

COMP 1630 - Project Two

Instructor: Mark Bacchus

Date: 26 November 2019

Matthew Simpson

Student Number: [REDACTED]



Table of Contents

Introduction	4
Part A	5
<i>Problem A-1</i>	5
<i>Problem A-2</i>	6
<i>Problem A-3</i>	7
<i>Problem A-4</i>	10
<i>Problem A-5</i>	13
<i>Problem A-6:</i>	15
Part B	17
<i>Problem B-1</i>	17
<i>Problem B-2</i>	18
<i>Problem B-3</i>	19
<i>Problem B-4</i>	20
<i>Problem B-5</i>	21
<i>Problem B-6</i>	22
<i>Problem B-7</i>	23
<i>Problem B-8</i>	24
<i>Problem B-9</i>	25
<i>Problem B-10</i>	26
Part C	27
<i>Problem C-1, C-2</i>	27
<i>Problem C-3</i>	28
<i>Problem C-4</i>	29
<i>Problem C-5</i>	30
<i>Problem C-6</i>	31
<i>Problem C-7</i>	32
<i>Problem C-8</i>	33
<i>Problem C-9</i>	34
<i>Problem C-10</i>	35
Part D	36
<i>Problem D-1</i>	36
<i>Problem D-2</i>	37
<i>Problem D-3</i>	38
<i>Problem D-4</i>	39

<i>Problem D-5</i>	41
<i>Problem D-6</i>	42
<i>Problem D-7</i>	43
<i>Problem D-8</i>	44
<i>Problem D-9</i>	45
<i>Problem D-10</i>	46
Challenges	47
<i>Problem C-8</i>	47
Appendix I	48
<i>Database Diagram 1</i>	48
<i>Database Diagram 2</i>	49
Appendix II	50
<i>The Code</i>	50

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

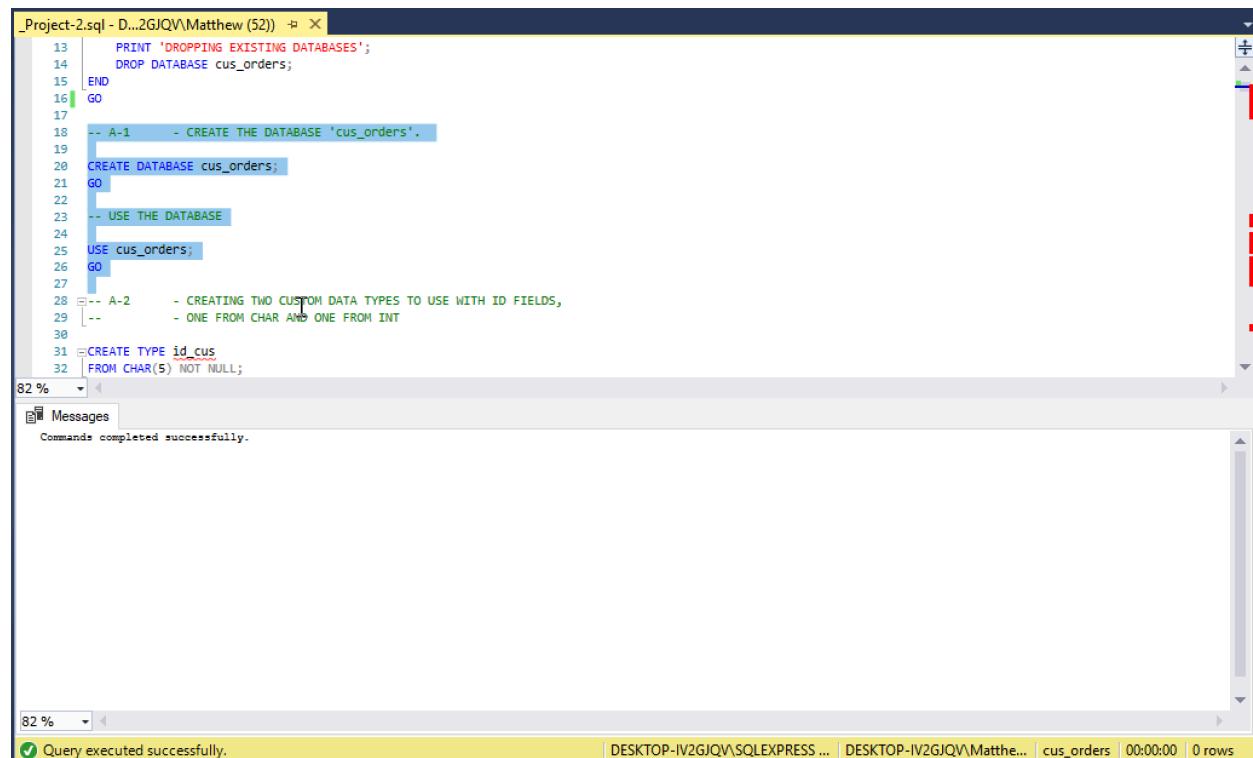
Create a database called **cus_orders**

Code:

```
CREATE DATABASE cus_orders;
GO

USE cus_orders;
GO
```

Completed Successfully:



The screenshot shows a SQL Server Management Studio (SSMS) window. The left pane displays a script named '_Project-2.sql' with lines of T-SQL code. The right pane shows the 'Messages' tab with a green success message: 'Commands completed successfully.' At the bottom, a status bar indicates 'Query executed successfully.' and other connection details.

```
13 PRINT 'DROPPING EXISTING DATABASES';
14 DROP DATABASE cus_orders;
15 END
16 GO

-- A-1      - CREATE THE DATABASE 'cus_orders'.
18 CREATE DATABASE cus_orders;
19 GO

-- USE THE DATABASE
23 USE cus_orders;
24 GO

-- A-2      - CREATING TWO CUSTOM DATA TYPES TO USE WITH ID FIELDS,
28   - ONE FROM CHAR AND ONE FROM INT
29
30
31 CREATE TYPE id_cus
32   FROM CHAR(5) NOT NULL;
```

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:

The screenshot shows a SQL query window titled 'Project-2.sql - D...2GJQ\Matthew (52)'. The code is as follows:

```
22  
23 -- USE THE DATABASE  
24  
25 USE cus_orders;  
26 GO  
27  
28 -- A-2      - CREATING TWO CUSTOM DATA TYPES TO USE WITH ID FIELDS,  
29 --      - ONE FROM CHAR AND ONE FROM INT  
30  
31 CREATE TYPE id_cus  
32     FROM CHAR(5) NOT NULL;  
33  
34 CREATE TYPE id_num  
35     FROM INT NOT NULL;  
36 GO  
37  
38  
39 -- A-3      - CREATE THE TABLES  
40 --      - PLEASE NOTE I HAVE CHANGED THE COLUMN NAMES SLIGHTLY  
41
```

The 'Messages' tab at the bottom shows the output:

```
Commands completed successfully.
```

A status bar at the bottom right indicates:

```
DESKTOP-IV2GJQ\SQLEXPRESS ... | DESKTOP-IV2GJQ\Matthe... | cus_orders | 00:00:00 | 0 rows
```

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:

The screenshot shows a SQL query window in SQL Server Management Studio. The code is a script named 'Project-2.sql' containing SQL commands to create tables and primary/foreign keys. The execution was successful, as indicated by the message 'Commands completed successfully.' in the 'Messages' pane.

```
105 GO
106
107 -- SUPPLIERS TABLE
108 CREATE TABLE suppliers (
109     supplier_id INT IDENTITY,
110     sup_name VARCHAR(40) NOT NULL,
111     sup_address VARCHAR(30),
112     sup_city VARCHAR(20),
113     sup_province CHAR(2)
114 );
115 GO
116
117 -- TITLES TABLE
118 CREATE TABLE titles (
119     title_id CHAR(3) NOT NULL,
120     title_description VARCHAR(35) NOT NULL
121 );
122 GO
123
124 -- A-4      - CREATE THE PRIMARY AND FOREIGN KEYS
```

82 %

Messages

Commands completed successfully.

82 %

Query executed successfully. | DESKTOP-IV2GJQV\SQLEXPRESS ... | DESKTOP-IV2GJQV\Matthe... | cus_orders | 00:00:00 | 0 rows

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:

The screenshot shows the SQL Editor window of SQL Server Management Studio. The code being run is:

```
137 GO  
138  
139 ALTER TABLE products  
140 ADD PRIMARY KEY (product_id);  
141 GO  
142  
143 ALTER TABLE shippers  
144 ADD PRIMARY KEY (shipper_id);  
145 GO  
146  
147 ALTER TABLE suppliers  
148 ADD PRIMARY KEY (supplier_id);  
149 GO  
150  
151 ALTER TABLE titles  
152 ADD PRIMARY KEY (title_id);  
153 GO  
154  
155 -- FOREIGN KEYS  
156
```

The status bar at the bottom indicates "Query executed successfully." and "0 rows".

The screenshot shows the SQL Editor window of SQL Server Management Studio. The code being run is:

```
168 ADD CONSTRAINT FK_orders_shippers FOREIGN KEY (shipper_id)  
169 REFERENCES shippers (shipper_id);  
170 GO  
171  
172 ALTER TABLE order_details  
173 ADD CONSTRAINT FK_orddet_order FOREIGN KEY (order_id)  
174 REFERENCES orders (order_id);  
175 GO  
176  
177 ALTER TABLE order_details  
178 ADD CONSTRAINT FK_orddet_product FOREIGN KEY (product_id)  
179 REFERENCES products (product_id);  
180 GO  
181  
182 ALTER TABLE products  
183 ADD CONSTRAINT FK_product_supplier FOREIGN KEY (supplier_id)  
184 REFERENCES suppliers (supplier_id);  
185 GO  
186  
187
```

The status bar at the bottom indicates "Query executed successfully." and "0 rows".

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 MORE
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:

The screenshot shows a SQL Server Management Studio (SSMS) window. The main pane displays a script named '_Project-2.sql' with lines 198 to 217. The script contains several ALTER TABLE statements with CHECK constraints for tables 'order_details', 'products', and 'suppliers'. The constraints define minimum order quantity (>= 1), minimum reorder point (>= 1), and maximum stock level (<= 150). Line 217 adds a default constraint for the 'sup_province' column. The status bar at the bottom indicates the command was executed successfully with 0 rows affected.

```
198 GO  
199  
200 -- SET DEFAULT ORDER DETAILS QUANTITY TO ONE OR MORE  
201 ALTER TABLE order_details  
202 ADD CONSTRAINT default_qty CHECK (quantity >= 1);  
203 GO  
204  
205 -- SET PRODUCTS DEFAULT REORDER POINT TO ONE OR MORE  
206 ALTER TABLE products  
207 ADD CONSTRAINT default_reorder CHECK (prod_reorder_level >= 1);  
208 GO  
209  
210 -- SET PRODUCT MAX IN STOCK TO 150 OR FEWER  
211 ALTER TABLE products  
212 ADD CONSTRAINT max_in_stock CHECK (prod_qty_in_stock <= 150);  
213 GO  
214  
215 -- SET SUPPLIERS DEFAULT PROVINCE TO BC  
216 ALTER TABLE suppliers  
217 ADD CONSTRAINT default_prov DEFAULT ('BC') FOR sup_province;
```

Messages

Commands completed successfully.

82 %

Query executed successfully. | DESKTOP-IV2GJQV\SQLEXPRESS ... | DESKTOP-IV2GJQV\Matthe... | cus_orders | 00:00:00 | 0 rows

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:

Project-2.sql - D...2GJQV\Matthew (52)* + X

```

280 GO
281
282 BULK INSERT titles
283 FROM 'C:\textfiles\titles.txt' WITH (
284     CODEPAGE = 1252,
285     DATAFILETYPE = 'char',
286     FIELDTERMINATOR = '\t',
287     KEEPNULLS,
288     ROWTERMINATOR = '\n'
289 );
290 GO
291
292
293 --PART B STARTS HERE
294
295 -- B-1      - LIST CUSTOMER ID, NAME, CITY, COUNTRY FROM THE
296 --           - CUSTOMERS TABLE ORDERED BY CUSTOMER ID.
297
298 SELECT
299     'Customer ID' = customer_id,

```

82 %

Messages

```

(91 rows affected)
(1078 rows affected)
(2820 rows affected)
(77 rows affected)
(3 rows affected)
(15 rows affected)
(12 rows affected)

```

82 %

Query executed successfully.

DESKTOP-IV2GJQV\SQLEXPRESS ... DESKTOP-IV2GJQV\Matthe... cus_orders 00:00:00 0 rows

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):



The screenshot shows a Windows-based application window titled 'Results' with a yellow status bar at the bottom. The main area displays a table with 12 rows of data, each containing four columns: Customer ID, Name, City, and Country. The first row is highlighted with a blue selection bar. The status bar at the bottom contains the message 'Query executed successfully.' followed by connection information: DESKTOP-IV2GJQV\SQLEXPRESS ..., DESKTOP-IV2GJQV\Matthe..., cus_orders, 00:00:00, and 91 rows.

	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

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:

The screenshot shows a SQL Server Management Studio window with the following details:

- Title Bar:** DESKTOP-IV2GJQV\S...orders - Diagram_0* | Project-2.sql - D...2GJQV\Matthew (58)*
- Script Area:** The code block above was pasted here. It includes comments explaining the purpose of each section: adding a column 'active' to the 'customers' table with a default value of 1, and listing orders between January 1 and December 31, 2001.
- Messages Area:** Shows the message "Commands completed successfully."
- Status Bar:** Displays "82 %", "Query executed successfully.", and connection information: DESKTOP-IV2GJQV\SQLEXPRESS ... | DESKTOP-IV2GJQV\Matthe... | cus_orders | 00:00:00 | 0 rows

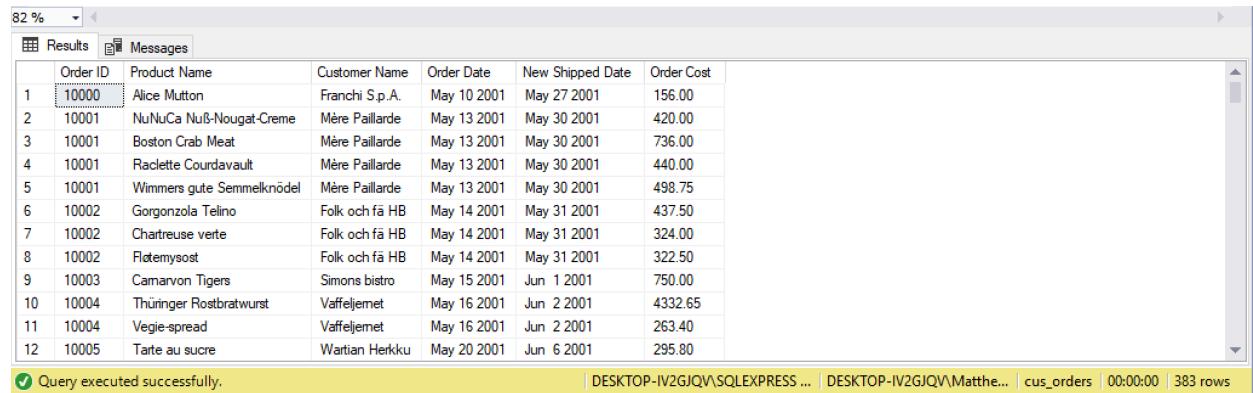
Problem B-3

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):



The screenshot shows a SQL query results window with the following details:

- Query:** 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';
- Results:** The results show 383 rows of data. The columns are Order ID, Product Name, Customer Name, Order Date, New Shipped Date, and Order Cost. The first few rows are:

	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	Flötämysost	Folk och fä HB	May 14 2001	May 31 2001	322.50
9	10003	Camarón Tigers	Simons bistro	May 15 2001	Jun 1 2001	750.00
10	10004	Thüringer Rostbratwurst	Vaffeljemet	May 16 2001	Jun 2 2001	4332.65
11	10004	Vegie-spread	Vaffeljemet	May 16 2001	Jun 2 2001	263.40
12	10005	Tarte au sucre	Wartian Herku	May 20 2001	Jun 6 2001	295.80

Status: Query executed successfully.

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):

	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-Delicatessen	(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):

	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):

	Supplier Name	Product Name	Reorder Level	Qty In Stock
1	Amstrong Company	Queso Cabrales	30	22
2	Cadbury Products Ltd.	Ipoh Coffee	25	17
3	Cadbury Products Ltd.	Røgdede 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):



The screenshot shows a SQL query results window with the following details:

- Query:** 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
- Results:** The results show 1057 rows of data. The columns are Order ID, Name, Contact Name, Shipped Date, and Years Elapsed. The data includes various customer names and their order details, such as Franchi S.p.A., Mère Paillarde, Folk och få HB, Simons bistro, Vaffeljernet, Wartian Herkku, Franchi S.p.A., Morgenstem Gesundkost, Fúria Bacalhau e Frutos do Mar, Seven Seas Imports, Simons bistro, and Wellington Importadora.
- Status Bar:** The status bar at the bottom indicates "Query executed successfully." and shows the connection information: 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):

	Name	Total
1	A	4
2	B	7
3	C	5
4	D	3
5	E	2
6	F	8
7	G	5
8	H	4
9	L	9
10	M	4
11	O	3
12	P	4
13	Q	3
14	R	6
15	T	6
16	V	3
17	W	5

Query executed successfully.

DESKTOP-IV2GJQV\SQLEXPRESS ... DESKTOP-IV2GJQV\Matthe... cus_orders 00:00:00 17 rows

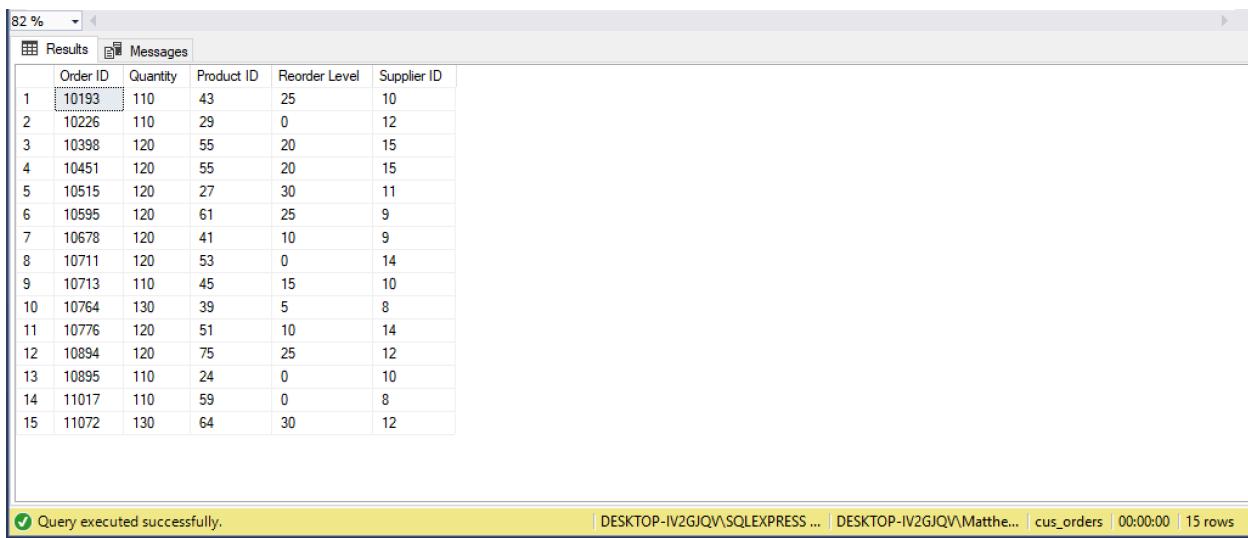
Problem B-9

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):



	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

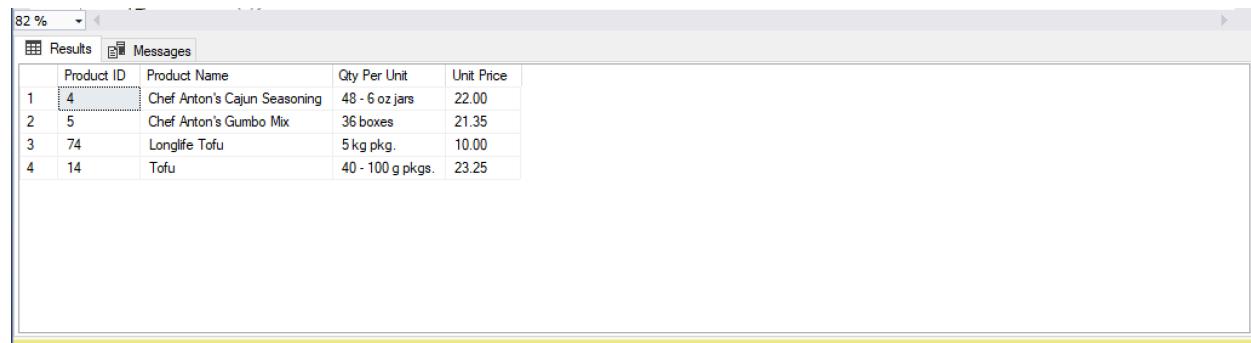
Problem B-10

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):



The screenshot shows a SQL query results window with the following details:

- Query:** 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
- Results:** A table with 4 rows of data:

	Product ID	Product Name	Qty Per Unit	Unit Price
1	4	Chef Anton's Cajun Seasoning	48 - 6 oz jars	22.00
2	5	Chef Anton's Gumbo Mix	36 boxes	21.35
3	74	Longlife Tofu	5 kg pkg.	10.00
4	14	Tofu	40 - 100 g pkgs.	23.25

- Status Bar:** Query executed successfully. | DESKTOP-IV2GJQV\SQLEXPRESS ... | DESKTOP-IV2GJQV\Matthe... | cus_orders | 00:00:00 | 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:

```
Project-2.sql - D...2GJQV\Matthew (52)*  ↗ X
454 -- GO
455
456 ALTER TABLE employee
457     ADD PRIMARY KEY (employee_id);
458 GO
459
460 -- C-3      - LOAD THE EMPLOYEE DATA INTO THE EMPLOYEE TABLE.
461 --          - SET UP THE FOREIGN KEY RELATIONSHIP
462
463 BULK INSERT employee
464 FROM 'C:\textfiles\employee.txt' WITH (
465     CODEPAGE = 1252,
466     DATAFILETYPE = 'char',
467     FIELDTERMINATOR = '\t',
468     KEEPNULLS,
469     ROWTERMINATOR = '\n'
470 );
471 GO
472
473 ALTER TABLE orders
474     ADD CONSTRAINT FK_ord_emp FOREIGN KEY (employee_id)
475     REFERENCES employee (employee_id);
476 GO
477

82 %  ↴
Messages
(9 rows affected)

I

82 %  ↴
Query executed successfully. | DESKTOP-IV2GJQV\SQLEXPRESS ... | DESKTOP-IV2GJQV\Matthe... | cus_orders | 00:00:00 | 0 rows
```

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 screenshot shows a SQL query window with the following content:

```
480  
481 INSERT INTO shippers  
482     VALUES('Quick Express');  
483 GO
```

The status bar at the bottom indicates "(1 row affected)".

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;
```



The screenshot shows a results grid for the SHIPPERS table:

shipper_id	ship_name
1	Speedy Express
2	United Package
3	Federal Shipping
4	Quick Express

The status bar at the bottom indicates "4 rows".

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

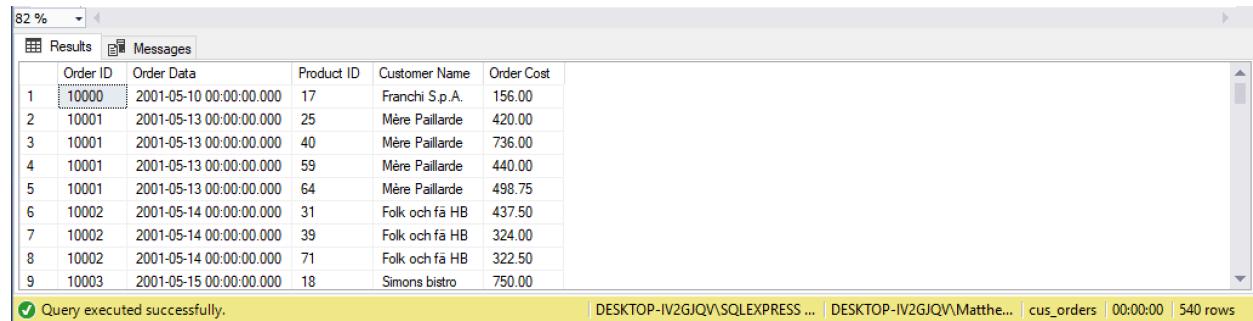
Create a view called **vw_order_cost** to list the cost of the orders. Display the order id and order_date from the orders table, the product id from the products table, the customer name from the customers table, and the order cost. To calculate the cost of the orders, use the formula (order_details.quantity * 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 Date'        = 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):



The screenshot shows a SQL Server Management Studio window with the 'Results' tab selected. The results grid displays 540 rows of data from the view 'vw_order_cost'. The columns are labeled: Order ID, Order Data, Product ID, Customer Name, and Order Cost. The data includes various order details such as order IDs 10000 through 10003, dates ranging from May 10 to May 15, product IDs like 17, 25, 40, etc., customer names like 'Franchi S.p.A.', 'Mère Paillarde', and 'Simons bistro', and order costs ranging from 156.00 to 750.00.

	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

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):

	Employee ID	Name	Birthdate
1	5	Buchanan, Steven	1967.03.04
2	7	King, Robert	1972.05.29
3	9	Dodsworth, Anne	1978.01.27

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):

	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 Taqueria	México D.F.	Mexico	Oct 26 2002
3	10137	ANTON	Antonio Moreno Taqueria	México D.F.	Mexico	Jan 22 2002
4	10142	ANTON	Antonio Moreno Taqueria	México D.F.	Mexico	Jan 08 2002
5	10218	ANTON	Antonio Moreno Taqueria	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	BSEEV	B's Beverages	London	United Kingdom	May 20 2002

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 Orleans Spices Ltd.	4	Chef Anton's Cajun Seasoning
5	2	New Orleans 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

Query executed successfully. | DESKTOP-IV2GJQV\SQLEXPRESS ... | DESKTOP-IV2GJQV\Matthe... | cus_orders | 00:00:00 | 77 rows

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.

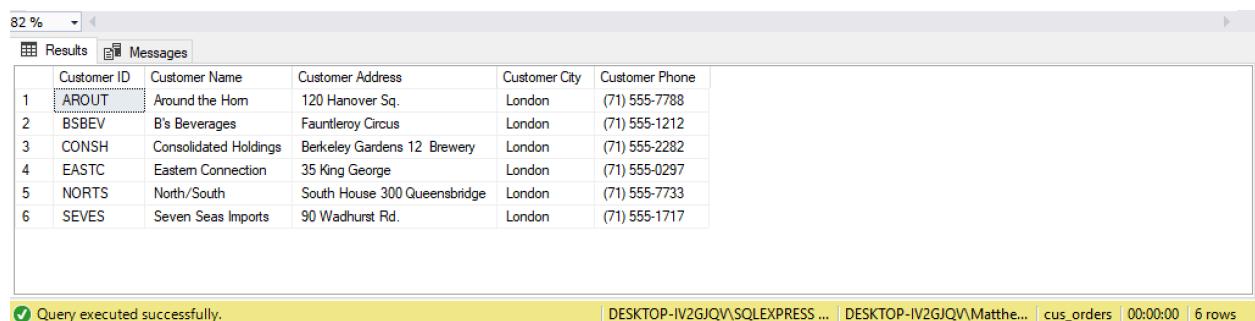
The **city** will be an **input parameter** for the stored procedure. Display the customer id, name, address, city and phone from the customers table. Run the stored procedure displaying customers living in **London**.

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):



A screenshot of the SQL Server Management Studio (SSMS) interface. The title bar shows '82 %'. The window has tabs for 'Results' and 'Messages'. The 'Results' tab is selected and displays a table with six rows of data. The columns are labeled 'Customer ID', 'Customer Name', 'Customer Address', 'Customer City', and 'Customer Phone'. The data shows customers from London. The 'Messages' tab at the bottom shows a green checkmark icon followed by the message 'Query executed successfully.' and other connection details.

	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

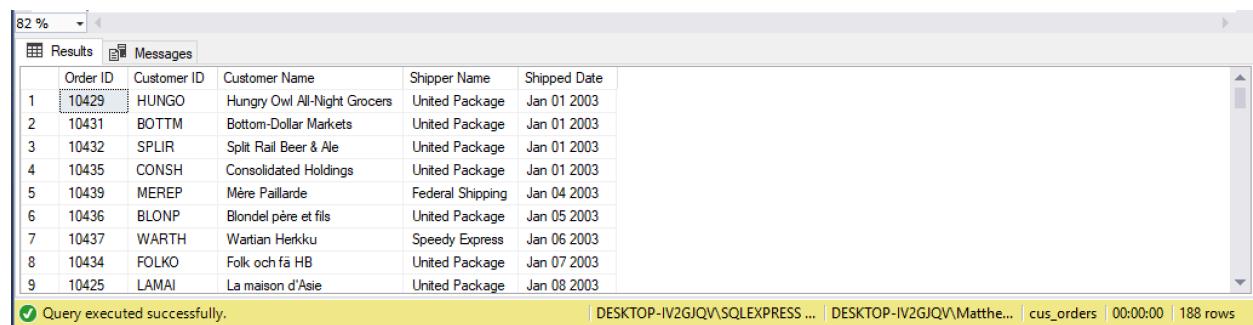
Create a stored procedure called **sp_orders_by_dates** displaying the orders shipped between particular dates. The **start** and **end** date will be **input parameters** for the stored procedure. Display the order id, customer id, and shipped date from the orders table, the customer name from the customer 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):



	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

Problem D-3

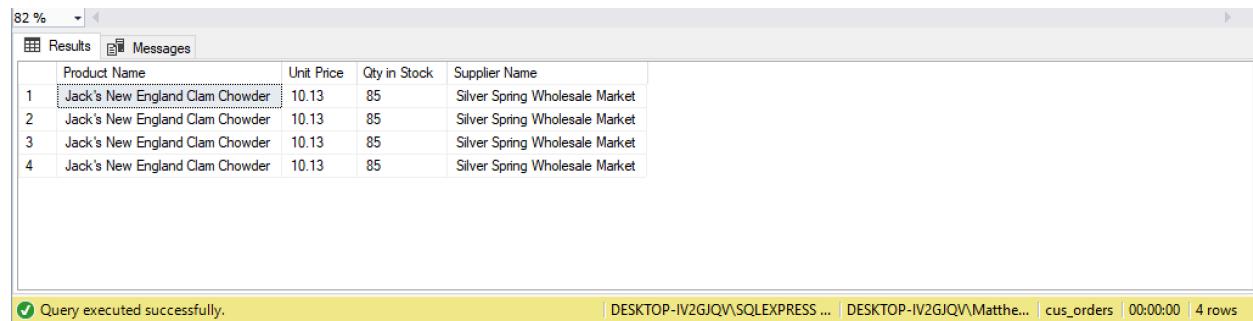
Create a stored procedure called **sp_product_listing** listing a specified product ordered during a specified month and year. The **product** and the **month** and **year** will be **input parameters** for the stored procedure. Display the product name, unit price, and quantity in stock from the products 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):



The screenshot shows the SQL Server Management Studio interface with the 'Results' tab selected. The query window displays the results of the stored procedure execution. The results are presented in a table with four columns: Product Name, Unit Price, Qty in Stock, and Supplier Name. There are four rows of data, all corresponding to the product 'Jack's New England Clam Chowder' with a unit price of 10.13, quantity in stock of 85, and supplier 'Silver Spring Wholesale Market'. The status bar at the bottom indicates 'Query executed successfully.' and '4 rows'.

	Product Name	Unit Price	Qty in Stock	Supplier Name
1	Jack's New England Clam Chowder	10.13	85	Silver Spring Wholesale Market
2	Jack's New England Clam Chowder	10.13	85	Silver Spring Wholesale Market
3	Jack's New England Clam Chowder	10.13	85	Silver Spring Wholesale Market
4	Jack's New England Clam Chowder	10.13	85	Silver Spring Wholesale Market

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):

Product ID	Product Name	Quantity Being Deleted	Quantity In Stock After Deletion
1 25	NuNuCa Nuß-Nougat-Creme	30	106

Query executed successfully. | DESKTOP-IV2GJQV\SQLEXPRESS ... | DESKTOP-IV2GJQV\Matthe... | cus_orders | 00:00:00 | 1 rows

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;
```



The screenshot shows the SQL Server Management Studio interface. The top window is titled 'Messages' and displays an error message: 'NOT ENOUGH STOCK. YOU NEED 7 MORE. Msg 3609, Level 16, State 1, Line 867 The transaction ended in the trigger. The batch has been aborted.' Below it, another window shows the status bar with 'Query completed with errors.' and '0 rows'.

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



The screenshot shows the SQL Server Management Studio interface. The top window is titled 'Results' and displays a table with one row: Name (Uncle Bob's Organic Dried Pears) and Updated Qty (3). Below it, another window shows the status bar with 'Query executed successfully.' and '1 rows'.

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

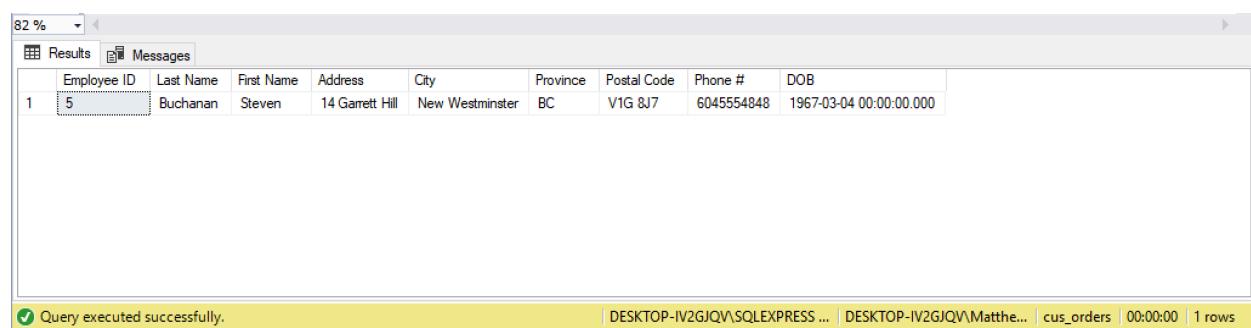
Create a stored procedure called **sp_employee_information** to display the employee information for a particular employee. The **employee id** will be an **input parameter** for the stored procedure. Run the stored procedure displaying information for employee id of 5.

Code:

```
CREATE PROCEDURE sp_employee_information
(
    @checkEmp int
)
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):



	Employee ID	Last Name	First Name	Address	City	Province	Postal Code	Phone #	DOB
1	5	Buchanan	Steven	14 Garrett Hill	New Westminster	BC	V1G 8J7	6045554848	1967-03-04 00:00:00.000

Query executed successfully. | DESKTOP-IV2GJQV\SQLEXPRESS ... | DESKTOP-IV2GJQV\Matthe... | cus_orders | 00:00:00 | 1 rows

Problem D-9

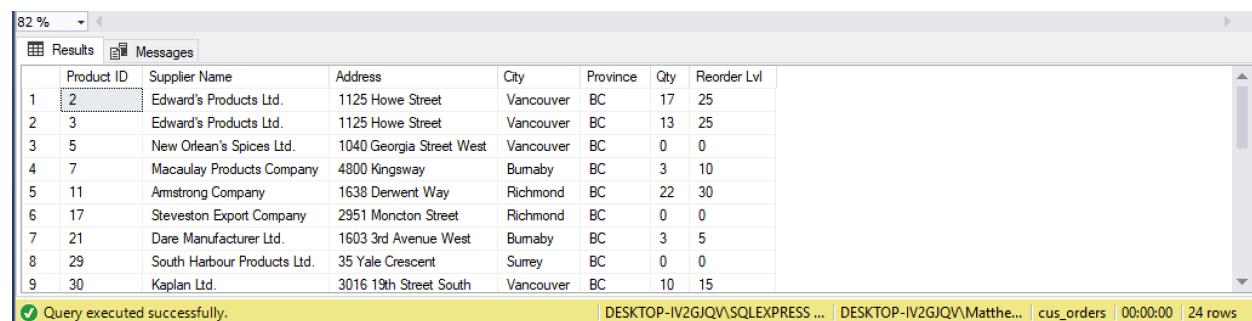
Create a stored procedure called **sp_reorder_qty** to show when the reorder level subtracted from the quantity in stock is less than a specified value. The **unit** value will be an **input parameter** for the stored procedure. Display the product id, quantity in stock, and reorder level from the products 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):



The screenshot shows the SQL Server Management Studio (SSMS) interface with the 'Results' tab selected. The query has executed successfully, returning 24 rows of data. The data is presented in a table with the following columns: Product ID, Supplier Name, Address, City, Province, Qty, and Reorder Lvl. The rows are numbered 1 through 9, corresponding to the entries in the products table.

	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 Orleans' Spices Ltd.	1040 Georgia Street West	Vancouver	BC	0	0
4	7	Macaulay Products Company	4800 Kingsway	Burnaby	BC	3	10
5	11	Armstrong 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

Problem D-10

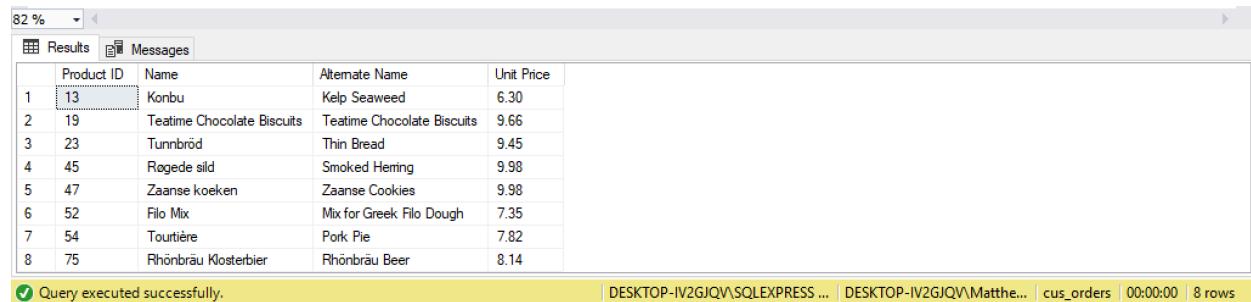
Create a stored procedure called **sp_unit_prices** for the product table where the **unit price is between particular values**. The **two unit prices** will be **input parameters** for the stored procedure. Display the product id, product name, alternate name, and unit price from the products table. 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):



	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	Rogede 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-IV2GJQ\SQLEXPRESS ... | DESKTOP-IV2GJQ\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:

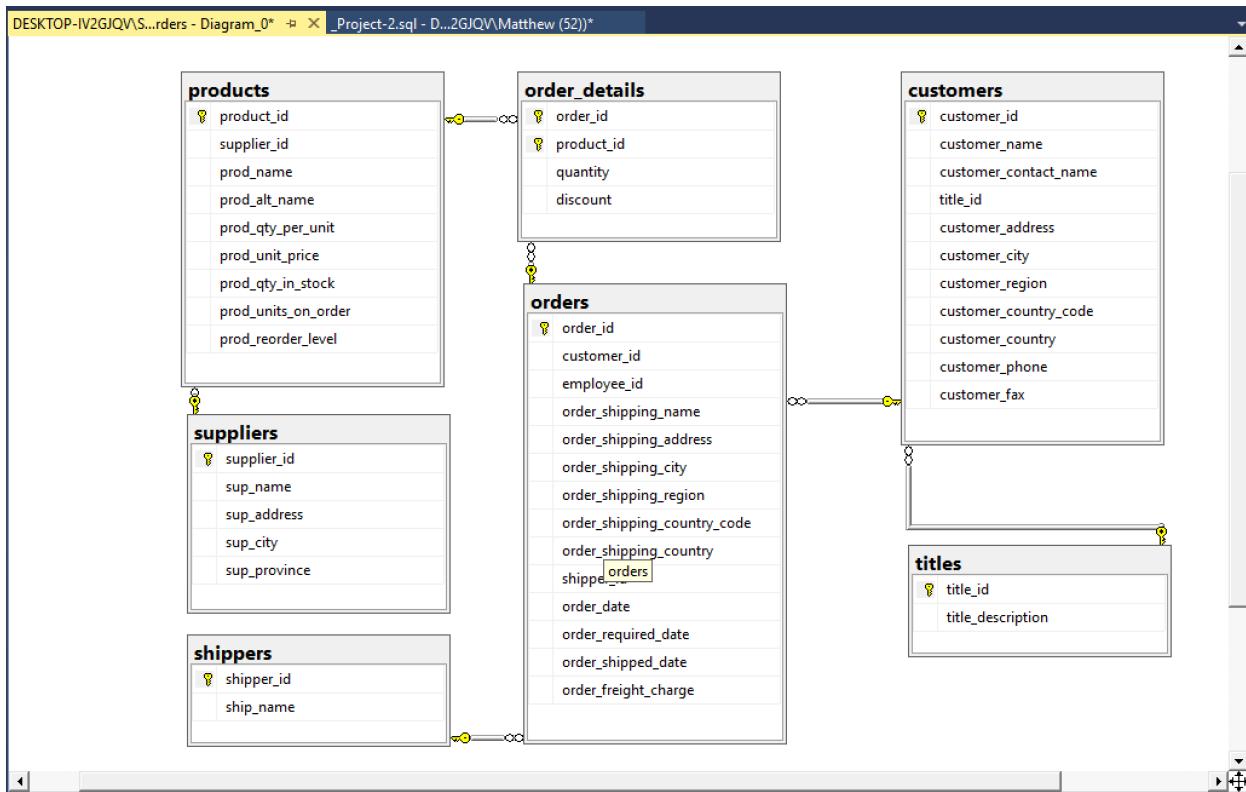
Employee ID	Name	Birthdate
5	Buchanan, Steven	1967.03.04
7	King, Robert	1972.05.29
9	Dodsworth, Anne	1978.01.27

Appendix I

Concerning Database Diagrams

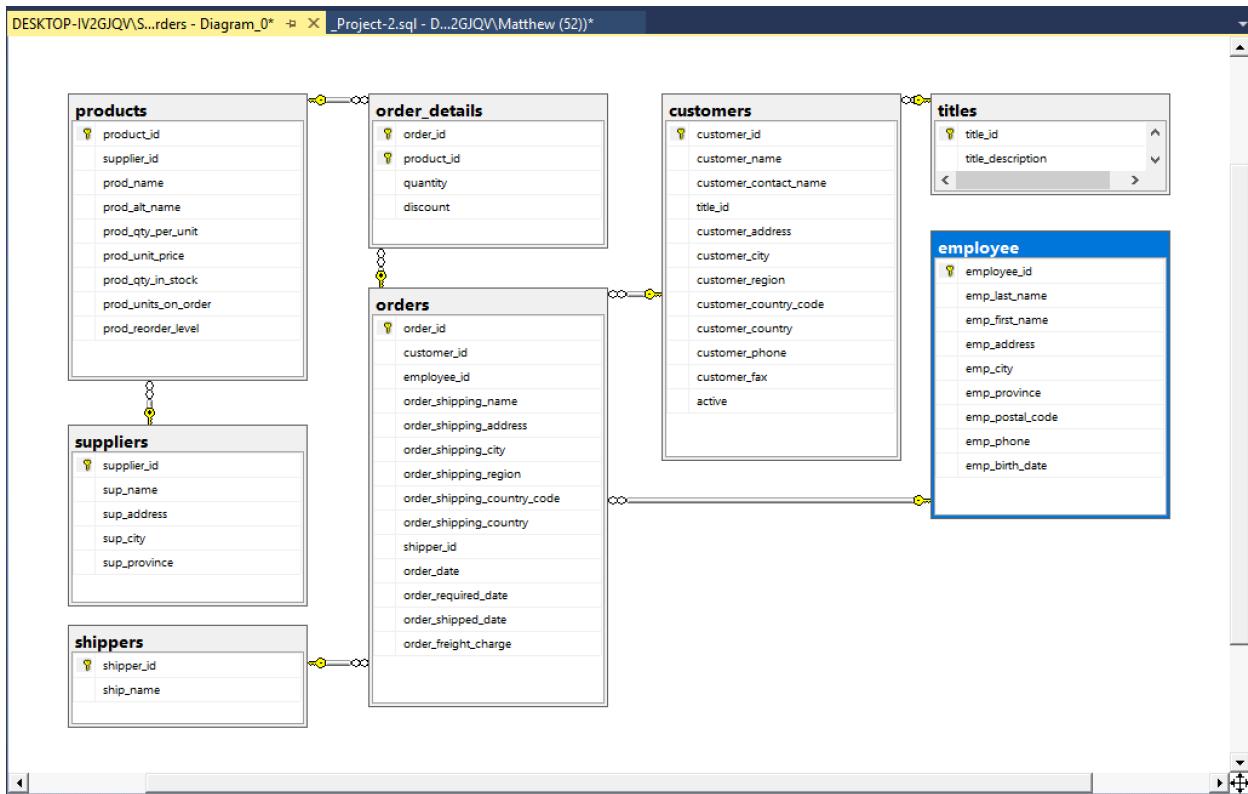
Database Diagram 1

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.

```
1  /*
2   By:          Matthew Simpson Student ID: A00820997
3   For:         COMP 1630           Instructor: Mark Bacchus
4   Date:        Fall 2019
5 */
6
7 USE master;
8
9 /*
10    CHECK FOR EXISTING DATABASE AND DROP IT IF IT EXISTS.
11    COPIED THIS FROM THE PUBS DATABASE SCRIPT.
12    SEEMED LIKE A GOOD IDEA FOR RE-RUNNING THE SCRIPT
13    OVER AND OVER.
14 */
15
16 IF EXISTS (
17     SELECT *
18     FROM sysdatabases
19     WHERE name = 'cus_orders'
20 )
21 BEGIN
22     PRINT 'DROPPING EXISTING DATABASES';
23     DROP DATABASE cus_orders;
24 END
25 GO
26
27 /*
28    A-1: CREATE THE DATABASE 'cus_orders'.
29 */
30
31 CREATE DATABASE cus_orders;
32 GO
33
34 USE cus_orders;
35 GO
36
37 /*
38    A-2: CREATING TWO CUSTOM DATA TYPES TO USE WITH
39    ID FIELDS, ONE FROM CHAR AND ONE FROM INT
40 */
41
42 CREATE TYPE id_cus
43     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
```

```
87 );
88 GO
89
90 -- ORDER_DETAILS TABLE
91 CREATE TABLE order_details (
92     order_id id_num,
93     product_id id_num,
94     quantity INT NOT NULL,
95     discount FLOAT NOT NULL
96 );
97 GO
98
99 -- PRODUCTS TABLE
100 CREATE TABLE products (
101     product_id id_num,
102     supplier_id INT NOT NULL,
103     prod_name VARCHAR(40),
104     prod_alt_name VARCHAR(40),
105     prod_qty_per_unit VARCHAR(25),
106     prod_unit_price MONEY,
107     prod_qty_in_stock INT,
108     prod_units_on_order INT,
109     prod_reorder_level INT
110 );
111 GO
112
113 -- SHIPPERS TABLE
114 CREATE TABLE shippers (
115     shipper_id INT IDENTITY,
116     ship_name VARCHAR(20) NOT NULL
117 );
118 GO
119
120 -- SUPPLIERS TABLE
121 CREATE TABLE suppliers (
122     supplier_id INT IDENTITY,
123     sup_name VARCHAR(40) NOT NULL,
124     sup_address VARCHAR(30),
125     sup_city VARCHAR(20),
126     sup_province CHAR(2)
127 );
128 GO
129
```

```
130 -- TITLES TABLE
131 CREATE TABLE titles (
132     title_id CHAR(3) NOT NULL,
133     title_description VARCHAR(35) NOT NULL
134 );
135 GO
136
137 /*
138     A-4: CREATE THE PRIMARY AND FOREIGN KEYS
139 */
140
141 -- PRIMARY KEYS
142
143 ALTER TABLE customers
144     ADD PRIMARY KEY (customer_id);
145 GO
146
147 ALTER TABLE orders
148     ADD PRIMARY KEY (order_id);
149 GO
150
151 ALTER TABLE order_details
152     ADD PRIMARY KEY (order_id, product_id);
153 GO
154
155 ALTER TABLE products
156     ADD PRIMARY KEY (product_id);
157 GO
158
159 ALTER TABLE shippers
160     ADD PRIMARY KEY (shipper_id);
161 GO
162
163 ALTER TABLE suppliers
164     ADD PRIMARY KEY (supplier_id);
165 GO
166
167 ALTER TABLE titles
168     ADD PRIMARY KEY (title_id);
169 GO
170
171 -- FOREIGN KEYS
172
```

```
173 ALTER TABLE customers
174     ADD CONSTRAINT FK_customer_title FOREIGN KEY (title_id)
175     REFERENCES titles (title_id);
176 GO
177
178 ALTER TABLE orders
179     ADD CONSTRAINT FK_orders_customer FOREIGN KEY (customer_id)
180     REFERENCES customers (customer_id);
181 GO
182
183 ALTER TABLE orders
184     ADD CONSTRAINT FK_orders_shippers FOREIGN KEY (shipper_id)
185     REFERENCES shippers (shipper_id);
186 GO
187
188 ALTER TABLE order_details
189     ADD CONSTRAINT FK_orddet_order FOREIGN KEY (order_id)
190     REFERENCES orders (order_id);
191 GO
192
193 ALTER TABLE order_details
194     ADD CONSTRAINT FK_orddet_product FOREIGN KEY (product_id)
195     REFERENCES products (product_id);
196 GO
197
198 ALTER TABLE products
199     ADD CONSTRAINT FK_product_supplier FOREIGN KEY (supplier_id)
200     REFERENCES suppliers (supplier_id);
201 GO
202
203 /*
204     A-5: ADD DEFAULT CONSTRAINTS
205 */
206
207 -- SET CUSTOMERS DEFAULT COUNTRY AS "CANADA"
208 ALTER TABLE customers
209     ADD CONSTRAINT default_country DEFAULT ('Canada') FOR
... customer_country;
210 GO
211
212 -- SET ORDERS DEFAULT REQUIRED DATE AS TODAY + 10 DAYS.
213 ALTER TABLE orders
214     ADD CONSTRAINT default_req_date DEFAULT (DATEADD(DAY, 10,
```

```
214... GETDATE())) FOR order_required_date;
215 GO
216
217 -- SET DEFAULT ORDER DETAILS QUANTITY TO ONE OR MORE
218 ALTER TABLE order_details
219     ADD CONSTRAINT default_qty CHECK (quantity >= 1);
220 GO
221
222 -- SET PRODUCTS DEFAULT REORDER POINT TO ONE OR MORE
223 ALTER TABLE products
224     ADD CONSTRAINT default_reorder CHECK (prod_reorder_level >=
225 ... 1);
226 GO
227
228 -- SET PRODUCT MAX IN STOCK TO 150 OR FEWER
229 ALTER TABLE products
230     ADD CONSTRAINT max_in_stock CHECK (prod_qty_in_stock <=
231 ... 150);
232 GO
233
234 -- SET SUPPLIERS DEFAULT PROVINCE TO BC
235 ALTER TABLE suppliers
236     ADD CONSTRAINT default_prov DEFAULT ('BC') FOR sup_province;
237 GO
238 /*
239     A-6: BULK LOAD THE DATA FOR CREATED TABLES
240 */
241
242 -- CUSTOMERS
243 BULK INSERT customers
244 FROM 'C:\textfiles\customers.txt' WITH (
245     CODEPAGE = 1252,
246     DATAFILETYPE = 'char',
247     FIELDTERMINATOR = '\t',
248     KEEPNULLS,
249     ROWTERMINATOR = '\n'
250 );
251 GO
252
253 -- ORDERS
254 BULK INSERT orders
255 FROM 'C:\textfiles\orders.txt' WITH (
```

```
255     CODEPAGE = 1252,  
256     DATAFILETYPE = 'char',  
257     FIELDTERMINATOR = '\t',  
258     KEEPNULLS,  
259     ROWTERMINATOR = '\n'  
260 );  
261 GO  
262  
263 -- ORDER DETAILS  
264 BULK INSERT order_details  
265 FROM 'C:\textfiles\order_details.txt' WITH (  
266     CODEPAGE = 1252,  
267     DATAFILETYPE = 'char',  
268     FIELDTERMINATOR = '\t',  
269     KEEPNULLS,  
270     ROWTERMINATOR = '\n'  
271 );  
272 GO  
273  
274 -- PRODUCTS  
275 BULK INSERT products  
276 FROM 'C:\textfiles\products.txt' WITH (  
277     CODEPAGE = 1252,  
278     DATAFILETYPE = 'char',  
279     FIELDTERMINATOR = '\t',  
280     KEEPNULLS,  
281     ROWTERMINATOR = '\n'  
282 );  
283 GO  
284  
285 -- SHIPPERS  
286 BULK INSERT shippers  
287 FROM 'C:\textfiles\shippers.txt' WITH (  
288     CODEPAGE = 1252,  
289     DATAFILETYPE = 'char',  
290     FIELDTERMINATOR = '\t',  
291     KEEPNULLS,  
292     ROWTERMINATOR = '\n'  
293 );  
294 GO  
295  
296 -- SUPPLIERS  
297 BULK INSERT suppliers
```

```
298 FROM 'C:\textfiles\suppliers.txt' WITH (
299     CODEPAGE = 1252,
300     DATAFILETYPE = 'char',
301     FIELDTERMINATOR = '\t',
302     KEEPNULLS,
303     ROWTERMINATOR = '\n'
304 );
305 GO
306
307 -- TITLES
308 BULK INSERT titles
309 FROM 'C:\textfiles\titles.txt' WITH (
310     CODEPAGE = 1252,
311     DATAFILETYPE = 'char',
312     FIELDTERMINATOR = '\t',
313     KEEPNULLS,
314     ROWTERMINATOR = '\n'
315 );
316 GO
317
318 /*
319     B-1: LIST CUSTOMER ID, NAME, CITY, COUNTRY FROM
320     THE CUSTOMERS TABLE ORDERED BY CUSTOMER ID.
321 */
322
323 SELECT
324     'Customer ID'      = customer_id,
325     'Name'              = customer_name,
326     'City'              = customer_city,
327     'Country'           = customer_country
328 FROM
329     customers
330 ORDER BY
331     customer_id;
332 GO
333
334 /*
335     B-2: ADD NEW COLUMN 'ACTIVE' TO CUSTOMERS TABLE.
336     ONLY VALID VALUES 1 OR 0.  DEFAULT VALUE 1.
337 */
338
339 ALTER TABLE customers
340 ADD active int NOT NULL
```

```

341 CONSTRAINT default_active DEFAULT 1 WITH VALUES
342 CONSTRAINT one_or_zero CHECK (active = 1 OR active = 0);
343 GO
344
345 /*
346     B-3: LIST ORDERS BETWEEN JANUARY 1 AND DECEMBER 31
347     2001. DISPLAY ORDER ID, PRODUCT NAME, CUSTOMER NAME,
348     NEW ORDER SHIPPED DATE (ORDER DATE + 17 DAYS) AND
349     ORDER COST (QUANTITY * UNIT PRICE)
350 */
351
352 SELECT
353     'Order ID'          = orders.order_id,
354     'Product Name'      = products.prod_name,
355     'Customer Name'     = customers.customer_name,
356     'Order Date'         = CONVERT(CHAR(12), orders.order_date,
357 ... 109),
358     'New Shipped Date'   = CONVERT(CHAR(12), DATEADD(DAY, 17,
359 ... orders.order_date)),
360     'Order Cost'          = order_details.quantity *
361 ... products.prod_unit_price
362 FROM
363     order_details
364 INNER JOIN orders ON order_details.order_id=orders.order_id
365 INNER JOIN products ON
366 ... order_details.product_id=products.product_id
367 INNER JOIN customers ON orders.customer_id=customers.customer_id
368 WHERE orders.order_date BETWEEN 'January 1 2001' AND 'December
369 ... 31 2001';
370 GO
371
372 /*
373     B-4: LIST ORDERS THAT HAVE NOT SHIPPED. ORDER BY SHIPPED
374     DATE.
375 */
376
377 SELECT
378     'Customer ID'        = customers.customer_id,
379     'Name'                = customers.customer_name,
380     'Phone'               = customers.customer_phone,
381     'Order ID'             = orders.order_id,
382     'Order Date'           = CONVERT(CHAR(12), orders.order_date, 109)
383 FROM

```

```
378     orders
379 INNER JOIN customers ON orders.customer_id=customers.customer_id
380 WHERE orders.order shipped_date IS NULL;
381
382 /*
383     B-5: LIST ALL CUSTOMERS WHERE REGION IS NULL
384 */
385
386 SELECT
387     'Customer ID'      = customer_id,
388     'Name'              = customer_name,
389     'City'               = customer_city,
390     'Description'       = title_description
391 FROM
392     customers
393 INNER JOIN titles ON customers.title_id=titles.title_id
394 WHERE customer_region IS NULL;
395 GO
396
397 /*
398     B-6: LIST ALL PRODUCTS WHERE REORDER POINT
399     IS HIGHER THAN QUANTITY IN STOCK
400 */
401
402 SELECT
403     'Supplier Name'    = sup_name,
404     'Product Name'     = prod_name,
405     'Reorder Level'    = prod_reorder_level,
406     'Qty In Stock'     = prod_qty_in_stock
407 FROM
408     products
409 INNER JOIN suppliers ON
... products.supplier_id=suppliers.supplier_id
410 WHERE prod_reorder_level > prod_qty_in_stock
411 ORDER BY [Supplier Name];
412 GO
413
414 /*
415     B-7: CALCULATE LENGTH IN YEARS FROM JANUARY 1 2008
416     TO WHEN AN ORDER WAS SHIPPED WHERE THE SHIPPED DATE
417     IS NOT NULL.
418 */
419
```

```

420 SELECT
421     'Order ID'      = orders.order_id,
422     'Name'          = customers.customer_name,
423     'Contact Name' = customers.customer_contact_name,
424     'Shipped Date' = CONVERT(CHAR(12), orders.order_date, 109),
425     'Years Elapsed' = DATEDIFF(year, orders.order_date, 'January
... 1 2008')
426 FROM
427     orders
428 INNER JOIN customers ON orders.customer_id=customers.customer_id
429 WHERE orders.order_shipped_date IS NOT NULL;
430 GO
431
432 /*
433     B-8: LIST NUMBER OF CUSTOMERS WITH NAMES BEGINNING
434     WITH EACH LETTER OF THE ALPHABET.
435     IGNORE THE LETTER S.
436     IGNORE LETTERS WITH FEWER THAN 2 ENTRIES.
437 */
438
439 SELECT
440     'Name'          = SUBSTRING(customers.customer_name, 1, 1),
441     'Total'          = count(*)
442 FROM
443     customers
444 GROUP BY SUBSTRING(customers.customer_name, 1, 1)
445 HAVING count(*) >= 2
446 AND SUBSTRING(customers.customer_name,1,1) != 's';
447
448 /*
449     B-9: LIST ORDER DETAILS FOR ORDERS WHERE QUANTITY
450     IS GREATER THAN 100 ORDER RESULT BY ORDER ID
451 */
452
453
454 SELECT
455     'Order ID'      = order_details.order_id,
456     'Quantity'       = order_details.quantity,
457     'Product ID'    = products.product_id,
458     'Reorder Level' = products.prod_reorder_level,
459     'Supplier ID'   = products.supplier_id
460 FROM
461     order_details

```

```
462 INNER JOIN products ON
463 ... order_details.product_id=products.product_id
464 WHERE order_details.quantity > 100
465 ORDER BY order_details.order_id;
466 GO
467 /*
468     B-10- LIST ALL PRODUCTS WITH SUBSTRING 'TOFU' OR
469     'CHEF' IN THE NAME ORDER BY PRODUCT NAME
470 */
471
472 SELECT
473     'Product ID'      = products.product_id,
474     'Product Name'    = products.prod_name,
475     'Qty Per Unit'   = products.prod_qty_per_unit,
476     'Unit Price'      = products.prod_unit_price
477 FROM
478     products
479 WHERE
480     products.prod_name LIKE '%chef%'
481     OR products.prod_name LIKE '%tofu%'
482 ORDER BY
483     products.prod_name;
484 GO
485
486 /*
487     C-1, C-2: CREATE THE EMPLOYEE TABLE AND SET THE
488     EMPLOYEE ID AS THE PRIMARY KEY.
489 */
490
491 CREATE TABLE employee (
492     employee_id INT NOT NULL,
493     emp_last_name varchar(30) NOT NULL,
494     emp_first_name varchar(15) NOT NULL,
495     emp_address varchar(30),
496     emp_city varchar(20),
497     emp_province char(2),
498     emp_postal_code varchar(7),
499     emp_phone varchar(10),
500     emp_birth_date datetime NOT NULL
501 );
502 GO
503
```

```
504 ALTER TABLE employee
505     ADD PRIMARY KEY (employee_id);
506 GO
507
508 /*
509     C-3: LOAD THE EMPLOYEE DATA INTO THE EMPLOYEE TABLE.
510     SET UP THE FOREIGN KEY RELATIONSHIP
511 */
512
513 BULK INSERT employee
514     FROM 'C:\textfiles\employee.txt' WITH (
515         CODEPAGE = 1252,
516         DATAFILETYPE = 'char',
517         FIELDTERMINATOR = '\t',
518         KEEPNULLS,
519         ROWTERMINATOR = '\n'
520     );
521 GO
522
523 ALTER TABLE orders
524     ADD CONSTRAINT FK_ord_emp FOREIGN KEY (employee_id)
525     REFERENCES employee (employee_id);
526 GO
527
528 /*
529     C-4: INSERT THE NEW SHIPPER 'QUICK EXPRESS' INTO THE
530     SHIPPERS TABLE. SINCE SHIPPERS HAS AN IDENTITY COLUMN,
531     EVERYTHING ELSE SHOULD BE TAKEN CARE OF AUTOMAGICALLY
532 */
533
534 INSERT INTO shippers
535     VALUES('Quick Express');
536 GO
537
538 SELECT * FROM shippers;
539
540 /*
541     C-5: INCREASE THE PRICE OF ALL PRODUCTS WITH CURRENT
542     PRICE BETWEEN 5$ AND 10$ BY 5%
543 */
544
545 UPDATE
546     products
```

```
547 SET products.prod_unit_price = ROUND(products.prod_unit_price *
... 1.05, 2)
548 WHERE products.prod_unit_price >= 5 AND products.prod_unit_price
... <= 10;
549 GO
550
551 /*
552     C-6: UPDATE THE FAX VALUE TO 'UNKNOWN' FOR ALL
553     CUSTOMERS WHO HAVE A NULL FAX VALUE.
554 */
555
556 UPDATE
557     customers
558 SET customers.customer_fax = 'UNKNOWN'
559 WHERE customers.customer_fax IS NULL;
560 GO
561
562 /*
563     C-7a: CREATE VIEW VW_ORDER_COST TO LIST THE COST OF ORDERS.
564 */
565
566 CREATE VIEW vw_order_cost
567 AS
568 SELECT
569     'Order ID'      = orders.order_id,
570     'Order Date'    = orders.order_date,
571     'Product ID'    = products.product_id,
572     'Customer Name' = customers.customer_name,
573     'Order Cost'    = (order_details.quantity *
... products.prod_unit_price)
574 FROM order_details
575 INNER JOIN orders ON order_details.order_id=orders.order_id
576 INNER JOIN products ON
... order_details.product_id=products.product_id
577 INNER JOIN customers ON
... customers.customer_id=orders.customer_id;
578 GO
579
580 /*
581     C-7b: RUN VW_ORDER_COST FOR ORDER IDS BETWEEN 10000 AND
... 10200
582 */
583
```

```
584 |SELECT *
585 |FROM vw_order_cost
586 |WHERE vw_order_cost.[Order ID] BETWEEN 10000 AND 10200;
587 |GO
588 |
589 |/*
590 |    C-8a: CREATE VIEW VW_LIST_EMPLOYEES TO LIST EMPLOYEE
591 |    ...
592 |    THIS QUESTION DESCRIPTION ASKS FOR ALL THE EMPLOYEE COLUMNS
593 |    TO BE LISTED, BUT THEN ALSO FOR FORMATTED OUTPUT FROM THE
594 |    VIEW. SEE CHALLENGES SECTION
595 */
596 |
597 |CREATE VIEW vw_list_employees
598 |AS
599 |SELECT
600 |    'Employee ID'      = employee.employee_id,
601 |    'Name'              = (employee.emp_last_name + ', ' +
... employee.emp_first_name),
602 |    'Birthdate'         = CONVERT(char(12), employee.emp_birth_date,
... 102)
603 |FROM employee;
604 |GO
605 |
606 |/*
607 |    C-8b RUN VW_LIST_EMPLOYEES FOR EMPLOYEE IDS 5, 7, 9
608 */
609 |
610 |SELECT *
611 |FROM vw_list_employees
612 |WHERE [Employee ID] = 5
613 |    OR [Employee ID] = 7
614 |    OR [Employee ID] = 9;
615 |GO
616 |
617 |/*
618 |    C-9a: CREATE VIEW VW_ALL_ORDERS,
619 |    FORMAT SHIPPED DATE MON DD YYYY
620 */
621 |
622 |CREATE VIEW vw_all_orders
623 |AS
```

```

624 SELECT
625   'Order ID'      = orders.order_id,
626   'Customer ID'  = customers.customer_id,
627   'Customer Name' = customers.customer_name,
628   'City'          = customers.customer_city,
629   'Country'       = customers.customer_country,
630   'Shipped Date' = orders.order_shipped_date
631 FROM orders
632 INNER JOIN customers ON
...  orders.customer_id=customers.customer_id;
633 GO
634
635 /*
636   C-9b: RUN VW_ALL_ORDERS FOR ALL ORDERS BETWEEN
637   JAN 1 2002 AND DEC 31 2002
638 */
639
640 SELECT
641   vw_all_orders.[Order ID],
642   vw_all_orders.[Customer ID],
643   vw_all_orders.[Customer Name],
644   vw_all_orders.[City],
645   vw_all_orders.[Country],
646   format(vw_all_orders.[Shipped Date], 'MMM dd yyyy') AS
... 'Shipped Date'
647 FROM vw_all_orders
648 WHERE [Shipped Date] BETWEEN 'Jan 1 2002' AND 'Dec 31 2002'
649 ORDER BY [Customer Name], [Country];
650 GO
651
652 /*
653   C-10a: CREATE VW_SUPPLIERS_SHIPPED DO DISPLAY
654   DETAILS OF ORDERS SHIPPED.
655 */
656
657 CREATE VIEW vw_suppliers_shipped
658 AS
659 SELECT
660   'Supplier ID'    = suppliers.supplier_id,
661   'Supplier Name'  = suppliers.sup_name,
662   'Product ID'     = products.product_id,
663   'Product Name'   = products.prod_name
664 FROM products

```

```
665 INNER JOIN suppliers ON
... products.supplier_id=suppliers.supplier_id;
666 GO
667
668 /*
669     C-10b: RUN VW_SUPPLIERS_SHIPPED
670 */
671
672 SELECT *
673 FROM vw_suppliers_shipped;
674 GO
675
676 /*
677     D-1a: CREATE STORED PROCEDURE SP_CUSTOMER_CITY TO LIST
678 CUSTOMERS BY SELECTED CITY.
... @CHECKCITY IS A STORED VARIABLE, THE CITY TO BE LISTED.
679 */
680
681 CREATE PROCEDURE sp_customer_city
682 (
683     @checkcity  varchar(20)
684 )
685 AS
686     SELECT
687         'Customer ID'      = customers.customer_id,
688         'Customer Name'    = customers.customer_name,
689         'Customer Address' = customers.customer_address,
690         'Customer City'   = customers.customer_city,
691         'Customer Phone'   = customers.customer_phone
692     FROM customers
693     WHERE customers.customer_city = @checkcity;
694 GO
695
696 /*
697     D-1b: RUN SP_CUSTOMER_CITY FOR THE CITY 'LONDON'
698 */
699
700 EXECUTE sp_customer_city 'London';
701 GO
702
703
704 /*
705     D-2a: CREATE SP_ORDERS_BY_DATES TO DISPLAY ORDERS
```

```
706      SHIPPED BETWEEN SPECIFIED DATES. @STARTDATE AND  
707      @ENDDATE ARE STORED VARIABLES, THE DATES TO CHECK  
708      BETWEEN.  
709  */  
710  
711 CREATE PROCEDURE sp_orders_by_dates  
712 (  
713     @startdate    datetime,  
714     @enddate      datetime  
715 )  
716 AS  
717 SELECT  
718     'Order ID'          = orders.order_id,  
719     'Customer ID'       = orders.customer_id,  
720     'Customer Name'     = customers.customer_name,  
721     'Shipper Name'       = shippers.ship_name,  
722     'Shipped Date'       =  
... FORMAT(orders.order_shipped_date, 'MMM dd yyyy')  
723   FROM orders  
724   INNER JOIN customers ON  
... orders.customer_id=customers.customer_id  
725   INNER JOIN shippers ON orders.shipper_id=shippers.shipper_id  
726   WHERE orders.order_shipped_date BETWEEN @startdate AND  
... @enddate  
    ORDER BY orders.order_shipped_date;  
727 GO  
729  
730 /*  
731      D-2b: RUN SP_ORDERS_BY_DATES FOR JAN 1 2003 THROUGH JUN 30  
... 2003  
732 */  
733  
734 EXECUTE sp_orders_by_dates 'January 1 2003', 'June 30 2003';  
735 GO  
736  
737 /*  
738      D-3a: CREATE SP_PRODUCT_LISTING TO LIST A PRODUCT ORDERED  
739      DURING SPECIFIED MONTH AND YEAR.  
740      @CHECKPROD IS THE SUBSTRING OF A PRODUCT NAME TO BE SEARCHED  
741      @CHECMONTH AND @CHECKYEAR ARE THE MONTH AND YEAR TO BE  
... SEARCHED.  
742 */  
743
```

```

744 CREATE PROCEDURE sp_product_listing
745 (
746     @checkprod  varchar(30),
747     @checkmonth datetime,
748     @checkyear  datetime
749 )
750 AS
751 SELECT
752     'Product Name'          = products.prod_name,
753     'Unit Price'            = products.prod_unit_price,
754     'Qty in Stock'          = products.prod_qty_in_stock,
755     'Supplier Name'         = suppliers.sup_name
756 FROM products
757 INNER JOIN suppliers ON
758     ...products.supplier_id=suppliers.supplier_id
759 INNER JOIN order_details ON
760     ...products.product_id=order_details.product_id
761 INNER JOIN orders ON order_details.order_id=orders.order_id
762 WHERE products.prod_name LIKE '%' + @checkprod + '%'
763     AND DATEPART(MONTH, orders.order_date) = @checkmonth
764     AND DATEPART(YEAR, orders.order_date) = @checkyear;
765 GO
766 /*
767     D-3b: RUN SP_PRODUCT_LISTING SEARCHING FOR PRODUCTS
768     CONTAINING 'JACK' IN JUNE 2001.
769 */
770 EXECUTE sp_product_listing 'jack', 06, 2001;
771 GO
772 /*
773     D-4a: CREATE DELETE TRIGGER ON ORDER_DETAILS TO SHOW
774     PRODUCT_ID, PRODUCT NAME, QUANTITY DELETED, AND STOCK
775     AFTER DELETION WHEN AN ORDER_DETAILS ROW IS DELETED.
776 */
777 CREATE TRIGGER tr_dl_qty_update
778 ON order_details
779 FOR DELETE
780 AS
781 DECLARE
782     @prod_id      id_num,
783     @prod_qty     int,
784     @prod_name    nvarchar(40),
785     @prod_uprice  decimal(10,2),
786     @prod_qstock  int
787 
```

```

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 =
798 ... products.prod_qty_in_stock + @qty_deleted
799          WHERE products.product_id = @prod_id
800      END
801      BEGIN
802          SELECT
803              'Product ID'           =
804 ... products.product_id,
805              'Product Name'        =
806 ... products.prod_name,
807              'Quantity Being Deleted' = @qty_deleted,
808              'Quantity In Stock After Deletion' = @qty_deleted
809          products.prod_qty_in_stock
810          FROM deleted
811          INNER JOIN products ON
812          deleted.product_id=products.product_id
813          WHERE products.product_id = 25;
814      END;
815  GO
816  /*
817      D-4b: TRIGGER THE TRIGGER WITH A DELETE STATEMENT.
818 */
819
820  /*
821      D-5a: CREATE TRIGGER TR_QTY_CHECK FOR ORDER_DETAILS
822      WHICH WILL REJECT ANY QUANTITY UPDATE THAT CANNOT

```

```
823      BE SUPPLIED BY THE PRODUCT QUANTITY IN STOCK WILL
824      ALSO REPORT SHORTFALL IF IN STOCK QUANTITY IS NOT
825      ENOUGH
826  */
827
828 CREATE TRIGGER tr_qty_check
829 ON order_details
830 FOR UPDATE
831 AS
832 DECLARE
833     @prod_id          id_num,
834     @quantity         INT,
835     @instock          INT
836 SELECT
837     @prod_id          = inserted.product_id,
838     @quantity         = inserted.quantity - deleted.quantity,
839     @instock          = products.prod_qty_in_stock
840 FROM inserted
841 INNER JOIN deleted ON inserted.product_id=deleted.product_id
842 INNER JOIN products ON inserted.product_id=products.product_id
843 IF @quantity > @instock
844     BEGIN
845         PRINT 'NOT ENOUGH STOCK. YOU NEED ' +
...      CONVERT(varchar(10), (@quantity - @instock)) + ' MORE.'
846         ROLLBACK TRANSACTION
847     END
848 ELSE
849     BEGIN
850         UPDATE products
851         SET products.prod_qty_in_stock =
...      products.prod_qty_in_stock - @quantity
852             WHERE products.product_id = @prod_id
853         SELECT
854             'Name'           = products.prod_name,
855             'Updated Qty'   = products.prod_qty_in_stock
856             FROM products
857             WHERE product_id = @prod_id
858     END
859 GO
860
861 /*
862     D-6a: RUN FIRST SUPPLIED QUERY
863 */
```

```
864  
865 UPDATE order_details  
866 SET quantity =50  
867 WHERE order_id = '10044'  
868     AND product_id = 7;  
869 GO  
870  
871 /*  
872     D-6B: RUN SECOND SUPPLIED QUERY  
873 */  
874  
875 UPDATE order_details  
876 SET quantity = 40  
877 WHERE order_id = '10044'  
878     AND product_id = 7;  
879 GO  
880  
881  
882 /*  
883     D-7: STORED PROCEDURE TO DELETE INACTIVE CUSTOMERS.  
884 */  
885  
886 CREATE PROCEDURE sp_del_inactive_cust  
887 AS  
888 DELETE  
889     FROM customers  
890     WHERE customers.customer_id NOT IN  
891     (  
892         SELECT orders.customer_id  
893         FROM orders  
894     );  
895 GO  
896  
897 -- RUN THIS PROCEDURE  
898  
899 EXECUTE sp_del_inactive_cust;  
900 GO  
901  
902 /*  
903     D-8: LIST DETAILS OF SPECIFIED EMPLOYEE. @checkEmp IS THE  
904     EMPLOYEE ID OF THE EMPLOYEE IN QUESTION  
905 */  
906
```

```

907 CREATE PROCEDURE sp_employee_information
908 (
909     @checkEmp int
910 )
911 AS
912 SELECT
913     'Employee ID'          = employee.employee_id,
914     'Last Name'            = employee.emp_last_name,
915     'First Name'           = employee.emp_first_name,
916     'Address'              = employee.emp_address,
917     'City'                 = employee.emp_city,
918     'Province'             = employee.emp_province,
919     'Postal Code'          = employee.emp_postal_code,
920     'Phone #'              = employee.emp_phone,
921     'DOB'                  = employee.emp_birth_date
922 FROM employee
923 WHERE employee.employee_id = @checkEmp;
924 GO
925
926 -- RUN THE PROCEDURE
927
928 EXECUTE sp_employee_information 5;
929 GO
930
931 /*
932 D-9: CREATE PROCEDURE sp_reorder_qty TO SHOW WHEN REORDER
933 LEVEL SUBTRACTED FROM QUANTITY IN STOCK IS LESS THAN A
934 SPECIFIED VALUE. @checkNum IS THE VALUE TO CHECK
935 */
936
937 CREATE PROCEDURE sp_reorder_qty
938 (
939     @checkNum int
940 )
941 AS
942 SELECT
943     'Product ID'           = products.product_id,
944     'Supplier Name'         = suppliers.sup_name,
945     'Address'               = suppliers.sup_address,
946     'City'                  = suppliers.sup_city,
947     'Province'              = suppliers.sup_province,
948     'Qty'                   = products.prod_qty_in_stock,
949     'Reorder Lvl'           = products.prod_reorder_level

```

```
950      FROM products
951      INNER JOIN suppliers ON
952      ... products.supplier_id=suppliers.supplier_id
953      WHERE (products.prod_qty_in_stock - prod_reorder_level) <
954      ... @checkNum;
955 GO
956
957 -- RUN THE PROCEDURE
958
959 EXECUTE sp_reorder_qty 5;
960 GO
961
962 /*
963 D-10: CREATE PRCEUDRE TO SHOW DETAILS FOR ALL PRODUCTS
964 WITH UNIT PRICE BETWEEN SPECIFIED UPPER AND LOWER VALUES.
965 @checkLow WILL BE THE LOW VALUE AND @checkHIGh WILL BE
966 THE HIGH VALUE.
967 */
968
969 CREATE PROCEDURE sp_unit_prices
970 (
971     @checkLow      money,
972     @checkHigh     money
973 )
974 AS
975 SELECT
976     'Product ID'      = products.product_id,
977     'Name'            = products.prod_name,
978     'Alternate Name' = products.prod_alt_name,
979     'Unit Price'      = products.prod_unit_price
980
981     FROM products
982     WHERE products.prod_unit_price > @checkLow AND
983     products.prod_unit_price < @checkHigh;
984     GO
985
986 -- RUN THE PROCEDURE.
987
988 EXECUTE sp_unit_prices 5.00, 10.00;
989 GO
```