Project 2 – Cus_Orders Database

Jas Sohi

COMP-1630 - Relational Database & SQL

Instructor - Mark Bacchus - BCIT

July 23, 2014

Table of Contents

Introduction

The overall goal of this project was to create a database related to Customer Orders from scratch and reproduce queries, views, triggers, and stored procedures.

I created separate SQL queries and executed them separately when first answering the questions. Then I combined together all the statements and entered the GO statements after each transaction block into one combined script file.

Finally, I entered descriptive code (RAISE ERROR) to describe what the script was doing to anyone who would eventually be running it.

All code was queried in Microsoft SQL Server Management Studio running SQL Server 2012.

Observations

- I found that highlighting code and then executing using the F5 keyboard shortcut was the most intuitive and user friendly way to run queries for me.
- I found that the benefit of creating one final script file as it makes all your work completely reproducible.
- I found that commenting out code was the best strategy for troubleshooting. I first tried executing the entire query, but if I got undesirable results I reduced the query by commenting out what I thought was incorrect and tried again. Then I gradually un-commented lines as I kept getting desirable results.

Questions

Part A - Database and Tables

1. Create a database called Cus_Orders.

```
CREATE DATABASE Cus_Orders;
```

2. Create a user defined data types for all similar Primary Key attribute columns.

```
CREATE TYPE idtype FROM int NOT NULL;
CREATE TYPE cus_idtype FROM char(5) NOT NULL;
```

3. Create the following tables: customers, orders, order_details, products, shippers, suppliers, titles.

```
CREATE TABLE customers
(
customer_id cus_idtype,
name varchar(50) NOT NULL,
contact_name varchar(30),
title_id char(3) NOT NULL,
address varchar(50),
city varchar(20),
region varchar(15),
country_code varchar(10),
country varchar(15),
phone varchar(20),
fax   varchar(20)
 );

 CREATE TABLE orders
(
order_id idtype,
customer_id     cus_idtype,
employee_id     int NOT NULL,
shipping_name varchar(50),
shipping_address varchar(50),
shipping_city varchar(20),
shipping_region varchar(15),
shipping_country_code varchar(10),
```

```
shipping_country varchar(15),
shipper_id int NOT NULL,
order_date datetime,
required_date datetime,
shipped_date datetime,
freight_charge money
 );

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

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

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

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

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

4. Set the primary keys and foreign keys for the tables.

```
ALTER TABLE customers
ADD PRIMARY KEY (customer_id);

ALTER TABLE orders
ADD PRIMARY KEY (order_id);

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

ALTER TABLE products
ADD PRIMARY KEY (product_id);

ALTER TABLE shippers
ADD PRIMARY KEY (shipper_id);

ALTER TABLE suppliers
ADD PRIMARY KEY (supplier_id);

ALTER TABLE titles
ADD PRIMARY KEY (title_id);

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

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

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

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

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

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

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

```
ALTER TABLE customers
ADD CONSTRAINT default_country
DEFAULT('Canada') FOR country;

ALTER TABLE orders
ADD CONSTRAINT default_required_date
DEFAULT GETDATE() + 10  FOR required_date;

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

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

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

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

6. Load the data into your created tables using the following files:

customers.txt	*into the customers table*	*(91 rows)*
orders.txt	*into the orders table*	*(1078 rows)*
order_details.txt	*into the order_details table*	*(2820 rows)*
products.txt	*into the products table*	*(77 rows)*
shippers.txt	*into the shippers table*	*(3 rows)*
suppliers.txt	*into the suppliers table*	*(15 rows)*
titles.txt	*into the titles table*	*(12 rows)*

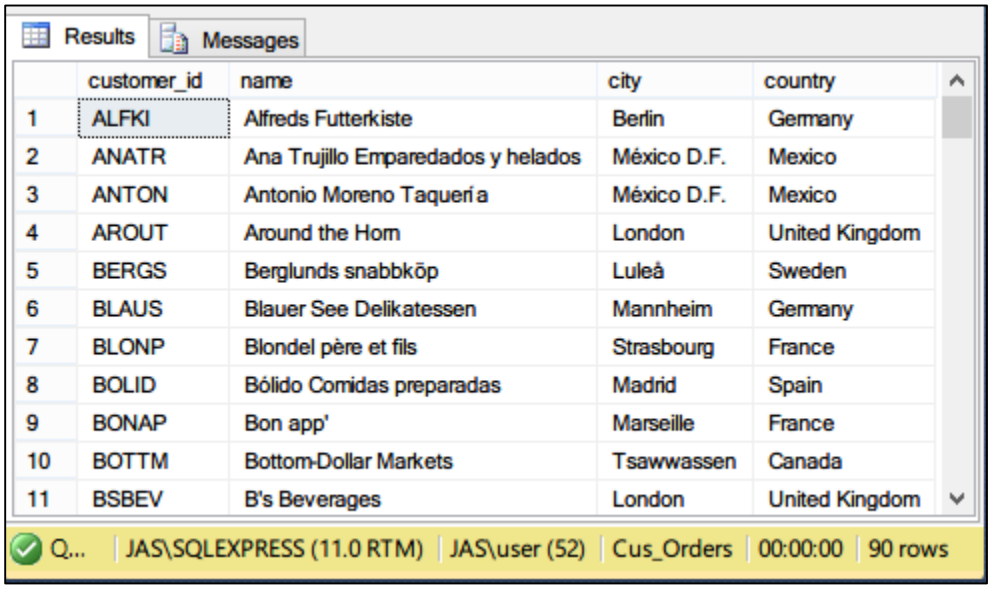
```
BULK INSERT titles
FROM 'C:\TextFiles\titles.txt'
WITH (
                CODEPAGE=1252,
            DATAFILETYPE = 'char',
            FIELDTERMINATOR = '\t',
            KEEPNULLS,
            ROWTERMINATOR = '\n'
       )

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


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

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

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

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

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

Part B - SQL Statements

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

```
SELECT customer_id, name, city, country
FROM customers
ORDER BY customer_id
;
```



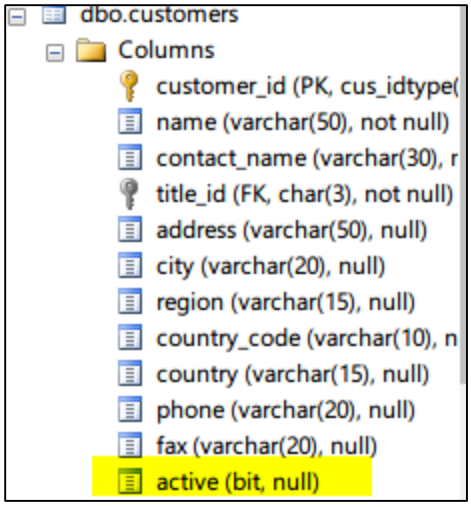
Results | Messages

	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

Q... | JAS\SQLEXPRESS (11.0 RTM) | JAS\user (52) | Cus_Orders | 00:00:00 | 90 rows

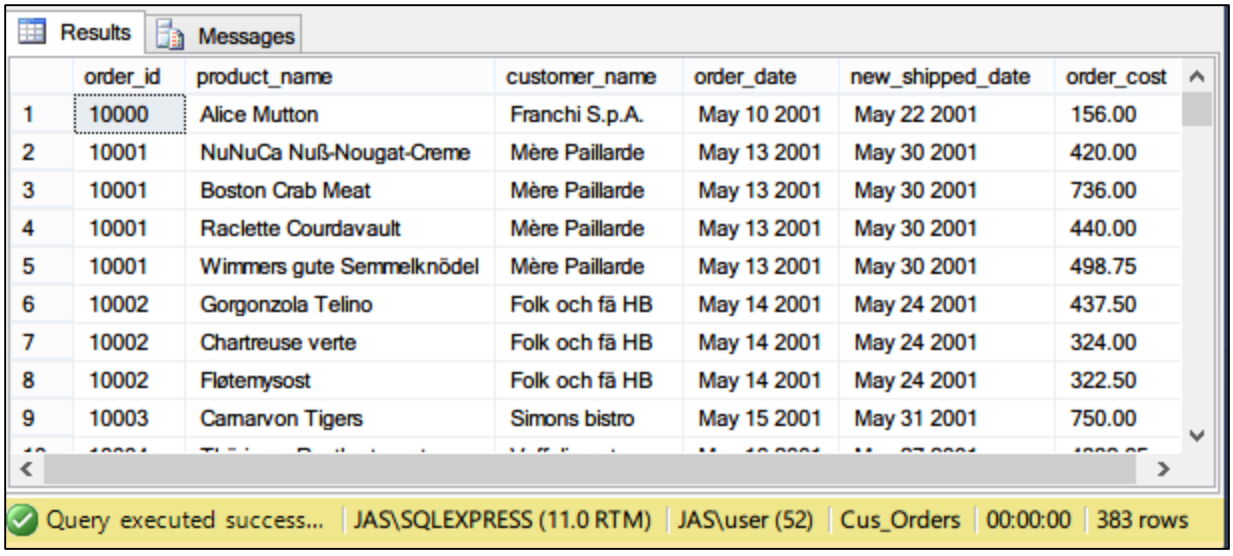
2. *Add a new column called active to the customers table using the ALTER statement. The only valid values are 1 or 0. The default should be 1.*

```
ALTER TABLE customers
ADD active bit
;

ALTER TABLE customers
ADD CONSTRAINT default_active
DEFAULT 1 FOR active
;
```



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 7 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.*

```
SELECT orders.order_id, 'product_name' = products.name,
'customer_name' = customers.name, 'order_date' =
CONVERT(char(12),orders.order_date, 0),
'new_shipped_date' = CONVERT(char(12),orders.shipped_date + 7, 0),
'order_cost' = order_details.quantity * products.unit_price
FROM customers
INNER JOIN orders ON orders.customer_id = customers.customer_id
INNER JOIN order_details ON order_details.order_id = orders.order_id
INNER JOIN products ON products.product_id = order_details.product_id
WHERE orders.order_date BETWEEN 'Jan 1, 2001' AND 'Dec 31, 2001';
```



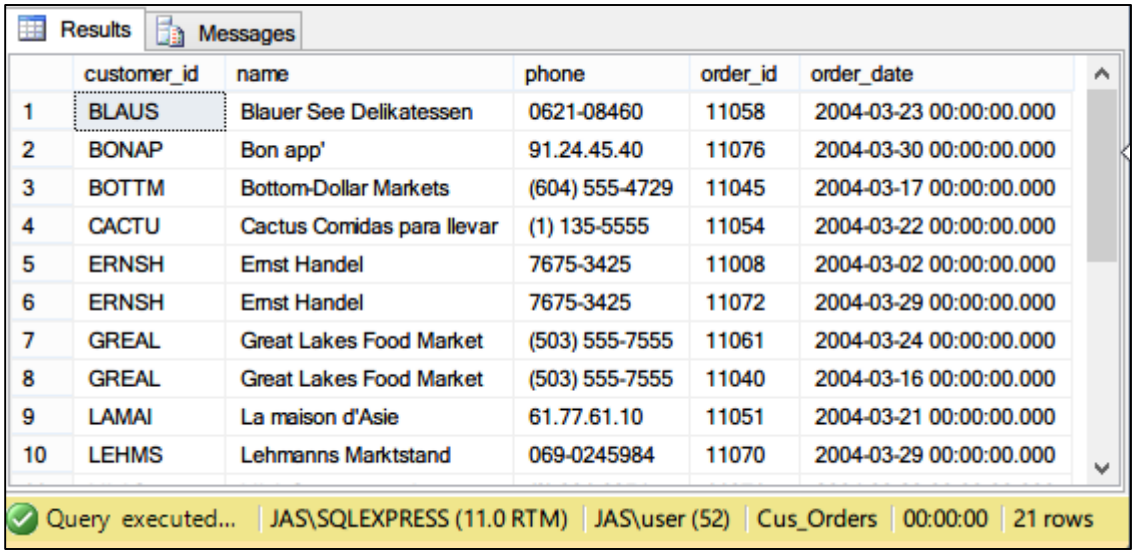
Results | Messages

	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 22 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 24 2001	437.50
7	10002	Chartreuse verte	Folk och fä HB	May 14 2001	May 24 2001	324.00
8	10002	Fløtemysost	Folk och fä HB	May 14 2001	May 24 2001	322.50
9	10003	Carnarvon Tigers	Simons bistro	May 15 2001	May 31 2001	750.00

Query executed success... | JAS\SQLEXPRESS (11.0 RTM) | JAS\user (52) | Cus_Orders | 00:00:00 | 383 rows

*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.*

```
SELECT customers.customer_id, customers.name, customers.phone,
orders.order_id, orders.order_date
FROM customers
INNER JOIN orders ON orders.customer_id = customers.customer_id
WHERE orders.shipped_date IS NULL
ORDER BY customers.name;
```



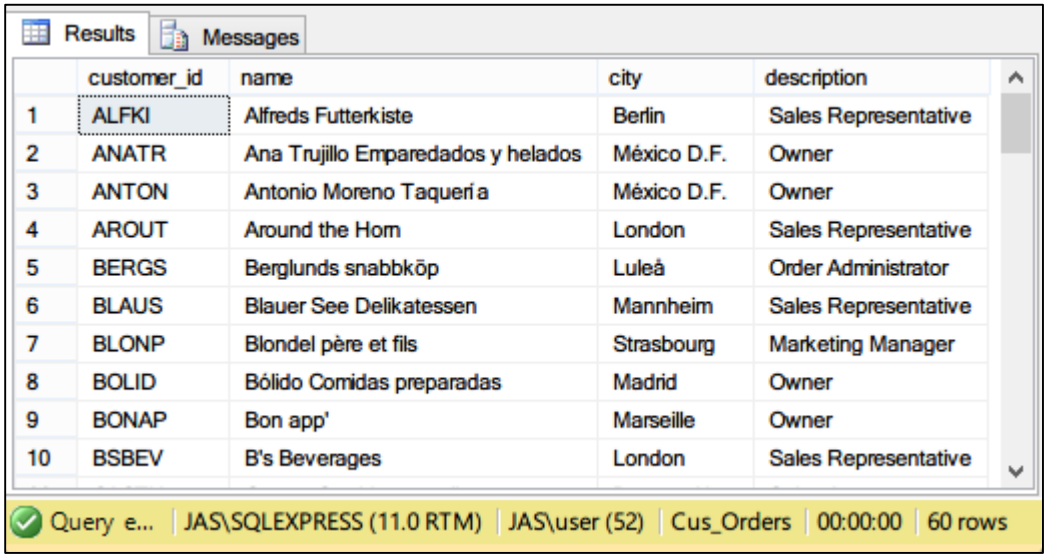
Results | Messages

	customer_id	name	phone	order_id	order_date
1	BLAUS	Blauer See Delikatessen	0621-08460	11058	2004-03-23 00:00:00.000
2	BONAP	Bon app'	91.24.45.40	11076	2004-03-30 00:00:00.000
3	BOTTM	Bottom-Dollar Markets	(604) 555-4729	11045	2004-03-17 00:00:00.000
4	CACTU	Cactus Comidas para llevar	(1) 135-5555	11054	2004-03-22 00:00:00.000
5	ERNSH	Ernst Handel	7675-3425	11008	2004-03-02 00:00:00.000
6	ERNSH	Ernst Handel	7675-3425	11072	2004-03-29 00:00:00.000
7	GREAL	Great Lakes Food Market	(503) 555-7555	11061	2004-03-24 00:00:00.000
8	GREAL	Great Lakes Food Market	(503) 555-7555	11040	2004-03-16 00:00:00.000
9	LAMAI	La maison d'Asie	61.77.61.10	11051	2004-03-21 00:00:00.000
10	LEHMS	Lehmanns Marktstand	069-0245984	11070	2004-03-29 00:00:00.000

Query executed... | JAS\SQLEXPRESS (11.0 RTM) | JAS\user (52) | Cus_Orders | 00:00:00 | 21 rows

*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.*

```
SELECT customers.customer_id, customers.name, customers.city,
titles.description
FROM customers
INNER JOIN titles ON titles.title_id = customers.title_id
WHERE customers.region IS NULL;
```



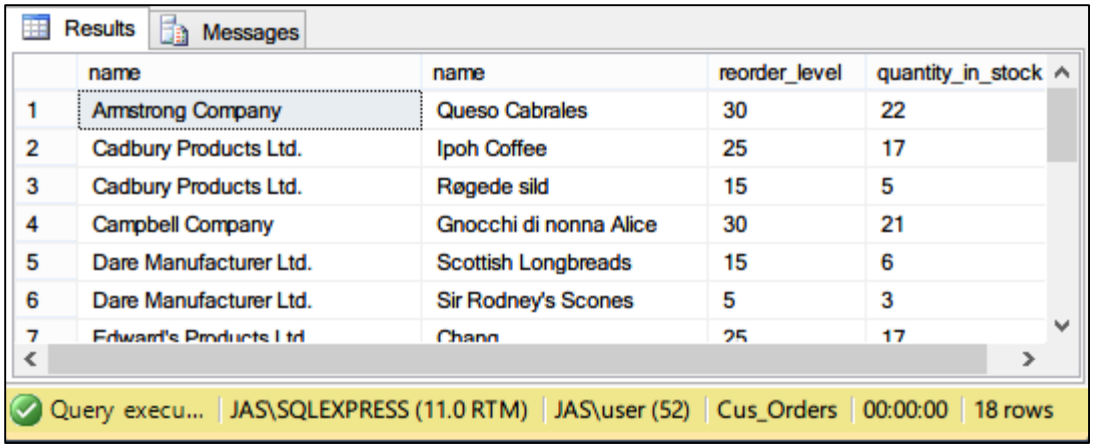
Results | Messages

	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

Query e... | JAS\SQLEXPRESS (11.0 RTM) | JAS\user (52) | Cus_Orders | 00:00:00 | 60 rows

*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.*

```
SELECT suppliers.name, products.name, products.reorder_level,
products.quantity_in_stock
FROM   products
INNER JOIN suppliers ON suppliers.supplier_id = products.supplier_id
WHERE products.reorder_level > products.quantity_in_stock
ORDER BY suppliers.name;
```



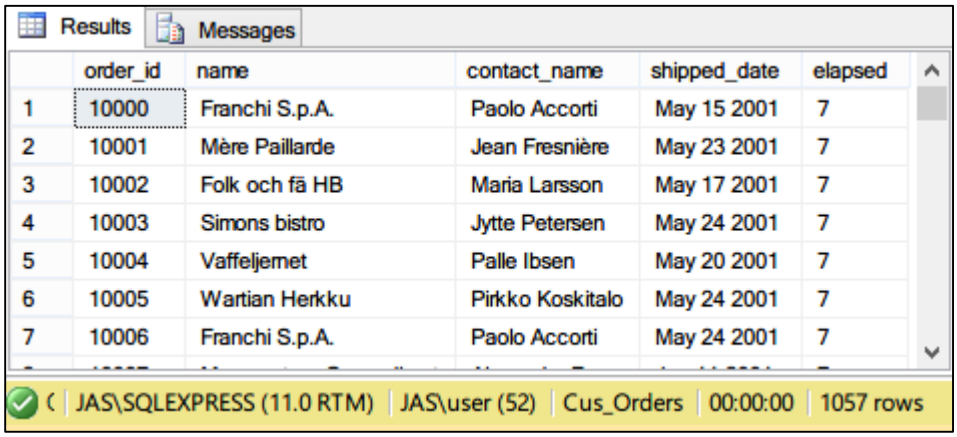
Results | Messages

	name	name	reorder_level	quantity_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

Query execu... | JAS\SQLEXPRESS (11.0 RTM) | JAS\user (52) | Cus_Orders | 00:00:00 | 18 rows

*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.*

```
SELECT orders.order_id, customers.name, customers.contact_name,
'shipped_date' = CONVERT(char(12), orders.shipped_date,0), 'elapsed' =
DATEDIFF(YEAR,orders.shipped_date,'Jan 1, 2008')
FROM orders
INNER JOIN customers ON customers.customer_id = orders.customer_id
WHERE orders.shipped_date IS NOT NULL;
```



Results | Messages

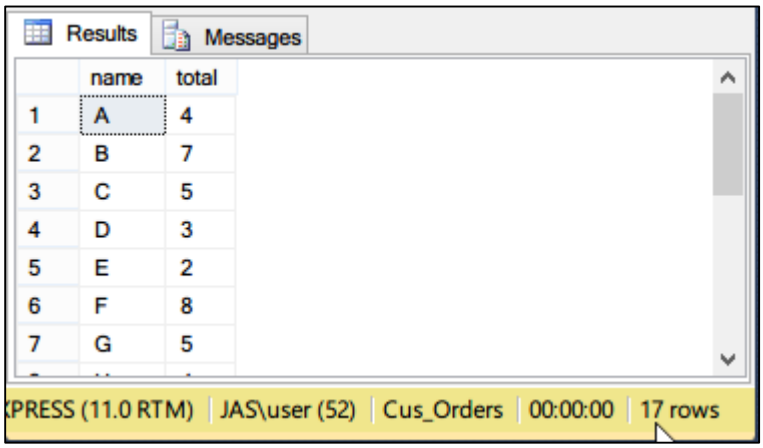
	order_id	name	contact_name	shipped_date	elapsed
1	10000	Franchi S.p.A.	Paolo Accorti	May 15 2001	7
2	10001	Mère Paillarde	Jean Fresnière	May 23 2001	7
3	10002	Folk och fä HB	Maria Larsson	May 17 2001	7
4	10003	Simons bistro	Jytte Petersen	May 24 2001	7
5	10004	Vaffeljernet	Palle Ibsen	May 20 2001	7
6	10005	Wartian Herkku	Pirkko Koskitalo	May 24 2001	7
7	10006	Franchi S.p.A.	Paolo Accorti	May 24 2001	7

(| JAS\SQLEXPRESS (11.0 RTM) | JAS\user (52) | Cus_Orders | 00:00:00 | 1057 rows

*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.*

```
SELECT 'name' = LEFT(name,1), 'total' = COUNT(*)
FROM customers
WHERE LEFT(name,1) <> 'S'
```

```
GROUP BY LEFT(name,1)
HAVING COUNT(*) > 1;
```



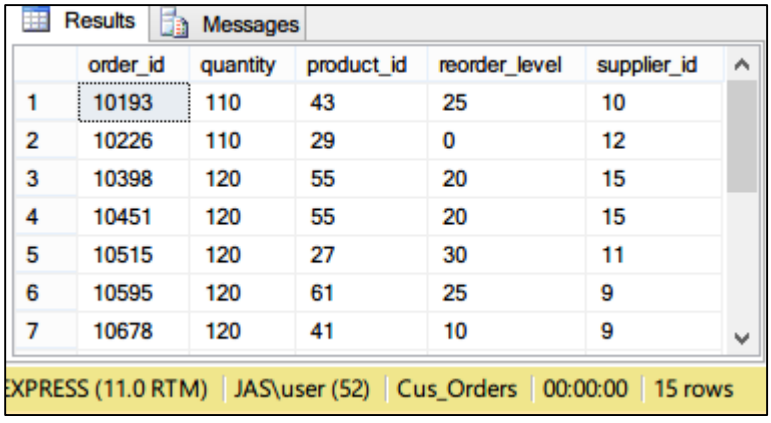
Results | Messages

	name	total
1	A	4
2	B	7
3	C	5
4	D	3
5	E	2
6	F	8
7	G	5

XPRESS (11.0 RTM) | JAS\user (52) | Cus_Orders | 00:00:00 | 17 rows

*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 and reorder level from the products table, and the supplier id from the suppliers table. Order the result set by the order id.*

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



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

EXPRESS (11.0 RTM) | JAS\user (52) | Cus_Orders | 00:00:00 | 15 rows

*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.*

```sql
SELECT product_id, name, quantity_per_unit, unit_price
FROM products
WHERE name LIKE '%tofu%' OR name LIKE '%chef%'
ORDER BY name;
```

Results | Messages

	product_id	name	quantity_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

JAS\SQLEXPRESS (11.0 RTM) | JAS\user (52) | Cus_Orders | 00:00:00 | 4 rows

Part C - INSERT, UPDATE, DELETE and VIEWS Statements

1. *Create an **employee** table with the following columns:*

Column Name	*Data Type*	*Length*	*Null Values*
employee_id	*int*		*No*
last_name	*varchar*	*30*	*No*
first_name	*varchar*	*15*	*No*
address	*varchar*	*30*	
city	*varchar*	*20*	
province	*char*	*2*	
postal_code	*varchar*	*7*	
phone	*varchar*	*10*	
birth_date	*datetime*		*No*

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

2. *The **primary key** for the employee table should be the employee id.*

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

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.*

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

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

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

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

5. *Using the UPDATE statement, increate the unit price in the products table of all rows with a current unit price between **$5.00** and **$10.00** by **5%**; 12 rows affected.*

```
UPDATE products
SET unit_price = 1.05 * unit_price
WHERE unit_price BETWEEN 5.00 AND 10.00;
```

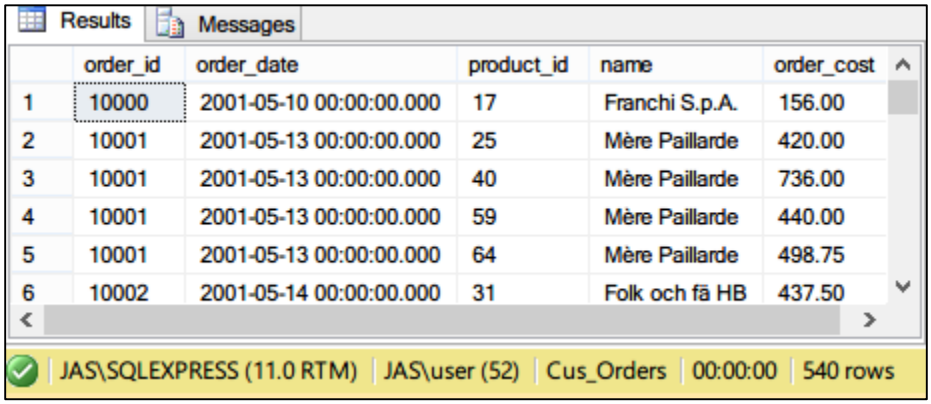
6. *Using the UPDATE statement, change the fax value to **Unknown** for all rows in the customers table where the current fax value is **NULL**; 22 rows affected.*

```
UPDATE customers
SET fax = 'Unknown'
WHERE fax IS NULL;
```

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 tble, 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**.*

```
CREATE VIEW vw_order_cost
AS
```

```
SELECT      orders.order_id,
            orders.order_date,
            products.product_id,
            customers.name,
            'order_cost' = order_details.quantity * products.unit_price
FROM customers
INNER JOIN orders ON orders.customer_id = customers.customer_id
INNER JOIN order_details ON order_details.order_id = orders.order_id
INNER JOIN products ON products.product_id = order_details.product_id
;

GO

SELECT *
FROM vw_order_cost
WHERE order_id BETWEEN 10000 AND 10200;
```



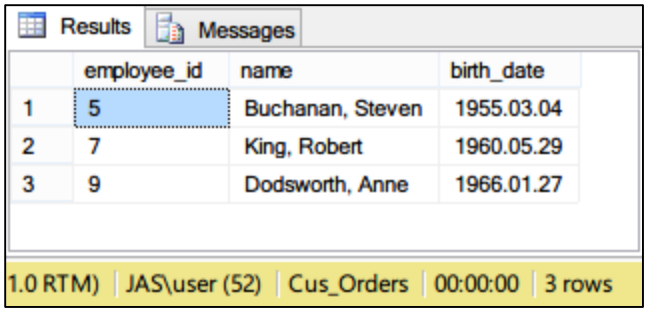
Results | Messages

	order_id	order_date	product_id	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

JAS\SQLEXPRESS (11.0 RTM) | JAS\user (52) | Cus_Orders | 00:00:00 | 540 rows

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**.*

```
CREATE VIEW vw_list_employees
AS
SELECT      *
FROM employee;

GO

SELECT employee_id, 'name' = last_name + ', ' + first_name,
'birth_date' = CONVERT(char(11),birth_date,102)
FROM vw_list_employees
WHERE employee_id IN (5,7,9);
```



Results | Messages

	employee_id	name	birth_date
1	5	Buchanan, Steven	1955.03.04
2	7	King, Robert	1960.05.29
3	9	Dodsworth, Anne	1966.01.27

1.0 RTM) | JAS\user (52) | Cus_Orders | 00:00:00 | 3 rows

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.*

```
CREATE VIEW vw_all_orders
(order_id, shipped_date, customer_id, customer_name, city, country)
AS
SELECT     orders.order_id,
           orders.shipped_date,
           customers.customer_id,
           customers.name,
           customers.city,
           customers.country
FROM customers
INNER JOIN orders ON orders.customer_id = customers.customer_id;

GO

SELECT order_id, customer_id, customer_name, city, country,
'shipped_date' = CONVERT(char(12),shipped_date,100)
FROM vw_all_orders
WHERE shipped_date BETWEEN 'Jan 1, 2002' AND 'Dec 31, 2002'
ORDER BY customer_name, country;
```

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 8 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

Query executed successf... | JAS\SQLEXPRESS (11.0 RTM) | JAS\user (52) | Cus_Orders | 00:00:00 | 293 rows

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.

```
CREATE VIEW vw_supplier_shipments
(supplier_id, supplier_name, product_id, product_name)
AS
SELECT     suppliers.supplier_id,
      suppliers.name,
      products.product_id,
      products.name
FROM suppliers
INNER JOIN products ON products.supplier_id = suppliers.supplier_id;

GO

SELECT *
FROM vw_supplier_shipments
```

Results | Messages

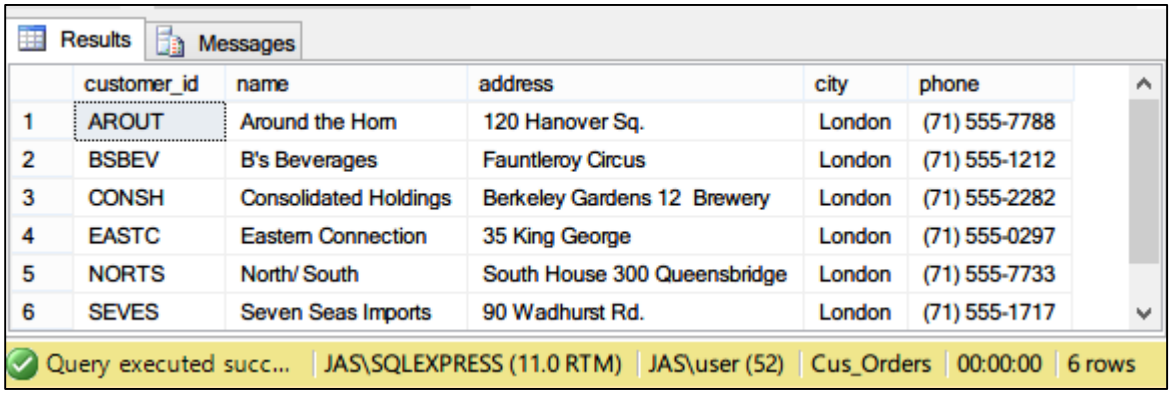
	supplier_id	supplier_name	product_id	product_name
1	1	Edward's Products Ltd.	1	Chai
2	1	Edward's Products Ltd.	2	Chang
3	1	Edward's Products Ltd.	3	Aniseed Syrup
4	2	New Orlean's Spices Ltd.	4	Chef Anton's Cajun Seasoning
5	2	New Orlean's Spices Ltd.	5	Chef Anton's Gumbo Mix
6	3	Macaulay Products Company	6	Grandma's Boysenberry Spread
7	3	Macaulay Products Company	7	Uncle Bob's Organic Dried Pears

Query exe... | JAS\SQLEXPRESS (11.0 RTM) | JAS\user (52) | Cus_Orders | 00:00:00 | 77 rows

Part D - Stored Procedures and Triggers

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***

```
CREATE PROCEDURE sp_customer_city
(
      @city varchar(20)
)
AS
SELECT     customer_id, name, address, city, phone
FROM customers
WHERE city = @city;
GO

EXECUTE sp_customer_city 'London'
```



Results | Messages

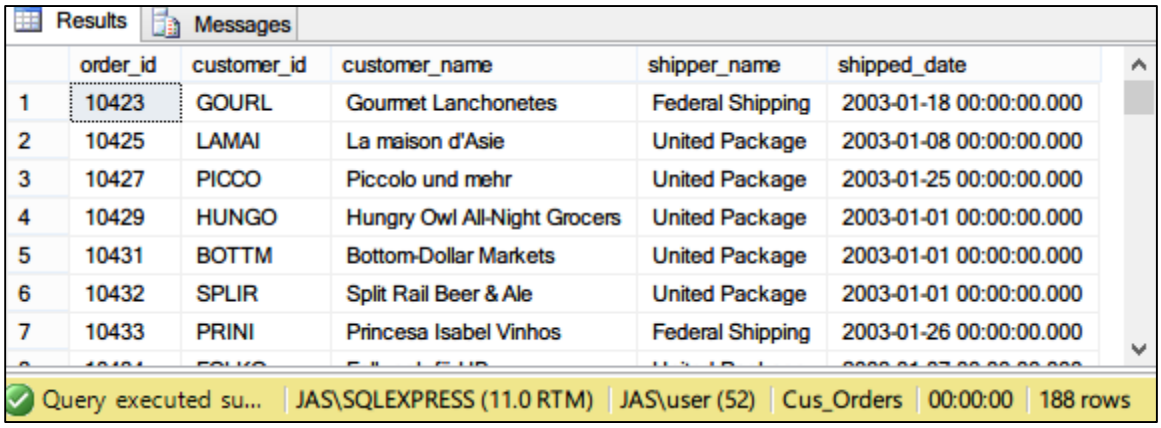
	customer_id	name	address	city	phone
1	AROUT	Around the Horn	120 Hanover Sq.	London	(71) 555-7788
2	BSBEV	B's Beverages	Fauntleroy Circus	London	(71) 555-1212
3	CONSH	Consolidated Holdings	Berkeley Gardens 12 Brewery	London	(71) 555-2282
4	EASTC	Eastern Connection	35 King George	London	(71) 555-0297
5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

Query executed succ... | JAS\SQLEXPRESS (11.0 RTM) | JAS\user (52) | Cus_Orders | 00:00:00 | 6 rows

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**.*

```
CREATE PROCEDURE sp_orders
(
      @startdate date,
      @enddate date
)
AS
```

```
SELECT orders.order_id, customers.customer_id, 'customer_name' =
customers.name, 'shipper_name' = shippers.name, 'shipped_date' =
orders.shipped_date
FROM customers
INNER JOIN orders ON customers.customer_id = orders.customer_id
INNER JOIN shippers ON orders.shipper_id = shippers.shipper_id
WHERE orders.shipped_date BETWEEN @startdate AND @enddate
GO

EXECUTE sp_orders 'Jan 1, 2003','June 30, 2003'
```



Results | Messages

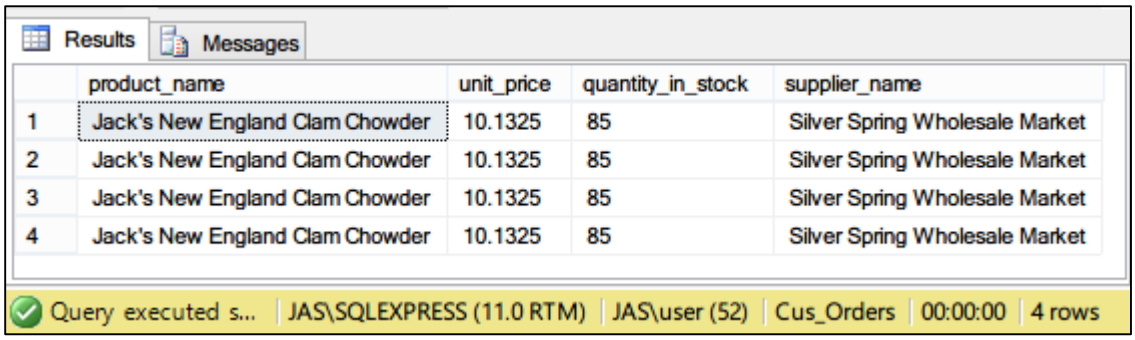
	order_id	customer_id	customer_name	shipper_name	shipped_date
1	10423	GOURL	Gourmet Lanchonetes	Federal Shipping	2003-01-18 00:00:00.000
2	10425	LAMAI	La maison d'Asie	United Package	2003-01-08 00:00:00.000
3	10427	PICCO	Piccolo und mehr	United Package	2003-01-25 00:00:00.000
4	10429	HUNGO	Hungry Owl All-Night Grocers	United Package	2003-01-01 00:00:00.000
5	10431	BOTTM	Bottom-Dollar Markets	United Package	2003-01-01 00:00:00.000
6	10432	SPLIR	Split Rail Beer & Ale	United Package	2003-01-01 00:00:00.000
7	10433	PRINI	Princesa Isabel Vinhos	Federal Shipping	2003-01-26 00:00:00.000

Query executed su... | JAS\SQLEXPRESS (11.0 RTM) | JAS\user (52) | Cus_Orders | 00:00:00 | 188 rows

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**.*

```
CREATE PROCEDURE sp_product_listing
(
@productname varchar(30),
@month varchar(9),
@year int
)
AS
SELECT 'product_name' = products.name,
products.unit_price,products.quantity_in_stock, 'supplier_name' =
suppliers.name
FROM suppliers
INNER JOIN products ON products.supplier_id = suppliers.supplier_id
INNER JOIN order_details ON order_details.product_id =
products.product_id
INNER JOIN orders ON orders.order_id = order_details.order_id
```

```
WHERE products.name LIKE '%' + @productname + '%' AND
datename(month,orders.order_date) = @month AND
datepart(year,orders.order_date) = @year
GO

EXECUTE sp_product_listing 'Jack','June','2001'
```



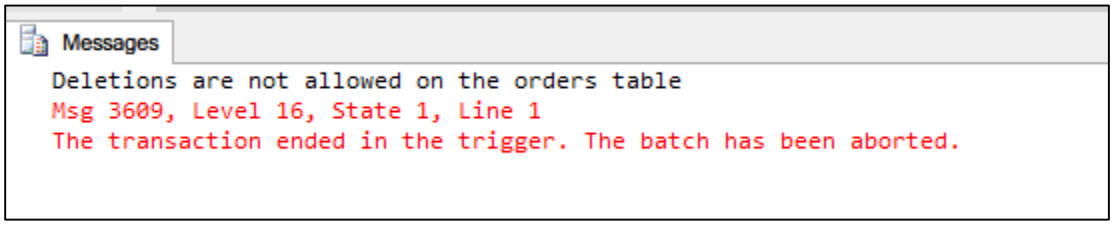
Results | Messages

	product_name	unit_price	quantity_in_stock	supplier_name
1	Jack's New England Clam Chowder	10.1325	85	Silver Spring Wholesale Market
2	Jack's New England Clam Chowder	10.1325	85	Silver Spring Wholesale Market
3	Jack's New England Clam Chowder	10.1325	85	Silver Spring Wholesale Market
4	Jack's New England Clam Chowder	10.1325	85	Silver Spring Wholesale Market

Query executed s... | JAS\SQLEXPRESS (11.0 RTM) | JAS\user (52) | Cus_Orders | 00:00:00 | 4 rows

4. *Create a **DELETE** trigger called **tr_delete_orders** on the orders table to display an error message if an order is deleted that has a value in the order_details table. (Since Referential Integrity constraints will normally prevent such deletions, this trigger needs to be an Instead of trigger.)*

```
CREATE TRIGGER tr_delete_orders
ON orders
INSTEAD OF DELETE
AS
BEGIN
      PRINT 'Deletions are not allowed on the orders table'
      ROLLBACK TRANSACTION
END

GO

DELETE orders
WHERE order_id = 10000
```

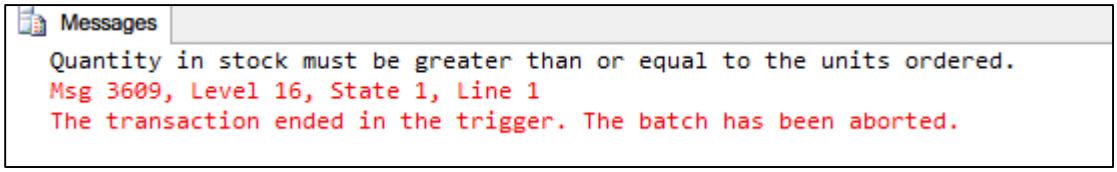


Messages

```
Deletions are not allowed on the orders table
Msg 3609, Level 16, State 1, Line 1
The transaction ended in the trigger. The batch has been aborted.
```

5. *Create an **INSERT** and **UPDATE** trigger called **tr_check_qty** on the order_details table to only allow orders of products that have a quantity in stock greater than or equal to the units ordered.*

```
CREATE TRIGGER tr_check_qty
ON order_details
FOR INSERT, UPDATE
AS
DECLARE @pid int
SELECT @pid = product_id
FROM inserted

IF (SELECT quantity FROM inserted) > (SELECT quantity_in_stock FROM
products WHERE product_id = @pid)
      BEGIN
            PRINT 'Quantity in stock must be greater than or equal to
the units ordered.'
            ROLLBACK TRANSACTION
      END
GO

UPDATE order_details
SET quantity = 30
WHERE order_id = '10044' AND product_id = 7
```



6. *Create a stored procedure called **sp_del_inactive_cust** to **delete** customers that have no orders. The stored procedure should delete **1** row.*

```
CREATE PROCEDURE sp_del_inactive_cust
AS
DELETE customers
FROM customers
LEFT JOIN  orders ON orders.customer_id = customers.customer_id
WHERE orders.order_id IS NULL
GO

--Ran this query before and after to see the effect of this procedure
SELECT customers.customer_id, orders.order_id
FROM customers
LEFT JOIN orders ON orders.customer_id = customers.customer_id
```

```
--PARIS was only customer_id without an order

EXECUTE sp_del_inactive_cust
```

BEFORE

Results | Messages

	customer_id	order_id
1069	QUEEN	11068
1070	TORTU	11069
1071	LEHMS	11070
1072	LILAS	11071
1073	ERNSH	11072
1074	PERIC	11073
1075	SIMOB	11074
1076	RICSU	11075
1077	BONAP	11076
1078	RATTC	11077
1079	PARIS	NULL

AFTER

Results | Messages

	customer_id	order_id
1068	DRACD	11067
1069	QUEEN	11068
1070	TORTU	11069
1071	LEHMS	11070
1072	LILAS	11071
1073	ERNSH	11072
1074	PERIC	11073
1075	SIMOB	11074
1076	RICSU	11075
1077	BONAP	11076
1078	RATTC	11077

7. 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**.

```
CREATE PROCEDURE sp_employee
```

```
(
@employeeid int
)
AS
SELECT employee_id, last_name, first_name, address, city, province,
postal_code, phone, birth_date
FROM employee
WHERE employee.employee_id = @employeeid
GO

EXECUTE sp_employee '5'
```

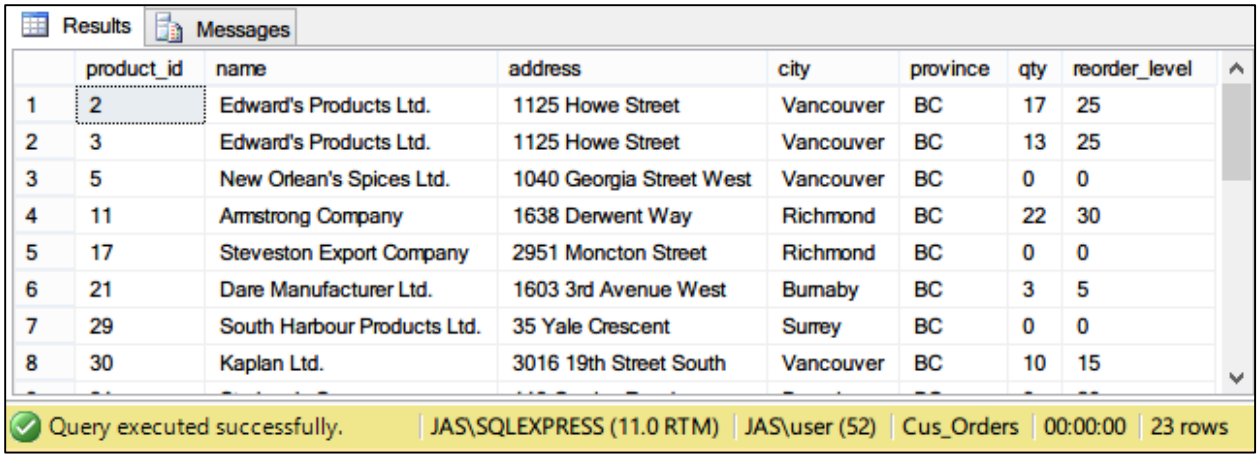
Results | Messages

	employee_id	last_name	first_name	address	city	province	postal_code	phone	birth_date
1	5	Buchanan	Steven	14 Garrett Hill	New Westminster	BC	V1G 8J7	6045554848	1955-03-04 00:00:00.000

Query executed successfully. | JAS\SQLEXPRESS (11.0 RTM) | JAS\user (52) | Cus_Orders | 00:00:00 | 1 rows

8. *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**.*

```
CREATE PROCEDURE sp_reorder_qty
(
@unitvalue int
)
AS
SELECT products.product_id, suppliers.name, suppliers.address,
suppliers.city, suppliers.province, 'qty' =
products.quantity_in_stock, products.reorder_level
FROM products
INNER JOIN suppliers ON suppliers.supplier_id = products.supplier_id
WHERE (products.quantity_in_stock - products.reorder_level) <
@unitvalue
GO

EXECUTE sp_reorder_qty '5'
```



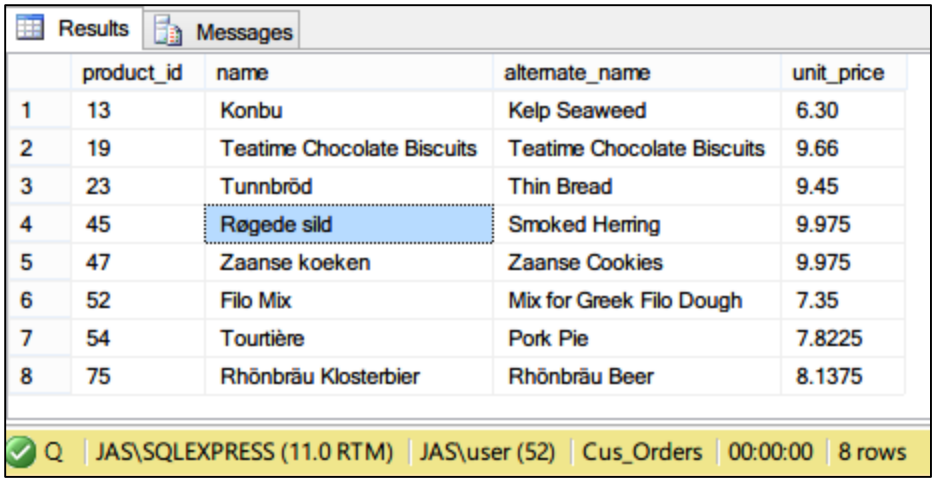
Results | Messages

	product_id	name	address	city	province	qty	reorder_level
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	11	Armstrong Company	1638 Derwent Way	Richmond	BC	22	30
5	17	Steveston Export Company	2951 Moncton Street	Richmond	BC	0	0
6	21	Dare Manufacturer Ltd.	1603 3rd Avenue West	Burnaby	BC	3	5
7	29	South Harbour Products Ltd.	35 Yale Crescent	Surrey	BC	0	0
8	30	Kaplan Ltd.	3016 19th Street South	Vancouver	BC	10	15

Query executed successfully. | JAS\SQLEXPRESS (11.0 RTM) | JAS\user (52) | Cus_Orders | 00:00:00 | 23 rows

9. *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**. The stored procedure should produce the result set listed below.*

```
CREATE PROCEDURE sp_unit_prices
(
@smallunitprice money,
@largeunitprice money
)
AS
SELECT product_id, name, alternate_name, unit_price
FROM products
WHERE unit_price BETWEEN @smallunitprice AND @largeunitprice
GO

EXECUTE sp_unit_prices '5.00','10.00'
```

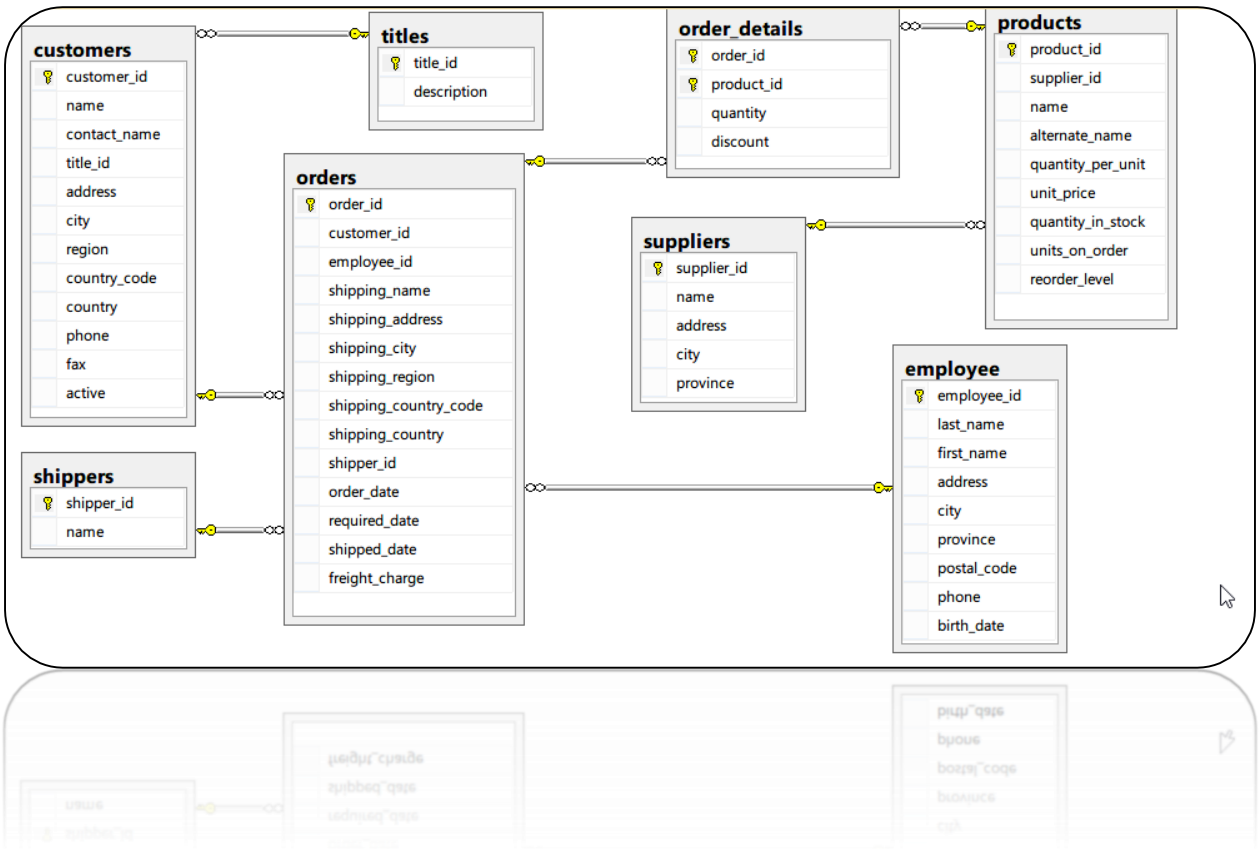


Results | Messages

	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	9.975
5	47	Zaanse koeken	Zaanse Cookies	9.975
6	52	Filo Mix	Mix for Greek Filo Dough	7.35
7	54	Tourtière	Pork Pie	7.8225
8	75	Rhönbräu Klosterbier	Rhönbräu Beer	8.1375

Q | JAS\SQLEXPRESS (11.0 RTM) | JAS\user (52) | Cus_Orders | 00:00:00 | 8 rows

Database Diagram



LOAD SCRIPT

```
/*                                                                    */
/*          Project2.SQL - Creates the Cus_Orders database            */
/*          Jas Sohi - July 23, 2014                                  */
/*                                                                    */

SET NOCOUNT ON
GO

set nocount    on
set dateformat mdy

USE master -- Use Master database

declare @dttm varchar(55)
select  @dttm=convert(varchar,getdate(),113)
raiserror('Beginning Project2.SQL at %s ....',1,1,@dttm) with nowait

GO

-- Check for existence of existing objects
if exists (SELECT * FROM sysdatabases WHERE name='Cus_Orders')
begin
  raiserror('Dropping existing Cus_Orders database ....',0,1)
  DROP database Cus_Orders

end
GO

CHECKPOINT

GO

raiserror('Creating Cus_Orders database....',0,1)
GO

-- 1. Create database called Cus_Orders
raiserror('Part A - Database and Tables....',0,1)
CREATE DATABASE Cus_Orders;

GO

Use Cus_Orders -- Use Cus_Orders

-- 2. Create user defined types
CREATE TYPE idtype FROM int NOT NULL;
CREATE TYPE cus_idtype FROM char(5) NOT NULL;
```

```
GO

-- 3. Create the following tables
raiserror('Creating the tables....',0,1)
CREATE TABLE customers
(
customer_id cus_idtype,
name varchar(50) NOT NULL,
contact_name varchar(30),
title_id char(3) NOT NULL,
address varchar(50),
city varchar(20),
region varchar(15),
country_code varchar(10),
country varchar(15),
phone varchar(20),
fax      varchar(20)
 );

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

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

 CREATE TABLE products
(
product_id idtype,
```

```
supplier_id     int NOT NULL,
name varchar(40) NOT NULL,
alternate_name varchar(40),
quantity_per_unit varchar(25),
unit_price money,
quantity_in_stock int,
units_on_order int,
reorder_level int
 );

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

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

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

GO

-- 4. Set the primary and foreign keys for the tables

raiserror('Creating the primary keys....',0,1)
--PRIMARY KEYS
ALTER TABLE customers
ADD PRIMARY KEY (customer_id);

ALTER TABLE orders
ADD PRIMARY KEY (order_id);

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

ALTER TABLE products
ADD PRIMARY KEY (product_id);
```

```
ALTER TABLE shippers
ADD PRIMARY KEY (shipper_id);

ALTER TABLE suppliers
ADD PRIMARY KEY (supplier_id);

ALTER TABLE titles
ADD PRIMARY KEY (title_id);

GO

--FOREIGN KEYS
raiserror('Creating the foreign keys....',0,1)
ALTER TABLE customers
ADD CONSTRAINT FK_customers_titles FOREIGN KEY (title_id)
REFERENCES  titles
(title_id);

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

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

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

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

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

--employee_id FK will be entered later as the employee table is currently not created
GO

-- 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
*/
raiserror('Creating the default constraints....',0,1)
ALTER TABLE customers
ADD CONSTRAINT default_country
DEFAULT('Canada') FOR country;

ALTER TABLE orders
ADD CONSTRAINT default_required_date
DEFAULT GETDATE() + 10  FOR required_date;

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

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

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

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

GO

-- 6. Load the data into your created tables using the following files:
raiserror('Loading the data into the created tables....',0,1)
BULK INSERT titles
FROM 'C:\TextFiles\titles.txt'
WITH (
            CODEPAGE=1252,
                  DATAFILETYPE = 'char',
                  FIELDTERMINATOR = '\t',
                  KEEPNULLS,
                  ROWTERMINATOR = '\n'
          )

BULK INSERT suppliers
FROM 'C:\TextFiles\suppliers.txt'
WITH (
            CODEPAGE=1252,
```

```
            DATAFILETYPE = 'char',
            FIELDTERMINATOR = '\t',
            KEEPNULLS,
            ROWTERMINATOR = '\n'
        )


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

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

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

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

BULK INSERT orders
FROM 'C:\TextFiles\orders.txt'
```

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

GO

-- Part B - SQL Statements
raiserror('Part B - SQL Statements....',0,1)
-- 1. List the customer id, name, city, and country from the customer table.  Order the result
set by the customer id.
SELECT customer_id, name, city, country
FROM customers
ORDER BY customer_id
;
GO

-- 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.

ALTER TABLE customers
ADD active bit
;

ALTER TABLE customers
ADD CONSTRAINT default_active
DEFAULT 1 FOR active
;

GO

/* 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 7 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.  The query should produce the result set listed below.
*/

SELECT orders.order_id, 'product_name' = products.name, 'customer_name' =
customers.name, 'order_date' = CONVERT(char(12),orders.order_date, 0),
'new_shipped_date' = CONVERT(char(12),orders.shipped_date + 7, 0), 'order_cost' =
order_details.quantity * products.unit_price
FROM customers
```

```
INNER JOIN orders ON orders.customer_id = customers.customer_id
INNER JOIN order_details ON order_details.order_id = orders.order_id
INNER JOIN products ON products.product_id = order_details.product_id
WHERE orders.order_date BETWEEN 'Jan 1, 2001' AND 'Dec 31, 2001';

GO
/* 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.  The query should produce the result
set listed below. Your displayed results may look slightly different to those shown below but
the query should still return 21 rows.
*/

SELECT customers.customer_id, customers.name, customers.phone, orders.order_id,
orders.order_date
FROM customers
INNER JOIN orders ON orders.customer_id = customers.customer_id
WHERE orders.shipped_date IS NULL
ORDER BY customers.name;

GO
/* 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.
*/

SELECT customers.customer_id, customers.name, customers.city, titles.description
FROM customers
INNER JOIN titles ON titles.title_id = customers.title_id
WHERE customers.region IS NULL;

GO
/* 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.  The query should produce the result set listed below.
*/

SELECT suppliers.name, products.name, products.reorder_level, products.quantity_in_stock
FROM   products
INNER JOIN suppliers ON suppliers.supplier_id = products.supplier_id
WHERE products.reorder_level > products.quantity_in_stock
ORDER BY suppliers.name;

GO

/* 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.  The query should produce the result
 set listed below.
*/

SELECT orders.order_id, customers.name, customers.contact_name, 'shipped_date' =
CONVERT(char(12), orders.shipped_date,0), 'elapsed' =
DATEDIFF(YEAR,orders.shipped_date,'Jan 1, 2008')
FROM orders
INNER JOIN customers ON customers.customer_id = orders.customer_id
WHERE orders.shipped_date IS NOT NULL;

GO
/* 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.  The query should
produce the result set listed below.
*/

SELECT 'name' = LEFT(name,1), 'total' = COUNT(*)
FROM customers
WHERE LEFT(name,1) <> 'S'
GROUP BY LEFT(name,1)
HAVING COUNT(*) > 1;

GO
/* 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 and reorder level from the
 products table, and the supplier id from the suppliers table.  Order the result set
 by the order id.  The query should produce the result set listed below.
*/

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

GO
/* 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.  The query should produce the result set listed below.
*/

SELECT product_id, name, quantity_per_unit, unit_price
FROM products
```

```
WHERE name LIKE '%tofu%' OR name LIKE '%chef%'
ORDER BY name;

GO
-- Part C - INSERT, UPDATE, DELETE and VIEWS Statements
raiserror('Part C - INSERT, UPDATE, DELETE and VIEWS Statements....',0,1)
-- 1. Create an employee table with the following columns:

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

GO

-- 2. The primary key for the employee table should be the employee id.

ALTER TABLE employee
ADD PRIMARY KEY (employee_id);

GO
/* 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.
*/

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

--Enforce referential integrity

ALTER TABLE orders
ADD CONSTRAINT FK_orders_employee FOREIGN KEY (employee_id)
```

```
REFERENCES  employee
(employee_id);

GO
-- 4. Using the INSERT statement, add the shipper Quick Express to the shippers table.

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

GO
-- 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%; 12 rows affected.

UPDATE products
SET unit_price = 1.05 * unit_price
WHERE unit_price BETWEEN 5.00 AND 10.00;

GO
-- 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; 22 rows affected.

UPDATE customers
SET fax = 'Unknown'
WHERE fax IS NULL;

GO
/* 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.  The view should produce the
result set listed below.
*/

CREATE VIEW vw_order_cost
AS
SELECTorders.order_id,
                orders.order_date,
                products.product_id,
                customers.name,
                'order_cost' = order_details.quantity * products.unit_price
FROM customers
INNER JOIN orders ON orders.customer_id = customers.customer_id
INNER JOIN order_details ON order_details.order_id = orders.order_id
INNER JOIN products ON products.product_id = order_details.product_id
;

GO
```

```
SELECT *
FROM vw_order_cost
WHERE order_id BETWEEN 10000 AND 10200;

GO
/* 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.  The view should
 produce the result set listed below.
 */

CREATE VIEW vw_list_employees
AS
SELECT*
FROM employee;

GO

SELECT employee_id, 'name' = last_name + ', ' + first_name, 'birth_date' =
CONVERT(char(11),birth_date,102)
FROM vw_list_employees
WHERE employee_id IN (5,7,9);

GO
/* 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. The view should
produce
 the result set listed below.
*/

CREATE VIEW vw_all_orders
(order_id, shipped_date, customer_id, customer_name, city, country)
AS
SELECTorders.order_id,
                orders.shipped_date,
                customers.customer_id,
                customers.name,
                customers.city,
                customers.country
FROM customers
INNER JOIN orders ON orders.customer_id = customers.customer_id;
```

```
GO

SELECT order_id, customer_id, customer_name, city, country, 'shipped_date' =
CONVERT(char(12),shipped_date,100)
FROM vw_all_orders
WHERE shipped_date BETWEEN 'Jan 1, 2002' AND 'Dec 31, 2002'
ORDER BY customer_name, country;

GO
/* 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.
The view should produce the result set listed below, although not necessarily in the same
order.
*/

CREATE VIEW vw_supplier_shipments
(supplier_id, supplier_name, product_id, product_name)
AS
SELECTsuppliers.supplier_id,
                suppliers.name,
                products.product_id,
                products.name
FROM suppliers
INNER JOIN products ON products.supplier_id = suppliers.supplier_id;

GO

SELECT *
FROM vw_supplier_shipments

GO

--Part D - Stored Procedures and Triggers
raiserror('Part D - Stored Procedures and Triggers',0,1)
GO
/* 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.  The stored
procedure should produce the result set listed below.
*/

CREATE PROCEDURE sp_customer_city
(
        @city   varchar(20)
```

```
)
AS
SELECTcustomer_id, name, address, city, phone
FROM customers
WHERE city = @city;
GO

EXECUTE sp_customer_city 'London'

GO

/* 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.
*/

CREATE PROCEDURE sp_orders
(
        @startdate date,
        @enddate date
)
AS
SELECT orders.order_id, customers.customer_id, 'customer_name' = customers.name,
'shipper_name' = shippers.name, 'shipped_date' = orders.shipped_date
FROM customers
INNER JOIN orders ON customers.customer_id = orders.customer_id
INNER JOIN shippers ON orders.shipper_id = shippers.shipper_id
WHERE orders.shipped_date BETWEEN @startdate AND @enddate
GO

EXECUTE sp_orders 'Jan 1, 2003','June 30, 2003'

GO

/* 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.
*/
```

```
CREATE PROCEDURE sp_product_listing
(
@productname varchar(30),
@month varchar(9),
@year int
)
AS
SELECT 'product_name' = products.name, products.unit_price,products.quantity_in_stock,
'supplier_name' = suppliers.name
FROM suppliers
INNER JOIN products ON products.supplier_id = suppliers.supplier_id
INNER JOIN order_details ON order_details.product_id = products.product_id
INNER JOIN orders ON orders.order_id = order_details.order_id
WHERE products.name LIKE '%' + @productname + '%' AND datename(month,orders.order_date)
= @month AND datepart(year,orders.order_date) = @year
GO

EXECUTE sp_product_listing 'Jack','June','2001'

GO

/*4. Create a DELETE trigger called tr_delete_orders on the orders table to display an error
message if an order
is deleted that has a value in the order_details table. ( Since Referential Integrity constraints
will normally
prevent such deletions, this trigger needs to be an Instead of trigger.)  Run the following
query to verify your trigger.

DELETE orders
WHERE order_id = 10000
*/

CREATE TRIGGER tr_delete_orders
ON orders
INSTEAD OF DELETE
AS
BEGIN
        PRINT 'Deletions are not allowed on the orders table'
        ROLLBACK TRANSACTION
END

GO

DELETE orders
WHERE order_id = 10000

GO
```

```
/* 5. Create an INSERT and UPDATE trigger called tr_check_qty on the order_details table to
only allow orders of
products that have a quantity in stock greater than or equal to the units ordered.  Run the
following query to verify your trigger.

UPDATE order_details
SET quantity = 30
WHERE order_id = '10044'
    AND product_id = 7
*/


CREATE TRIGGER tr_check_qty
ON order_details
FOR INSERT, UPDATE
AS
DECLARE @pid int
SELECT @pid = product_id
FROM inserted

IF (SELECT quantity FROM inserted) > (SELECT quantity_in_stock FROM products WHERE
product_id = @pid)
        BEGIN
                PRINT 'Quantity in stock must be greater than or equal to the units ordered.'
                ROLLBACK TRANSACTION
        END
GO

UPDATE order_details
SET quantity = 30
WHERE order_id = '10044' AND product_id = 7

GO

/* 6. Create a stored procedure called sp_del_inactive_cust to delete customers that have no
orders.
  The stored procedure should delete 1 row.
*/

CREATE PROCEDURE sp_del_inactive_cust
AS
DELETE customers
FROM customers
LEFT JOIN  orders ON orders.customer_id = customers.customer_id
WHERE orders.order_id IS NULL
GO

--Ran this query before and after to see the effect of this procedure
SELECT customers.customer_id, orders.order_id
```

```
FROM customers
LEFT JOIN orders ON orders.customer_id = customers.customer_id

--PARIS was only customer_id without an order

EXECUTE sp_del_inactive_cust

GO

/* 7. 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.
*/

CREATE PROCEDURE sp_employee
(
@employeeid int
)
AS
SELECT employee_id, last_name, first_name, address, city, province, postal_code, phone,
birth_date
FROM employee
WHERE employee.employee_id = @employeeid
GO

EXECUTE sp_employee '5'
GO

/* 8. 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.
*/

CREATE PROCEDURE sp_reorder_qty
(
@unitvalue int
)
AS
SELECT products.product_id, suppliers.name, suppliers.address, suppliers.city,
suppliers.province, 'qty' = products.quantity_in_stock, products.reorder_level
FROM products
INNER JOIN suppliers ON suppliers.supplier_id = products.supplier_id
```

```
WHERE (products.quantity_in_stock - products.reorder_level) < @unitvalue
GO

EXECUTE sp_reorder_qty '5'
GO

/* 9. 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.
*/

CREATE PROCEDURE sp_unit_prices
(
@smallunitprice money,
@largeunitprice money
)
AS
SELECT product_id, name, alternate_name, unit_price
FROM products
WHERE unit_price BETWEEN @smallunitprice AND @largeunitprice
GO

EXECUTE sp_unit_prices '5.00','10.00'
GO
```