Version 1.0

PROJECT 2

RELATIONAL DATABASE DESIGN AND SQL

Project Report

Submitted to teacher Mark Bacchus

As a requirement of the COMP 1630 – CRN 59004

British Columbia Institute of Technology

July 2016

by: lILIEK SIDHARTA

A00821393

# Table of content

# Introduction

In this final project of COMP 1630 CRN 59004 we implement SQL (Structured Query Language) DDL (Data Definition Language) and DML (Data Manipulation Language) that we have learned in the class to manage structured data in a relational database.

The purpose of this implementation is to use DDL and DML to interact with a relational database. DDL (CREATE, ALTER, DROP) will be used for creating database, creating tables, defining tables’ primary keys and foreign keys, and setting the column constraints. DML (INSERT, UPDATE, DELETE) will be used for querying several tables to select records that have matching value; creating virtual table VIEW to allow users to access data without granting permission to access underlying tables; creating stored procedures and triggers which are very useful and can have parameters and passing output values.

The project gives 35 tasks to work on and provides table design and requirements. To accomplish these tasks, we implement SQL in Microsoft SQL Server 2016 Professional Edition on SQL Server Management Studio (SSMS) 17 environment. For reporting, SQL code is formatted using Poor Man’s T-SQL Formatter Extension on Visual Studio Code.

# DATABASE AND TABLES

## Create database Cus\_Orders

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

### sql statements

USE master

GO

*-- if there is any Cus\_Orders database, drop it*

IF EXISTS (

        SELECT \*

        FROM sysdatabases

        WHERE name = 'Cus\_Orders'

        )

BEGIN

    RAISERROR (

            'Dropping existing Cus\_Orders...',

            0,

            1

            )

    DROP DATABASE Cus\_Orders

END

GO

RAISERROR (

        'Creating Cus\_Orders database...',

        0,

        1

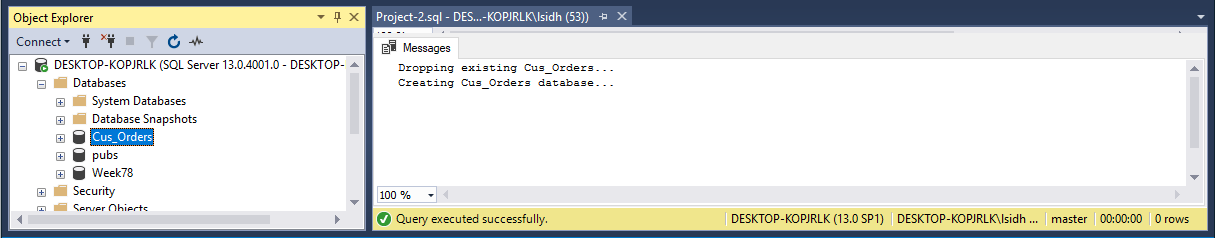
        )

*-- create a new Cus\_Orders database*

CREATE DATABASE Cus\_Orders

GO

### result



## Create a user defined data types

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.

There are three data types used by paimary keys of eight tables in database Cus\_Orders.

|  |  |  |
| --- | --- | --- |
| User-define Data Type | Data Type | For attributes |
| numid | integer | order\_id, product\_id, employee\_id, shipper\_id |
| char3id | char(3) | title\_id |
| char5id | char(5) | customer\_id |

### SQL Statements

EXECUTE sp\_addtype numid,

    'int',

    'NOT NULL';

EXECUTE sp\_addtype char3id,

    'char(3)',

    'NOT NULL';

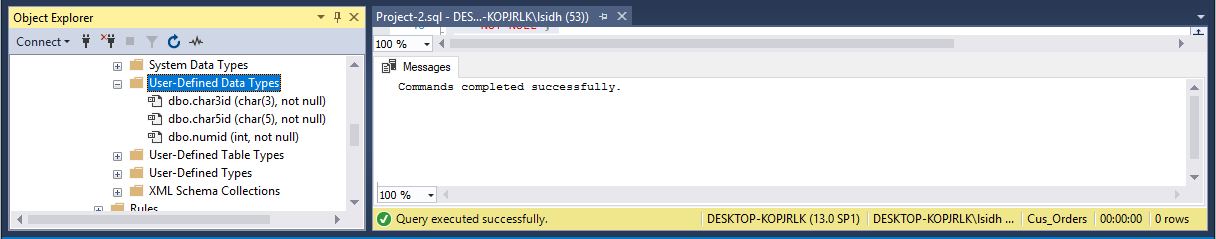
EXECUTE sp\_addtype char5id,

    'char(5)',

    'NOT NULL';

GO

### result



## Create seven tables in database cus\_orders

Create the following tables (see column information in Apendix A): customers, orders, order\_details, products, shippers, suppliers, titles.

### SQL Statements

CREATE TABLE customers (

    customer\_id char5id,

    name VARCHAR(50),

    contact\_name VARCHAR(30),

    title\_id char3id,

    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 numid,

    customer\_id char5id,

    employee\_id numid,

    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,

    order\_date DATETIME,

    required\_date DATETIME,

    shipped\_date DATETIME,

    freight\_charge MONEY

    );

CREATE TABLE order\_details (

    order\_id numid,

    product\_id numid,

    quantity INT,

    discount FLOAT

    );

CREATE TABLE products (

    product\_id numid,

    supplier\_id INT,

    name VARCHAR(40),

    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(1, 1),

    name VARCHAR(20)

    );

CREATE TABLE suppliers (

    supplier\_id INT IDENTITY(1, 1),

    name VARCHAR(40),

    address VARCHAR(30),

    city VARCHAR(20),

    province CHAR(2)

    );

CREATE TABLE titles (

    title\_id char3id,

    description VARCHAR(35)

    );

GO

Based on the table design requirement, we can construct the following table to help us defining the NOT NULL constraints.

### CONSTRAINT NOT NULL

|  |  |  |  |
| --- | --- | --- | --- |
| TABLE | COLUMN | DATA TYPE | CONSTRAINT TYPE |
| titles | description | varchar(35) | NOT NULL |
| customers | name | varchar(50) | NOT NULL |
| shippers | name | varchar(20) | NOT NULL |
| products | name | varchar(40) | NOT NULL |
| suppliers | name | varchar(40) | NOT NULL |
| order\_details | quantity | int | NOT NULL |
| order\_details | discount | float | NOT NULL |
| employee | last\_name | varchar(30) | NOT NULL |
| employee | first\_name | varchar(15) | NOT NULL |
| employee | birth\_date | datetime | NOT NULL |

*-- Set tables not null constraints*

ALTER TABLE titles

ALTER COLUMN description VARCHAR(35) NOT NULL;

ALTER TABLE customers

ALTER COLUMN name VARCHAR(50) NOT NULL;

ALTER TABLE shippers

ALTER COLUMN name VARCHAR(20) NOT NULL;

ALTER TABLE products

ALTER COLUMN name VARCHAR(40) NOT NULL;

ALTER TABLE suppliers

ALTER COLUMN name VARCHAR(40) NOT NULL;

ALTER TABLE order\_details

ALTER COLUMN quantity INT NOT NULL;

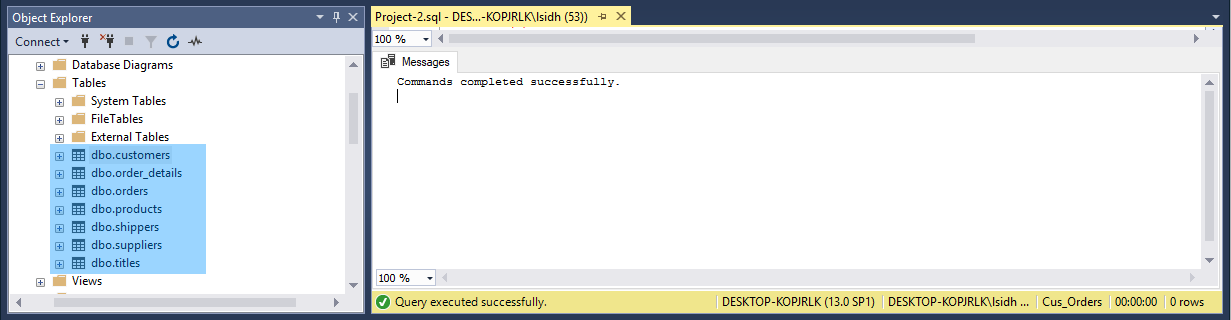
ALTER TABLE order\_details

ALTER COLUMN discount FLOAT NOT NULL;

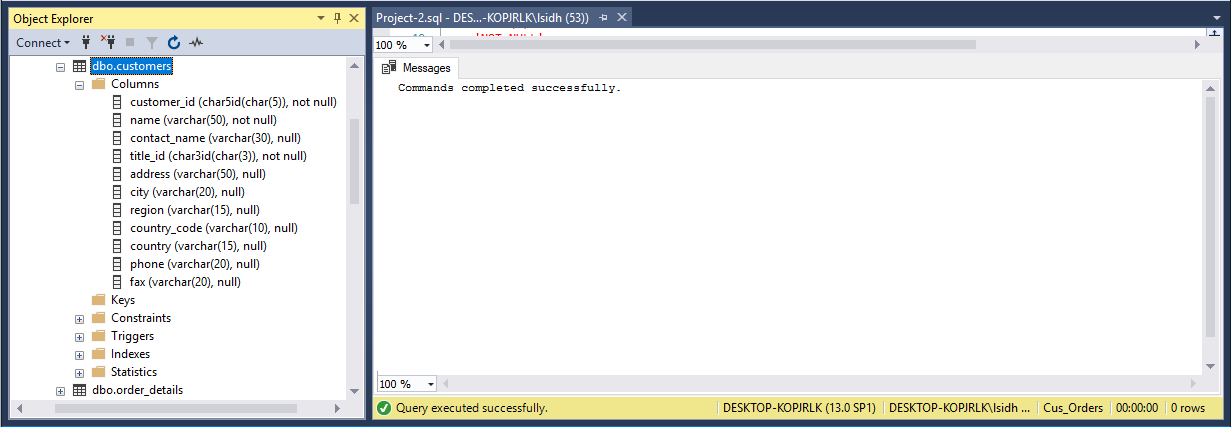
GO

### result

Seven new tables created are shown in the Object Explorer section of SSMS window.



The columns created together with the table are shown under the Column section. The picture below shows table customers, its columns, data type and column properties.



## Set primary keys and foreign keys

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

Based on the table design provided in the project requirements, we can construct these table before implement the SQL code to define primary keys and foreign keys.

### CONSTRAINT PRIMARY KEY

|  |  |  |
| --- | --- | --- |
| TABLE | COLUMN | CONSTRAINT TYPE |
| title | title\_id | PRIMARY KEY |
| customer | customer\_id | PRIMARY KEY |
| shippers | shipper\_id | PRIMARY KEY |
| suppliers | supplier\_id | PRIMARY KEY |
| products | product\_id | PRIMARY KEY |
| orders | order\_id | PRIMARY KEY |
| employee | employee\_id | PRIMARY KEY |

### CONSTRAINT FOREIGN KEY

|  |  |  |  |
| --- | --- | --- | --- |
| TABLE | COLUMN | REFERENCE TO | CONSTRAINT TYPE |
| customers | title\_id | Titles | FOREIGN KEY |
| products | supplier\_id | suppliers | FOREIGN KEY |
| orders | customer\_id | customers | FOREIGN KEY |
| orders | shipper\_id | shippers | FOREIGN KEY |
| order\_details | order\_id | orders | FOREIGN KEY |
| order\_details | product\_id | products | FOREIGN KEY |

### Sql statements

In this project, primary keys and foreign keys are defined using ALTER TABLE.

ALTER TABLE titles ADD PRIMARY KEY (title\_id);

ALTER TABLE customers ADD PRIMARY KEY (customer\_id);

ALTER TABLE customers ADD CONSTRAINT FK\_customers FOREIGN KEY (title\_id) REFERENCES titles (title\_id);

ALTER TABLE shippers ADD PRIMARY KEY (shipper\_id);

ALTER TABLE suppliers ADD PRIMARY KEY (supplier\_id);

ALTER TABLE products ADD PRIMARY KEY (product\_id);

ALTER TABLE products ADD CONSTRAINT FK\_products FOREIGN KEY (supplier\_id) REFERENCES suppliers (supplier\_id);

ALTER TABLE orders ADD PRIMARY KEY (order\_id);

ALTER TABLE orders ADD CONSTRAINT FK\_orders1 FOREIGN KEY (customer\_id) REFERENCES customers (customer\_id);

ALTER TABLE orders ADD CONSTRAINT FK\_orders2 FOREIGN KEY (shipper\_id) REFERENCES shippers (shipper\_id);

ALTER TABLE order\_details ADD CONSTRAINT PK\_order\_details PRIMARY KEY (

    order\_id,

    product\_id

    );

ALTER TABLE order\_details ADD CONSTRAINT FK\_order\_details1 FOREIGN KEY (order\_id) REFERENCES orders (order\_id);

ALTER TABLE order\_details ADD CONSTRAINT FK\_order\_details2 FOREIGN KEY (product\_id) REFERENCES products (product\_id);

GO

### RESULT

MMSM’s Object Explorer lists primary keys and foreign keys in a table’s key section. The following picture shows table **customers** and **order\_details** primary keys and foreign keys in two blue rectangles.

### 

## Set the constraints

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

### CONSTRAINT DEFAULT

|  |  |  |  |
| --- | --- | --- | --- |
| TABLE | COLUMN | Default value | CONSTRAINT TYPE |
| customers | country | Canada | DEFAULT |
| suppliers | province | BC | DEFAULT |
| orders | required\_date | getdate() + 10 | DEFAULT |

### CONSTRAINT CHECK

|  |  |  |  |
| --- | --- | --- | --- |
| TABLE | COLUMN | check value | CONSTRAINT TYPE |
| products | quantity\_in\_stock | <= 150 | CHECK |
| products | reorder\_level | >= 1 | CHECK |
| order\_details | quantity | >= 1 | CHECK |

### sql statements

*-- Set table customers' country default value*

ALTER TABLE customers ADD CONSTRAINT DF\_country DEFAULT 'Canada'

FOR country;

*-- Set table suppliers' province default value*

ALTER TABLE suppliers ADD CONSTRAINT DF\_province DEFAULT 'BC'

FOR province;

*-- Set table products constraint values*

ALTER TABLE products ADD CONSTRAINT CH\_stockqty CHECK (quantity\_in\_stock <= 150);

ALTER TABLE products ADD CONSTRAINT CH\_reorderlv CHECK (reorder\_level >= 1);

*-- Set table orders default values*

ALTER TABLE orders ADD CONSTRAINT DF\_date DEFAULT(**getdate**() + 10)

FOR required\_date;

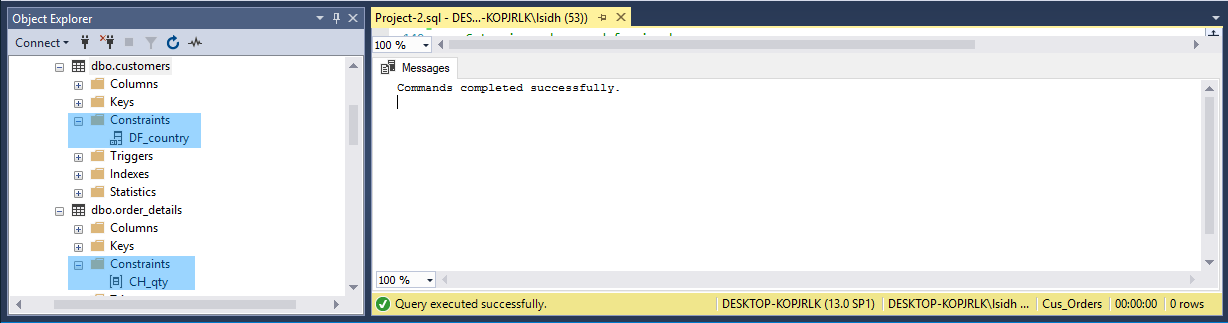
*-- Set table order\_details constraint values*

ALTER TABLE order\_details ADD CONSTRAINT CH\_qty CHECK (quantity >= 1);

GO

### Result

The **check** and **default** column constraints for a table are listed under constraints section of each table, as it is shown in the following table for **customers** and **order\_details** tables.



## Load data into created tables except table employees

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)

### SQL Statements

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,

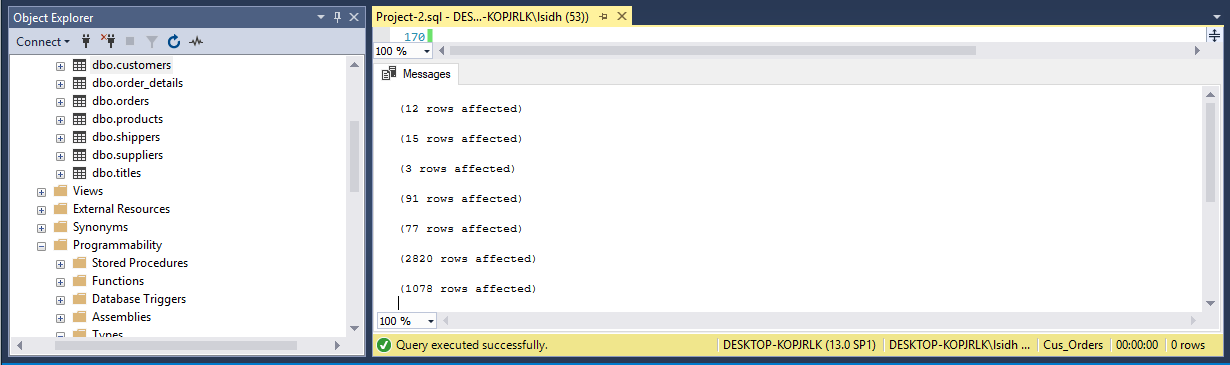
        ROWTERMINATOR = '\n'

        );

GO

### Result

The number of records added into the tables are displayed in the Messages section.



# sql statements

## List of table customers

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.

customer\_id name city country

--------------- ---------------------------------- ------------- ---------------

ALFKI Alfreds Futterkiste Berlin Germany

ANATR Ana Trujillo Emparedados y helados México D.F. Mexico

ANTON Antonio Moreno Taquería México D.F. Mexico

...

(91 row(s) affected)

### Sql statements

SELECT customer\_id,

    name,

    city,

    country

FROM customers

ORDER BY customer\_id;

GO

### Result

### 

## Add New column active to table customers.

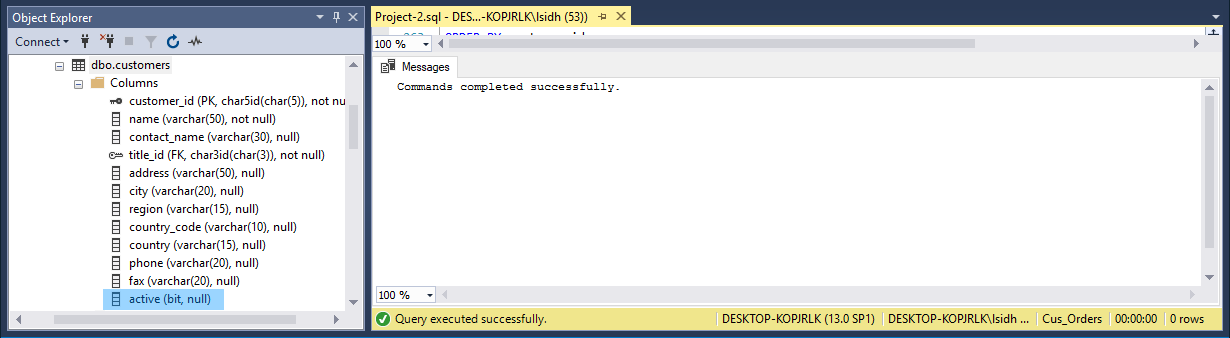
Add a new column called **active** to the customers table using the ALTER statement. The only valid values are 1 or 0. The default should be **1**.

### Sql statements

ALTER TABLE customers ADD active BIT DEFAULT 1;

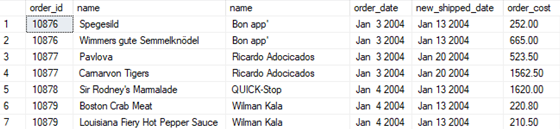
GO

### result



## List all the orders between January and February 2004

List all the orders where the order date is sometime in January or February 2004. 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.



…

(306 rows should be returned)

### sql statements

SELECT o.order\_id,

    p.name,

    c.name,

    'order\_date' = **CONVERT**(VARCHAR(12), o.order\_date, 107),

    'new shipped date' = **CONVERT**(VARCHAR(12), **DATEADD**(**DAY**, 7, o.shipped\_date), 107),

    'order\_cost' = (od.quantity - od.discount) \* p.unit\_price

FROM orders o

INNER JOIN order\_details od

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

INNER JOIN products p

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

INNER JOIN customers c

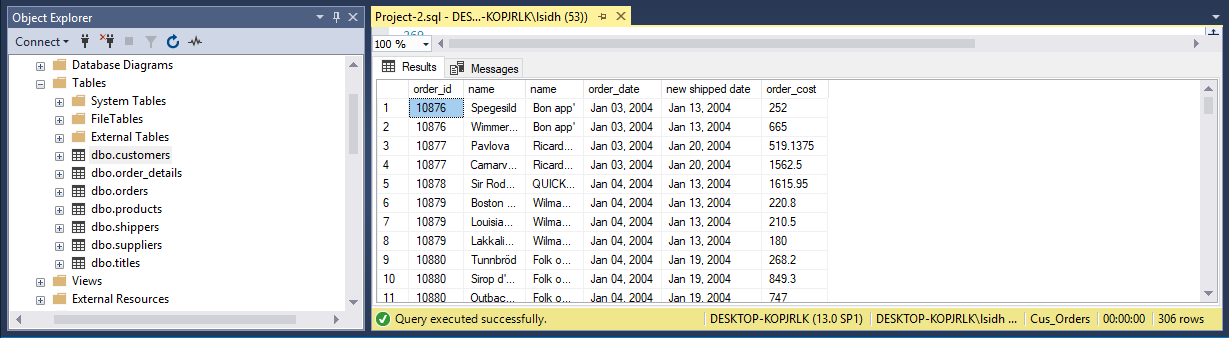
    ON orders.customer\_id = customers.customer\_id

WHERE orders.order\_date BETWEEN 'Jan 1, 2004'

        AND 'Feb 28, 2004';

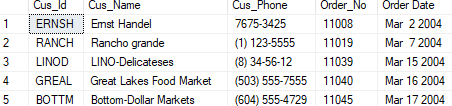
GO

### result



## List all the orders that have not been shipped

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 order date. The query should produce the result set listed below.



…

(21 row(s) affected)

### sql statements

SELECT c.customer\_id,

    c.name,

    c.phone,

    o.order\_id,

    'Order Date' = **CONVERT**(VARCHAR(12), o.order\_date, 109)

FROM customers c

INNER JOIN orders o

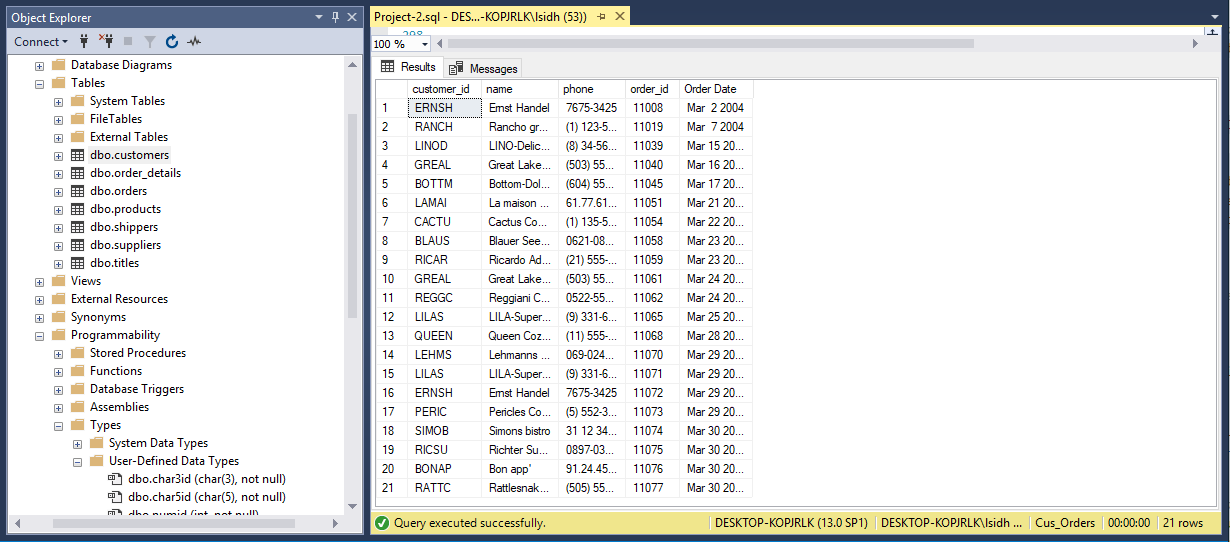
    ON c.customer\_id = o.customer\_id

WHERE o.shipped\_date IS NULL

ORDER BY o.order\_date;

GO

### result



## List all the customers where the region is null

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.

customer\_id name city description

-------------- ----------------------------------- --------------- ----------------------

ALFKI Alfreds Futterkiste Berlin Sales Representative

ANATR Ana Trujillo Emparedados y helados México D.F. Owner

ANTON Antonio Moreno Taquería México D.F. Owner

…

(60 row(s) affected)

### sql statements

SELECT customer\_id,

    name,

    city,

    t.description

FROM customers c

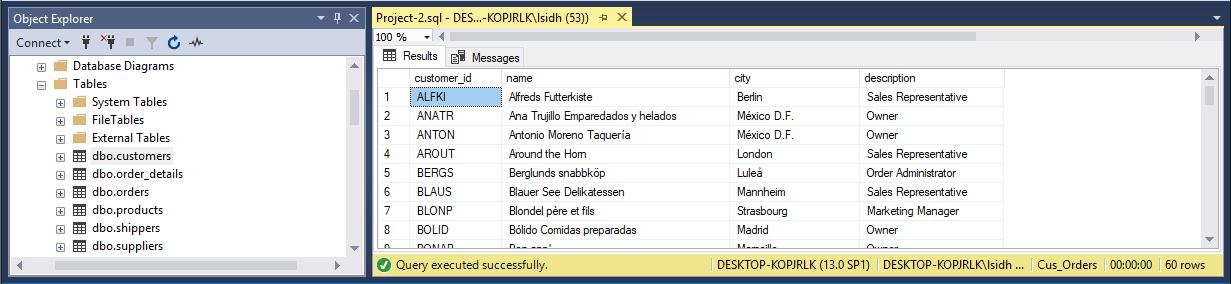
INNER JOIN titles t

    ON t.title\_id = c.title\_id

WHERE region IS NULL;

GO

### result



## List the products where the reorder level is higher than stock

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.

supplier\_name product\_name reorder\_level quantity\_in\_stock

------------------------ --------------------- ---------------- -----------------

Armstrong Company Queso Cabrales 30 22

Cadbury Products Ltd. Ipoh Coffee 25 17

Cadbury Products Ltd. Røgede sild 15 5

...

(18 row(s) affected)

### sql statements

SELECT s.name AS 'supplier\_name',

    p.name AS 'product\_name',

    p.reorder\_level,

    p.quantity\_in\_stock

FROM products p

INNER JOIN suppliers s

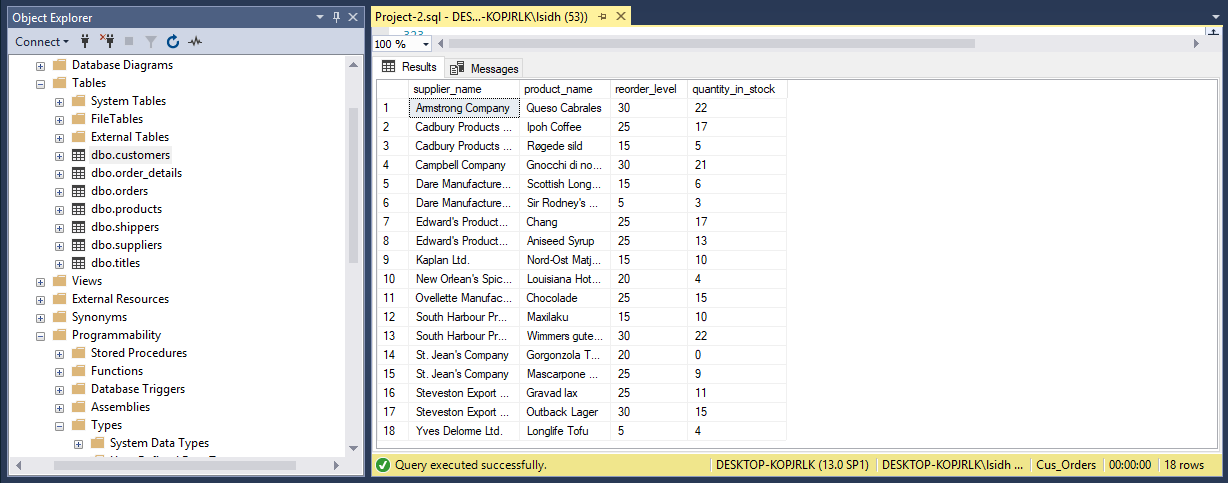
    ON s.supplier\_id = p.supplier\_id

WHERE reorder\_level > quantity\_in\_stock

ORDER BY s.name;

GO

### result



## Calculate the length in years for shipped orders

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.

order\_id name contact\_name shipped\_date elapsed

----------- --------------------------- --------------------------- ------------ -------

10000 Franchi S.p.A. Paolo Accorti May 15 2001 7

10001 Mère Paillarde Jean Fresnière May 23 2001 7

10002 Folk och fä HB Maria Larsson May 17 2001 7

...

(1057 row(s) affected)

### sql statements

SELECT o.order\_id,

    c.name,

    c.contact\_name,

**CONVERT**(VARCHAR(12), o.shipped\_date, 109),

**DATEDIFF**(**year**, o.shipped\_date, 'January 1, 2008') AS elapsed

FROM orders o

INNER JOIN customers c

    ON o.customer\_id = c.customer\_id

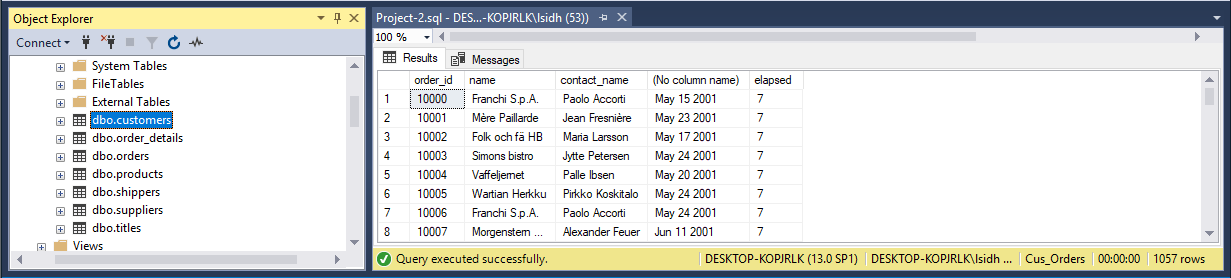
WHERE shipped\_date IS NOT NULL

ORDER BY o.order\_id,

    elapsed;

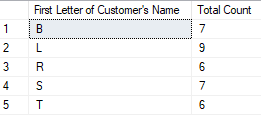
GO

### result



## List of customers’ names with their first letter

List number of customers with names beginning with each letter of the alphabet. Ignore customers whose name begins with the letter F or G. Do not display the letter and count unless **at least six** customer’s names begin with the letter. The query should produce the result set listed below.



### sql statements

SELECT LEFT(c.name, 1),

**COUNT**(LEFT(c.name, 1)) AS 'Total\_Count'

FROM customers c

WHERE (

        LEFT(c.name, 1) <> 'F'

        AND LEFT(c.name, 1) <> 'G'

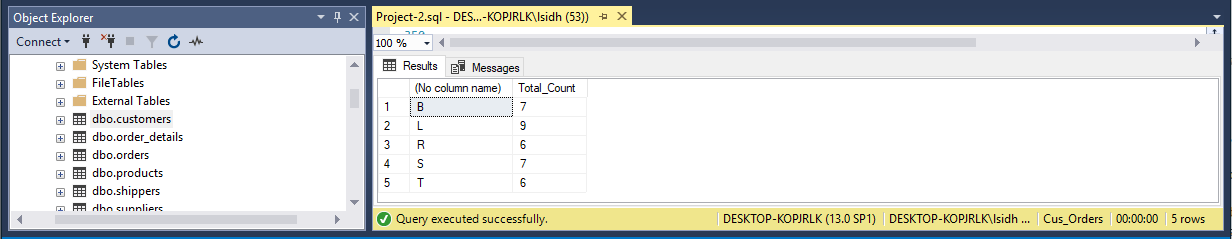
        )

GROUP BY (LEFT(c.name, 1))

HAVING **COUNT**(LEFT(c.name, 1)) >= 6;

GO

### result



## List the order details where the quantity is greater than 100

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.

order\_id quantity product\_id reorder\_level supplier\_id

----------- ----------- -------------- ----------------- -----------

10193 110 43 25 10

10226 110 29 0 12

10398 120 55 20 15

...

(15 row(s) affected)

### sql statements

SELECT od.order\_id,

    od.quantity,

    p.product\_id,

    p.reorder\_level,

    s.supplier\_id

FROM order\_details od

INNER JOIN products p

    ON od.product\_id = p.product\_id

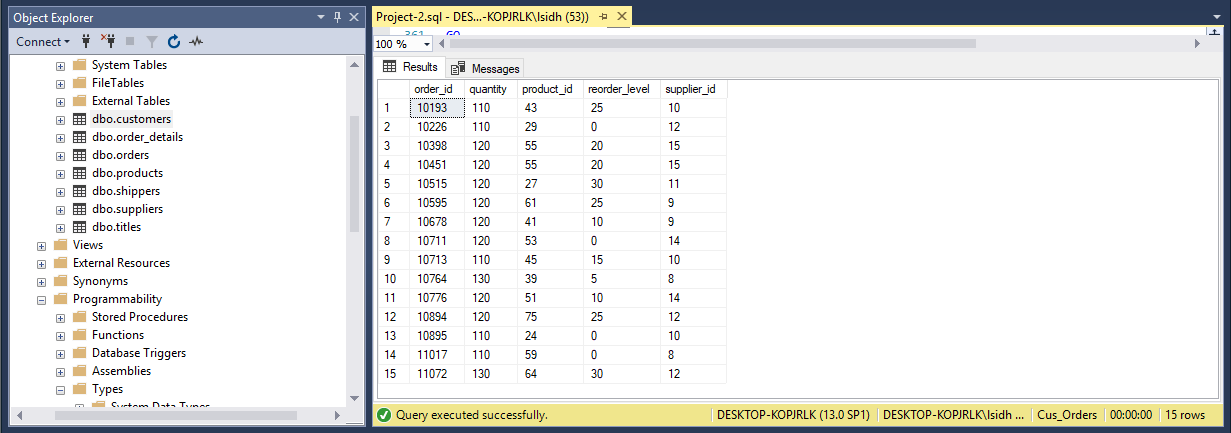
INNER JOIN suppliers s

    ON p.supplier\_id = s.supplier\_id

WHERE quantity > 100;

GO

### result



## List the products which contain ‘tofu’ or ‘chef’ in their name

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.

product\_id name quantity\_per\_unit unit\_price

-------------- ------------------------------------- ---------------------- --------------

4 Chef Anton's Cajun Seasoning 48 - 6 oz jars 22.0000

5 Chef Anton's Gumbo Mix 36 boxes 21.3500

74 Longlife Tofu 5 kg pkg. 10.0000

14 Tofu 40 - 100 g pkgs. 23.2500

(4 row(s) affected)

### sql statements

SELECT product\_id,

    name,

    quantity\_per\_unit,

    unit\_price

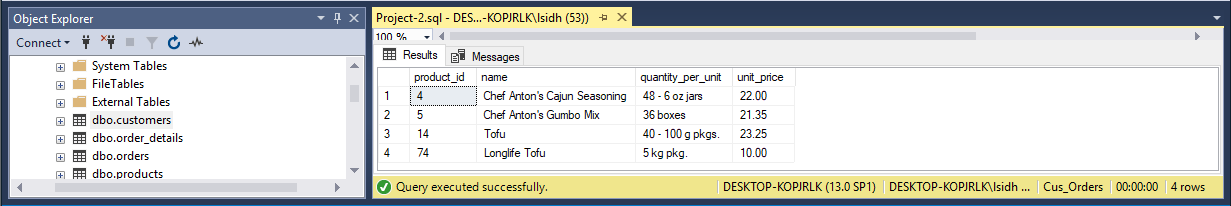
FROM products

WHERE name LIKE '%tofu%'

    OR name LIKE '%chef%';

GO

### result



# INSERT, UPDATE, DELETE AND VIEWS STATEMENTS

## Create employee table

### sql statements

CREATE TABLE employee (

    employee\_id numid NOT NULL,

    last\_name VARCHAR(30) NOT NULL,

    first\_name VARCHAR(15) NOT NULL,

    address VARCHAR(30),

    city VARCHAR(20),

    province VARCHAR(2),

    postal\_code VARCHAR(7),

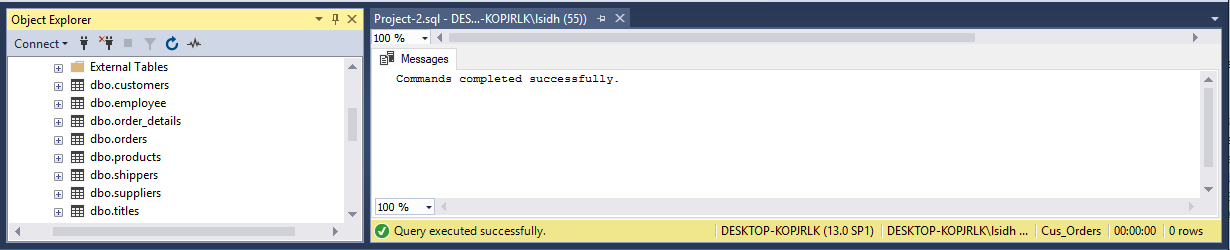
    phone VARCHAR(10),

    birth\_date DATETIME NOT NULL

    );

GO

### result



## Set primary key for the employee

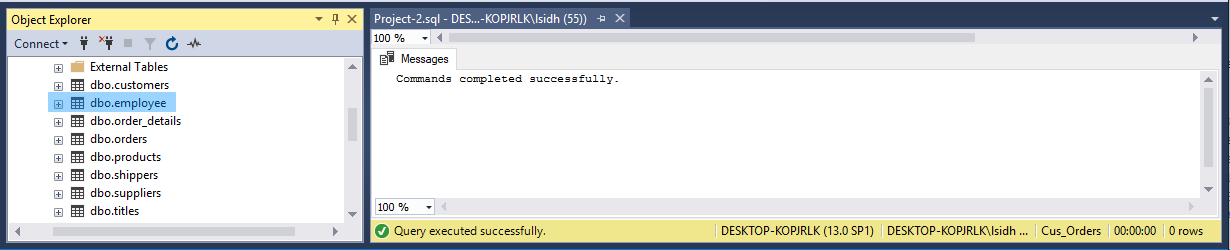
The **primary key** for the employee table should be the employee id

### sql statement

ALTER TABLE employee ADD PRIMARY KEY (employee\_id);

GO

### result



## Load the data into the employee table and create relationship

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.

### sql statements

BULK INSERT employee

FROM 'C:\TextFiles\employee.txt' WITH (

        CODEPAGE = 1252,

        DATAFILETYPE = 'char',

        FIELDTERMINATOR = '\t',

        KEEPNULLS,

        ROWTERMINATOR = '\n'

        );

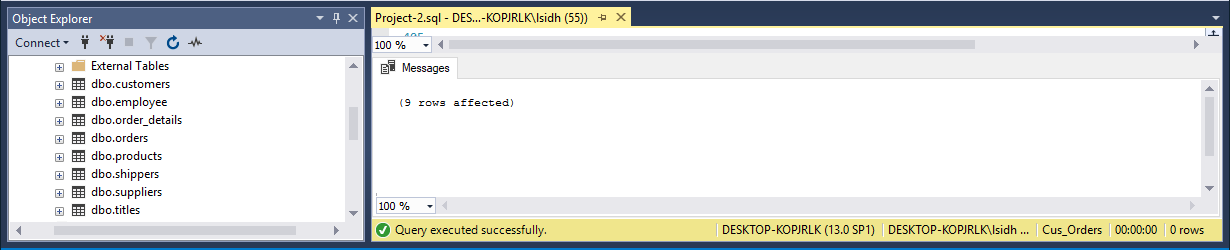
GO

*-- Create relationship between employee table and order table*

ALTER TABLE orders ADD CONSTRAINT FK\_orders3 FOREIGN KEY (employee\_id) REFERENCES employee (employee\_id);

GO

### result



## Add the shipper Quick Express to the shippers table

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

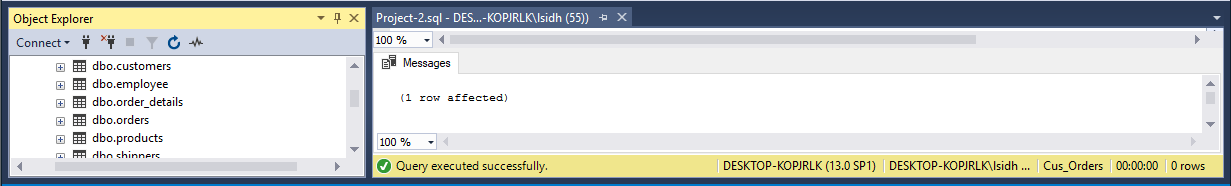
### sql statement

INSERT INTO shippers

VALUES ('Quick Express');

GO

### result



## Increase the unit price in the products table

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.

### sql statement

UPDATE products

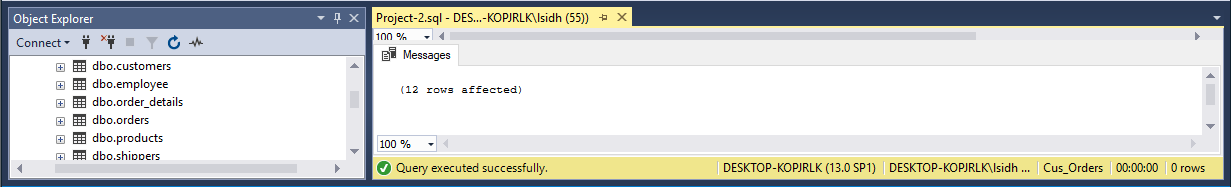
SET unit\_price = unit\_price \* 1.05

WHERE unit\_price BETWEEN 5.00

        AND 10.00;

GO

### result



## Change the fax value in the customers table

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.

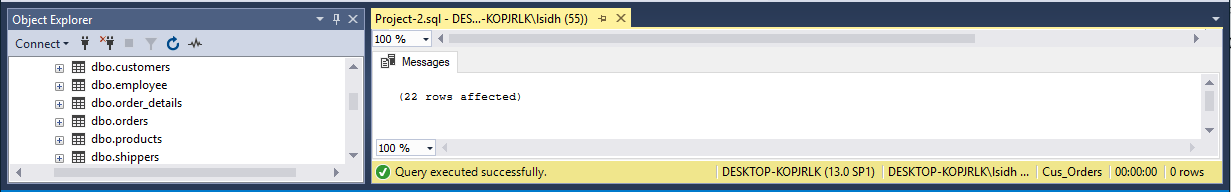
### sql statement

SET fax = 'Unknown'

WHERE fax IS NULL;

GO

### result



## Create a view called vw\_order\_cost

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.

order\_id order\_date product\_id name order\_cost

--------- ------------------------------ -------------- ------------------------------ --------------

10000 2001-05-10 00:00:00.000 17 Franchi S.p.A. 156.0000

10001 2001-05-13 00:00:00.000 25 Mère Paillarde 420.0000

...

(540 row(s) affected)

### sql statement

CREATE VIEW vs\_order\_cost

AS

SELECT o.order\_id,

    o.order\_date,

    p.product\_id,

    c.name,

    od.quantity \* p.unit\_price AS 'order\_cost'

FROM orders o

INNER JOIN order\_details od

    ON od.order\_id = o.order\_id

INNER JOIN products p

    ON p.product\_id = od.product\_id

INNER JOIN customers c

    ON c.customer\_id = o.customer\_id

WHERE o.order\_id BETWEEN 10000

        AND 10200;

GO

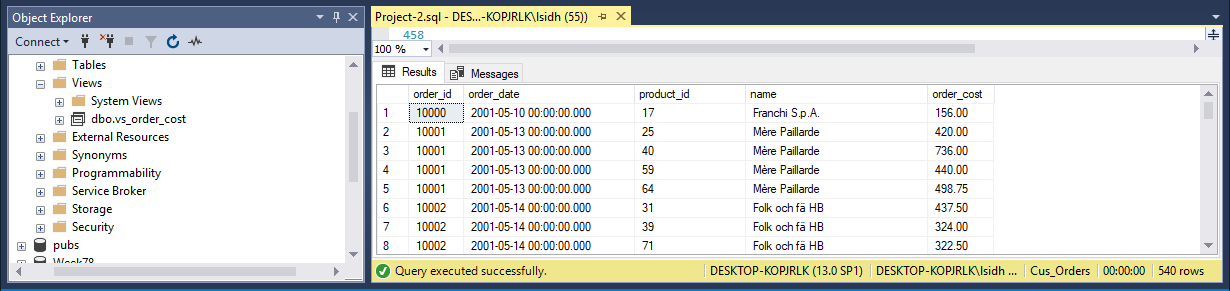
*-- To display the list using the vw\_order\_cost view*

SELECT \*

FROM vs\_order\_cost;

GO

### result



## Create a view called vw\_list\_employees

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.

employee\_id name birth\_date

---------------- -------------------------------- ------------

5 Buchanan, Steven 1955.03.04

7 King, Robert 1960.05.29

9 Dodsworth, Anne 1966.01.27

(3 row(s) affected)

### sql statement

CREATE VIEW vw\_list\_employee

AS

SELECT \*

FROM employee;

GO

*-- To display the list using the above view*

SELECT employee\_id,

    (last\_name + ', ' + first\_name) AS name,

**CONVERT**(VARCHAR(12), birth\_date, 102) AS birth\_date

FROM vw\_list\_employee

WHERE employee\_id IN (

        5,

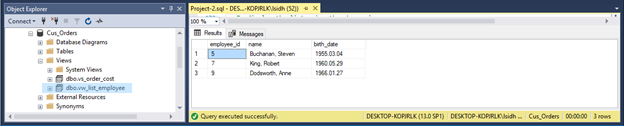
        7,

        9

        );

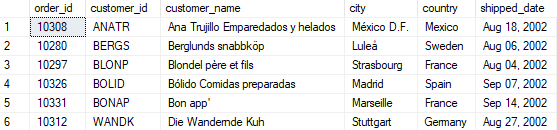
GO

### result



## Create a view called vw\_all\_orders

Create a view called **vw\_all\_orders** to list the columns shown below. 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 **August** **1, 2002** and **September 30, 2002**, formatting the shipped date as shown. Order the result set by customer name and country. The view should produce the result set listed below.



(51 row(s) affected)

### sql statements

CREATE VIEW vw\_all\_orders

AS

SELECT o.order\_id,

    c.customer\_id,

    c.name,

    c.city,

    c.country,

    o.shipped\_date

FROM orders o

INNER JOIN customers c

    ON o.customer\_id = c.customer\_id;

GO

*-- To display the list using the above view*

SELECT order\_id,

    customer\_id,

    name,

    city,

    country,

**CONVERT**(VARCHAR(12), shipped\_date, 102) AS shipped\_date

FROM vw\_all\_orders

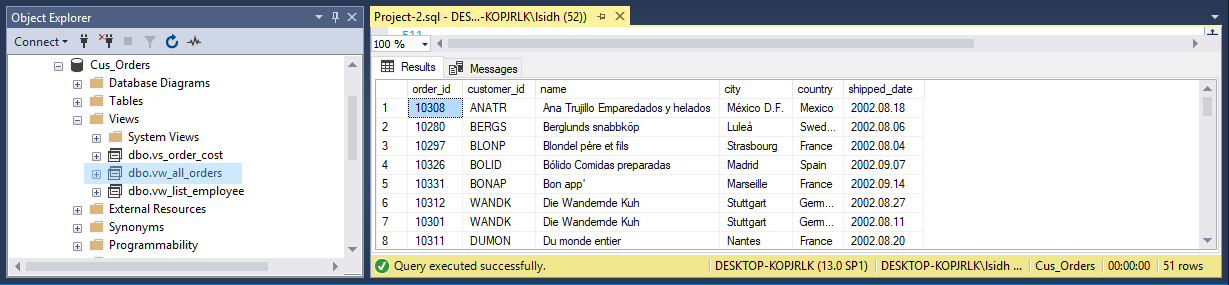
WHERE shipped\_date >= 'Aug 01, 2002'

    AND shipped\_date <= 'Sep 30,2002'

ORDER BY name;

GO

### result



## Create a view listing the suppliers and their shipped items

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.*

supplier\_id supplier\_name product\_id product\_name

--------------- ---------------------------------------- -------------- ------------------------------

9 Silver Spring Wholesale Market 23 Tunnbröd

11 Ovellette Manufacturer Company 46 Spegesild

15 Campbell Company 69 Gudbrandsdalsost

12 South Harbour Products Ltd. 77 Original Frankfurter grüne Soße

14 St. Jean's Company 31 Gorgonzola Telino

…

(77 row(s) affected)

### sql statement

CREATE VIEW vw\_list\_suppliers

AS

SELECT s.supplier\_id,

    s.name AS supplier\_name,

    p.product\_id,

    p.name AS product\_name

FROM suppliers s

INNER JOIN products p

    ON s.supplier\_id = p.supplier\_id;

GO

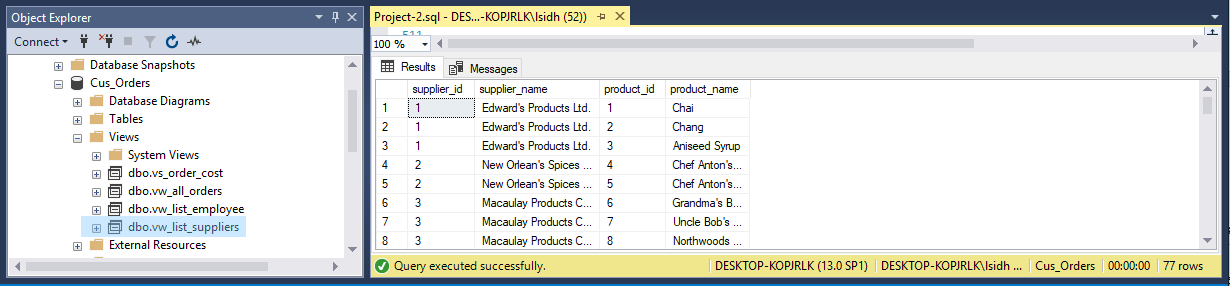
*-- To display the list using the above view*

SELECT \*

FROM vw\_list\_suppliers;

GO

### result



# STORED PROCEDURES AND TRIGGERS

## Create stored procedure sp\_customer\_city

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.

customer\_id name address city phone

--------------- ------------------------ ---------- ------------------------------------- ------ ------------

AROUT Around the Horn 120 Hanover Sq. London (71) 555-7788

BSBEV B's Beverages Fauntleroy Circus London (71) 555-1212

CONSH Consolidated Holdings Berkeley Gardens 12 Brewery London (71) 555-2282

EASTC Eastern Connection 35 King George London (71) 555-0297

NORTS North/South South House 300 Queensbridge London (71) 555-7733

SEVES Seven Seas Imports 90 Wadhurst Rd. London (71) 555-1717

(6 row(s) affected)

### sql statements

CREATE PROCEDURE sp\_customer\_city (@city VARCHAR(20))

AS

SELECT customer\_id,

    name,

    address,

    city,

    phone

FROM customers

WHERE city = @city;

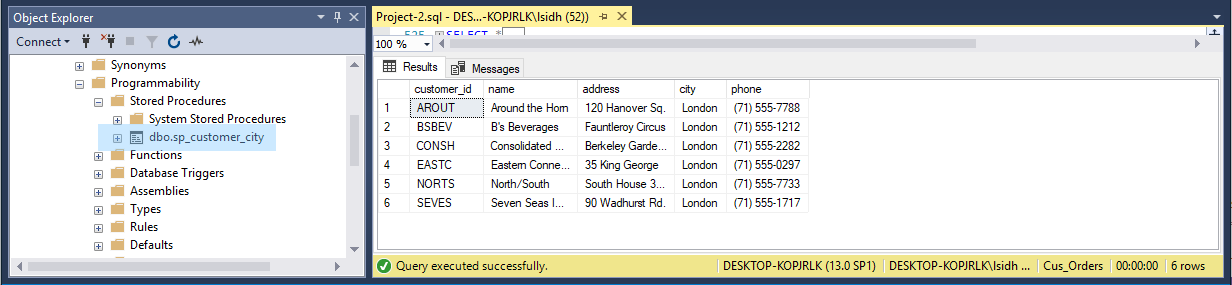
GO

*-- To list the customers using sp\_customer\_city*

EXECUTE sp\_customer\_city 'London';

GO

### result



## Create stored procedure sp\_orders\_by\_dates

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.

order\_id customer\_id customer\_name shipper\_name shipped\_date

---------- --------------- ------------------------------------- -------------------- ----------------------

10423 GOURL Gourmet Lanchonetes Federal Shipping 2003-01-18 00:00:00.000

10425 LAMAI La maison d'Asie United Package 2003-01-08 00:00:00.000

10427 PICCO Piccolo und mehr United Package 2003-01-25 00:00:00.000

...

(188 row(s) affected)

### sql statements

CREATE PROCEDURE sp\_orders\_by\_dates (

    @start DATETIME,

    @end DATETIME

    )

AS

SELECT o.order\_id,

    o.customer\_id,

    c.name,

    s.name,

    o.shipped\_date

FROM orders o

INNER JOIN customers c

    ON o.customer\_id = c.customer\_id

INNER JOIN shippers s

    ON o.shipper\_id = s.shipper\_id

WHERE o.shipped\_date >= @start

    AND o.shipped\_date <= @end;

GO

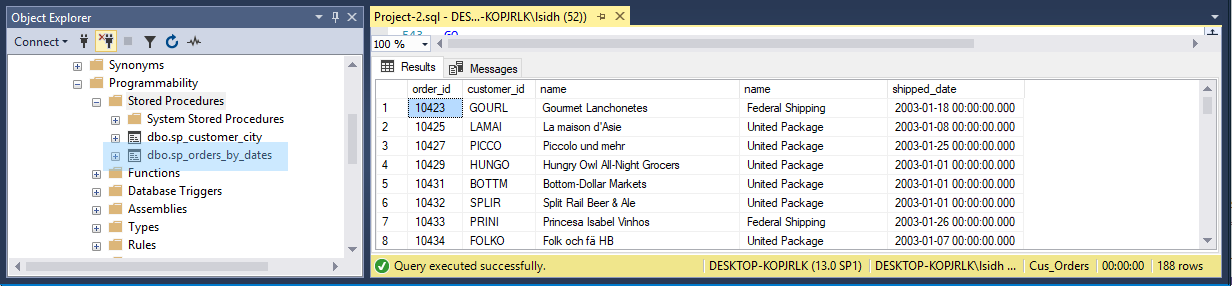
*-- display data using sp\_orders\_by\_dates*

EXECUTE sp\_orders\_by\_dates 'January 1, 2003',

    'June 30, 2003';

GO

### result



## Create stored procedure sp\_product\_listing

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.

product\_name unit\_price quantity\_in\_stock supplier\_name

--------------------------------------------- ------------- --------------------- -----------------------------------

Jack's New England Clam Chowder 10.1325 85 Silver Spring Wholesale Market

Jack's New England Clam Chowder 10.1325 85 Silver Spring Wholesale Market

Jack's New England Clam Chowder 10.1325 85 Silver Spring Wholesale Market

Jack's New England Clam Chowder 10.1325 85 Silver Spring Wholesale Market

(4 row(s) affected)

### sql statements

CREATE PROCEDURE sp\_product\_listing (

    @product VARCHAR(20),

    @month VARCHAR(3),

    @year INT

    )

AS

SELECT p.name,

    p.unit\_price,

    p.quantity\_in\_stock,

    s.name

FROM products p

INNER JOIN suppliers s

    ON p.supplier\_id = s.supplier\_id

INNER JOIN order\_details od

    ON od.product\_id = p.product\_id

INNER JOIN orders o

    ON o.order\_id = od.order\_id

WHERE p.name LIKE '%' + @product + '%'

    AND **CONVERT**(CHAR(3), **DATENAME**(**MONTH**, o.order\_date)) = @month

    AND **CONVERT**(CHAR(4), **DATEPART**(**YEAR**, o.order\_date)) = @year;

GO

*-- Display product using sp\_product\_listing*

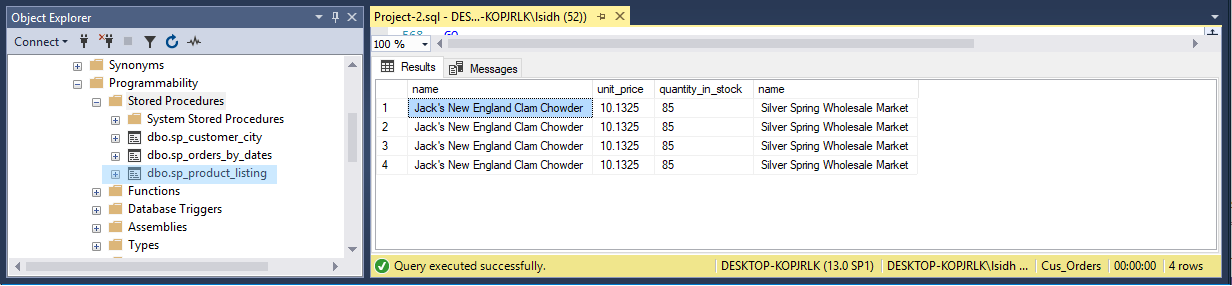
EXECUTE sp\_product\_listing 'Jack',

    'Jun',

    '2001';

GO

### result



## Create a delete trigger on the order\_details table

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

DELETE order\_details

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

You should get the following results:



### SQL STATEMENTS

CREATE TRIGGER tr\_UpdateQty\_AftDelete ON order\_details

FOR DELETE

AS

DECLARE @qtyDelete INT,

    @orderId INT,

    @productId INT;

SELECT @orderId = order\_id,

    @productId = product\_id,

    @qtyDelete = quantity

FROM deleted;

UPDATE products

SET quantity\_in\_stock = quantity\_in\_stock + @qtyDelete

WHERE product\_id = @productId;

SELECT product\_id,

    name,

    'Quantity being deleted form Order' = @qtyDelete,

    'In stock Quantity after Deletion' = quantity\_in\_stock

FROM products

WHERE product\_id = @productId;

GO

*-- use tr\_UpdateQty\_AftDelete*

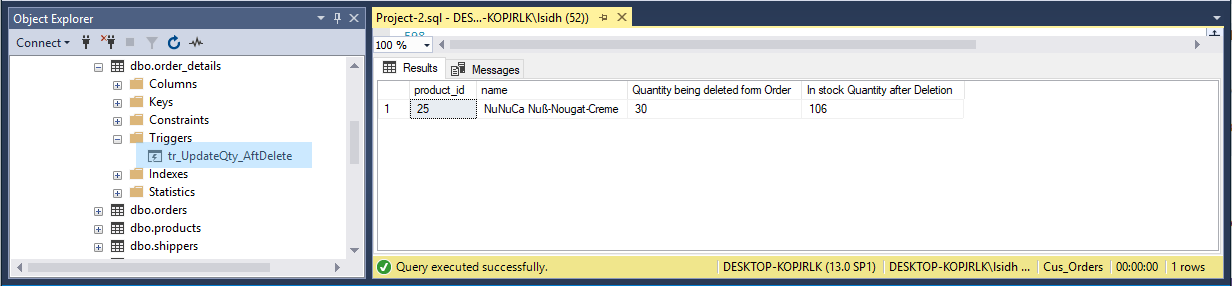
DELETE order\_details

WHERE order\_id = 10001

    AND product\_id = 25;

GO

### RESULT



## Create an insert and update trigger tr\_check\_qty

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

### SQL STATEMENTS

CREATE TRIGGER tr\_check\_qty ON order\_details

FOR INSERT,

    UPDATE

AS

DECLARE @newOrdQty INT, *-- new order value*

    @initOrdQty INT, *-- initial order value*

    @stockQty INT,

    @productId numid,

    @orderId numid;

*-- take the initial order value from virtual deleted table*

SELECT @initOrdQty = quantity,

    @orderId = order\_id,

    @productId = product\_id

FROM deleted;

*-- take the new order value from virtual inserted table*

SELECT @newOrdQty = quantity,

    @orderId = order\_id,

    @productId = product\_id

FROM inserted;

*-- take the product's stock qty*

SELECT @stockQty = quantity\_in\_stock

FROM products

WHERE product\_id = @productId;

IF ((@initOrdQty - @newOrdQty) > @stockQty)

BEGIN

*-- if the stock qty is not enough for order correction*

    PRINT 'Order not allowed: Quantity in stock is not enough'

    ROLLBACK TRANSACTION

END

ELSE

BEGIN

*-- if the stock qty is enough to accomodate new order*

*-- update order\_details table with the new qty value*

    UPDATE order\_details

    SET quantity = @newOrdQty

    WHERE order\_id = @orderId

        AND product\_id = @productId;

*-- update product's stock qty with (@initOrdQty - @newOrdQty)*

    UPDATE products

    SET quantity\_in\_stock = quantity\_in\_stock + (@initOrdQty - @newOrdQty)

    WHERE product\_id = @productId;

END

GO

*-- use trigger tr\_check\_qty*

UPDATE order\_details

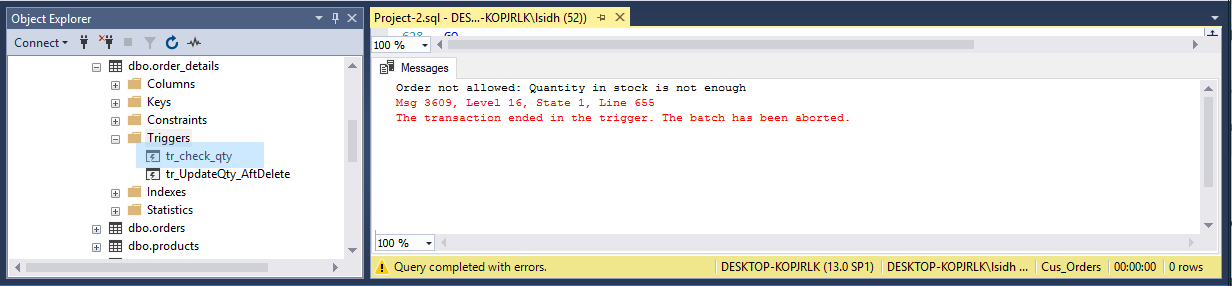
SET quantity = 30

WHERE order\_id = '10044'

    AND product\_id = 7;

GO

### RESULT



## Create a stored procedure sp\_inactive\_cust

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

### SQL STATEMENTS

CREATE PROCEDURE sp\_del\_inactive\_cust

AS

DECLARE @custId char5id;

SELECT @custId = customers.customer\_id

FROM customers

LEFT JOIN orders

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

WHERE orders.customer\_id IS NULL;

*-- to use trigger sp\_del\_inactive\_cust*

DELETE

FROM customers

WHERE customer\_id = @custId;

GO

### RESULT

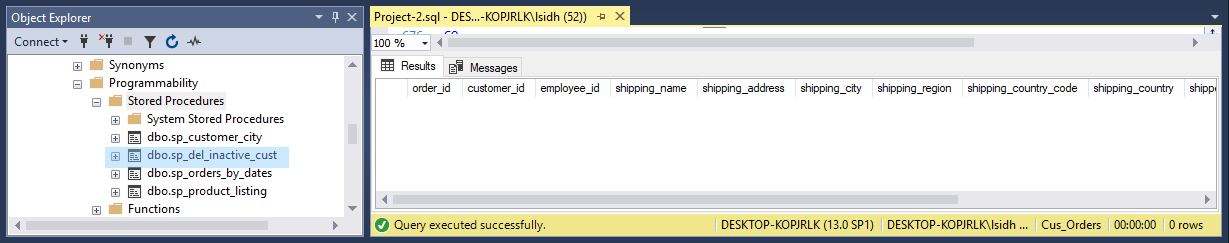
It is only one customer that is inactive, and it has id PARIS. After the above DELETE statement is executed, the following SELECT statement shows that there is no customer\_id PARIS in orders table.

SELECT \*

FROM orders

WHERE customer\_id = 'PARIS'

GO



## Create a store procedure sp\_employee\_information

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



(1 row(s) affected)

### sql statements

CREATE PROCEDURE sp\_employee\_information (@empId INT)

AS

SELECT last\_name,

    first\_name,

    address,

    city,

    province,

    postal\_code,

    'DATE OF BIRTH' = **CONVERT**(VARCHAR(12), birth\_date, 109)

FROM employee

WHERE employee\_id = @empId;

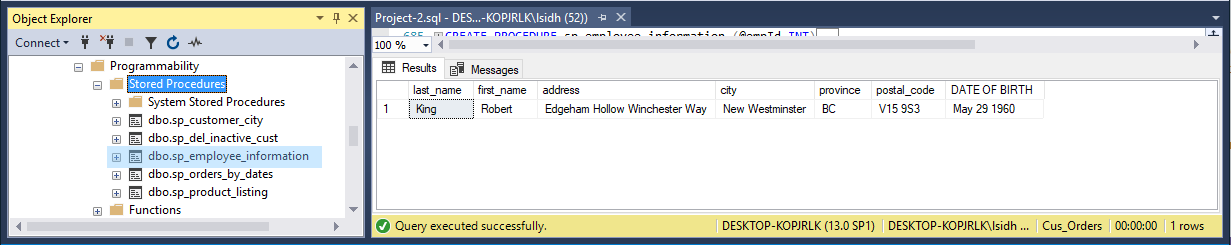
GO

*-- to execute sp\_employee\_information*

EXECUTE sp\_employee\_information 7

GO

### result



## Create a stored procedure sp\_reorder\_qty

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.

product\_id name address city province qty reorder\_level

-------------- ---------------------------------- ------------------------ ------------------ -------- --- -------------

2 Edward's Products Ltd. 1125 Howe Street Vancouver BC 17 25

3 Edward's Products Ltd. 1125 Howe Street Vancouver BC 13 25

5 New Orlean's Spices Ltd. 1040 Georgia Street West Vancouver BC 0 0

…

(23 row(s) affected)

### sql statements

CREATE PROCEDURE sp\_reorder\_qty (@unit INT)

AS

SELECT p.product\_id,

    s.name,

    s.address,

    s.city,

    s.province,

    p.quantity\_in\_stock,

    p.reorder\_level

FROM products p

INNER JOIN suppliers s

    ON s.supplier\_id = p.supplier\_id

WHERE (p.quantity\_in\_stock - p.reorder\_level) < 5;

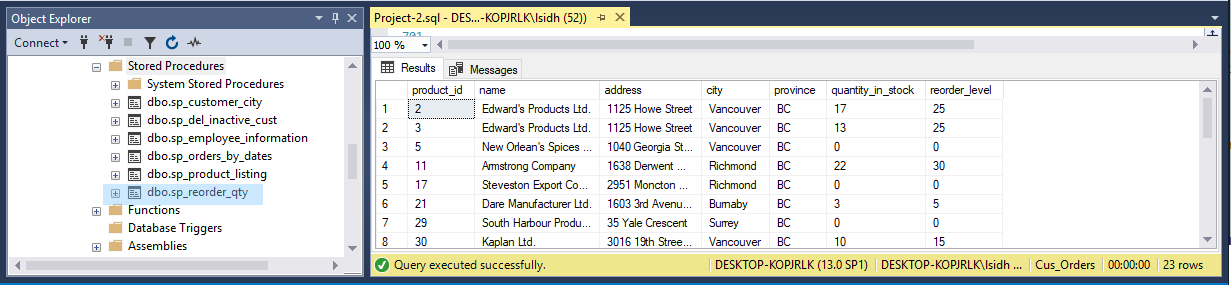
GO

*-- execute sp\_reorder\_qty*

EXECUTE sp\_reorder\_qty 5

GO

### result



## Create a stored procedure sp\_unit\_prices

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.

product\_id name alternate\_name unit\_price

-------------- --------------------------------- -------------------------------- --------------

13 Konbu Kelp Seaweed 6.30

19 Teatime Chocolate Biscuits Teatime Chocolate Biscuits 9.66

23 Tunnbr÷d Thin Bread 9.45

45 R°gede sild Smoked Herring 9.975

47 Zaanse koeken Zaanse Cookies 9.975

52 Filo Mix Mix for Greek Filo Dough 7.35

54 TourtiÞre Pork Pie 7.8225

75 Rh÷nbrõu Klosterbier Rh÷nbrõu Beer 8.1375

1. row(s) affected)

### sql statements

CREATE PROCEDURE sp\_unit\_prices (

    @bottomVal MONEY,

    @topvalue MONEY

    )

AS

SELECT product\_id,

    name,

    alternate\_name,

    unit\_price

FROM products

WHERE unit\_price BETWEEN @bottomVal

        AND @topvalue;

GO

*-- execute sp\_unit\_prices*

EXECUTE sp\_unit\_prices 5.00,

    10.00

GO

### result



# SUMMARY

# APPENDIX A – TABLE DESIGN

**customers**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Column Name | Data Type | Length |  | Null Values |
| customer\_id | char | 5 | User-Defined Data Type | No |
| name | varchar | 50 |  | No |
| contact\_name | varchar | 30 |  |  |
| title\_id | char | 3 |  | No |
| address | varchar | 50 |  |  |
| city | varchar | 20 |  |  |
| region | varchar | 15 |  |  |
| country\_code | varchar | 10 |  |  |
| country | varchar | 15 |  |  |
| phone | varchar | 20 |  |  |
| fax | varchar | 20 |  |  |

**orders**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Column Name | Data Type | Length |  | Null Values |
| order\_id | int |  | User-Defined Data Type | No |
| customer\_id | char | 5 | User-Defined Data Type | No |
| employee\_id | int |  |  | No |
| 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 |  |  | No |
| order\_date | datetime |  |  |  |
| required\_date | datetime |  |  |  |
| shipped\_date | datetime |  |  |  |
| freight\_charge | money |  |  |  |

**order\_details**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Column Name | *Data Type* | *Length* |  | *Null Values* |
| order\_id | int |  | User-Defined Data Type | No |
| product\_id | int |  | User-Defined Data Type | No |
| quantity | int |  |  | No |
| discount | float |  |  | No |

**products**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Column Name | Data Type | Length |  | Null Values |
| product\_id | int |  | User-Defined Data Type | No |
| supplier\_id | int |  |  | No |
| name | varchar | 40 |  | No |
| alternate\_name | varchar | 40 |  |  |
| quantity\_per\_unit | varchar | 25 |  |  |
| unit\_price | money |  |  |  |
| quantity\_in\_stock | int |  |  |  |
| units\_on\_order | int |  |  |  |
| reorder\_level | int |  |  |  |

**shippers**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Column Name | Data Type | Length |  | Null Values |
| shipper\_id | int |  | IDENTITY | No |
| name | varchar | 20 |  | No |

**suppliers**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Column Name | Data Type | Length |  | Null Values |
| supplier\_id | int |  | IDENTITY | No |
| name | varchar | 40 |  | No |
| address | varchar | 30 |  |  |
| city | varchar | 20 |  |  |
| province | char | 2 |  |  |

**titles**

|  |  |  |  |
| --- | --- | --- | --- |
| Column Name | *Data Type* | *Length* | *Null Values* |
| title\_id | char | 3 | No |
| description | varchar | 35 | No |

Employee

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |

# APPENDIX B – ERD

