Wanderson De Castro

BCIT – A01062023

Mark bacchus

FINAL PROJECT

COMP 1630 – RELATIONAL DATABASE & SQL

PROJECT CONTENT

[PROJECT DESCRIPTION 3](#_Toc5626992)

[Part A - Database and Tables 4](#_Toc5626993)

[Q.1 4](#_Toc5626994)

[Q.2 5](#_Toc5626995)

[Q.3 6](#_Toc5626996)

[Q.4 8](#_Toc5626997)

[Q.5 10](#_Toc5626998)

[Q.6 12](#_Toc5626999)

[Part B - SQL Statements 13](#_Toc5627000)

[Q.1 13](#_Toc5627001)

[Q.2 14](#_Toc5627002)

[Q.3 15](#_Toc5627003)

[Q.4 16](#_Toc5627004)

[Q.5 17](#_Toc5627005)

[Q.6 18](#_Toc5627006)

[Q.7 19](#_Toc5627007)

[Q.8 20](#_Toc5627008)

[Q.9 21](#_Toc5627009)

[Q.10 22](#_Toc5627010)

[Part C - INSERT, UPDATE, DELETE and VIEWS Statements 23](#_Toc5627011)

[Q.1 23](#_Toc5627012)

[Q.2 24](#_Toc5627013)

[Q.3 24](#_Toc5627014)

[Q.4 25](#_Toc5627015)

[Q.5 25](#_Toc5627016)

[Q.6 26](#_Toc5627017)

[Q.7 27](#_Toc5627018)

[Q.8 28](#_Toc5627019)

[Q.9 29](#_Toc5627020)

[Q.10 30](#_Toc5627021)

[Part D - Stored Procedures and Triggers 31](#_Toc5627022)

[Q.1 31](#_Toc5627023)

[Q.2 32](#_Toc5627024)

[Q.3 33](#_Toc5627025)

[Q.4 34](#_Toc5627026)

[Q.5 35](#_Toc5627027)

[Q.6 36](#_Toc5627028)

[Q.7 37](#_Toc5627029)

[Q.8 38](#_Toc5627030)

[Q.9 39](#_Toc5627031)

[SCRIP COPY – ALL QUESTIONS SCRIPTS 40](#_Toc5627032)

[BULK INSERT SCRIPT 68](#_Toc5627033)

# **PROJECT DESCRIPTION**

**•Final Project for COMP 1630.**

**•This project will have instructions to how create, edit, manipulee and insert most of the principal’s commands of SQL SERVER.**

**•You can follow all steps to learn each command and execute.**

**•At the very end you will see all scripts to execute.**

**•The Bulk insert script will need to be inserted separate from the main script.**

**•After done all questions you will be able to create a DATABASE and edit following the requirements of what you will need.**

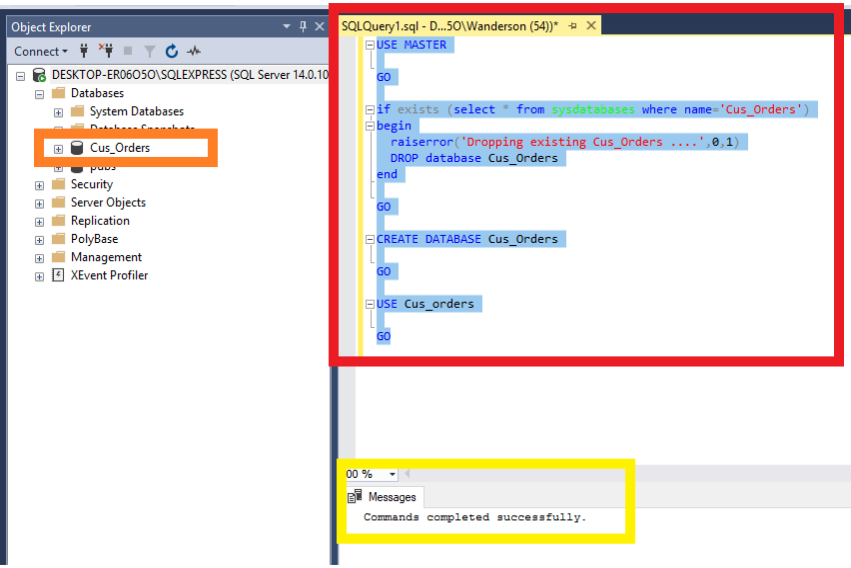
**•This project has only proposed for COMP 1630 and do not can be apply for a real company in your real script. You should create you own DATABASE following your company rules.**

**•This project also can be used to show in resume for interview in a company to show what you are able to do.**

# **Part A - Database and Tables**

## **Q.1**

1. Create a database called **Cus\_Orders**.



•THIS IS THE PROCESS TO CREATE A DATABASE CALLED Cus\_Orders

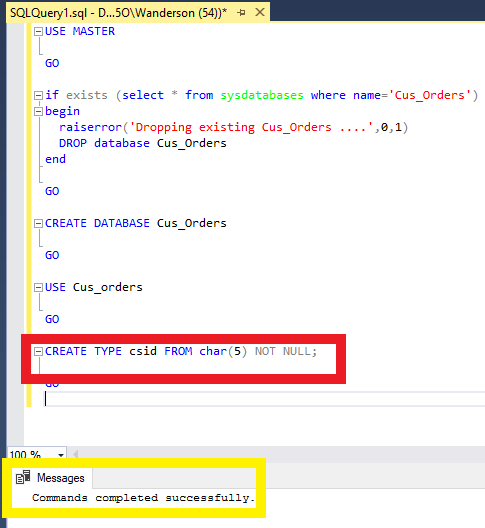
•RED – SCRIPT THAT WAS INSERTED

•YELLOW – MESSAGE PRODUCED

•ORANGE – AFTER REFRESH YOU CAN SEE THAT THE DATABASE WAS CREATED

## **Q.2**

1. 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. See pages 12/13 for specifications.



•THIS SCRIPT WAS USED TO CREATE A UNIQUE SPECIFICATION FOR THIS TYPE.

•RED – SCRIPT THAT WAS INSERTED

•YELLOW – MESSAGE PRODUCED

## **Q.3**

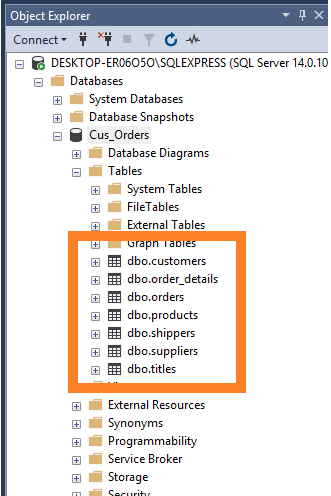
1. Create the following tables (see column information on pages 12 and 13 ):



•FOR CREATE A TABLE I USED THIS SCRIPT, YOU HAVE TO USE THIS SCRIPT FOR EACH TABLE AND CHANGING THE TABLE NAME AND COLUMNS NAME AND SPECIFICATIONS.

•RED – SCRIPT THAT WAS INSERTED

•YELLOW – MESSAGE PRODUCED

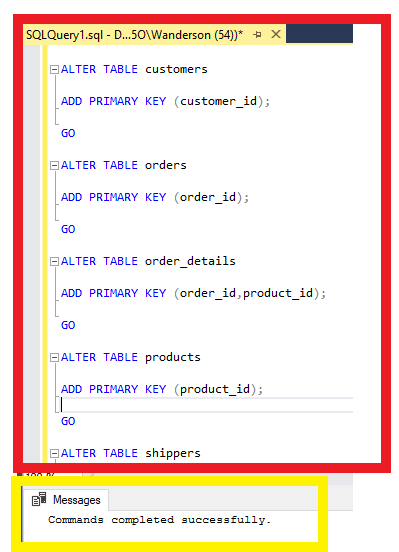


•ORANGE – AFTER REFRESH YOU CAN SEE THAT THE TABLES WERE CREATED.

GO TO “Databases > Cus\_Orders > Tables”

## **Q.4**

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



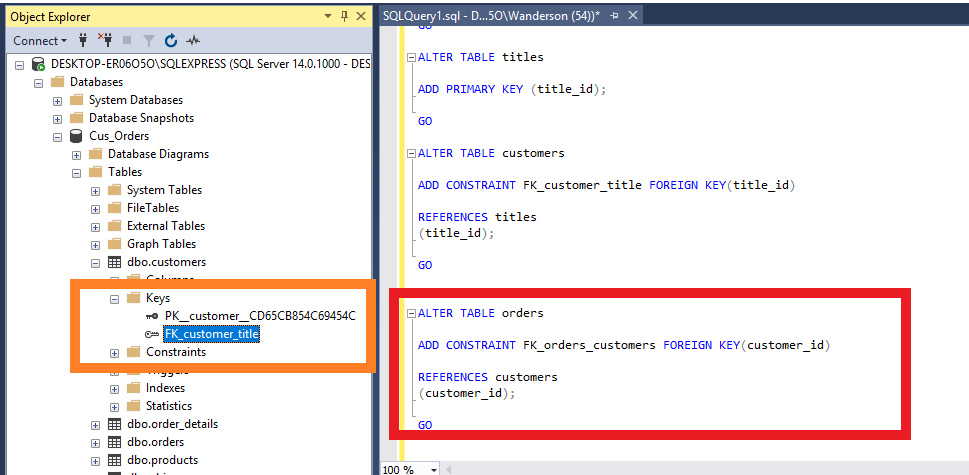
•FOR CREATE PRIMARY KEYS YOU SHOULD USE THIS SCRIPT WITH THE INFORMATION THAT YOU WANT.

ATER TABLE – TO CHOOSE TABLE THAT WILL BE USED.

ADD PRIMARY KEY – TO INSERT PRIMARY KEY (COLUMN NAME);

•RED – SCRIPT THAT WAS INSERTED

•YELLOW – MESSAGE PRODUCED



•FOR ADD FOREIGN KEYS YOU SHOULD USE THIS SCRIPT.

ATER TABLE – TO CHOOSE TABLE THAT WILL BE USED.

ADD CONSTRAINT “CONSTRAINT NAME” – WILL ADD A CONSTRAINT TO THIS TABLE WITH THE NAME THAT YOU CHOOSED.

FOREIGN KEY (COLUMN NAME) – WILL TRANSFORM THIS COLUMN IN A FOREIGN KEY.

REFERENCES “TABLE THAT FOREIGN IS PRIMARY”

(COLUMN THAT IN THIS TABLE IS PRIMARY KEY);

•RED – SCRIPT THAT WAS INSERTED

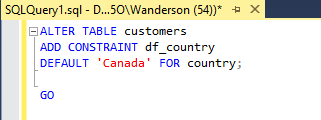
•ORANGE – AFTER REFRESH YOU CAN SEE THAT THE PRIMARY AND FOREIGN WERE CREATED.

OPEN TABLE > KEYS

## **Q.5**

1. Set the **constraints** as follows:

**customers** **table** - country should default to Canada]

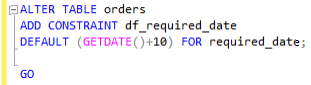


•USE THIS SCRIPT TO CREATE A CONSTRAINT WITH A DEFAULT VALUE.

DEFAULT ‘NAME’

FOR “COLUMN”

**orders** **table** - required\_date should default to today’s date plus ten days

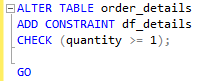


•USE THIS SCRIPT TO CREAT A CONSTRAINT WITH YOUR RULE.

•IN THIS CASE WE USED “GETDATE()” THAT WILL TAKE THE ATUAL DATE

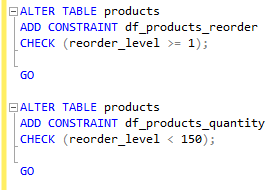
AND “+10” THAT WILL USE TO ADD 10 DAYS.

**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

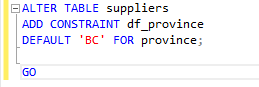


•IN THIS SCRIP WE SHOULD USE “CHECK” TO DETERMINE IF THIS VALUE IT IS TRUE AND IF WILL BE ABLE TO USE.

•HOW YOU CAN SEE FOR “GREATER OR EQUAL THAN 1” WE USE THE SIGNAL “>=”

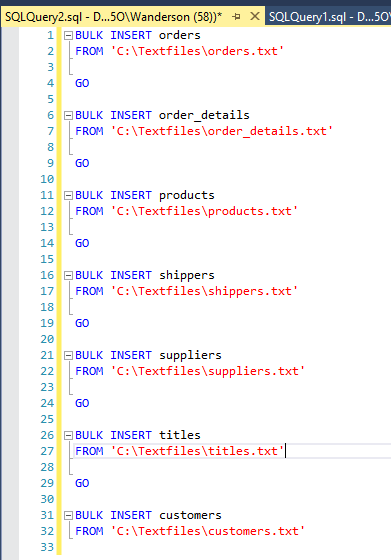
AND FOR THAT ONE THAT CAN’T BE GREATER THAN 150 WE USE ”<”.

**suppliers table** - province should default to BC



## **Q.6**

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



•FOR IMPORT DATA FROM SOMEWHERE FROM YOU COMPUTER OR DATACENTER WE CAN USE THE FUNCTION “BULK INSERT”

BULK INSERT “table name”

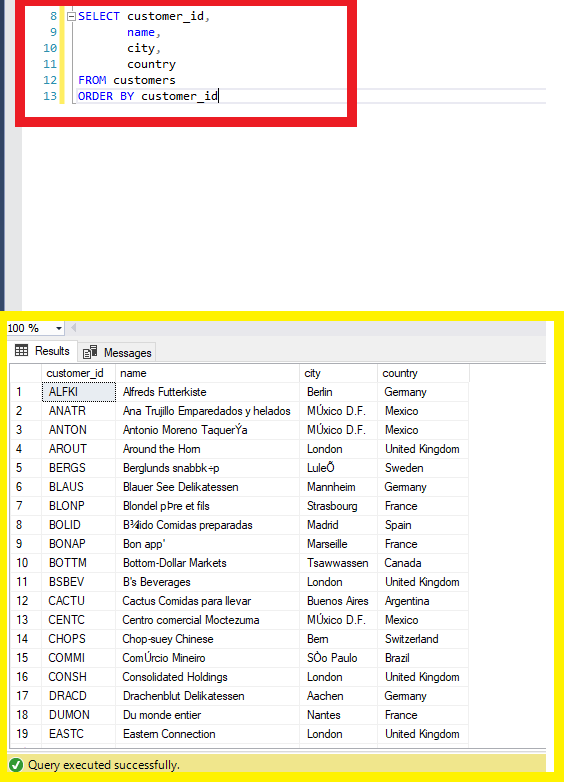
FROM “Where they are in your storage”

•THE VALUE PRODUCED FOR THIS CASE WILL BE NUBER OF ROLLS AFFECTED THAT WILL BE DIFFERENT FOR EACH INSERT.

# **Part B - SQL Statements**

## **Q.1**

1. List the customer id, name, city, and country from the customer table. Order the result set by the **customer id**. The query should produce the result set listed below.



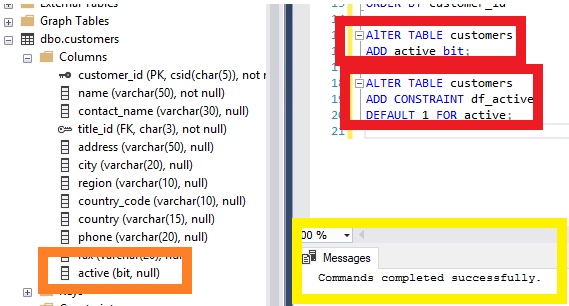
•RED – SCRIPT THAT WAS INSERTED

•YELLOW – MESSAGE PRODUCED

•ON THE SCRIPT YOU SHOULD SELECT COLUMNS THAT YOU WANT, FROM THE TABER THAT THEY ARE AND USE “ORDER BY” TO ORDER FOLLOWING WHAT YOU NEED.

## **Q.2**

1. 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**.



•RED – SCRIPT THAT WAS INSERTED

•YELLOW – MESSAGE PRODUCED

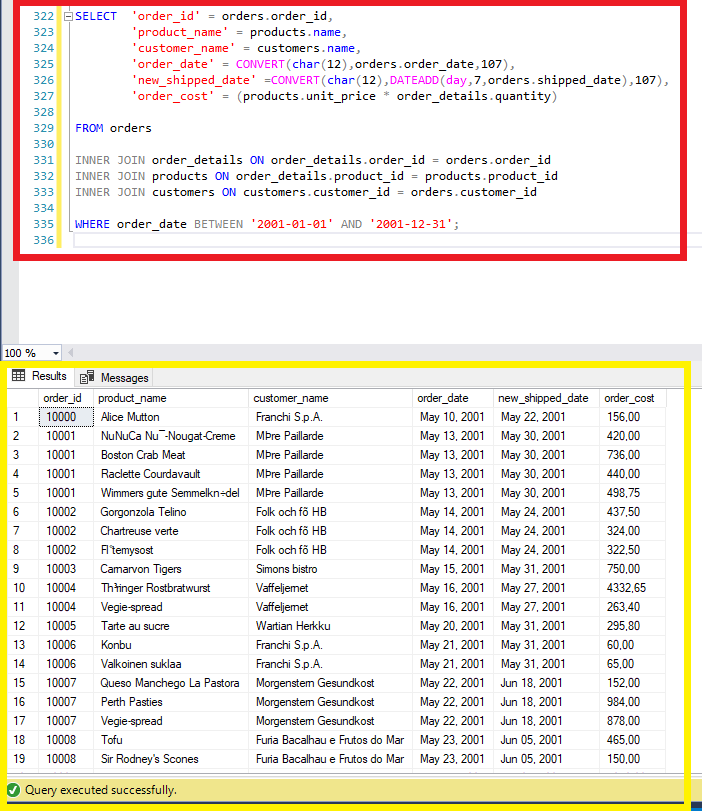
•ORANGE – AFTER REFRESH YOU CAN SEE THAT THE DATABASE WAS CREATED

•IN THIS CASE I USED AGAIN O ALTER TO ADD A COLUMN TO A CUSTOMERS TABLE ALSO I CHOOSED TYPE “bit” THAT HAS VALUE BETWEEN 0-1 AS WAS ASKED.

•USING ALTER AGAIN I CHANGED DEFAULT VALUE FOR 1 TO COLUMN active.

## **Q.3**

1. List all the orders where the order date is between **January 1** and **December 31, 200**1. 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.



•RED – SCRIPT THAT WAS INSERTED

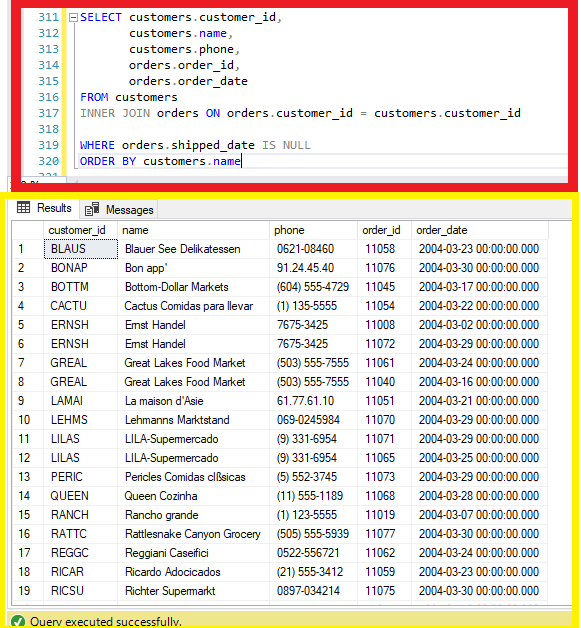
•YELLOW – MESSAGE PRODUCED

•THIS VIEW IT’S MORE COMPLEX BECAUSE YOU WILL NEED USE INNER JOIN TO TAKE VALUES FROM ANOTHER TABLES. BUT THE PROCESS IT’S ALMOST THE SAME , NAME THE COLUMN HOW YOU NEED AND USE BEFORE THE COLUMN NAME THE NAME FROM TABLE WHERE IT CAME FROM AS “TABLE.COLUMN\_NAME”.

•TO ADD 7 DAYS TO SHIPPED\_DATE I USED “DATEADD(day,number of days, column) AND TO CONVERT TO MON DD YYYY I USED DATE TYPE 107.

## **Q.4**

1. 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.



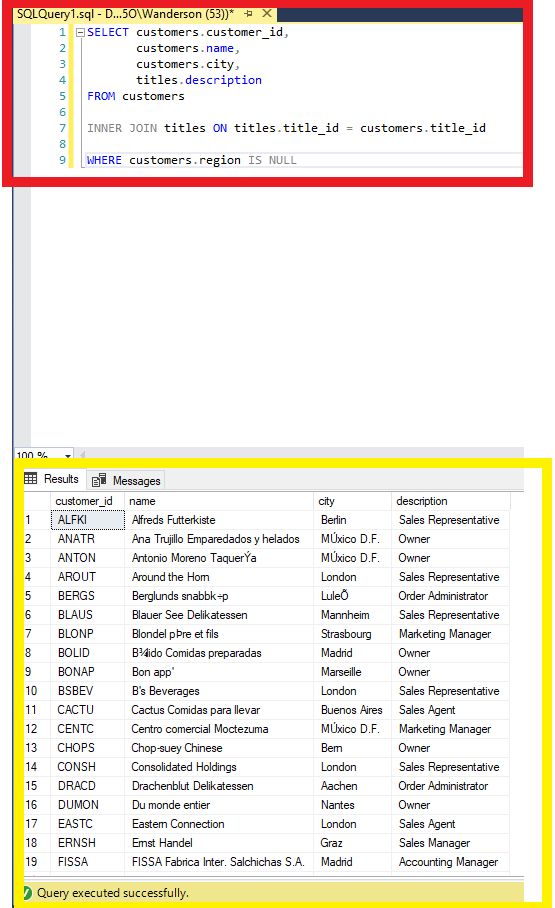
•RED – SCRIPT THAT WAS INSERTED

•YELLOW – MESSAGE PRODUCED

•THIS CASE IS VERY SIMILAR AS SHOWED ON QUESTION NUMBER 3, BUT IN THIS CASE AT THE END I USED TO ORDER BY A ESPECIFIC COLUMN.

## **Q.5**

1. 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.



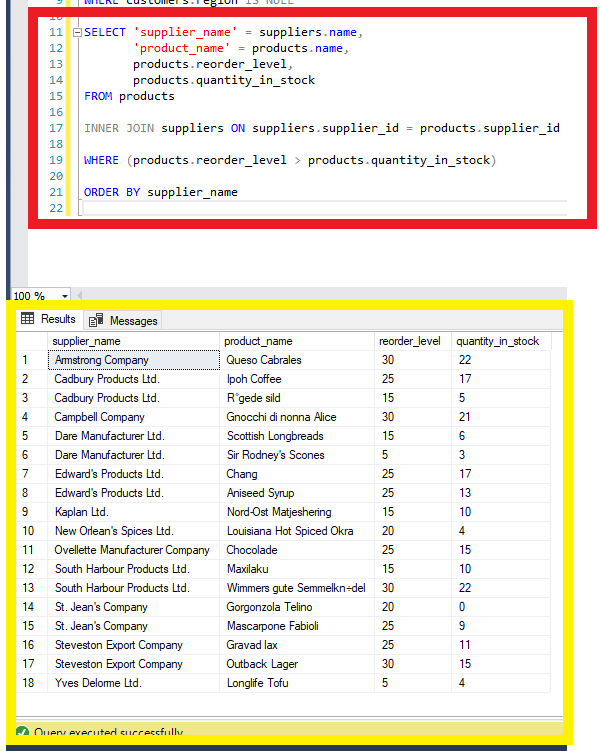
•RED – SCRIPT THAT WAS INSERTED

•YELLOW – MESSAGE PRODUCED

•IN THIS CASE I USED WHERE TO SELECT ALL REGIONS THAT ARE NULL.

## **Q.6**

1. 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.



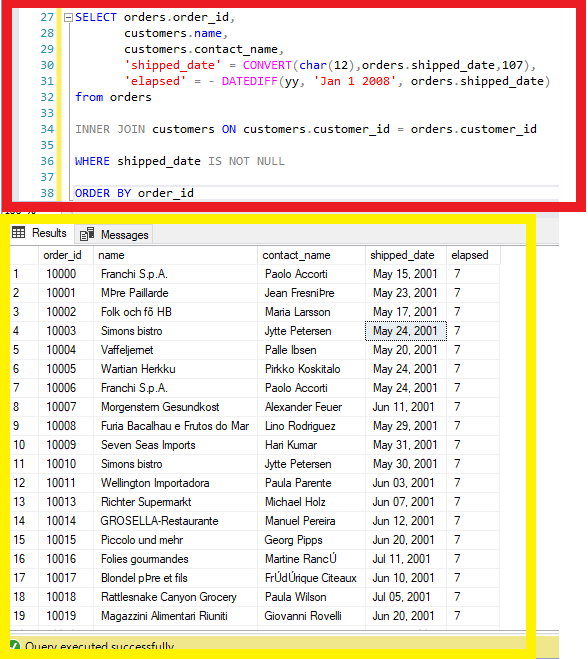
•RED – SCRIPT THAT WAS INSERTED

•YELLOW – MESSAGE PRODUCED

•THIS CASE AFTER SELECTED ALL COLUMNS THAT I WANT AND WHERE THEY ARE FROM, I DID ONE INNER JOIN TO GET MY SUPPLIER\_ID AND ON WHERE I USED “>” BETWEEN COLUMNS THAT I WANT FOR SHOW ONLY VALUES THAT ONE IS GREATER THAN OTHER.

## **Q.7**

1. 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.



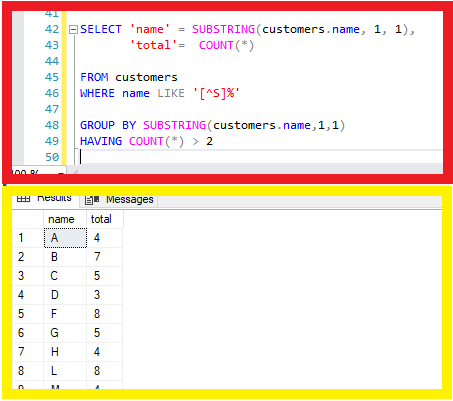
•RED – SCRIPT THAT WAS INSERTED

•YELLOW – MESSAGE PRODUCED

•IN THIS QUESTION WAS USED “CONVERT” TO SHOW DATE RESULTS ON FORMAT THAT WAS ASKED AND “DATEDIFF” TO GET THE DIFERENCE BETWEEN TIME THAT I WANTED.

## **Q.8**

1. 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.



•RED – SCRIPT THAT WAS INSERTED

•YELLOW – MESSAGE PRODUCED

•THIS QUESTION I USED “SUBSTRING” TO USE JUST THE FIRST LETTER OF NAMES AND

“COUNT” TO COUNT THEM.

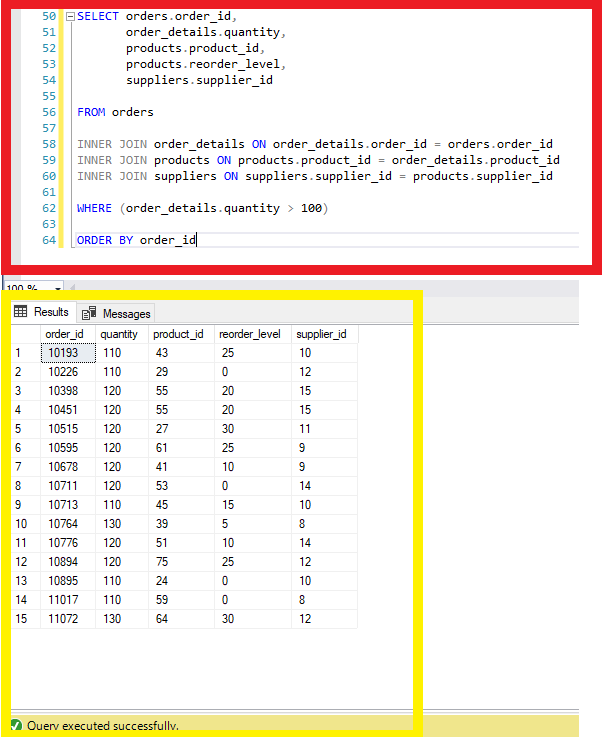
•ON WHERE I USED “LIKE” AND “[^S]%” TO NOT SHOW LETTERS S.

• “GROUP BY” TO SHOW THEM IN GROUPS.

•AND TO SHOW THAT ONES THAT HAVE JUST MORE THAN 2 I USED HAVING.

## **Q.9**

1. 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.



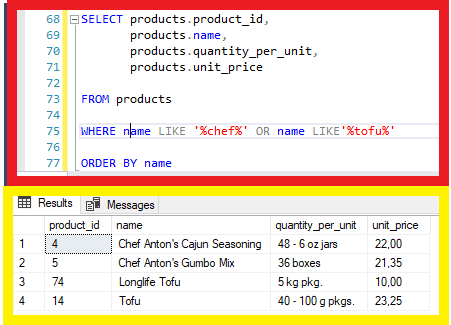
•RED – SCRIPT THAT WAS INSERTED

•YELLOW – MESSAGE PRODUCED

•THIS QUESTION I SLECTED THE COLUMNS AND MADE HIS RESPCTIVES INNER JOINS AND TO SHOW ONLY QUANTITY GREATER THAN 100 I USED WHERE WITH “>”.

## **Q.10**

1. 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.



•RED – SCRIPT THAT WAS INSERTED

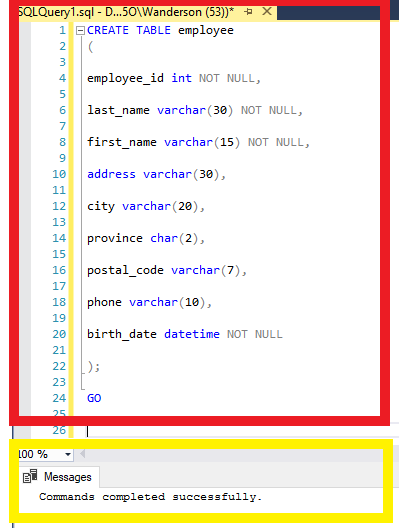
•YELLOW – MESSAGE PRODUCED

•FOR THIS QUESTION I SELECTED COLUMNS AND TO SHOW ONLY THAT NAMES WHERE HAVE “CHEF” OR “TOFU” I USED “WHERE” WITH “LIKE” CONDITION AND “%LOOKFOR%” SO THE SISTEM WILL LOOK JUST FOR NAMES THAT HAVE “LOOKFOR”.

# **Part C - INSERT, UPDATE, DELETE and VIEWS Statements**

## **Q.1**

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



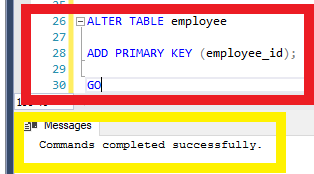
•RED – SCRIPT THAT WAS INSERTED

•YELLOW – MESSAGE PRODUCED

• SAME AS USED ON PART A WE USE “CREATE TABLE” FOLLOWING FOR THE COLUMNS.

## **Q.2**

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

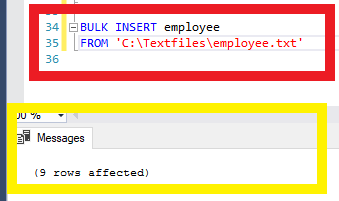


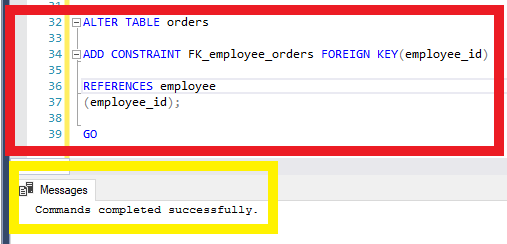
•RED – SCRIPT THAT WAS INSERTED

•YELLOW – MESSAGE PRODUCED

## **Q.3**

1. 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.



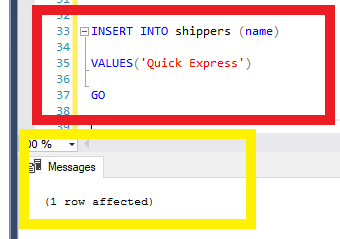


•RED – SCRIPT THAT WAS INSERTED

•YELLOW – MESSAGE PRODUCE

## **Q.4**

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



•RED – SCRIPT THAT WAS INSERTED

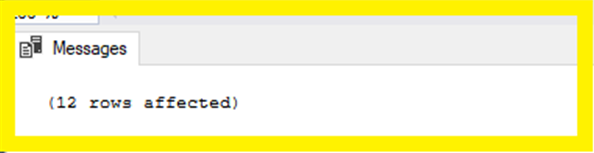
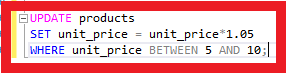
•YELLOW – MESSAGE PRODUCED

•HERE WE ARE USING “INSERT INTO” TO INSERT SOME VALUE AT THE “TABLE”

FOLLOWING BY “VALUE” THAT YOU MUST INSERT THE VALUE.

## **Q.5**

1. 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.



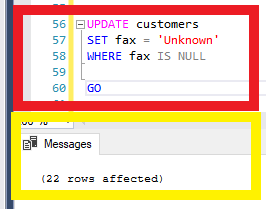
•RED – SCRIPT THAT WAS INSERTED

•YELLOW – MESSAGE PRODUCED

•FOR “UPDATE” WE CAN SET SOME RULE AS YOU CAN SEE.

## **Q.6**

1. 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.

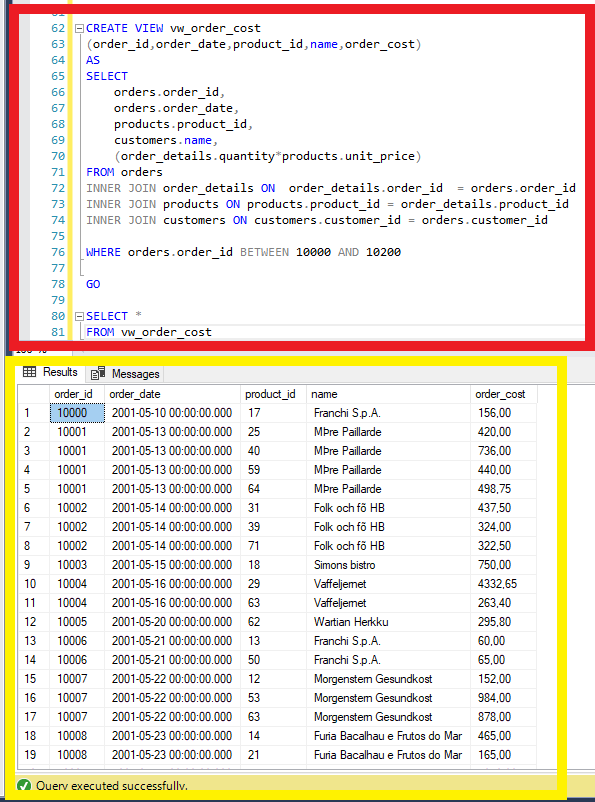


•RED – SCRIPT THAT WAS INSERTED

•YELLOW – MESSAGE PRODUCED

## **Q.7**

1. 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**. The view should produce the result set listed below.



•RED – SCRIPT THAT WAS INSERTED

•YELLOW – MESSAGE PRODUCED

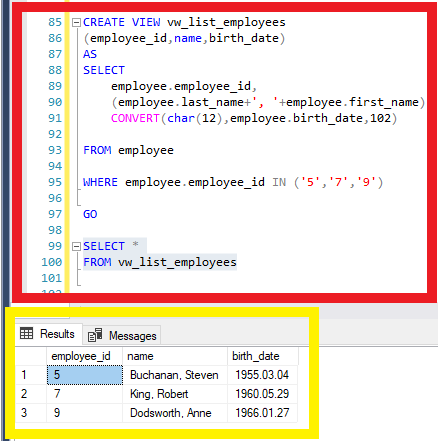
•CREATE A VIEW WILL HELP YOU TO EXECUTE JUST THIS VIEW COMAND TO SEE SOME CONTENT THAT WAS DEFINED BEFORE.

•AS YOU SEE ON IMAGE, WE CREATED A VIEW TO SEE ORDERS BETWEEN 10000 AND 10200,

SO, YOU CAN UPDATE THE DATA AND YOU CAN JUST EXECUTE THE VIEW TO SHOW RESULTS.

## **Q.8**

1. 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.



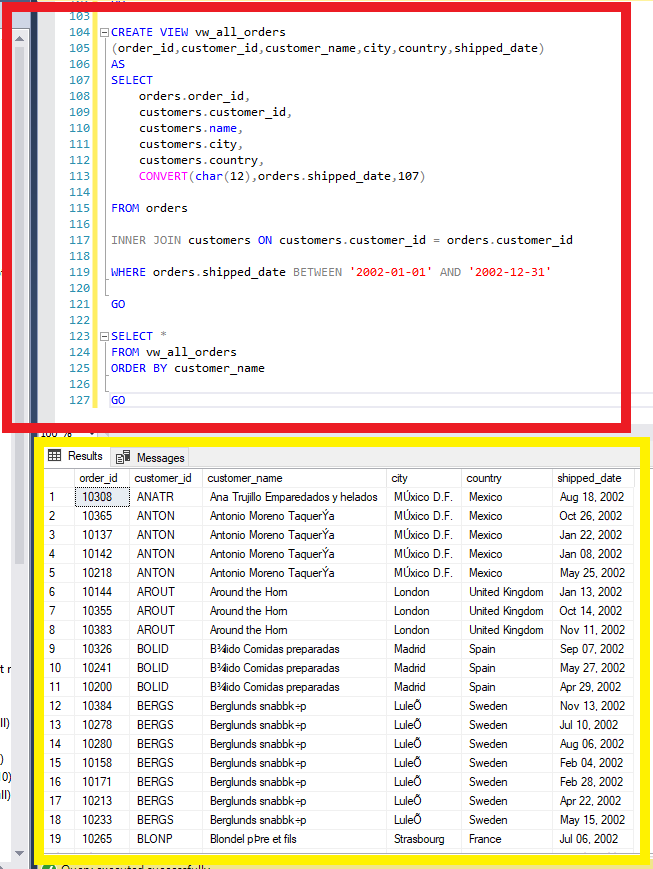
•RED – SCRIPT THAT WAS INSERTED

•YELLOW – MESSAGE PRODUCED

•THIS VIEW WAS CREATED FOR SEE EMPLOYEES WITH ID “5,7,9”.

## **Q.9**

1. 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.



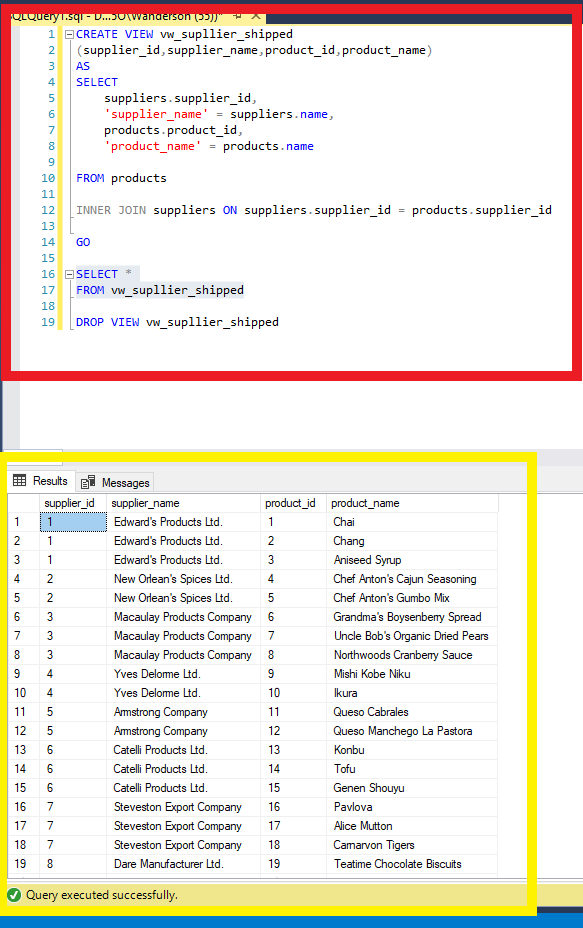
•RED – SCRIPT THAT WAS INSERTED

•YELLOW – MESSAGE PRODUCED

•THIS VIEW WAS CREATED TO SEE ORDERS THAT WAS SHIPPED BETWEEN THE DATE THAT WAS DETERMINED.

## **Q.10**

1. 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.*



•RED – SCRIPT THAT WAS INSERTED

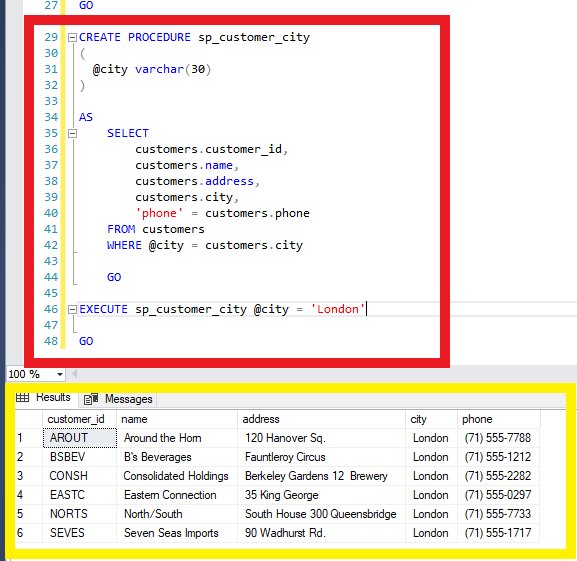
•YELLOW – MESSAGE PRODUCED

•VIEW CREATED TO SEE SUPPLIERS AND THE ITEMS THAT THEY HAVE SHIPPED.

# **Part D - Stored Procedures and Triggers**

# **Q.1**

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.



•RED – SCRIPT THAT WAS INSERTED

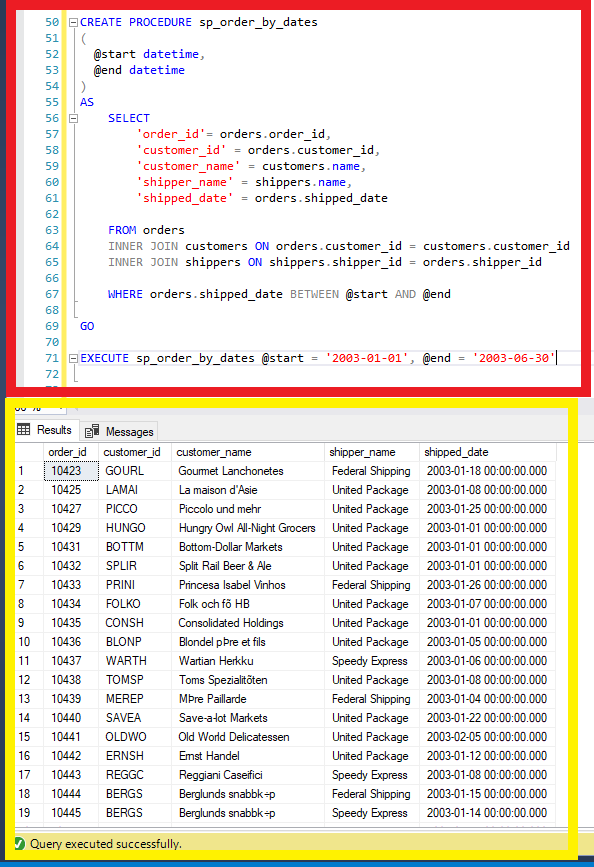
•YELLOW – MESSAGE PRODUCED

•”PROCEDURE” IS CREATE TO SHOW DETERMINED VALUE THAT YOU CAN INSERT WHEN EXECUTE.

•IN THIS CASE THE VALUE WILL BE CITY, SO EVERY TIME THAT YOU WILL EXECUTE THIS PROCEDURE YOU MUST DETERMINE THE CITY SO THE RESULTS WILL BE ONLY THE CUSTOMERS THAT LIVES IN THE CITY INSERTED.

## **Q.2**

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



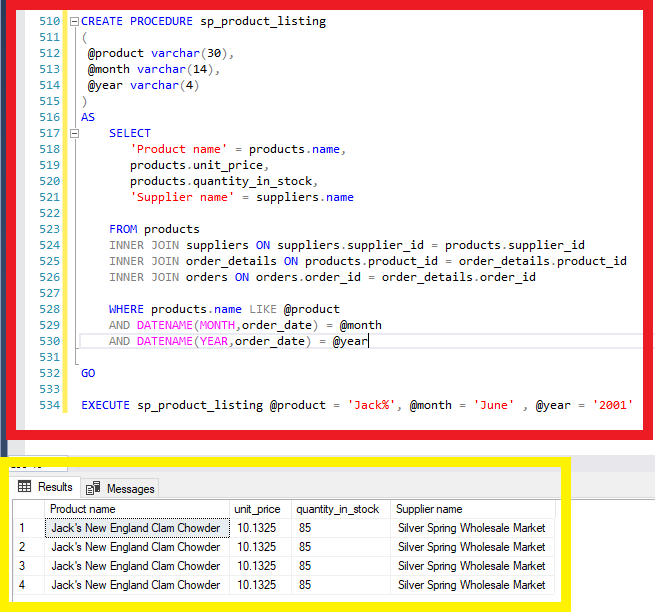
•RED – SCRIPT THAT WAS INSERTED

•YELLOW – MESSAGE PRODUCED

•THIS PROCEDURE YOU WILL FIND A ORDER BETWEEN SOME DATES SO YOU MUST DETERMINE WHEN EXECUTE THE START DATE END THE END DATE.

## **Q.3**

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



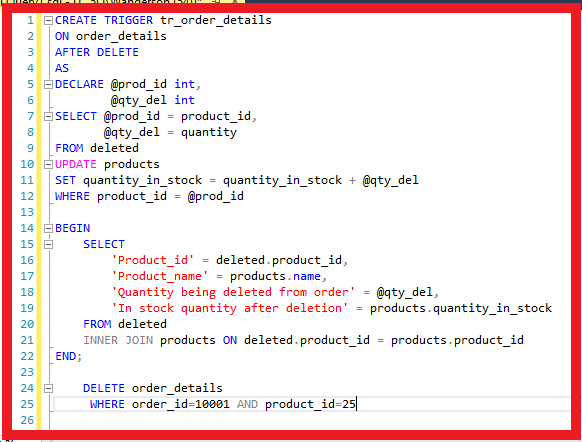
•RED – SCRIPT THAT WAS INSERTED

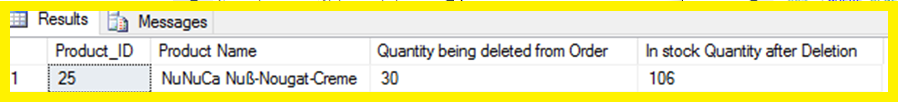
•YELLOW – MESSAGE PRODUCED

•THIS PROCEDURE WILL BE ABLE TO SHOW A PRODUCT LIST WITH ESPECIFIC NAME AND MONTH AND YEAR.

## **Q.4**

1. Create a **DELETE** trigger on the order\_details table to display the information shown below when you issue the following statement:





•RED – SCRIPT THAT WAS INSERTED

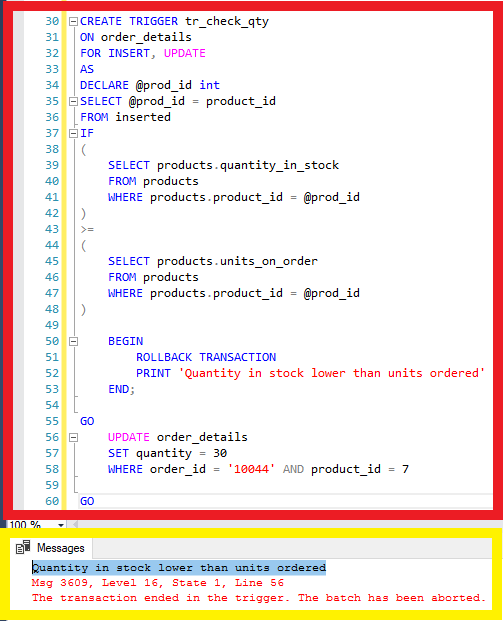
•YELLOW – MESSAGE PRODUCED

•”TRIGGER” IS AN AUTO EXECUTABLE COMMAND THAT WILL BE EXECUTED EVERY TIME THAT YOU DO SOMETHING THAT WAS DETERMINED, LIKE INSERT, DELETE, UPDATE.

•IN THIS CASE EVERY TIME THAT YOU DELETE SOME PRODUCT QUANTITY THIS WILL SHOW YOU THE QUANTITY DELETED AND QUANTITY IN STOCK AFTER DELETE.

## **Q.5**

1. 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.



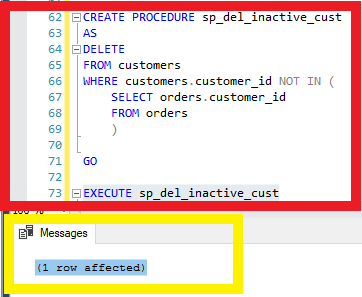
•RED – SCRIPT THAT WAS INSERTED

•YELLOW – MESSAGE PRODUCED

•THIS TRIGGER WAS CREATED TO INSERT AND UPDATE TABLE IF THE FOLLOWING RULES ARE ABLE TO EXECUTE. IN THIS CASE THE QUANTITY IN STOCK SHOULD BE GREATER OR EQUAL TO UNITS ON ORDER.

## **Q.6**

1. Create a stored procedure called **sp\_del\_inactive\_cust** to **delete** customers that have no orders. The stored procedure should delete **1** row.



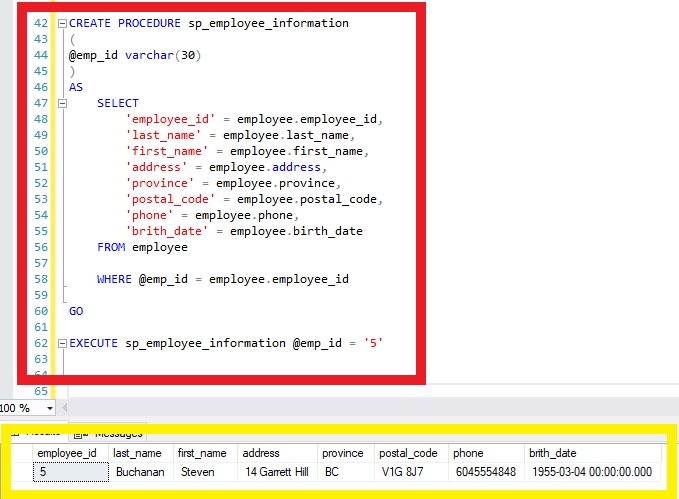
•RED – SCRIPT THAT WAS INSERTED

•YELLOW – MESSAGE PRODUCED

•THIS PROCEDURE INCLUDES A TRIGGER TO DELETE EVERY CUSTOMER THAT HAVE NO ORDERS.

## **Q.7**

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



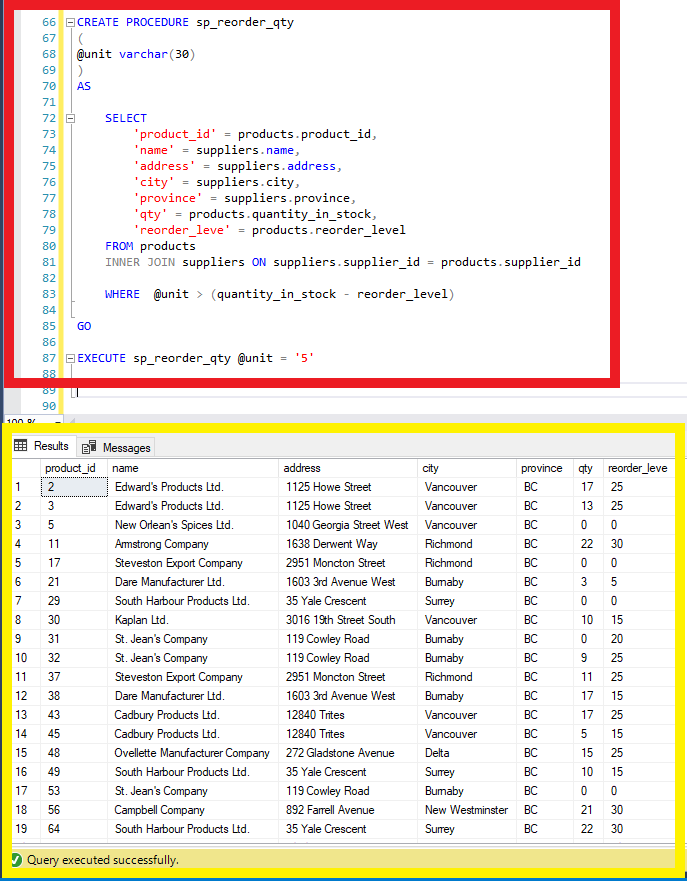
•RED – SCRIPT THAT WAS INSERTED

•YELLOW – MESSAGE PRODUCED

•THIS PROCEDURE WILL SHOW INFORMATION ABOUT A SPECIFIC EMPLOYEE ID.

## **Q.8**

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

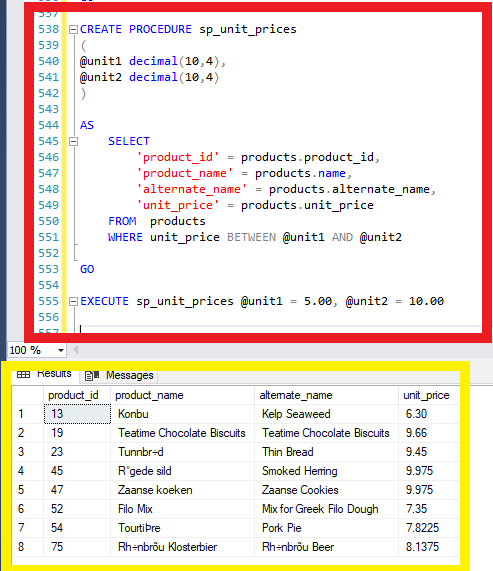


•RED – SCRIPT THAT WAS INSERTED

•YELLOW – MESSAGE PRODUCED

## **Q.9**

1. 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.



•RED – SCRIPT THAT WAS INSERTED

•YELLOW – MESSAGE PRODUCED

•THIS PROCEDURE WILL SHOW PRODUCTS BETWEEN THE PRICES THAT YOU CAN INPUT WHEN EXECUTED.

# **SCRIP COPY – ALL QUESTIONS SCRIPTS**

USE MASTER

GO

if exists (select \* from sysdatabases where name='Cus\_Orders')

begin

raiserror('Dropping existing Cus\_Orders ....',0,1)

DROP database Cus\_Orders

end

GO

-- 1)

CREATE DATABASE Cus\_Orders

GO

--2)

USE Cus\_orders

GO

CREATE TABLE customers

(

customer\_id char(5) NOT NULL,

name varchar(50) NOT NULL,

contact\_name varchar(30),

title\_id char(3) NOT NULL,

address varchar(50),

city varchar(20),

region varchar(15),

country\_code varchar(10),

country varchar(15),

phone varchar(20),

fax varchar(20)

);

GO

CREATE TABLE orders

(

order\_id int NOT NULL,

customer\_id char(5) NOT NULL,

employee\_id int NOT NULL,

shipping\_name varchar(50),

shipping\_address varchar(50),

shipping\_city varchar(20),

shipping\_region varchar(15),

shipping\_country\_code varchar(10),

shipping\_country varchar(15),

shipper\_id int NOT NULL,

order\_date datetime,

required\_date datetime,

shipped\_date datetime,

freight\_charge money

);

GO

CREATE TABLE order\_details

(

order\_id int NOT NULL,

product\_id int NOT NULL,

quantity int NOT NULL,

discount float NOT NULL

);

GO

CREATE TABLE products

(

product\_id int NOT NULL,

supplier\_id int NOT NULL,

name varchar(40) NOT NULL,

alternate\_name varchar(40),

quantity\_per\_unit varchar(25),

unit\_price money,

quantity\_in\_stock int,

units\_on\_order int,

reorder\_level int,

);

GO

CREATE TABLE shippers

(

shipper\_id int IDENTITY NOT NULL,

name varchar(20) NOT NULL

);

GO

CREATE TABLE suppliers

(

supplier\_id int IDENTITY NOT NULL,

name varchar(40) NOT NULL,

address varchar(30),

city varchar(20),

province char(2)

);

GO

CREATE TABLE titles

(

title\_id char(3) NOT NULL,

description varchar(35) NOT NULL,

);

-- 4)

GO

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

ALTER TABLE customers

ADD CONSTRAINT FK\_customer\_title FOREIGN KEY(title\_id)

REFERENCES titles

(title\_id);

GO

ALTER TABLE orders

ADD CONSTRAINT FK\_orders\_customers 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\_order\_dt\_orders FOREIGN KEY(order\_id)

REFERENCES orders

(order\_id);

GO

ALTER TABLE order\_details

ADD CONSTRAINT FK\_oder\_dt\_products FOREIGN KEY(product\_id)

REFERENCES products

(product\_id);

GO

ALTER TABLE products

ADD CONSTRAINT FK\_products\_suppliers FOREIGN KEY(supplier\_id)

REFERENCES suppliers

(supplier\_id);

GO

-- 5)

ALTER TABLE customers

ADD CONSTRAINT df\_country

DEFAULT 'Canada' FOR country;

GO

ALTER TABLE orders

ADD CONSTRAINT df\_required\_date

DEFAULT DATEADD(day,10,GETDATE()) FOR required\_date;

GO

ALTER TABLE order\_details

ADD CONSTRAINT df\_details

CHECK (quantity >= 1);

GO

ALTER TABLE products

ADD CONSTRAINT df\_products\_reorder

CHECK (reorder\_level >= 1);

GO

ALTER TABLE products

ADD CONSTRAINT df\_products\_quantity

CHECK (reorder\_level < 150);

GO

ALTER TABLE suppliers

ADD CONSTRAINT df\_province

DEFAULT 'BC' FOR province;

GO

-- PART B 1)

SELECT customer\_id,

name,

city,

country

FROM customers

ORDER BY customer\_id

GO

-- 2)

AlTER TABLE customers

ADD active bit;

GO

ALTER TABLE customers

ADD CONSTRAINT df\_active

DEFAULT 1 FOR active;

GO

-- 3)

SELECT 'order\_id' = orders.order\_id,

'product\_name' = products.name,

'customer\_name' = customers.name,

'order\_date' = CONVERT(char(12),orders.order\_date,107),

'new\_shipped\_date' =CONVERT(char(12),DATEADD(day,7,orders.shipped\_date),107),

'order\_cost' = (products.unit\_price \* order\_details.quantity)

FROM orders

INNER JOIN order\_details ON order\_details.order\_id = orders.order\_id

INNER JOIN products ON order\_details.product\_id = products.product\_id

INNER JOIN customers ON customers.customer\_id = orders.customer\_id

WHERE order\_date BETWEEN '2001-01-01' AND '2001-12-31';

-- 4)

GO

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)

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)

SELECT 'supplier\_name' = suppliers.name,

'product\_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 supplier\_name

GO

-- 7)

SELECT orders.order\_id,

customers.name,

customers.contact\_name,

'shipped\_date' = CONVERT(char(12),orders.shipped\_date,107),

'elapsed' = - DATEDIFF(yy, 'Jan 1 2008', orders.shipped\_date)

from orders

INNER JOIN customers ON customers.customer\_id = orders.customer\_id

WHERE shipped\_date IS NOT NULL

ORDER BY order\_id

GO

-- 8)

SELECT 'name' = SUBSTRING(customers.name, 1, 1),

'total'= COUNT(\*)

FROM customers

WHERE name LIKE '[^S]%'

GROUP BY SUBSTRING(customers.name,1,1)

HAVING COUNT(\*) > 2

GO

-- 9)

SELECT orders.order\_id,

order\_details.quantity,

products.product\_id,

products.reorder\_level,

suppliers.supplier\_id

FROM orders

INNER JOIN order\_details ON order\_details.order\_id = orders.order\_id

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\_id

GO

-- 10)

SELECT products.product\_id,

products.name,

products.quantity\_per\_unit,

products.unit\_price

FROM products

WHERE name LIKE '%chef%' OR name LIKE'%tofu%'

ORDER BY name

GO

-- PART C 1)

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)

ALTER TABLE employee

ADD PRIMARY KEY (employee\_id);

GO

--BULK INSERT BEFORE 3)

ALTER TABLE orders

ADD CONSTRAINT FK\_employee\_orders FOREIGN KEY(employee\_id)

REFERENCES employee

(employee\_id);

GO

-- 4)

INSERT INTO shippers (name)

VALUES('Quick Express')

GO

-- 5)

UPDATE products

SET unit\_price = unit\_price\*1.05

WHERE unit\_price BETWEEN 5 AND 10;

GO

-- 6)

UPDATE customers

SET fax = 'Unknown'

WHERE fax IS NULL

GO

-- 7)

CREATE VIEW vw\_order\_cost

(order\_id,order\_date,product\_id,name,order\_cost)

AS

SELECT

orders.order\_id,

orders.order\_date,

products.product\_id,

customers.name,

(order\_details.quantity\*products.unit\_price)

FROM orders

INNER JOIN order\_details ON order\_details.order\_id = orders.order\_id

INNER JOIN products ON products.product\_id = order\_details.product\_id

INNER JOIN customers ON customers.customer\_id = orders.customer\_id

WHERE orders.order\_id BETWEEN 10000 AND 10200

GO

SELECT \*

FROM vw\_order\_cost

GO

-- 8)

CREATE VIEW vw\_list\_employees

(employee\_id,name,birth\_date)

AS

SELECT

employee.employee\_id,

(employee.last\_name+', '+employee.first\_name),

CONVERT(char(12),employee.birth\_date,102)

FROM employee

WHERE employee.employee\_id IN ('5','7','9')

GO

SELECT \*

FROM vw\_list\_employees

GO

-- 9)

CREATE VIEW vw\_all\_orders

(order\_id,customer\_id,customer\_name,city,country,shipped\_date)

AS

SELECT

orders.order\_id,

customers.customer\_id,

customers.name,

customers.city,

customers.country,

CONVERT(char(12),orders.shipped\_date,107)

FROM orders

INNER JOIN customers ON customers.customer\_id = orders.customer\_id

WHERE orders.shipped\_date BETWEEN '2002-01-01' AND '2002-12-31'

GO

SELECT \*

FROM vw\_all\_orders

ORDER BY customer\_name

GO

-- 10)

CREATE VIEW vw\_supllier\_shipped

(supplier\_id,supplier\_name,product\_id,product\_name)

AS

SELECT

suppliers.supplier\_id,

suppliers.name,

products.product\_id,

products.name

FROM products

INNER JOIN suppliers ON suppliers.supplier\_id = products.supplier\_id

GO

SELECT \*

FROM vw\_supllier\_shipped

GO

-- PART D 1)

CREATE PROCEDURE sp\_customer\_city

(

@city varchar(30)

)

AS

SELECT

customers.customer\_id,

customers.name,

customers.address,

customers.city,

'phone' = customers.phone

FROM customers

WHERE @city = customers.city

GO

EXECUTE sp\_customer\_city @city = 'London'

GO

-- 2)

CREATE PROCEDURE sp\_order\_by\_dates

(

@start datetime,

@end datetime

)

AS

SELECT

'order\_id'= orders.order\_id,

'customer\_id' = orders.customer\_id,

'customer\_name' = customers.name,

'shipper\_name' = shippers.name,

'shipped\_date' = orders.shipped\_date

FROM orders

INNER JOIN customers ON orders.customer\_id = customers.customer\_id

INNER JOIN shippers ON shippers.shipper\_id = orders.shipper\_id

WHERE orders.shipped\_date BETWEEN @start AND @end

GO

EXECUTE sp\_order\_by\_dates @start = '2003-01-01', @end = '2003-06-30'

GO

-- 3)

CREATE PROCEDURE sp\_product\_listing

(

@product varchar(30),

@month varchar(14),

@year varchar(4)

)

AS

SELECT

'Product name' = products.name,

products.unit\_price,

products.quantity\_in\_stock,

'Supplier name' = suppliers.name

FROM products

INNER JOIN suppliers ON suppliers.supplier\_id = products.supplier\_id

INNER JOIN order\_details ON products.product\_id = order\_details.product\_id

INNER JOIN orders ON orders.order\_id = order\_details.order\_id

WHERE products.name LIKE @product

AND DATENAME(MONTH,order\_date) = @month

AND DATENAME(YEAR,order\_date) = @year

GO

EXECUTE sp\_product\_listing @product = 'Jack%', @month = 'June' , @year = '2001'

GO

-- 4)

CREATE TRIGGER tr\_order\_details

ON order\_details

AFTER DELETE

AS

DECLARE @prod\_id int,

@qty\_del int

SELECT @prod\_id = product\_id,

@qty\_del = quantity

FROM deleted

UPDATE products

SET quantity\_in\_stock = quantity\_in\_stock + @qty\_del

WHERE product\_id = @prod\_id

BEGIN

SELECT

'Product\_id' = deleted.product\_id,

'Product\_name' = products.name,

'Quantity being deleted from order' = @qty\_del,

'In stock quantity after deletion' = products.quantity\_in\_stock

FROM deleted

INNER JOIN products ON deleted.product\_id = products.product\_id

END;

GO

DELETE order\_details

WHERE order\_id=10001 AND product\_id=25

GO

-- 5)

CREATE TRIGGER tr\_check\_qty

ON order\_details

FOR INSERT, UPDATE

AS

DECLARE @prod\_id int

SELECT @prod\_id = product\_id

FROM inserted

IF

(

SELECT products.quantity\_in\_stock

FROM products

WHERE products.product\_id = @prod\_id

)

>=

(

SELECT products.units\_on\_order

FROM products

WHERE products.product\_id = @prod\_id

)

BEGIN

ROLLBACK TRANSACTION

PRINT 'Quantity in stock lower than units ordered'

END;

GO

UPDATE order\_details

SET quantity = 30

WHERE order\_id = '10044' AND product\_id = 7

GO

-- 6)

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

GO

-- 7)

CREATE PROCEDURE sp\_employee\_information

(

@emp\_id varchar(30)

)

AS

SELECT

'employee\_id' = employee.employee\_id,

'last\_name' = employee.last\_name,

'first\_name' = employee.first\_name,

'address' = employee.address,

'province' = employee.province,

'postal\_code' = employee.postal\_code,

'phone' = employee.phone,

'brith\_date' = employee.birth\_date

FROM employee

WHERE @emp\_id = employee.employee\_id

GO

EXECUTE sp\_employee\_information @emp\_id = '5'

GO

-- 8)

CREATE PROCEDURE sp\_reorder\_qty

(

@unit varchar(30)

)

AS

SELECT

'product\_id' = products.product\_id,

'name' = suppliers.name,

'address' = suppliers.address,

'city' = suppliers.city,

'province' = suppliers.province,

'qty' = products.quantity\_in\_stock,

'reorder\_leve' = products.reorder\_level

FROM products

INNER JOIN suppliers ON suppliers.supplier\_id = products.supplier\_id

WHERE @unit > (quantity\_in\_stock - reorder\_level)

GO

EXECUTE sp\_reorder\_qty @unit = '5'

GO

-- 9)

CREATE PROCEDURE sp\_unit\_prices

(

@unit1 decimal(10,4),

@unit2 decimal(10,4)

)

AS

SELECT

'product\_id' = products.product\_id,

'product\_name' = products.name,

'alternate\_name' = products.alternate\_name,

'unit\_price' = products.unit\_price

FROM products

WHERE unit\_price BETWEEN @unit1 AND @unit2

GO

EXECUTE sp\_unit\_prices @unit1 = 5.00, @unit2 = 10.00

# BULK INSERT SCRIPT

BULK INSERT orders

FROM 'C:\Textfiles\orders.txt'

GO

BULK INSERT order\_details

FROM 'C:\Textfiles\order\_details.txt'

GO

BULK INSERT products

FROM 'C:\Textfiles\products.txt'

GO

BULK INSERT shippers

FROM 'C:\Textfiles\shippers.txt'

GO

BULK INSERT suppliers

FROM 'C:\Textfiles\suppliers.txt'

GO

BULK INSERT titles

FROM 'C:\Textfiles\titles.txt'

GO

BULK INSERT customers

FROM 'C:\Textfiles\customers.txt'

GO

BULK INSERT employee

FROM 'C:\Textfiles\employee.txt'