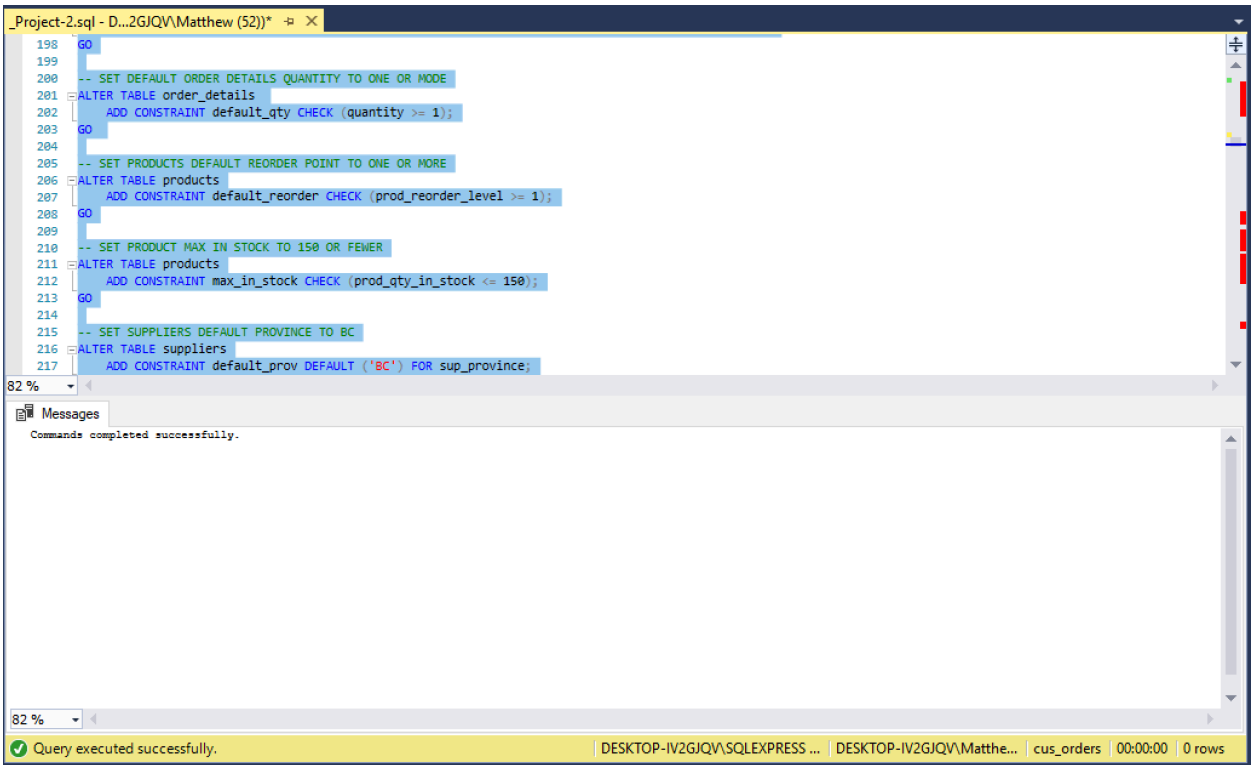
Set the **constraints** as follows:

customers table	- country should default to Canada
orders table	- required_date should default to today's date plus ten days
order details table	- quantity must be greater than or equal to 1
products table	- reorder_level must be greater than or equal to 1
	- quantity_in_stock value must not be greater than 150
suppliers table	- province should default to BC

Code:

```
-- SET CUSTOMERS DEFAULT COUNTRY AS "CANADA"
ALTER TABLE customers
  ADD CONSTRAINT default_country DEFAULT ('Canada') FOR customer_country;
GO

-- SET ORDERS DEFAULT REQUIRED DATE AS TODAY + 10 DAYS.
ALTER TABLE orders
  ADD CONSTRAINT default_req_date DEFAULT (DATEADD(DAY, 10, GETDATE())) FOR
order_required_date;
GO

-- SET DEFAULT ORDER DETAILS QUANTITY TO ONE OR MODE
ALTER TABLE order_details
  ADD CONSTRAINT default_qty CHECK (quantity >= 1);
GO

-- SET PRODUCTS DEFAULT REORDER POINT TO ONE OR MORE
ALTER TABLE products
  ADD CONSTRAINT default_reorder CHECK (prod_reorder_level >= 1);
GO

-- SET PRODUCT MAX IN STOCK TO 150 OR FEWER
ALTER TABLE products
  ADD CONSTRAINT max_in_stock CHECK (prod_qty_in_stock <= 150);
GO

-- SET SUPPLIERS DEFAULT PROVINCE TO BC
ALTER TABLE suppliers
  ADD CONSTRAINT default_prov DEFAULT ('BC') FOR sup_province;
GO
```

Competed Successfully:



Problem A-6:

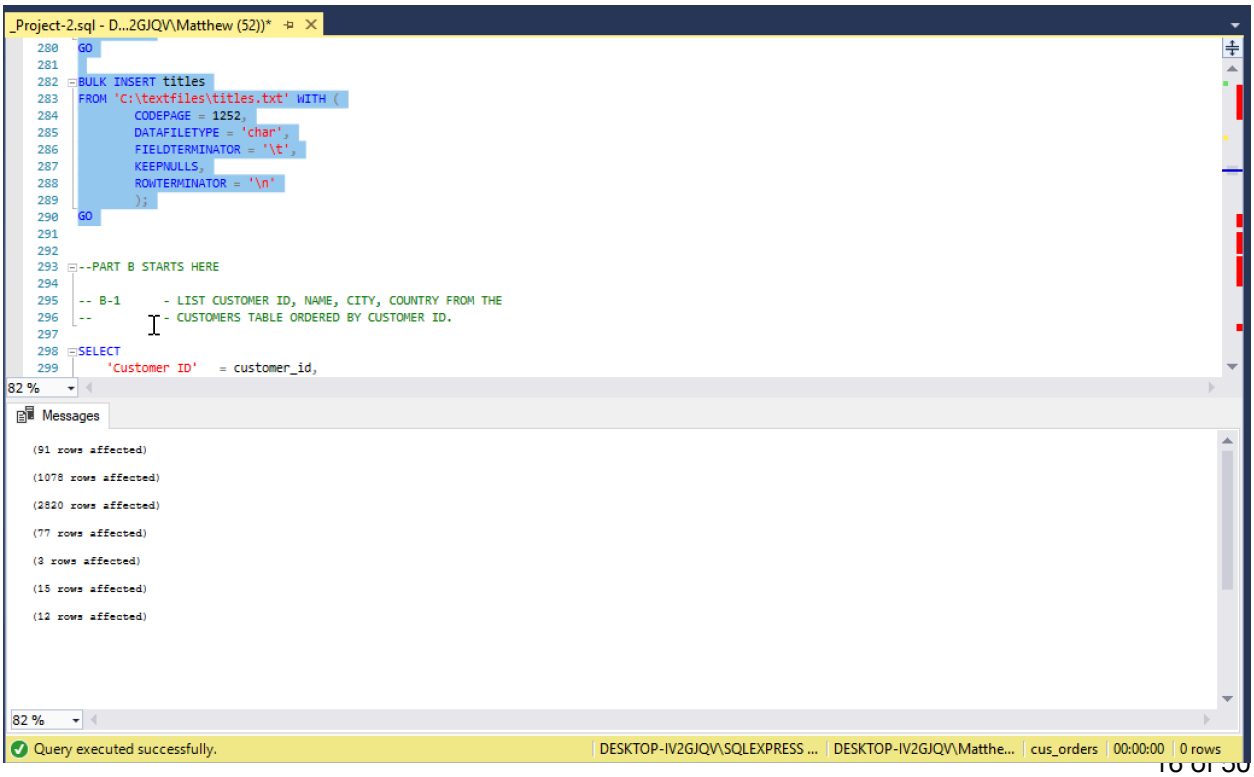
Bulk load the data into the created tables.

Code:

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

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

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

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

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

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

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

Completed Successfully:



Part B

Concerning various SQL Statements.

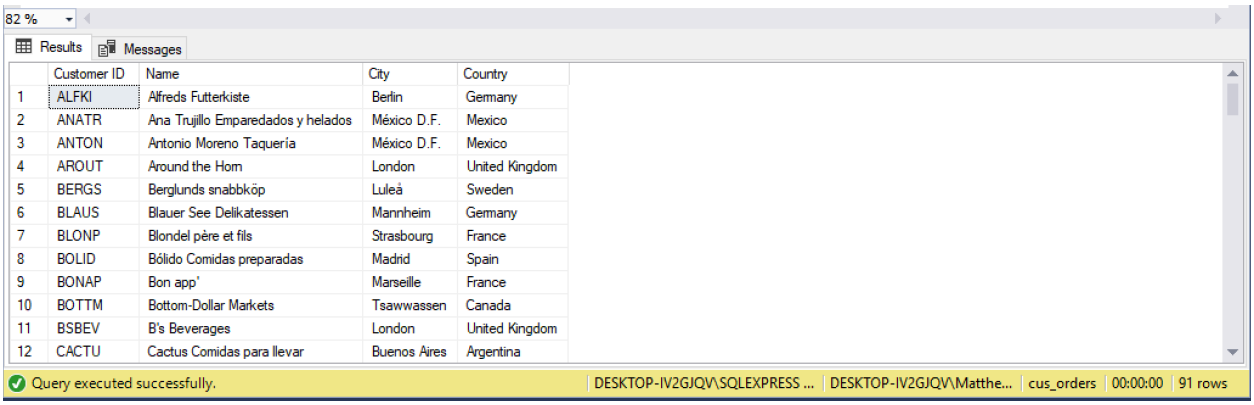
Problem B-1

List the customer id, name, city, and country from the customer table. Order the result set by the **customer id**.

Code:

```
SELECT
  'Customer ID'          = customer_id,
  'Name'                 = customer_name,
  'City'                 = customer_city,
  'Country'              = customer_country
FROM
  customers
ORDER BY
  customer_id;
GO
```

Completed Successfully (91 rows):



82 %

Results | Messages

	Customer ID	Name	City	Country
1	ALFKI	Alfreds Futterkiste	Berlin	Germany
2	ANATR	Ana Trujillo Emparedados y helados	México D.F.	Mexico
3	ANTON	Antonio Moreno Taquería	México D.F.	Mexico
4	AROUT	Around the Horn	London	United Kingdom
5	BERGS	Berglunds snabbköp	Luleå	Sweden
6	BLAUS	Blauer See Delikatessen	Mannheim	Germany
7	BLONP	Blondel père et fils	Strasbourg	France
8	BOLID	Bólido Comidas preparadas	Madrid	Spain
9	BONAP	Bon app'	Marseille	France
10	BOTTM	Bottom-Dollar Markets	Tsawwassen	Canada
11	BSBEV	B's Beverages	London	United Kingdom
12	CACTU	Cactus Comidas para llevar	Buenos Aires	Argentina

Query executed successfully. | DESKTOP-IV2GJQV\SQLEXPRESS ... | DESKTOP-IV2GJQV\Matthe... | cus_orders | 00:00:00 | 91 rows

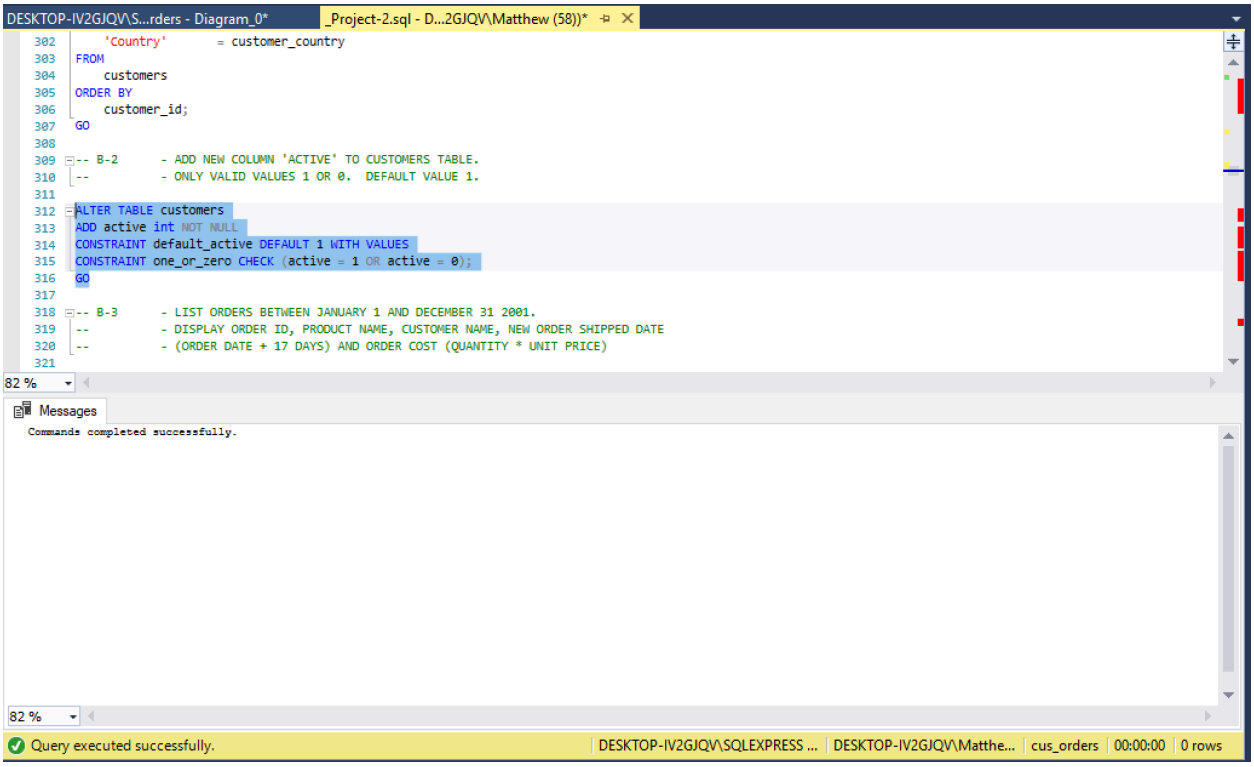
Problem B-2

Add a new column called **active** to the customers table using the ALTER statement. The only valid values are 1 or 0. The default should be **1**.

Code:

```
ALTER TABLE customers
  ADD active int NOT NULL
  CONSTRAINT default_active DEFAULT 1 WITH VALUES
  CONSTRAINT one_or_zero CHECK (active = 1 OR active = 0);
GO
```

Completed Successfully:



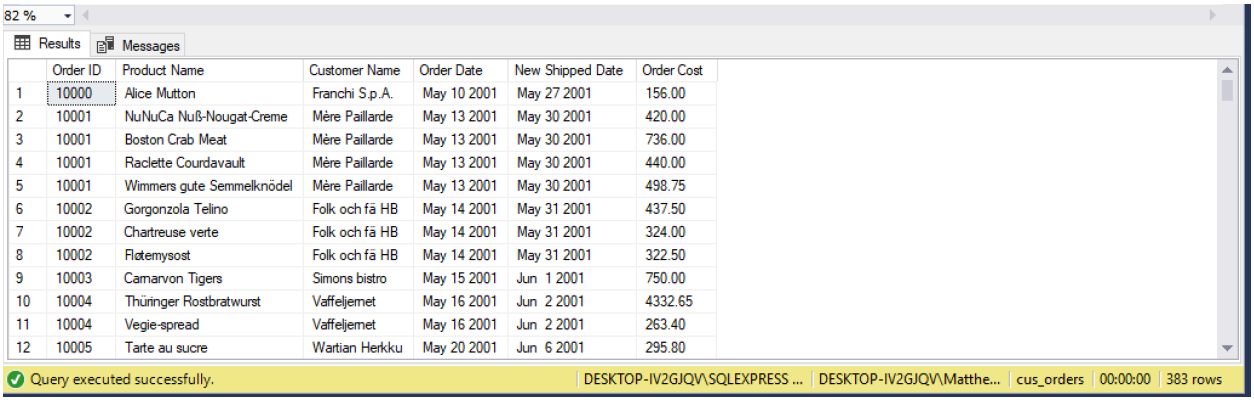
Problem B-3

List all the orders where the order date is between **January 1** and **December 31, 2001**. Display the order id, order date, and a new shipped date calculated by adding 17 days to the shipped date from the orders table, the product name from the product table, the customer 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 (quantity * unit_price) to calculate the cost of the order.

Code:

```
SELECT
  'Order ID'            = orders.order_id,
  'Product Name'        = products.prod_name,
  'Customer Name'       = customers.customer_name,
  'Order Date'          = CONVERT(CHAR(12), orders.order_date, 109),
  'New Shipped Date'    = CONVERT(CHAR(12), DATEADD(DAY, 17, orders.order_date)),
  'Order Cost'          = order_details.quantity * products.prod_unit_price
FROM
  order_details
INNER JOIN orders 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
WHERE orders.order_date BETWEEN 'January 1 2001' AND 'December 31 2001';
```

Completed Successfully (383 rows):



82 %

Results | Messages

	Order ID	Product Name	Customer Name	Order Date	New Shipped Date	Order Cost
1	10000	Alice Mutton	Franchi S.p.A.	May 10 2001	May 27 2001	156.00
2	10001	NuNuCa Nuß-Nougat-Creme	Mère Paillarde	May 13 2001	May 30 2001	420.00
3	10001	Boston Crab Meat	Mère Paillarde	May 13 2001	May 30 2001	736.00
4	10001	Raclette Courdavault	Mère Paillarde	May 13 2001	May 30 2001	440.00
5	10001	Wimmers gute Semmelknödel	Mère Paillarde	May 13 2001	May 30 2001	498.75
6	10002	Gorgonzola Telino	Folk och fä HB	May 14 2001	May 31 2001	437.50
7	10002	Chartreuse verte	Folk och fä HB	May 14 2001	May 31 2001	324.00
8	10002	Fløtemysost	Folk och fä HB	May 14 2001	May 31 2001	322.50
9	10003	Camarvon Tigers	Simons bistro	May 15 2001	Jun 1 2001	750.00
10	10004	Thüringer Rostbratwurst	Vaffeljernet	May 16 2001	Jun 2 2001	4332.65
11	10004	Vegie-spread	Vaffeljernet	May 16 2001	Jun 2 2001	263.40
12	10005	Tarte au sucre	Wartian Herkku	May 20 2001	Jun 6 2001	295.80

Query executed successfully. | DESKTOP-IV2GJQV\SQLEXPRESS ... | DESKTOP-IV2GJQV\Matthe... | cus_orders | 00:00:00 | 383 rows

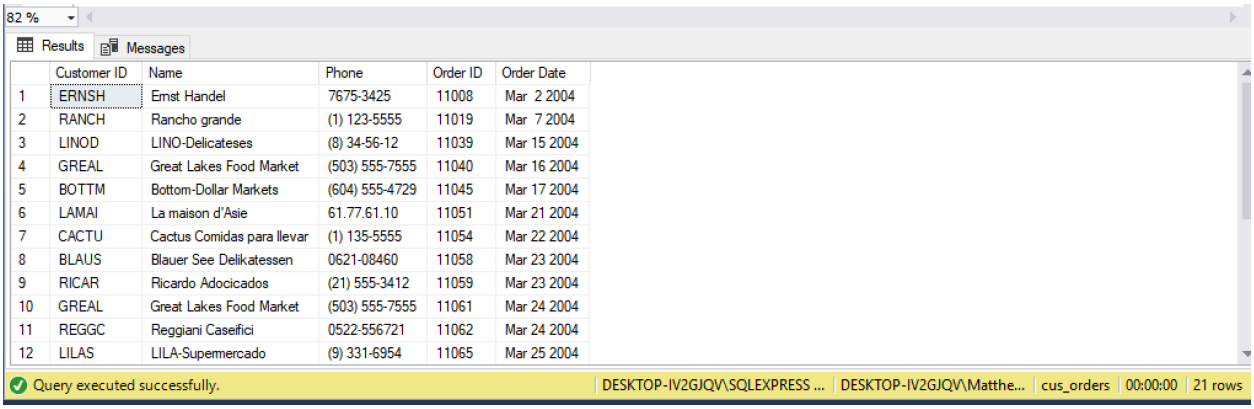
Problem B-4

List all the orders that have **not** been shipped. Display the customer id, name and phone number from the customers table, and the order id and order date from the orders table. Order the result set by the customer name.

Code:

```
SELECT
  'Customer ID'          = customers.customer_id,
  'Name'                 = customers.customer_name,
  'Phone'                = customers.customer_phone,
  'Order ID'             = orders.order_id,
  'Order Date'           = CONVERT(CHAR(12), orders.order_date, 109)
FROM
  orders
INNER JOIN customers ON orders.customer_id=customers.customer_id
WHERE orders.order_shipped_date IS NULL;
```

Completed Successfully (21 rows):



82 %

Results | Messages

	Customer ID	Name	Phone	Order ID	Order Date
1	ERNSH	Ernst Handel	7675-3425	11008	Mar 2 2004
2	RANCH	Rancho grande	(1) 123-5555	11019	Mar 7 2004
3	LINOD	LINO-Delicateses	(8) 34-56-12	11039	Mar 15 2004
4	GREAL	Great Lakes Food Market	(503) 555-7555	11040	Mar 16 2004
5	BOTTM	Bottom-Dollar Markets	(604) 555-4729	11045	Mar 17 2004
6	LAMAI	La maison d'Asie	61.77.61.10	11051	Mar 21 2004
7	CACTU	Cactus Comidas para llevar	(1) 135-5555	11054	Mar 22 2004
8	BLAUS	Blauer See Delikatessen	0621-08460	11058	Mar 23 2004
9	RICAR	Ricardo Adocicados	(21) 555-3412	11059	Mar 23 2004
10	GREAL	Great Lakes Food Market	(503) 555-7555	11061	Mar 24 2004
11	REGGC	Reggiani Caseifici	0522-556721	11062	Mar 24 2004
12	LILAS	LILA-Supermercado	(9) 331-6954	11065	Mar 25 2004

Query executed successfully. | DESKTOP-IV2GJQV\SQLEXPRESS ... | DESKTOP-IV2GJQV\Matthe... | cus_orders | 00:00:00 | 21 rows

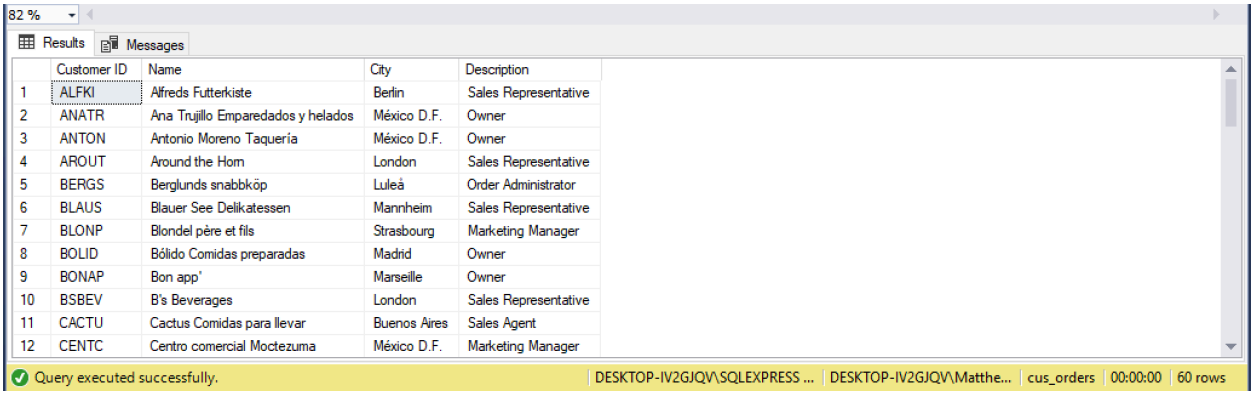
Problem B-5

List all the customers where the region is **NULL**. Display the customer id, name, and city from the customers table, and the title description from the titles table. The query should produce the result set listed below.

Code:

```
SELECT
  'Customer ID'          = customer_id,
  'Name'                 = customer_name,
  'City'                 = customer_city,
  'Description'          = title_description
FROM
  customers
INNER JOIN titles ON customers.title_id=titles.title_id
WHERE customer_region IS NULL;
GO
```

Completed Successfully (60 rows):



82 %

Results | Messages

	Customer ID	Name	City	Description
1	ALFKI	Alfreds Futterkiste	Berlin	Sales Representative
2	ANATR	Ana Trujillo Emparedados y helados	México D.F.	Owner
3	ANTON	Antonio Moreno Taquería	México D.F.	Owner
4	AROUT	Around the Horn	London	Sales Representative
5	BERGS	Berglunds snabbköp	Luleå	Order Administrator
6	BLAUS	Blauer See Delikatessen	Mannheim	Sales Representative
7	BLONP	Blondel père et fils	Strasbourg	Marketing Manager
8	BOLID	Bólido Comidas preparadas	Madrid	Owner
9	BONAP	Bon app'	Marseille	Owner
10	BSBEV	B's Beverages	London	Sales Representative
11	CACTU	Cactus Comidas para llevar	Buenos Aires	Sales Agent
12	CENTC	Centro comercial Moctezuma	México D.F.	Marketing Manager

Query executed successfully. | DESKTOP-IV2GJQV\SQLEXPRESS ... | DESKTOP-IV2GJQV\Matthe... | cus_orders | 00:00:00 | 60 rows

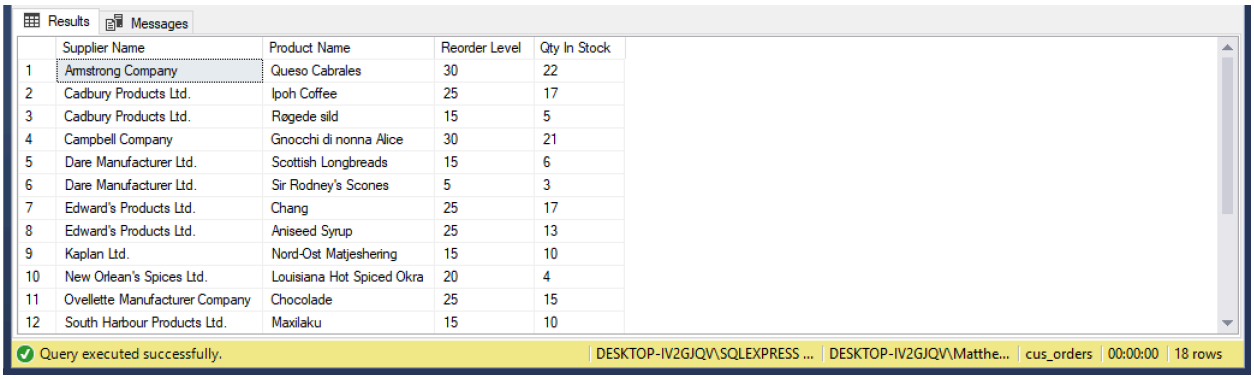
Problem B-6

List the products where the reorder level is **higher than** the quantity in stock. Display the supplier name from the suppliers table, the product name, reorder level, and quantity in stock from the products table. Order the result set by the supplier name.

Code:

```
SELECT
  'Supplier Name'        = sup_name,
  'Product Name'         = prod_name,
  'Reorder Level'        = prod_reorder_level,
  'Qty In Stock'         = prod_qty_in_stock
FROM
  products
INNER JOIN suppliers ON products.supplier_id=suppliers.supplier_id
WHERE prod_reorder_level > prod_qty_in_stock
ORDER BY [Supplier Name];
GO
```

Completed Successfully (18 rows):



Results | Messages

	Supplier Name	Product Name	Reorder Level	Qty In Stock
1	Armstrong Company	Queso Cabrales	30	22
2	Cadbury Products Ltd.	Ipoh Coffee	25	17
3	Cadbury Products Ltd.	Røgede sild	15	5
4	Campbell Company	Gnocchi di nonna Alice	30	21
5	Dare Manufacturer Ltd.	Scottish Longbreads	15	6
6	Dare Manufacturer Ltd.	Sir Rodney's Scones	5	3
7	Edward's Products Ltd.	Chang	25	17
8	Edward's Products Ltd.	Aniseed Syrup	25	13
9	Kaplan Ltd.	Nord-Ost Matjeshering	15	10
10	New Orlean's Spices Ltd.	Louisiana Hot Spiced Okra	20	4
11	Ovellette Manufacturer Company	Chocolade	25	15
12	South Harbour Products Ltd.	Maxilaku	15	10

Query executed successfully. | DESKTOP-IV2GJQV\SQLEXPRESS ... | DESKTOP-IV2GJQV\Matthe... | cus_orders | 00:00:00 | 18 rows

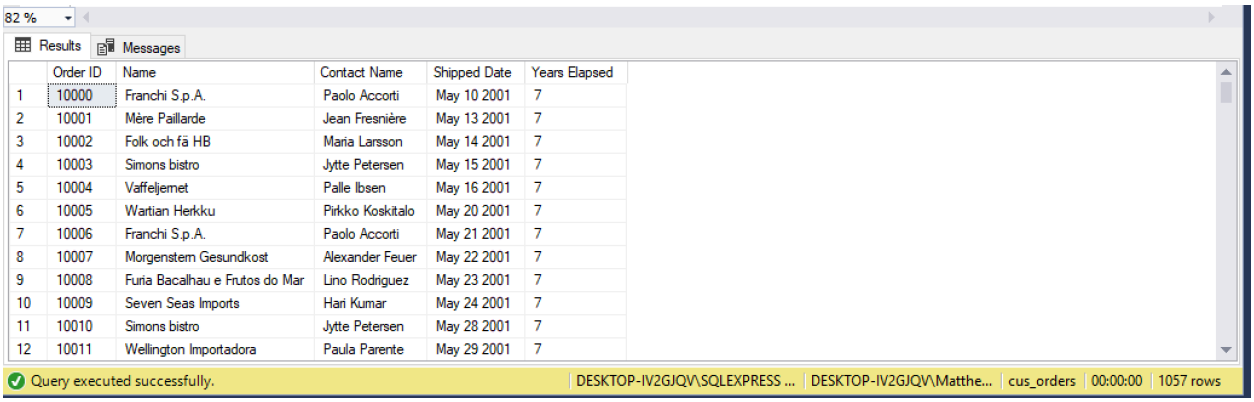
Problem B-7

Calculate the length in years from **January 1, 2008** and when an order was shipped where the shipped date is **not null**. Display the order id, and the shipped date from the orders table, the customer name, and the contact name from the customers table, and the 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.

Code:

```
SELECT
  'Order ID'            = orders.order_id,
  'Name'                = customers.customer_name,
  'Contact Name'        = customers.customer_contact_name,
  'Shipped Date'        = CONVERT(CHAR(12), orders.order_date, 109),
  'Years Elapsed'       = DATEDIFF(year, orders.order_date, 'January 1 2008')
FROM
  orders
INNER JOIN customers ON orders.customer_id=customers.customer_id
WHERE orders.order_shipped_date IS NOT NULL;
GO
```

Completed Successfully (1057 rows):



82 %

Results | Messages

	Order ID	Name	Contact Name	Shipped Date	Years Elapsed
1	10000	Franchi S.p.A.	Paolo Accorti	May 10 2001	7
2	10001	Mère Paillarde	Jean Fresnière	May 13 2001	7
3	10002	Folk och fä HB	Maria Larsson	May 14 2001	7
4	10003	Simons bistro	Jytte Petersen	May 15 2001	7
5	10004	Vaffeljernet	Palle Ibsen	May 16 2001	7
6	10005	Wartian Herkku	Pirkko Koskitalo	May 20 2001	7
7	10006	Franchi S.p.A.	Paolo Accorti	May 21 2001	7
8	10007	Morgenstern Gesundkost	Alexander Feuer	May 22 2001	7
9	10008	Furia Bacalhau e Frutos do Mar	Lino Rodriguez	May 23 2001	7
10	10009	Seven Seas Imports	Hari Kumar	May 24 2001	7
11	10010	Simons bistro	Jytte Petersen	May 28 2001	7
12	10011	Wellington Importadora	Paula Parente	May 29 2001	7

Query executed successfully. | DESKTOP-IV2GJQV\SQLEXPRESS ... | DESKTOP-IV2GJQV\Matthe... | cus_orders | 00:00:00 | 1057 rows

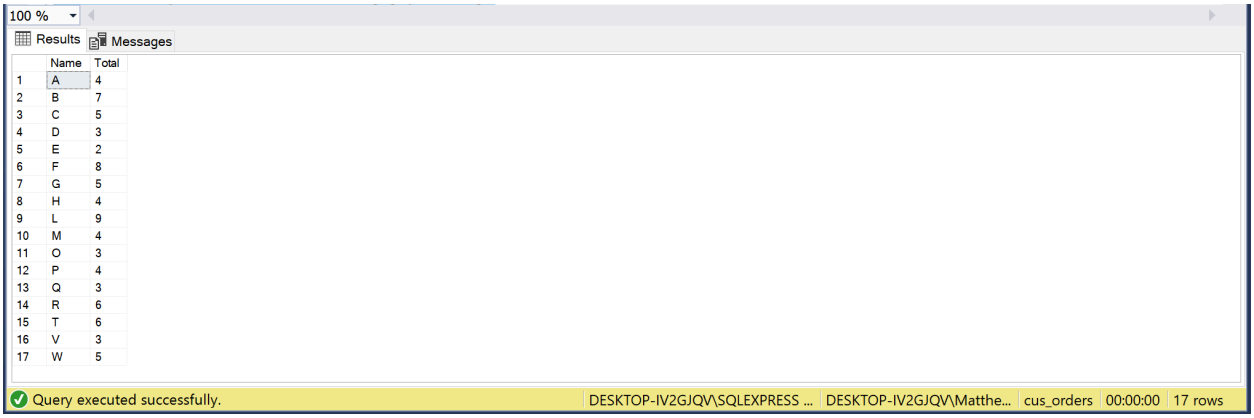
Problem B-8

List number of customers with names beginning with each letter of the alphabet. Ignore customers whose name begins with the letter **S**. Do not display the letter and count unless **at least two** customer's names begin with the letter.

Code:

```
SELECT
  'Name'          = SUBSTRING(customers.customer_name, 1, 1),
  'Total'         = count(*)
FROM
  customers
GROUP BY SUBSTRING(customers.customer_name, 1, 1)
HAVING count(*) >= 2
AND SUBSTRING(customers.customer_name,1,1) != 's';
```

Completed Successfully (16 rows):



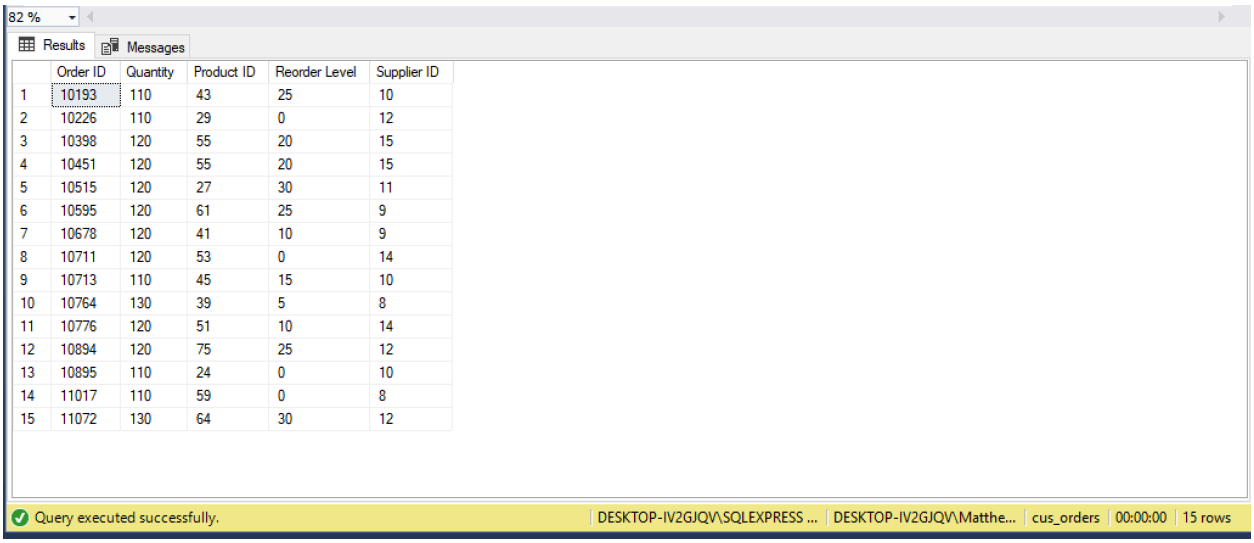
Problem B-9

List the order details where the quantity is **greater than 100**. Display the order id and quantity from the order_details table, the product id, the supplier_id and reorder level from the products table. Order the result set by the order id.

Code:

```
SELECT
  'Order ID'            = order_details.order_id,
  'Quantity'            = order_details.quantity,
  'Product ID'          = products.product_id,
  'Reorder Level'       = products.prod_reorder_level,
  'Supplier ID'         = products.supplier_id
FROM
  order_details
INNER JOIN products ON order_details.product_id=products.product_id
WHERE order_details.quantity > 100
ORDER BY order_details.order_id;
GO
```

Completed Successfully (15 rows):



82 %

Results | Messages

	Order ID	Quantity	Product ID	Reorder Level	Supplier ID
1	10193	110	43	25	10
2	10226	110	29	0	12
3	10398	120	55	20	15
4	10451	120	55	20	15
5	10515	120	27	30	11
6	10595	120	61	25	9
7	10678	120	41	10	9
8	10711	120	53	0	14
9	10713	110	45	15	10
10	10764	130	39	5	8
11	10776	120	51	10	14
12	10894	120	75	25	12
13	10895	110	24	0	10
14	11017	110	59	0	8
15	11072	130	64	30	12

Query executed successfully. | DESKTOP-IV2GJQV\SQLEXPRESS ... | DESKTOP-IV2GJQV\Matthe... | cus_orders | 00:00:00 | 15 rows

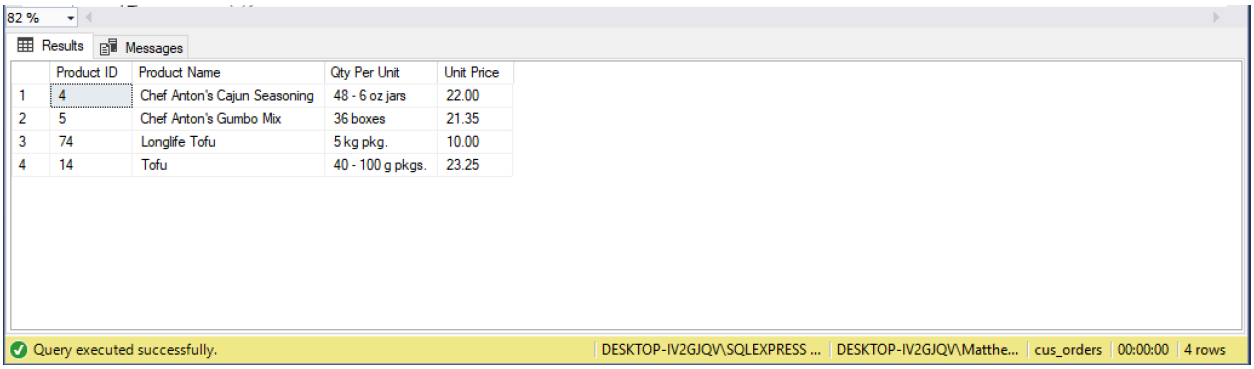
Problem B-10

List the products which contain **tofu** or **chef** 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 name.

Code:

```
SELECT
  'Product ID'    = products.product_id,
  'Product Name'  = products.prod_name,
  'Qty Per Unit'  = products.prod_qty_per_unit,
  'Unit Price'    = products.prod_unit_price
FROM
  products
WHERE
  products.prod_name LIKE '%chef%'
  OR products.prod_name LIKE '%tofu%'
ORDER BY
  products.prod_name;
GO
```

Completed Successfully (4 rows):



Part C

Insert, Update, Delete, and Views.

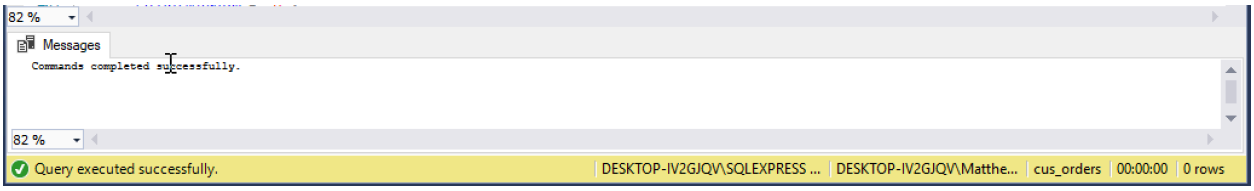
Problem C-1, C-2

Create an **employee** table and create the primary key.

Code:

```
CREATE TABLE employee (
  employee_id id_num PRIMARY KEY,
  emp_last_name varchar(30) NOT NULL,
  emp_first_name varchar(15) NOT NULL,
  emp_address varchar(30),
  emp_city varchar(20),
  emp_province char(2),
  emp_postal_code varchar(7),
  emp_phone varchar(10),
  emp_birth_date datetime NOT NULL
);
GO

ALTER TABLE employee
  ADD PRIMARY KEY (employee_id);
GO
```

Completed Successfully:



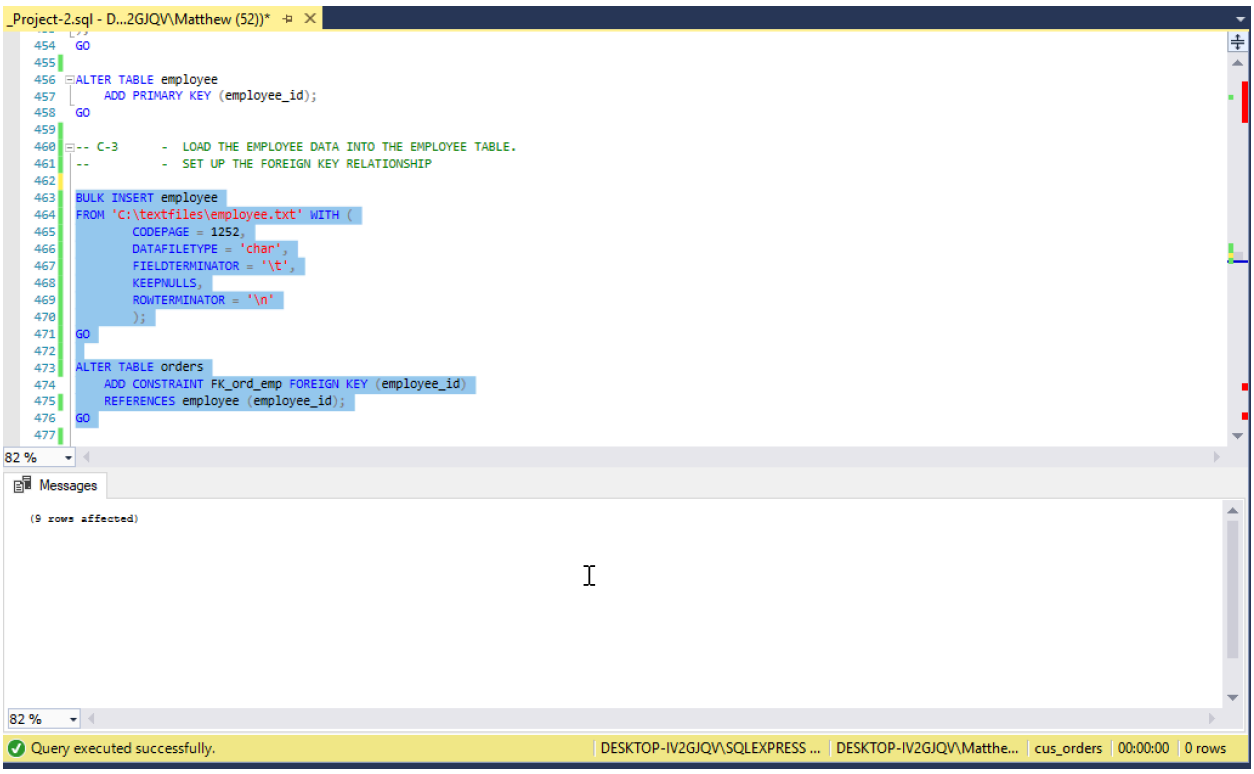
Problem C-3

Load the data into the employee table using the employee.txt file; **9** rows. In addition, **create the relationship** to enforce referential integrity between the employee and orders tables.

Code:

```
BULK INSERT employee
FROM 'C:\textfiles\employee.txt' WITH (
          CODEPAGE = 1252,
          DATAFILETYPE = 'char',
          FIELDTERMINATOR = '\t',
          KEEPNULLS,
          ROWTERMINATOR = '\n'
          );
GO

ALTER TABLE orders
  ADD CONSTRAINT FK_ord_emp FOREIGN KEY (employee_id)
  REFERENCES employee (employee_id);
GO
```

Completed Successfully:



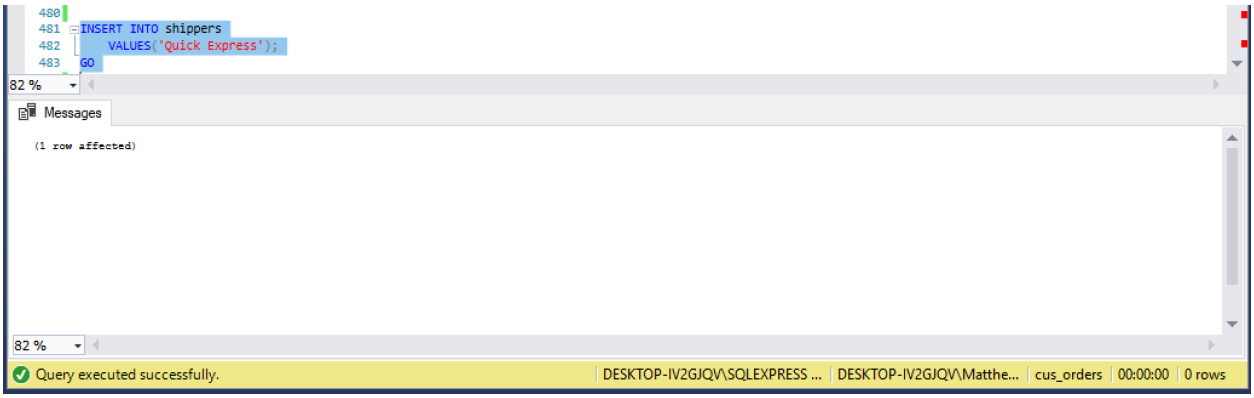
Problem C-4

Using the INSERT statement, add the shipper **Quick Express** to the shippers table.

Code:

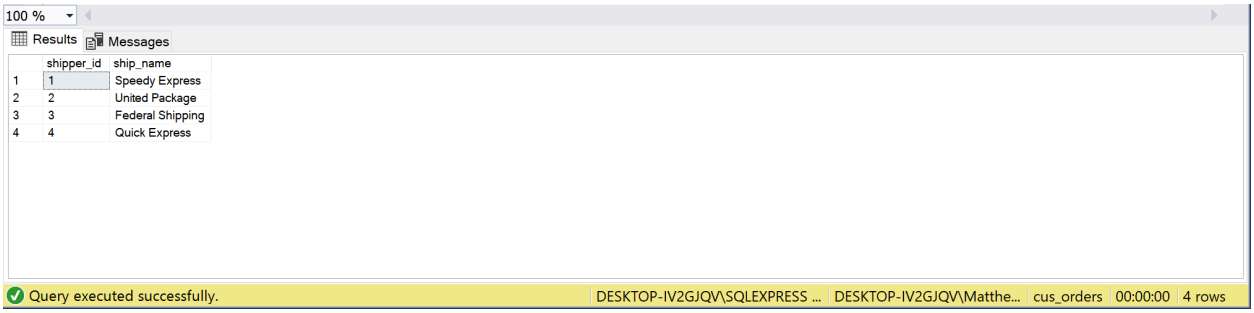
```
INSERT INTO shippers
 VALUES('Quick Express');
GO
```

Completed Successfully:



The Shippers table has an IDENTITY flag on the primary key so shipper_id should be auto-generated. To prove the insert worked:

```
SELECT * FROM SHIPPERS;
```



Problem C-5

Using the UPDATE statement, increase the unit price in the products table of all rows with a current unit price between **$5.00** and **$10.00** by **5%.**

Code:

```
UPDATE
  products
SET products.prod_unit_price = ROUND(products.prod_unit_price * 1.05, 2)
WHERE products.prod_unit_price >= 5 AND products.prod_unit_price <= 10;
GO
```

Completed Successfully (12 rows):

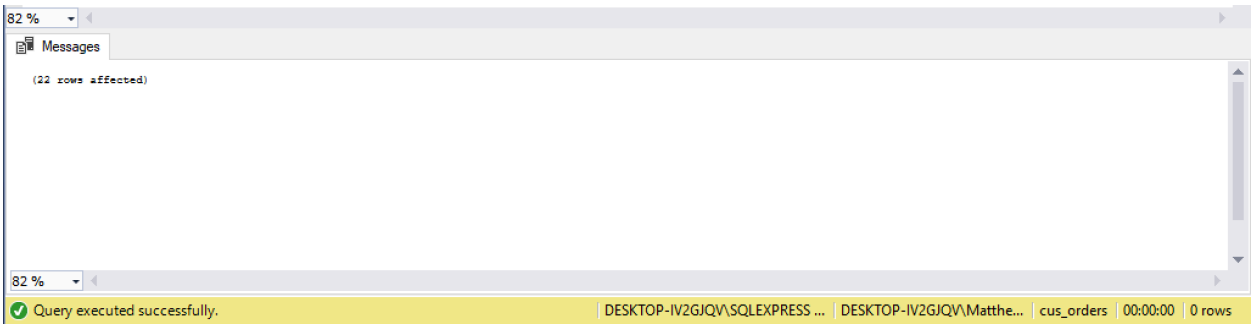
Problem C-6

Using the UPDATE statement, change the fax value to **Unknown** for all rows in the customers table where the current fax value is **NULL**;

Code:

```
UPDATE
  customers
SET customers.customer_fax = 'UNKNOWN'
WHERE customers.customer_fax IS NULL;
GO
```

Completed Successfully (22 rows);



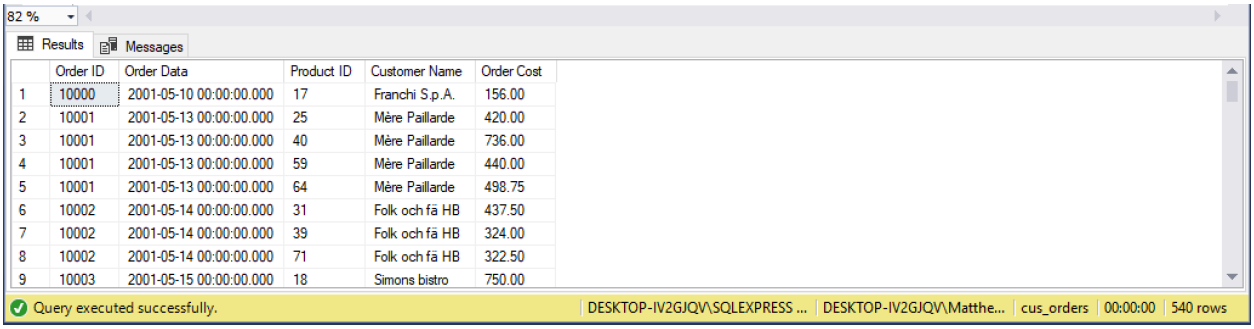
Problem C-7

Create a view called **vw_order_cost** to list the cost of the orders. Display the order id and order_date from the orders table, the product id from the products table, the customer name from the customers table, and the order cost. To calculate the cost of the orders, use the formula (order_details.quantity * products.unit_price). Run the view for the order ids between **10000** and **10200**.

Code:

```
CREATE VIEW vw_order_cost
AS
SELECT
  'Order ID'            = orders.order_id,
  'Order Data'          = orders.order_date,
  'Product ID'          = products.product_id,
  'Customer Name'       = customers.customer_name,
  'Order Cost'          = (order_details.quantity * products.prod_unit_price)
FROM order_details
INNER JOIN orders ON order_details.order_id=orders.order_id
INNER JOIN products ON order_details.product_id=products.product_id
INNER JOIN customers ON customers.customer_id=orders.customer_id;
GO

SELECT *
FROM vw_order_cost
WHERE vw_order_cost.[Order ID] BETWEEN 10000 AND 10200;
GO
```

Successfully Completed(540 rows):



82 %

Results | Messages

	Order ID	Order Data	Product ID	Customer Name	Order Cost
1	10000	2001-05-10 00:00:00.000	17	Franchi S.p.A.	156.00
2	10001	2001-05-13 00:00:00.000	25	Mère Paillarde	420.00
3	10001	2001-05-13 00:00:00.000	40	Mère Paillarde	736.00
4	10001	2001-05-13 00:00:00.000	59	Mère Paillarde	440.00
5	10001	2001-05-13 00:00:00.000	64	Mère Paillarde	498.75
6	10002	2001-05-14 00:00:00.000	31	Folk och fä HB	437.50
7	10002	2001-05-14 00:00:00.000	39	Folk och fä HB	324.00
8	10002	2001-05-14 00:00:00.000	71	Folk och fä HB	322.50
9	10003	2001-05-15 00:00:00.000	18	Simons bistro	750.00

Query executed successfully. | DESKTOP-IV2GJQV\SQLEXPRESS ... | DESKTOP-IV2GJQV\Matthe... | cus_orders | 00:00:00 | 540 rows

Problem C-8

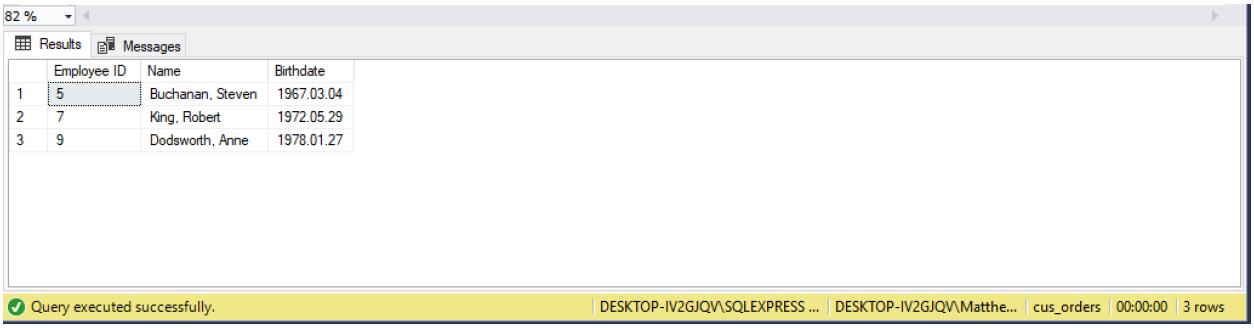
Create a view called **vw_list_employees** to list all the employees and all the columns in the employee table. Run the view for employee ids **5**, **7**, and **9**. Display the employee id, last name, first name, and birth date. Format the name as last name followed by a comma and a space followed by the first name. Format the birth date as **YYYY.MM.DD**.

See Challenges Section

Code:

```
CREATE VIEW vw_list_employees
AS
SELECT
  'Employee ID'  = employee.employee_id,
  'Name'                 = (employee.emp_last_name + ', ' + employee.emp_first_name),
  'Birthdate'            = CONVERT(char(12), employee.emp_birth_date, 102)
FROM employee;
GO

SELECT *
FROM vw_list_employees
WHERE [Employee ID] = 5
  OR [Employee ID] = 7
  OR [Employee ID] = 9;
GO
```

Completed Successfully (3 rows):



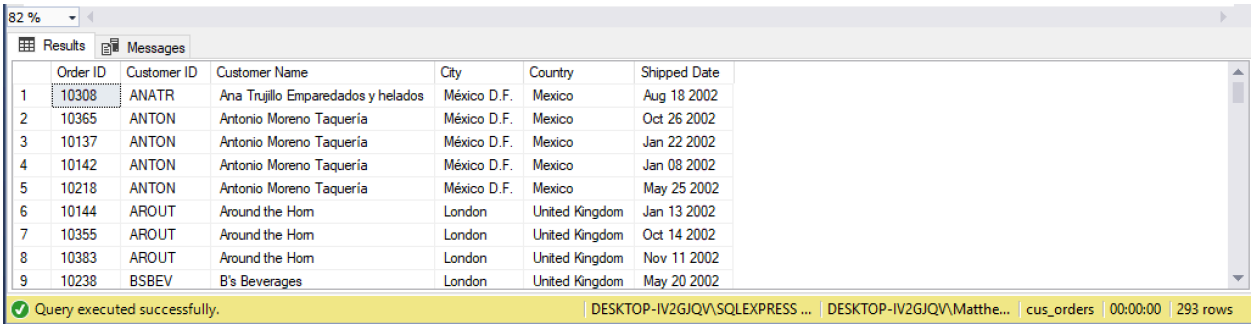
Problem C-9

Create a view called **vw_all_orders** to list all the orders. Display the order id and shipped date from the orders table, and the customer id, name, city, and country from the customers table. Run the view for orders shipped from **January 1, 2002** and **December 31, 2002**, formatting the shipped date as **MON DD YYYY**. Order the result set by customer name and country.

Code:

```
CREATE VIEW vw_all_orders
AS
SELECT
  'Order ID'            = orders.order_id,
  'Customer ID'         = customers.customer_id,
  'Customer Name'       = customers.customer_name,
  'City'                = customers.customer_city,
  'Country'             = customers.customer_country,
  'Shipped Date'        = orders.order_shipped_date
FROM orders
INNER JOIN customers ON orders.customer_id=customers.customer_id;
GO

SELECT
  vw_all_orders.[Order ID],
  vw_all_orders.[Customer ID],
  vw_all_orders.[Customer Name],
  vw_all_orders.[City],
  vw_all_orders.[Country],
  format(vw_all_orders.[Shipped Date], 'MMM dd yyyy') AS 'Shipped Date'
FROM vw_all_orders
WHERE [Shipped Date] BETWEEN 'Jan 1 2002' AND 'Dec 31 2002'
ORDER BY [Customer Name], [Country];
GO
```

Completed Successfully (293 rows):



82 %

Results | Messages

	Order ID	Customer ID	Customer Name	City	Country	Shipped Date
1	10308	ANATR	Ana Trujillo Emparedados y helados	México D.F.	Mexico	Aug 18 2002
2	10365	ANTON	Antonio Moreno Taquería	México D.F.	Mexico	Oct 26 2002
3	10137	ANTON	Antonio Moreno Taquería	México D.F.	Mexico	Jan 22 2002
4	10142	ANTON	Antonio Moreno Taquería	México D.F.	Mexico	Jan 08 2002
5	10218	ANTON	Antonio Moreno Taquería	México D.F.	Mexico	May 25 2002
6	10144	AROUT	Around the Horn	London	United Kingdom	Jan 13 2002
7	10355	AROUT	Around the Horn	London	United Kingdom	Oct 14 2002
8	10383	AROUT	Around the Horn	London	United Kingdom	Nov 11 2002
9	10238	BSBEV	B's Beverages	London	United Kingdom	May 20 2002

Query executed successfully. | DESKTOP-IV2GJQV\SQLEXPRESS ... | DESKTOP-IV2GJQV\Matthe... | cus_orders | 00:00:00 | 293 rows

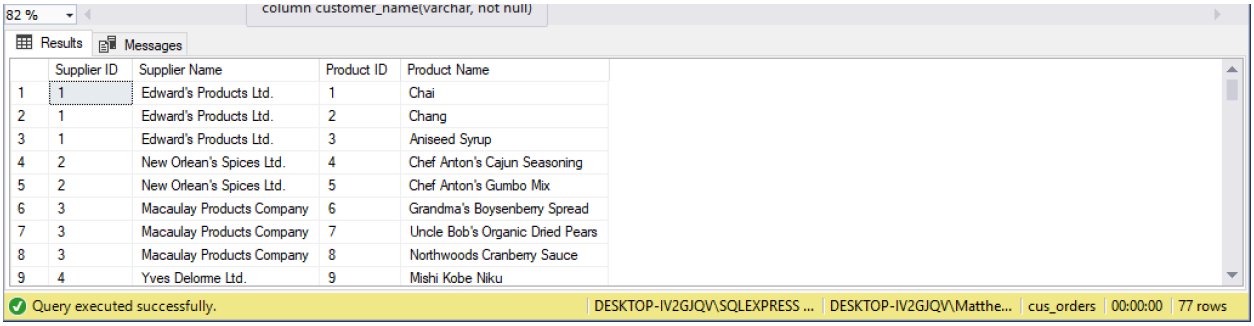
Problem C-10

Create a view listing the suppliers and the items they have shipped. Display the supplier id and name from the suppliers table, and the product id and name from the products table. Run the view.

Code:

```
CREATE VIEW vw_suppliers_shipped
AS
SELECT
  'Supplier ID'          = suppliers.supplier_id,
  'Supplier Name'        = suppliers.sup_name,
  'Product ID'           = products.product_id,
  'Product Name'         = products.prod_name
FROM products
INNER JOIN suppliers ON products.supplier_id=suppliers.supplier_id;
GO

SELECT *
FROM vw_suppliers_shipped;
GO
```

Completed Successfully (77 rows):



	Supplier ID	Supplier Name	Product ID	Product Name
1	1	Edward's Products Ltd.	1	Chai
2	1	Edward's Products Ltd.	2	Chang
3	1	Edward's Products Ltd.	3	Aniseed Syrup
4	2	New Orlean's Spices Ltd.	4	Chef Anton's Cajun Seasoning
5	2	New Orlean's Spices Ltd.	5	Chef Anton's Gumbo Mix
6	3	Macaulay Products Company	6	Grandma's Boysenberry Spread
7	3	Macaulay Products Company	7	Uncle Bob's Organic Dried Pears
8	3	Macaulay Products Company	8	Northwoods Cranberry Sauce
9	4	Yves Delorme Ltd.	9	Mishi Kobe Niku

Part D

Concerning stored procedures and triggers.

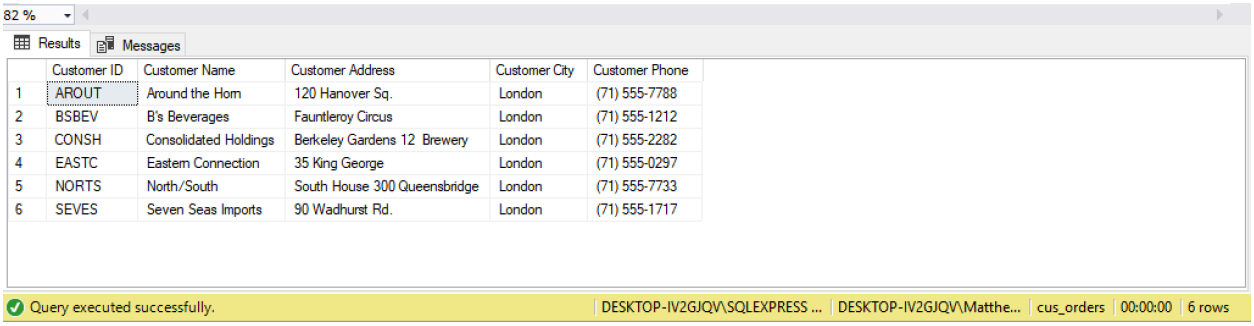
Problem D-1

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, name, address, city and phone from the customers table. Run the stored procedure displaying customers living in **London**.

Code:

```
CREATE PROCEDURE sp_customer_city
(
  @checkcity     varchar(20)
)
AS
  SELECT
         'Customer ID'          = customers.customer_id,
         'Customer Name'        = customers.customer_name,
         'Customer Address'     = customers.customer_address,
         'Customer City'        = customers.customer_city,
         'Customer Phone'       = customers.customer_phone
  FROM customers
  WHERE customers.customer_city = @checkcity;
GO

EXECUTE sp_customer_city 'London';
GO
```

Completed Successfully (6 rows):



	Customer ID	Customer Name	Customer Address	Customer City	Customer Phone
1	AROUT	Around the Horn	120 Hanover Sq.	London	(71) 555-7788
2	BSBEV	B's Beverages	Fauntleroy Circus	London	(71) 555-1212
3	CONSH	Consolidated Holdings	Berkeley Gardens 12 Brewery	London	(71) 555-2282
4	EASTC	Eastern Connection	35 King George	London	(71) 555-0297
5	NORTS	North/South	South House 300 Queensbridge	London	(71) 555-7733
6	SEVES	Seven Seas Imports	90 Wadhurst Rd.	London	(71) 555-1717

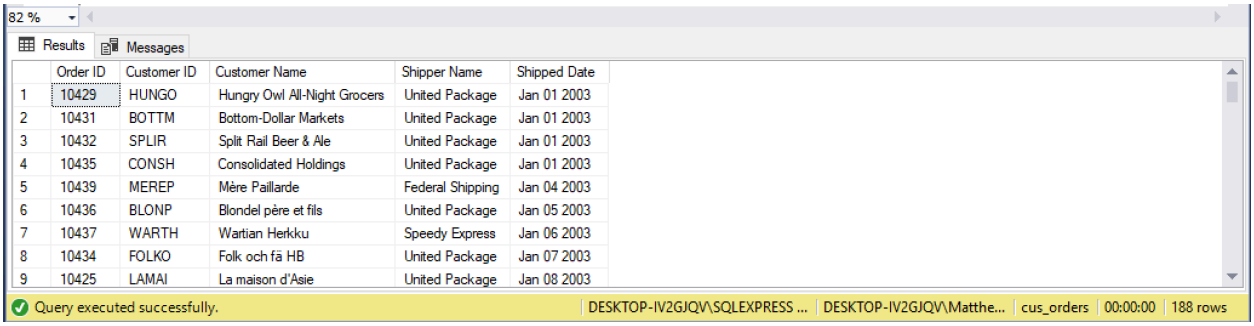
Problem D-2

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 customer name from the customer table, and the shipper name from the shippers table. Run the stored procedure displaying orders from **January 1, 2003** to **June 30, 2003**.

Code:

```
CREATE PROCEDURE sp_orders_by_dates
(
  @startdate     datetime,
  @enddate       datetime
)
AS
  SELECT
         'Order ID'                    = orders.order_id,
         'Customer ID'                 = orders.customer_id,
         'Customer Name'               = customers.customer_name,
         'Shipper Name'                = shippers.ship_name,
         'Shipped Date'                = FORMAT(orders.order_shipped_date, 'MMM dd yyyy')
  FROM orders
  INNER JOIN customers ON orders.customer_id=customers.customer_id
  INNER JOIN shippers ON orders.shipper_id=shippers.shipper_id
  WHERE orders.order_shipped_date BETWEEN @startdate AND @enddate
  ORDER BY orders.order_shipped_date;
GO

EXECUTE sp_orders_by_dates 'January 1 2003', 'June 30 2003';
GO
```

Completed Successfully (188 rows):



82 %

Results | Messages

	Order ID	Customer ID	Customer Name	Shipper Name	Shipped Date
1	10429	HUNGO	Hungry Owl All-Night Grocers	United Package	Jan 01 2003
2	10431	BOTTM	Bottom-Dollar Markets	United Package	Jan 01 2003
3	10432	SPLIR	Split Rail Beer & Ale	United Package	Jan 01 2003
4	10435	CONSH	Consolidated Holdings	United Package	Jan 01 2003
5	10439	MEREP	Mère Paillarde	Federal Shipping	Jan 04 2003
6	10436	BLONP	Blondel père et fils	United Package	Jan 05 2003
7	10437	WARTH	Wartian Herkku	Speedy Express	Jan 06 2003
8	10434	FOLKO	Folk och fä HB	United Package	Jan 07 2003
9	10425	LAMAI	La maison d'Asie	United Package	Jan 08 2003

Query executed successfully. | DESKTOP-IV2GJQV\SQLEXPRESS ... | DESKTOP-IV2GJQV\Matthe... | cus_orders | 00:00:00 | 188 rows

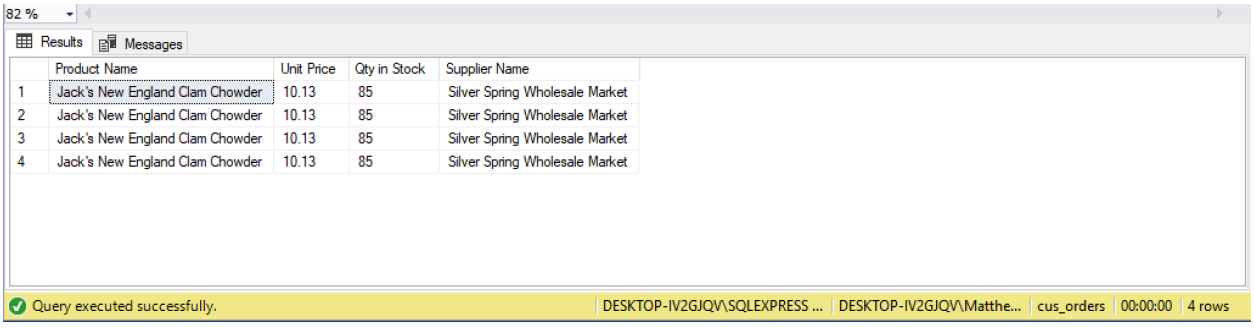
Problem D-3

Create a stored procedure called **sp_product_listing** listing a specified product ordered during a specified month and year. The **product** and the **month** and **year** will be **input parameters** for the stored procedure. Display the product name, unit price, and quantity in stock from the products table, and the supplier name from the suppliers table. Run the stored procedure displaying a product name containing **Jack** and the month of the order date is **June** and the year is **2001**.

Code:

```
CREATE PROCEDURE sp_product_listing
(
  @checkprod     varchar(30),
  @checkmonth datetime,
  @checkyear     datetime
)
AS
  SELECT
          'Product Name'                = products.prod_name,
          'Unit Price'                  = products.prod_unit_price,
          'Qty in Stock'                = products.prod_qty_in_stock,
          'Supplier Name'               = suppliers.sup_name
  FROM products
  INNER JOIN suppliers ON products.supplier_id=suppliers.supplier_id
  INNER JOIN order_details ON products.product_id=order_details.product_id
  INNER JOIN orders ON order_details.order_id=orders.order_id
  WHERE products.prod_name LIKE '%' + @checkprod + '%'
  AND DATEPART(MONTH, orders.order_date) = @checkmonth
  AND DATEPART(YEAR, orders.order_date) = @checkyear;
GO

EXECUTE sp_product_listing 'jack', 06, 2001;
GO
```

Completed Successfully (4 rows):



Problem D-4

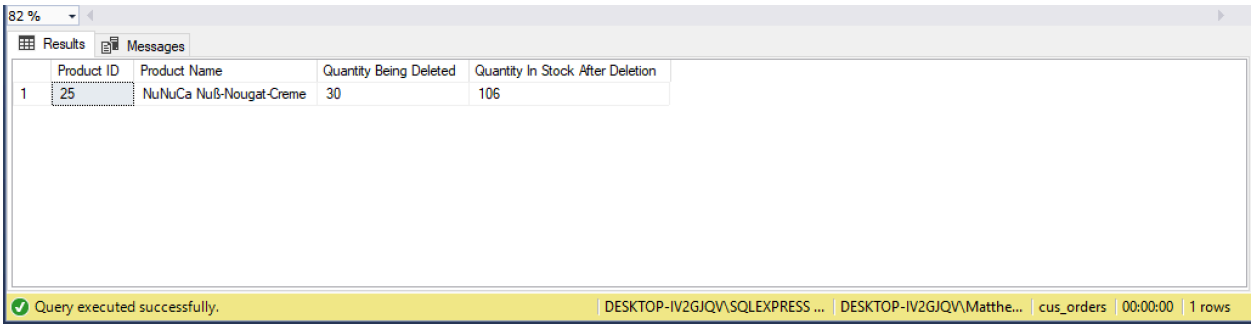
Create a **DELETE** trigger on the order_details table to display the information shown below when you issue the following statement:

```
DELETE order_details WHERE order_id=10001 AND product_id=25
```

Code:

```
CREATE TRIGGER tr_dl_qty_update
ON order_details
FOR DELETE
AS
DECLARE
  @prod_id              id_num,
  @ord_id               id_num,
  @qty_deleted          INT
SELECT
  @prod_id              = deleted.product_id,
  @ord_id               = deleted.order_id,
  @qty_deleted          = deleted.quantity
FROM deleted
IF @@ROWCOUNT = 0
  PRINT 'NOTHING FOUND!'
ELSE
  BEGIN
  -- UPDATE THE PRODUCTS QUANTITY IN STOCK.
  UPDATE products
         SET products.prod_qty_in_stock = products.prod_qty_in_stock + @qty_deleted
         WHERE products.product_id = @prod_id
  END
  BEGIN
         -- DISPLAY THE DETAILS
         SELECT
                'Product ID'                        = products.product_id,
                'Product Name'                      = products.prod_name,
                'Quantity Being Deleted'            = @qty_deleted,
                'Quantity In Stock After Deletion'  = products.prod_qty_in_stock
         FROM deleted
         INNER JOIN products ON deleted.product_id=products.product_id
         WHERE products.product_id = 25;
  END;
GO

DELETE order_details
WHERE order_id = 10001 AND product_id = 25;
GO
```

Completed Successfully (1 row):



Problem D-5

Create an **UPDATE** trigger called **tr_qty_check** on the order_details table which will reject any update to the quantity column if an addition to the original quantity cannot be supplied from the existing quantity in stock. The trigger should also report on the additional quantity needed and the quantity available. If there is enough stock, the trigger should update the stock value in the products table by subtracting the additional quantity from the original stock value and display the updated stock value.

Code:

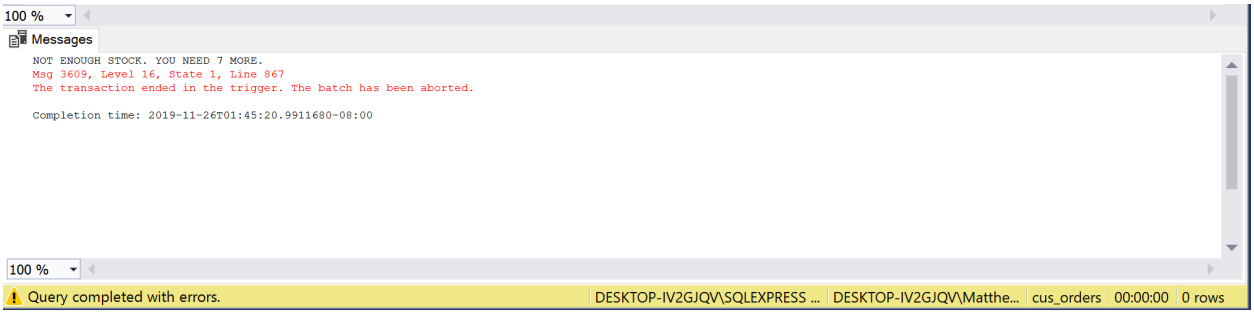
```
CREATE TRIGGER tr_qty_check
ON order_details
FOR UPDATE
AS
DECLARE
  @prod_id              id_num,
  @quantity             INT,
  @instock              INT
SELECT
  @prod_id              = inserted.product_id,
  @quantity             = inserted.quantity - deleted.quantity,
  @instock              = products.prod_qty_in_stock
FROM inserted
INNER JOIN deleted ON inserted.product_id=deleted.product_id
INNER JOIN products ON inserted.product_id=products.product_id
IF @quantity > @instock
  BEGIN
         PRINT 'NOT ENOUGH STOCK. YOU NEED ' + CONVERT(varchar(10), (@quantity - @instock))
+ ' MORE.'
         ROLLBACK TRANSACTION
  END
ELSE
  BEGIN
         UPDATE products
         SET products.prod_qty_in_stock = products.prod_qty_in_stock - @quantity
         WHERE products.product_id = @prod_id
         SELECT
                'Name'          = products.prod_name,
                'Updated Qty'   = products.prod_qty_in_stock
         FROM products
         WHERE product_id = @prod_id
  END
GO
```

Completed Successfully:

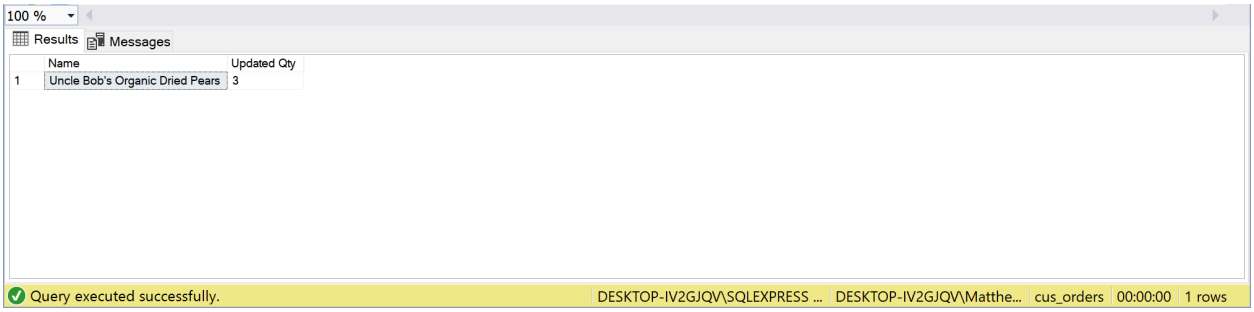
Problem D-6

Run the following 2 queries separately to verify your trigger

```
UPDATE order_details
SET quantity =50
WHERE order_id = '10044'
     AND product_id = 7;
```




```
UPDATE order_details
SET quantity =40
WHERE order_id = '10044'
     AND product_id = 7;
```



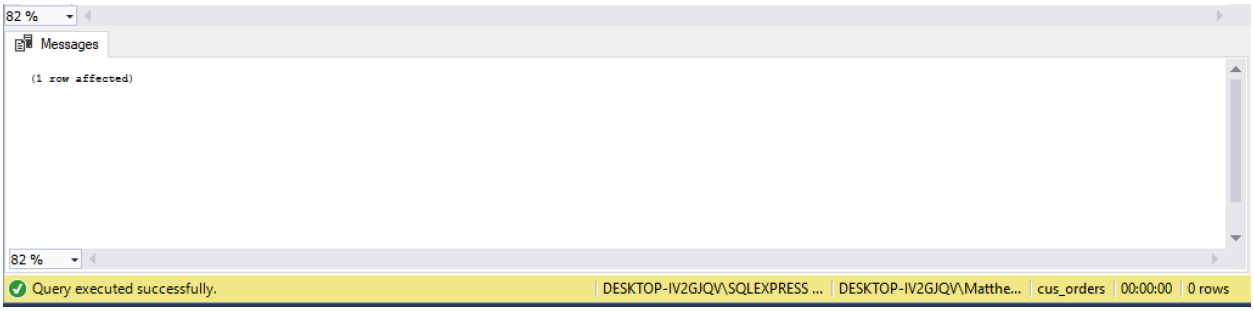
Problem D-7

Create a stored procedure called **sp_del_inactive_cust** to **delete** customers that have no orders.

Code:

```
CREATE PROCEDURE sp_del_inactive_cust
AS
DELETE
  FROM customers
  WHERE customers.customer_id NOT IN
  (
         SELECT orders.customer_id
         FROM orders
  );
GO

EXECUTE sp_del_inactive_cust;
```

Completed Successfully (1 row):



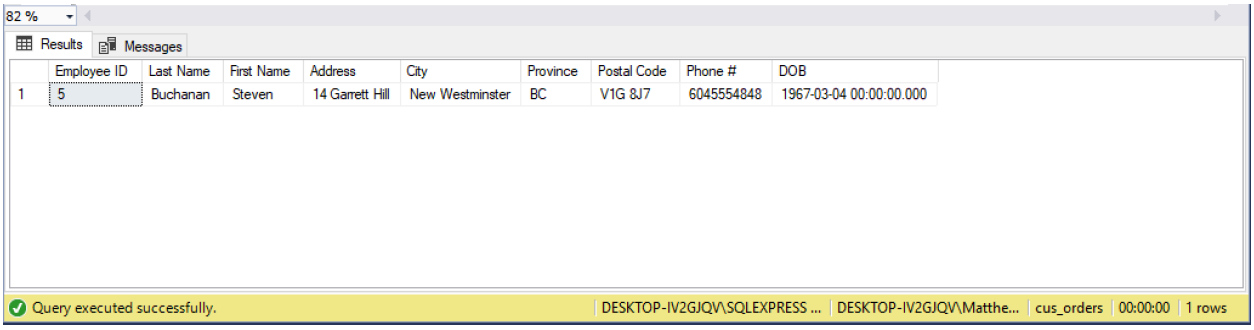
Problem D-8

Create a stored procedure called **sp_employee_information** to display the employee information for a particular employee. The **employee id** will be an **input parameter** for the stored procedure. Run the stored procedure displaying information for employee id of **5**.

Code:

```
CREATE PROCEDURE sp_employee_information
(
  @checkEmp id_num
)
AS
SELECT
  'Employee ID'          = employee.employee_id,
  'Last Name'            = employee.emp_last_name,
  'First Name'           = employee.emp_first_name,
  'Address'              = employee.emp_address,
  'City'                 = employee.emp_city,
  'Province'             = employee.emp_province,
  'Postal Code'          = employee.emp_postal_code,
  'Phone #'              = employee.emp_phone,
  'DOB'                  = employee.emp_birth_date
FROM employee
WHERE employee.employee_id = @checkEmp;
GO

EXECUTE sp_employee_information 5;
GO
```

Completed Successfully (1 row):



Problem D-9

Create a stored procedure called **sp_reorder_qty** to show when the reorder level subtracted from the quantity in stock is less than a specified value. The **unit** value will be an **input parameter** for the stored procedure. Display the product id, quantity in stock, and reorder level from the products table, and the supplier name, address, city, and province from the suppliers table. Run the stored procedure displaying the information for a value of **5**.

Code:

```
CREATE PROCEDURE sp_reorder_qty
(
  @checkNUm int
)
AS
  SELECT
          'Product ID'          = products.product_id,
          'Supplier Name'       = suppliers.sup_name,
          'Address'             = suppliers.sup_address,
          'City'                = suppliers.sup_city,
          'Province'            = suppliers.sup_province,
          'Qty'                 = products.prod_qty_in_stock,
          'Reorder Lvl'         = products.prod_reorder_level
  FROM products
  INNER JOIN suppliers ON products.supplier_id=suppliers.supplier_id
  WHERE (products.prod_qty_in_stock - prod_reorder_level) < @checkNUm;
GO

EXECUTE sp_reorder_qty 5;
GO
```

Completed Successfully (24 rows):

82 %

Results | Messages

	Product ID	Supplier Name	Address	City	Province	Qty	Reorder Lvl
1	2	Edward's Products Ltd.	1125 Howe Street	Vancouver	BC	17	25
2	3	Edward's Products Ltd.	1125 Howe Street	Vancouver	BC	13	25
3	5	New Orlean's Spices Ltd.	1040 Georgia Street West	Vancouver	BC	0	0
4	7	Macaulay Products Company	4800 Kingsway	Burnaby	BC	3	10
5	11	Amstrong Company	1638 Derwent Way	Richmond	BC	22	30
6	17	Steveston Export Company	2951 Moncton Street	Richmond	BC	0	0
7	21	Dare Manufacturer Ltd.	1603 3rd Avenue West	Burnaby	BC	3	5
8	29	South Harbour Products Ltd.	35 Yale Crescent	Surrey	BC	0	0
9	30	Kaplan Ltd.	3016 19th Street South	Vancouver	BC	10	15

Query executed successfully. | DESKTOP-IV2GJQV\SQLEXPRESS ... | DESKTOP-IV2GJQV\Matthe... | cus_orders | 00:00:00 | 24 rows

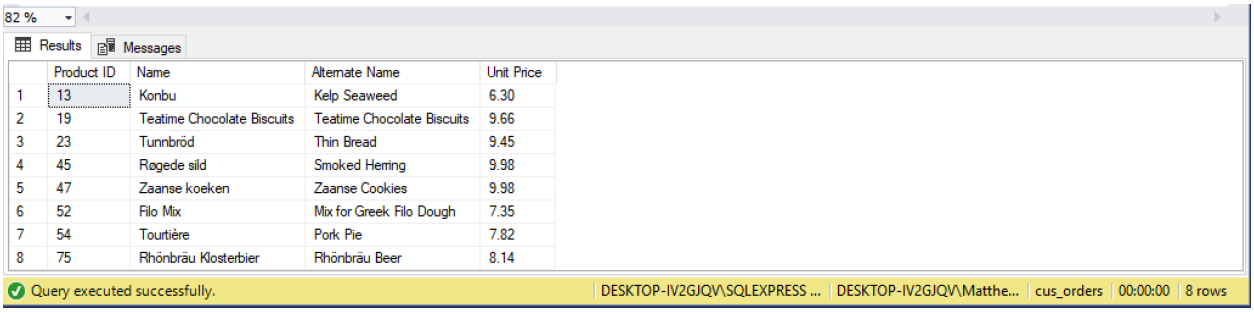
Problem D-10

Create a stored procedure called **sp_unit_prices** for the product table 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, alternate name, and unit price from the products table. Run the stored procedure to display products where the unit price is between **$5.00** and **$10.00**.

Code:

```
CREATE PROCEDURE sp_unit_prices
(
  @checkLow      money,
  @checkHigh     money
)
AS
  SELECT
         'Product ID'          = products.product_id,
         'Name'                = products.prod_name,
         'Alternate Name'      = products.prod_alt_name,
         'Unit Price'          = products.prod_unit_price
  FROM products
  WHERE products.prod_unit_price > @checkLow AND products.prod_unit_price < @checkHigh;
  GO

EXECUTE sp_unit_prices 5.00, 10.00;
GO
```

Completed Successfully (8 rows):



82 %

Results | Messages

	Product ID	Name	Alternate Name	Unit Price
1	13	Konbu	Kelp Seaweed	6.30
2	19	Teatime Chocolate Biscuits	Teatime Chocolate Biscuits	9.66
3	23	Tunnbröd	Thin Bread	9.45
4	45	Røgede sild	Smoked Herring	9.98
5	47	Zaanse koeken	Zaanse Cookies	9.98
6	52	Filo Mix	Mix for Greek Filo Dough	7.35
7	54	Tourtière	Pork Pie	7.82
8	75	Rhönbräu Klosterbier	Rhönbräu Beer	8.14

Query executed successfully. | DESKTOP-IV2GJQV\SQLEXPRESS ... | DESKTOP-IV2GJQV\Matthe... | cus_orders | 00:00:00 | 8 rows

Challenges

Problem C-8

This problem tasks us with: *list all the employees and all the columns in the employee table. Run the view for employee ids **5**, **7**, and **9**. Display the employee id, last name, first name, and birth date. Format the name as last name followed by a comma and a space followed by the first name. Format the birth date as **YYYY.MM.DD**.*

I found this one tough to reconcile as it asks for a formatted output of four columns and not to display everything else despite saying to select all columns from the table. I wasn't sure how to accomplish that within a view, but also for the sake of maximum reusability to generate the requested output I opted to simply SELECT the four columns for output for the problem. This means that when running the view it does not require a complex SELECT statement, just the basic variables for the data you want displayed.

Here is a version that selects everything from the table followed by the statement to generate the requested output.

Code:

```
CREATE VIEW vw_list_all_employees
AS
SELECT
  'Employee ID'          = employee.employee_id,
  'Name'                 = (employee.emp_last_name + ', ' + employee.emp_first_name),
  'Birthdate'            = CONVERT(char(12), employee.emp_birth_date, 102),
  'Address'              = employee.emp_address,
  'City'                 = employee.emp_city,
  'Province'             = employee.emp_province,
  'Postal'               = employee.emp_postal_code,
  'Phone'                = employee.emp_phone,
  'Birth Date'           = employee.emp_birth_date
FROM employee;

SELECT
  [Employee ID],
  [Name],
  [Birthdate]
FROM vw_list_all_employees
WHERE [Employee ID] = 5
  OR [Employee ID] = 7
  OR [Employee ID] = 9;
```

Result:

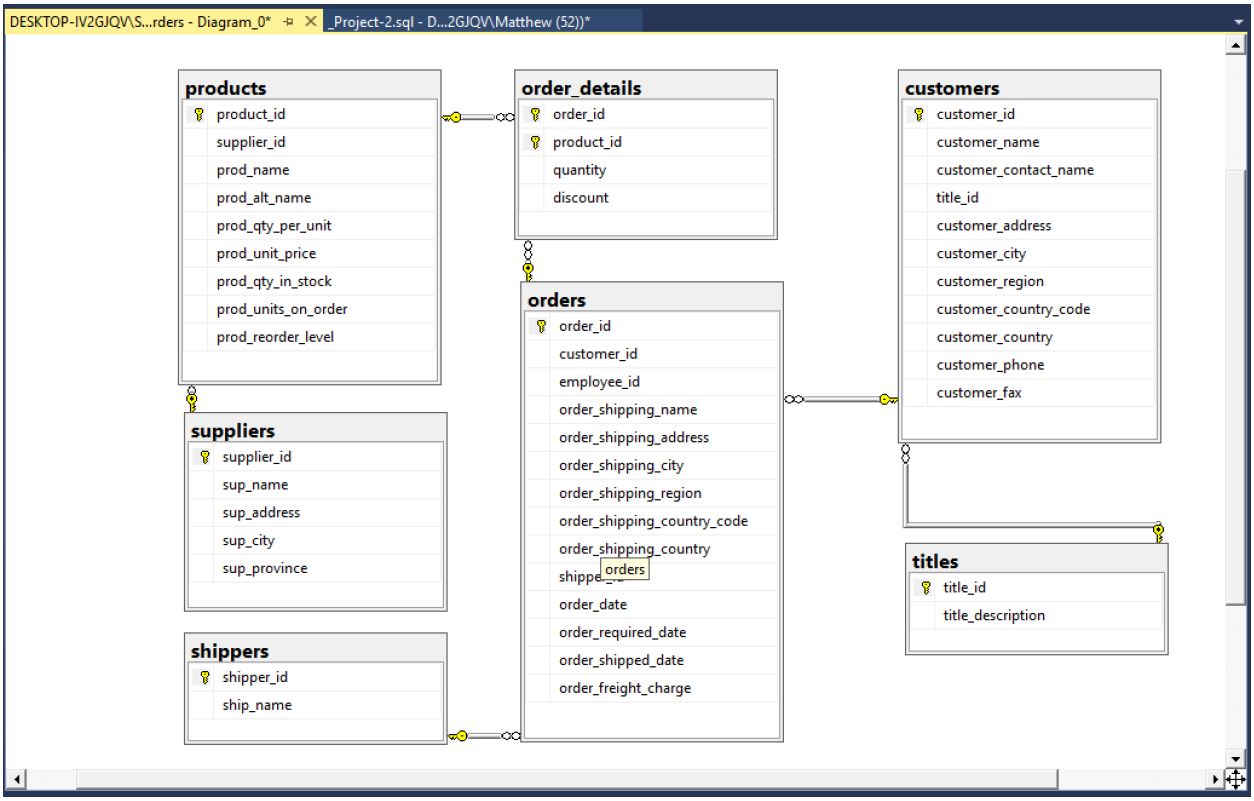
Employee ID	Name	Birthdate
5	Buchanan, Steven	1967.03.04
7	King, Robert	1972.05.29
9	Dodsworth, Anne	1978.01.27

Appendix I

Concerning Database Diagrams

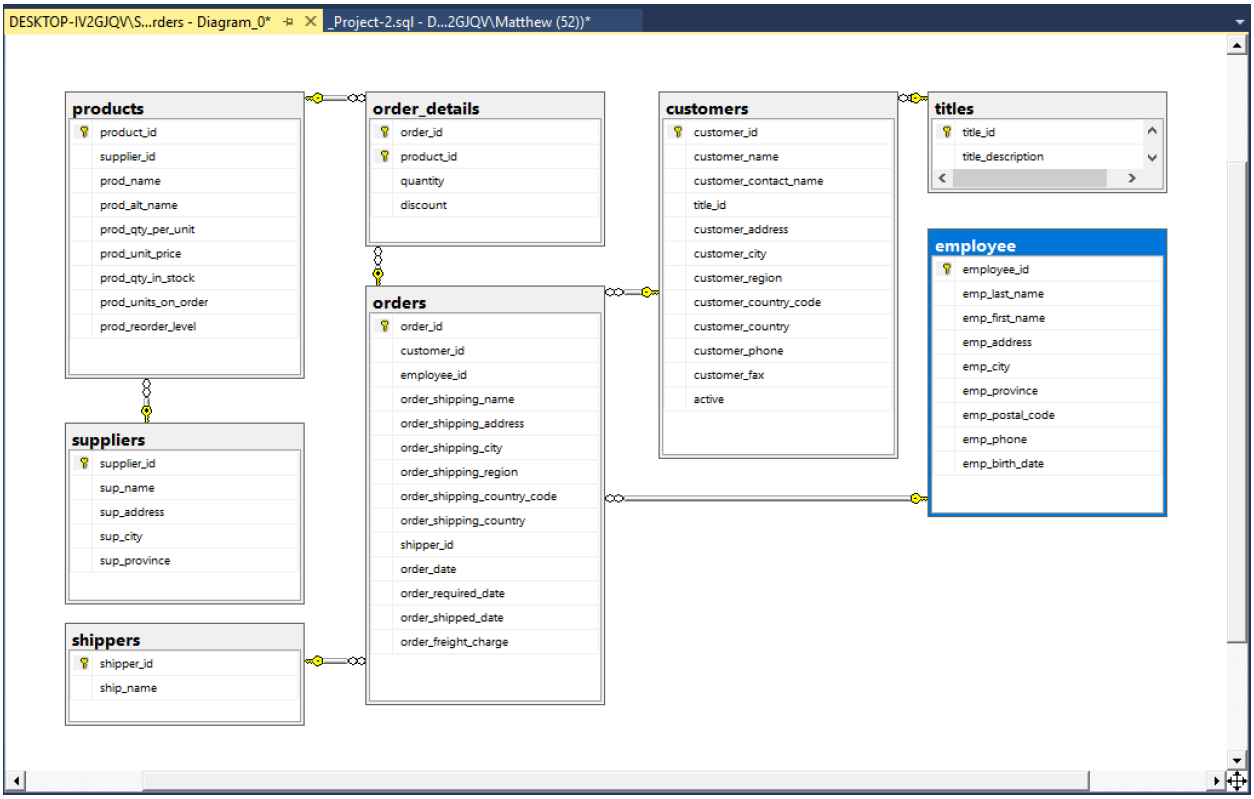
Database Diagram 1

Full diagram of the database at the end of Part A.



Database Diagram 2

Full diagram of the database after Problem C-3



Appendix II
The Code

Because of the way I inserted the code file, code will begin on the following page.

```sql
/*
    By:         Matthew Simpson Student ID: A00820997
    For:        COMP 1630       Instructor: Mark Bacchus
    Date:       Fall 2019
*/

USE master;

/*
    CHECK FOR EXISTING DATABASE AND DROP IT IF IT EXISTS.
    COPIED THIS FROM THE PUBS DATABASE SCRIPT.
    SEEMED LIKE A GOOD IDEA FOR RE-RUNNING THE SCRIPT
    OVER AND OVER.
*/

IF EXISTS (
        SELECT *
        FROM sysdatabases
        WHERE name = 'cus_orders'
        )
BEGIN
    PRINT 'DROPPING EXISTING DATABASES';
    DROP DATABASE cus_orders;
END
GO

/*
    A-1: CREATE THE DATABASE 'cus_orders'.
*/

CREATE DATABASE cus_orders;
GO

USE cus_orders;
GO

/*
    A-2: CREATING TWO CUSTOM DATA TYPES TO USE WITH
    ID FIELDS, ONE FROM CHAR AND ONE FROM INT
*/

CREATE TYPE id_cus
FROM CHAR(5) NOT NULL;
```

```
44
45 CREATE TYPE id_num
46 FROM INT NOT NULL;
47 GO
48
49
50 /*
51     A-3: CREATE THE TABLES PLEASE NOTE I HAVE CHANGED
52     THE COLUMN NAMES SLIGHTLY
53 */
54
55 -- CUSTOMERS TABLE
56 CREATE TABLE customers (
57     customer_id id_cus,
58     customer_name VARCHAR(50) NOT NULL,
59     customer_contact_name VARCHAR(30),
60     title_id CHAR(3),
61     customer_address VARCHAR(50) NOT NULL,
62     customer_city VARCHAR(20),
63     customer_region VARCHAR(15),
64     customer_country_code VARCHAR(10),
65     customer_country VARCHAR(15),
66     customer_phone VARCHAR(20),
67     customer_fax VARCHAR(20)
68     );
69 GO
70
71 -- ORDERS TABLE
72 CREATE TABLE orders (
73     order_id id_num,
74     customer_id id_cus,
75     employee_id INT NOT NULL,
76     order_shipping_name VARCHAR(50),
77     order_shipping_address VARCHAR(50),
78     order_shipping_city VARCHAR(20),
79     order_shipping_region VARCHAR(15),
80     order_shipping_country_code VARCHAR(10),
81     order_shipping_country VARCHAR(15),
82     shipper_id INT NOT NULL,
83     order_date DATETIME,
84     order_required_date DATETIME,
85     order_shipped_date DATETIME,
86     order_freight_charge MONEY
```

```
    );
GO

-- ORDER_DETAILS TABLE
CREATE TABLE order_details (
    order_id id_num,
    product_id id_num,
    quantity INT NOT NULL,
    discount FLOAT NOT NULL
    );
GO

-- PRODUCTS TABLE
CREATE TABLE products (
    product_id id_num,
    supplier_id INT NOT NULL,
    prod_name VARCHAR(40),
    prod_alt_name VARCHAR(40),
    prod_qty_per_unit VARCHAR(25),
    prod_unit_price MONEY,
    prod_qty_in_stock INT,
    prod_units_on_order INT,
    prod_reorder_level INT
    );
GO

-- SHIPPERS TABLE
CREATE TABLE shippers (
    shipper_id INT IDENTITY,
    ship_name VARCHAR(20) NOT NULL
    );
GO

-- SUPPLIERS TABLE
CREATE TABLE suppliers (
    supplier_id INT IDENTITY,
    sup_name VARCHAR(40) NOT NULL,
    sup_address VARCHAR(30),
    sup_city VARCHAR(20),
    sup_province CHAR(2)
    );
GO

```

```
-- TITLES TABLE
CREATE TABLE titles (
    title_id CHAR(3) NOT NULL,
    title_description VARCHAR(35) NOT NULL
    );
GO

/*
    A-4: CREATE THE PRIMARY AND FOREIGN KEYS
*/

-- PRIMARY KEYS

ALTER TABLE customers
    ADD PRIMARY KEY (customer_id);
GO

ALTER TABLE orders
    ADD PRIMARY KEY (order_id);
GO

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

ALTER TABLE products
    ADD PRIMARY KEY (product_id);
GO

ALTER TABLE shippers
    ADD PRIMARY KEY (shipper_id);
GO

ALTER TABLE suppliers
    ADD PRIMARY KEY (supplier_id);
GO

ALTER TABLE titles
    ADD PRIMARY KEY (title_id);
GO

-- FOREIGN KEYS

```

```
ALTER TABLE customers
    ADD CONSTRAINT FK_customer_title FOREIGN KEY (title_id)
    REFERENCES titles (title_id);
GO

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

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

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

ALTER TABLE order_details
    ADD CONSTRAINT FK_orddet_product FOREIGN KEY (product_id)
    REFERENCES products (product_id);
GO

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

/*
    A-5: ADD DEFAULT CONSTRAINTS
*/

-- SET CUSTOMERS DEFAULT COUNTRY AS "CANADA"
ALTER TABLE customers
    ADD CONSTRAINT default_country DEFAULT ('Canada') FOR
customer_country;
GO

-- SET ORDERS DEFAULT REQUIRED DATE AS TODAY + 10 DAYS.
ALTER TABLE orders
    ADD CONSTRAINT default_req_date DEFAULT (DATEADD(DAY, 10,
```

```
GETDATE())) FOR order_required_date;
GO

-- SET DEFAULT ORDER DETAILS QUANTITY TO ONE OR MODE
ALTER TABLE order_details
    ADD CONSTRAINT default_qty CHECK (quantity >= 1);
GO

-- SET PRODUCTS DEFAULT REORDER POINT TO ONE OR MORE
ALTER TABLE products
    ADD CONSTRAINT default_reorder CHECK (prod_reorder_level >=
1);
GO

-- SET PRODUCT MAX IN STOCK TO 150 OR FEWER
ALTER TABLE products
    ADD CONSTRAINT max_in_stock CHECK (prod_qty_in_stock <=
150);
GO

-- SET SUPPLIERS DEFAULT PROVINCE TO BC
ALTER TABLE suppliers
    ADD CONSTRAINT default_prov DEFAULT ('BC') FOR sup_province;
GO

/*
    A-6: BULK LOAD THE DATA FOR CREATED TABLES
*/

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

-- ORDERS
BULK INSERT orders
FROM 'C:\textfiles\orders.txt' WITH (
```

```
        CODEPAGE = 1252,
        DATAFILETYPE = 'char',
        FIELDTERMINATOR = '\t',
        KEEPNULLS,
        ROWTERMINATOR = '\n'
        );
GO

-- ORDER DETAILS
BULK INSERT order_details
FROM 'C:\textfiles\order_details.txt' WITH (
        CODEPAGE = 1252,
        DATAFILETYPE = 'char',
        FIELDTERMINATOR = '\t',
        KEEPNULLS,
        ROWTERMINATOR = '\n'
        );
GO

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

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

-- SUPPLIERS
BULK INSERT suppliers
```

```
FROM 'C:\textfiles\suppliers.txt' WITH (
        CODEPAGE = 1252,
        DATAFILETYPE = 'char',
        FIELDTERMINATOR = '\t',
        KEEPNULLS,
        ROWTERMINATOR = '\n'
        );
GO

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

/*
    B-1: LIST CUSTOMER ID, NAME, CITY, COUNTRY FROM
    THE CUSTOMERS TABLE ORDERED BY CUSTOMER ID.
*/

SELECT
    'Customer ID'   = customer_id,
    'Name'          = customer_name,
    'City'          = customer_city,
    'Country'       = customer_country
FROM
    customers
ORDER BY
    customer_id;
GO

/*
    B-2: ADD NEW COLUMN 'ACTIVE' TO CUSTOMERS TABLE.
    ONLY VALID VALUES 1 OR 0.  DEFAULT VALUE 1.
*/

ALTER TABLE customers
ADD active int NOT NULL
```

```
CONSTRAINT default_active DEFAULT 1 WITH VALUES
CONSTRAINT one_or_zero CHECK (active = 1 OR active = 0);
GO

/*
    B-3: LIST ORDERS BETWEEN JANUARY 1 AND DECEMBER 31
    2001. DISPLAY ORDER ID, PRODUCT NAME, CUSTOMER NAME,
    NEW ORDER SHIPPED DATE (ORDER DATE + 17 DAYS) AND
    ORDER COST (QUANTITY * UNIT PRICE)
*/

SELECT
    'Order ID'          = orders.order_id,
    'Product Name'      = products.prod_name,
    'Customer Name'     = customers.customer_name,
    'Order Date'        = CONVERT(CHAR(12), orders.order_date,
109),
    'New Shipped Date'  = CONVERT(CHAR(12), DATEADD(DAY, 17,
orders.order_date)),
    'Order Cost'        = order_details.quantity *
products.prod_unit_price
FROM
    order_details
INNER JOIN orders 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
WHERE orders.order_date BETWEEN 'January 1 2001' AND 'December
31 2001';
GO

/*
    B-4: LIST ORDERS THAT HAVE NOT SHIPPED. ORDER BY SHIPPED
DATE.
*/

SELECT
    'Customer ID'   = customers.customer_id,
    'Name'          = customers.customer_name,
    'Phone'         = customers.customer_phone,
    'Order ID'      = orders.order_id,
    'Order Date'    = CONVERT(CHAR(12), orders.order_date, 109)
FROM
```

```
    orders
INNER JOIN customers ON orders.customer_id=customers.customer_id
WHERE orders.order_shipped_date IS NULL;

/*
    B-5: LIST ALL CUSTOMERS WHERE REGION IS NULL
*/

SELECT
    'Customer ID'    = customer_id,
    'Name'           = customer_name,
    'City'           = customer_city,
    'Description'    = title_description
FROM
    customers
INNER JOIN titles ON customers.title_id=titles.title_id
WHERE customer_region IS NULL;
GO

/*
    B-6: LIST ALL PRODUCTS WHERE REORDER POINT
    IS HIGHER HTAN QUANTITY IN STOCK
*/

SELECT
    'Supplier Name' = sup_name,
    'Product Name'  = prod_name,
    'Reorder Level' = prod_reorder_level,
    'Qty In Stock'  = prod_qty_in_stock
FROM
    products
INNER JOIN suppliers ON
products.supplier_id=suppliers.supplier_id
WHERE prod_reorder_level > prod_qty_in_stock
ORDER BY [Supplier Name];
GO

/*
    B-7: CALCULATE LENGTH IN YEARS FROM JANUARY 1 2008
    TO WHEN AN ORDER WAS SHIPPED WHERE THE SHIPPED DATE
    IS NOT NULL.
*/

```

```
SELECT
    'Order ID'      = orders.order_id,
    'Name'          = customers.customer_name,
    'Contact Name'  = customers.customer_contact_name,
    'Shipped Date'  = CONVERT(CHAR(12), orders.order_date, 109),
    'Years Elapsed' = DATEDIFF(year, orders.order_date, 'January
1 2008')
FROM
    orders
INNER JOIN customers ON orders.customer_id=customers.customer_id
WHERE orders.order_shipped_date IS NOT NULL;
GO

/*
    B-8: LIST NUMBER OF CUSTOMERS WITH NAMES BEGINNING
    WITH EACH LETTER OF THE ALPHABET.
    IGNORE THE LETTER S.
    IGNORE LETTERS WITH FEWER THAN 2 ENTRIES.
*/

SELECT
    'Name'          = SUBSTRING(customers.customer_name, 1, 1),
    'Total'         = count(*)
FROM
    customers
GROUP BY SUBSTRING(customers.customer_name, 1, 1)
HAVING count(*) >= 2
AND SUBSTRING(customers.customer_name,1,1) != 's';


/*
    B-9: LIST ORDER DETAILS FOR ORDERS WHERE QUANTITY
    IS GREATER THAN 100 ORDER RESULT BY ORDER ID
*/

SELECT
    'Order ID'      = order_details.order_id,
    'Quantity'      = order_details.quantity,
    'Product ID'    = products.product_id,
    'Reorder Level' = products.prod_reorder_level,
    'Supplier ID'   = products.supplier_id
FROM
    order_details
```

```
INNER JOIN products ON
order_details.product_id=products.product_id
WHERE order_details.quantity > 100
ORDER BY order_details.order_id;
GO

/*
    B-10- LIST ALL PRODUCTS WITH SUBSTRING 'TOFU' OR
    'CHEF' IN THE NAME ORDER BY PRODUCT NAME
*/

SELECT
    'Product ID'    = products.product_id,
    'Product Name'  = products.prod_name,
    'Qty Per Unit'  = products.prod_qty_per_unit,
    'Unit Price'    = products.prod_unit_price
FROM
    products
WHERE
    products.prod_name LIKE '%chef%'
    OR products.prod_name LIKE '%tofu%'
ORDER BY
    products.prod_name;
GO

/*
    C-1, C-2: CREATE THE EMPLOYEE TABLE AND SET THE
    EMPLOYEE ID AS THE PRIMARY KEY.
*/

CREATE TABLE employee (
    employee_id INT NOT NULL,
    emp_last_name varchar(30) NOT NULL,
    emp_first_name varchar(15) NOT NULL,
    emp_address varchar(30),
    emp_city varchar(20),
    emp_province char(2),
    emp_postal_code varchar(7),
    emp_phone varchar(10),
    emp_birth_date datetime NOT NULL
);
GO

```

```
ALTER TABLE employee
    ADD PRIMARY KEY (employee_id);
GO

/*
    C-3: LOAD THE EMPLOYEE DATA INTO THE EMPLOYEE TABLE.
    SET UP THE FOREIGN KEY RELATIONSHIP
*/

BULK INSERT employee
FROM 'C:\textfiles\employee.txt' WITH (
        CODEPAGE = 1252,
        DATAFILETYPE = 'char',
        FIELDTERMINATOR = '\t',
        KEEPNULLS,
        ROWTERMINATOR = '\n'
        );
GO

ALTER TABLE orders
    ADD CONSTRAINT FK_ord_emp FOREIGN KEY (employee_id)
    REFERENCES employee (employee_id);
GO

/*
    C-4: INSERT THE NEW SHIPPER 'QUICK EXPRESS' INTO THE
    SHIPPERS TABLE. SINCE SHIPPERS HAS AN IDENTITY COLUMN,
    EVERYTHING ELSE SHOULD BE TAKEN CARE OF AUTOMAGICALLY
*/

INSERT INTO shippers
    VALUES('Quick Express');
GO

SELECT * FROM shippers;

/*
    C-5: INCREASE THE PRICE OF ALL PRODUCTS WITH CURRENT
    PRICE BETWEEN 5$ AND 10$ BY 5%
*/

UPDATE
    products
```

```
SET products.prod_unit_price = ROUND(products.prod_unit_price * 1.05, 2)
WHERE products.prod_unit_price >= 5 AND products.prod_unit_price <= 10;
GO

/*
    C-6: UPDATE THE FAX VALUE TO 'UNKNOWN' FOR ALL
    CUSTOMERS WHO HAVE A NULL FAX VALUE.
*/

UPDATE
    customers
SET customers.customer_fax = 'UNKNOWN'
WHERE customers.customer_fax IS NULL;
GO

/*
    C-7a: CREATE VIEW VW_ORDER_COST TO LIST THE COST OF ORDERS.
*/

CREATE VIEW vw_order_cost
AS
SELECT
    'Order ID'      = orders.order_id,
    'Order Data'    = orders.order_date,
    'Product ID'    = products.product_id,
    'Customer Name' = customers.customer_name,
    'Order Cost'    = (order_details.quantity * products.prod_unit_price)
FROM order_details
INNER JOIN orders ON order_details.order_id=orders.order_id
INNER JOIN products ON order_details.product_id=products.product_id
INNER JOIN customers ON customers.customer_id=orders.customer_id;
GO

/*
    C-7b: RUN VW_ORDER_COST FOR ORDER IDS BETWEEN 10000 AND 10200
*/

```

```
SELECT *
FROM vw_order_cost
WHERE vw_order_cost.[Order ID] BETWEEN 10000 AND 10200;
GO

/*
    C-8a: CREATE VIEW VW_LIST_EMPLOYEES TO LIST EMPLOYEE
DETAILS.

    THIS QUESTION DESCRIPTION ASKS FOR ALL THE EMPLOYEE COLUMNS
    TO BE LISTED,BUT THEN ALSO FOR FORMATTED OUTPUT FROM THE
    VIEW.  SEE CHALLENGES SECTION
*/

CREATE VIEW vw_list_employees
AS
SELECT
    'Employee ID'    = employee.employee_id,
    'Name'           = (employee.emp_last_name + ', ' +
employee.emp_first_name),
    'Birthdate'      = CONVERT(char(12), employee.emp_birth_date,
102)
FROM employee;
GO

/*
    C-8b RUN VW_LIST_EMPLOYEES FOR EMPLOYEE IDS 5, 7, 9
*/

SELECT *
FROM vw_list_employees
WHERE [Employee ID] = 5
    OR [Employee ID] = 7
    OR [Employee ID] = 9;
GO

/*
    C-9a: CREATE VIEW VW_ALL_ORDERS,
    FORMAT SHIPPED DATE MON DD YYYY
*/

CREATE VIEW vw_all_orders
AS
```

```sql
SELECT
    'Order ID'      = orders.order_id,
    'Customer ID'   = customers.customer_id,
    'Customer Name' = customers.customer_name,
    'City'          = customers.customer_city,
    'Country'       = customers.customer_country,
    'Shipped Date'  = orders.order_shipped_date
FROM orders
INNER JOIN customers ON
orders.customer_id=customers.customer_id;
GO

/*
    C-9b: RUN VW_ALL_ORDERS FOR ALL ORDERS BETWEEN
    JAN 1 2002 AND DEC 31 2002
*/

SELECT
    vw_all_orders.[Order ID],
    vw_all_orders.[Customer ID],
    vw_all_orders.[Customer Name],
    vw_all_orders.[City],
    vw_all_orders.[Country],
    format(vw_all_orders.[Shipped Date], 'MMM dd yyyy') AS
'Shipped Date'
FROM vw_all_orders
WHERE [Shipped Date] BETWEEN 'Jan 1 2002' AND 'Dec 31 2002'
ORDER BY [Customer Name], [Country];
GO

/*
    C-10a: CREATE VW_SUPPLIERS_SHIPPED DO DISPLAY
    DETAILS OF ORDERS SHIPPED.
*/

CREATE VIEW vw_suppliers_shipped
AS
SELECT
    'Supplier ID'   = suppliers.supplier_id,
    'Supplier Name' = suppliers.sup_name,
    'Product ID'    = products.product_id,
    'Product Name'  = products.prod_name
FROM products
```

```sql
INNER JOIN suppliers ON
products.supplier_id=suppliers.supplier_id;
GO

/*
    C-10b: RUN VW_SUPPLIERS_SHIPPED
*/

SELECT *
FROM vw_suppliers_shipped;
GO

/*
    D-1a: CREATE STORED PROCEDURE SP_CUSTOMER_CITY  TO LIST
CUSTOMERS BY SELECTED CITY.
    @CHECKCITY IS A STORED VARIABLE, THE CITY TO BE LISTED.
*/

CREATE PROCEDURE sp_customer_city
(
    @checkcity  varchar(20)
)
AS
    SELECT
        'Customer ID'       = customers.customer_id,
        'Customer Name'     = customers.customer_name,
        'Customer Address'  = customers.customer_address,
        'Customer City'     = customers.customer_city,
        'Customer Phone'    = customers.customer_phone
    FROM customers
    WHERE customers.customer_city = @checkcity;
GO

/*
    D-1b: RUN SP_CUSTOMER_CITY FOR THE CITY 'LONDON'
*/

EXECUTE sp_customer_city 'London';
GO


/*
    D-2a: CREATE SP_ORDERS_BY_DATES TO DISPLAY ORDERS
```

```
    SHIPPED BETWEEN SPECIFIED DATES. @STARTDATE AND
    @ENDDATE ARE STORED VARIABLES, THE DATES TO CHECK
    BETWEEN.
*/

CREATE PROCEDURE sp_orders_by_dates
(
    @startdate  datetime,
    @enddate    datetime
)
AS
    SELECT
        'Order ID'              = orders.order_id,
        'Customer ID'           = orders.customer_id,
        'Customer Name'         = customers.customer_name,
        'Shipper Name'          = shippers.ship_name,
        'Shipped Date'          =
FORMAT(orders.order_shipped_date, 'MMM dd yyyy')
    FROM orders
    INNER JOIN customers ON
orders.customer_id=customers.customer_id
    INNER JOIN shippers ON orders.shipper_id=shippers.shipper_id
    WHERE orders.order_shipped_date BETWEEN @startdate AND
@enddate
    ORDER BY orders.order_shipped_date;
GO

/*
    D-2b: RUN SP_ORDERS_BY_DATES FOR JAN 1 2003 THROUGH JUN 30
2003
*/

EXECUTE sp_orders_by_dates 'January 1 2003', 'June 30 2003';
GO

/*
    D-3a: CREATE SP_PRODUCT_LISTING TO LIST A PRODUCT ORDERED
    DURING SPECIFIED MONTH AND YEAR.
    @CHECKPROD IS THE SUBSTRING OF A PRODUCT NAME TO BE SEARCHED
    @CHECMONTH AND @CHECKYEAR ARE THE MONTH AND YEAR TO BE
SEARCHED.
*/

```

```
CREATE PROCEDURE sp_product_listing
(
    @checkprod  varchar(30),
    @checkmonth datetime,
    @checkyear  datetime
)
AS
SELECT
    'Product Name'              = products.prod_name,
    'Unit Price'                = products.prod_unit_price,
    'Qty in Stock'              = products.prod_qty_in_stock,
    'Supplier Name'             = suppliers.sup_name
FROM products
INNER JOIN suppliers ON
products.supplier_id=suppliers.supplier_id
INNER JOIN order_details ON
products.product_id=order_details.product_id
INNER JOIN orders ON order_details.order_id=orders.order_id
WHERE products.prod_name LIKE '%' + @checkprod + '%'
    AND DATEPART(MONTH, orders.order_date) = @checkmonth
    AND DATEPART(YEAR, orders.order_date) = @checkyear;
GO

/*
    D-3b: RUN SP_PRODUCT_LISTING SEARCHING FOR PRODUCTS
    CONTAINING 'JACK' IN JUNE 2001.
*/

EXECUTE sp_product_listing 'jack', 06, 2001;
GO

/*
    D-4a: CREATE DELETE TRIGGER ON ORDER_DETAILS TO SHOW
    PRODUCT_ID, PRODUCT NAME, QUANTITY DELETED, AND STOCK
    AFTER DELETION WHEN AN ORDER_DETAILS ROW IS DELETED.
*/

CREATE TRIGGER tr_dl_qty_update
ON order_details
FOR DELETE
AS
DECLARE
    @prod_id        id_num,
```

```
785     @ord_id          id_num,
786     @qty_deleted     int
787 SELECT
788     @prod_id         = deleted.product_id,
789     @ord_id          = deleted.order_id,
790     @qty_deleted     = deleted.quantity
791 FROM deleted
792 IF @@ROWCOUNT = 0
793     PRINT 'NOTHING FOUND!'
794 ELSE
795     BEGIN
796     UPDATE products
797         SET products.prod_qty_in_stock =
  … products.prod_qty_in_stock + @qty_deleted
798         WHERE products.product_id = @prod_id
799     END
800     BEGIN
801         SELECT
802             'Product ID'                          =
  … products.product_id,
803             'Product Name'                        =
  … products.prod_name,
804             'Quantity Being Deleted'              = @qty_deleted,
805             'Quantity In Stock After Deletion'    =
  … products.prod_qty_in_stock
806         FROM deleted
807         INNER JOIN products ON
  … deleted.product_id=products.product_id
808         WHERE products.product_id = 25;
809     END;
810 GO
811
812 /*
813     D-4b: TRIGGER THE TRIGGER WITH A DELETE STATEMENT.
814 */
815
816 DELETE order_details
817 WHERE order_id = 10001 AND product_id = 25;
818 GO
819
820 /*
821     D-5a: CREATE TRIGGER TR_QTY_CHECK FOR ORDER_DETAILS
822     WHICH WILL REJECT ANY QUANTITY UPDATE THAT CANNOT
```

```
    BE SUPPLIED BY THE PRODUCT QUANTITY IN STOCK WILL
    ALSO REPORT SHORTFALL IF IN STOCK QUANTITY IS NOT
    ENOUGH
*/

CREATE TRIGGER tr_qty_check
ON order_details
FOR UPDATE
AS
DECLARE
    @prod_id        id_num,
    @quantity       INT,
    @instock        INT
SELECT
    @prod_id        = inserted.product_id,
    @quantity       = inserted.quantity - deleted.quantity,
    @instock        = products.prod_qty_in_stock
FROM inserted
INNER JOIN deleted ON inserted.product_id=deleted.product_id
INNER JOIN products ON inserted.product_id=products.product_id
IF @quantity > @instock
    BEGIN
        PRINT 'NOT ENOUGH STOCK. YOU NEED ' +
CONVERT(varchar(10), (@quantity - @instock)) + ' MORE.'
        ROLLBACK TRANSACTION
    END
ELSE
    BEGIN
        UPDATE products
        SET products.prod_qty_in_stock =
products.prod_qty_in_stock - @quantity
        WHERE products.product_id = @prod_id
        SELECT
            'Name'          = products.prod_name,
            'Updated Qty'   = products.prod_qty_in_stock
        FROM products
        WHERE product_id = @prod_id
    END
GO

/*
    D-6a: RUN FIRST SUPPLIED QUERY
*/
```

```
UPDATE order_details
SET quantity =50
WHERE order_id = '10044'
     AND product_id = 7;
GO

/*
    D-6B: RUN SECOND SUPPLIED QUERY
*/

UPDATE order_details
SET quantity = 40
WHERE order_id = '10044'
     AND product_id = 7;
GO


/*
    D-7: STORED PROCEDURE TO DELETE INACTIVE CUSTOMERS.
*/

CREATE PROCEDURE sp_del_inactive_cust
AS
DELETE
    FROM customers
    WHERE customers.customer_id NOT IN
    (
        SELECT orders.customer_id
        FROM orders
    );
GO

--  RUN THIS PROCEDURE

EXECUTE sp_del_inactive_cust;
GO

/*
    D-8: LIST DETAILS OF SPECIFIED EMPLOYEE. @checkEmp IS THE
    EMPLOYEE ID OF THE EMPLOYEE IN QUESTION
*/

```

```
CREATE PROCEDURE sp_employee_information
(
    @checkEmp id_num
)
AS
SELECT
    'Employee ID'       = employee.employee_id,
    'Last Name'         = employee.emp_last_name,
    'First Name'        = employee.emp_first_name,
    'Address'           = employee.emp_address,
    'City'              = employee.emp_city,
    'Province'          = employee.emp_province,
    'Postal Code'       = employee.emp_postal_code,
    'Phone #'           = employee.emp_phone,
    'DOB'               = employee.emp_birth_date
FROM employee
WHERE employee.employee_id = @checkEmp;
GO

-- RUN THE PROCEDURE

EXECUTE sp_employee_information 5;
GO

/*
    D-9: CREATE PROCEDURE sp_reorder_qty TO SHOW WHEN REORDER
    LEVEL SUBTRACTED FROM QUANTITY IN STOCK IS LESS THAN A
    SPECIFIED VALUE. @checkNum IS THE VALUE TO CHECK
*/

CREATE PROCEDURE sp_reorder_qty
(
    @checkNUm int
)
AS
    SELECT
        'Product ID'        = products.product_id,
        'Supplier Name'     = suppliers.sup_name,
        'Address'           = suppliers.sup_address,
        'City'              = suppliers.sup_city,
        'Province'          = suppliers.sup_province,
        'Qty'               = products.prod_qty_in_stock,
        'Reorder Lvl'       = products.prod_reorder_level
```

```
    FROM products
    INNER JOIN suppliers ON
products.supplier_id=suppliers.supplier_id
    WHERE (products.prod_qty_in_stock - prod_reorder_level) <
@checkNUm;
GO

--   RUN THE PROCEDURE

EXECUTE sp_reorder_qty 5;
GO

/*
    D-10: CREATE PRCEUDRE TO SHOW  DETAILS FOR ALL PRODUCTS
    WITH UNIT PRICE BETWEEN SPECIFIED UPPER AND LOWER VALUES.
    @checkLow WILL BE THE LOW VALUE AND @checkHIgh WILL BE
    THE HIGH VALUE.
*/

CREATE PROCEDURE sp_unit_prices
(
    @checkLow       money,
    @checkHigh      money
)
AS
    SELECT
        'Product ID'        = products.product_id,
        'Name'              = products.prod_name,
        'Alternate Name'    = products.prod_alt_name,
        'Unit Price'        = products.prod_unit_price
    FROM products
    WHERE products.prod_unit_price > @checkLow AND
products.prod_unit_price < @checkHigh;
    GO

--  RUN THE PROCEDURE.

EXECUTE sp_unit_prices 5.00, 10.00;
GO
```