Raquel H. Rodriguez

[Dirección de correo electrónico]

Descripción breve

[Dibujar su lector con un resumen de la participación. Normalmente es un breve resumen del documento.   
Cuando esté listo para agregar contenido, haga clic aquí y empiece a escribir.]

DATABASE MANAGEMENT SYSTEM

HYBRID DATABASE

1. **OVERVIEW**

This project is based on the development of a Hybrid Database System for a start-up restaurant called “Pickosito”, which is a new brand in the food industry that specialises in Mexican cuisine. This company is interested in innovating and implementing technological tools to ensure continuous improvement of their processes and services. It is expected that the restaurant will open its doors in Dublin in the spring of 2021**.**

1. **SCOPE**

The main objective of the project is to create a Hybrid Database Management System that provides the company with the necessary technological tools to create, control and maintain a database system, which includes the monitoring and control of the main areas of the restaurant.

The Management System of Pickosito, as a whole, should act as an integrated system, capable of communicating and managing the totality of the restaurant operations;nevertheless, given the time and resources available for the development of this project, this database design will be focused on three main areas only:

* **Sales**. For the management and processing of different transactions, which include orders, payments and other transactions.
* **Inventory Management.** For the management of suppliers, menu items and products/ingredients and stock take.
* **Human Resources.** For the correct management of employee records, their general details and granting of permissions to access the system.

1. **BUSINESS RULES**

This section describes the business rules, to provide a better understanding of the expected outcomes of this project.

Business rules were implemented through integrity constraints, which are “a method to restrict database updates to a set of specified values or ranges of values” to prevent the data and inter-table relationships to become corrupted by improper management of data (Connolly, T.M. and Begg, C.E., 2005).

The following table contains the full list of business rules:

**Table No. Business Rules.**

|  |  |
| --- | --- |
| **#** | **RULE** |
| **1** | Every branch should have a unique number used as identifier. |
| **2** | Every employee should have a unique number used as identifier. |
| **3** | Every customer should have a unique number used as identifier. |
| **5** | Each product record must match at least one supplier record. |
| **6** | Every employee should pertain to one branch only. |
| **7** | Every employee should have an employee record that includes their position and salary, accordingly. |
| **8** | Every order should be assigned a unique number used as identifier and should match one and only one order transaction. |
| **9** | Every payment should be assigned a unique number used as identifier and should be linked to one transaction. |
| **10** | Every sale/transaction in the restaurant should be recorded. |

As mentioned before, rules were defined during the analysis and design stage to ensure that the business’ data was accurately incorporated into the database.

The following are examples for rules 1, 2, 3 and 4 and their implementation:

* 1. **RULE 1. EVERY BRANCH SHOULD HAVE A UNIQUE NUMBER USED AS IDENTIFIER.**

**BRANCH CONSTRAINTS:**

*CREATE TABLE tbl\_Branch(*

*branch\_id varchar(5) not null,*

*branch\_name varchar(20) not null,*

*branch\_location varchar(50) not null,*

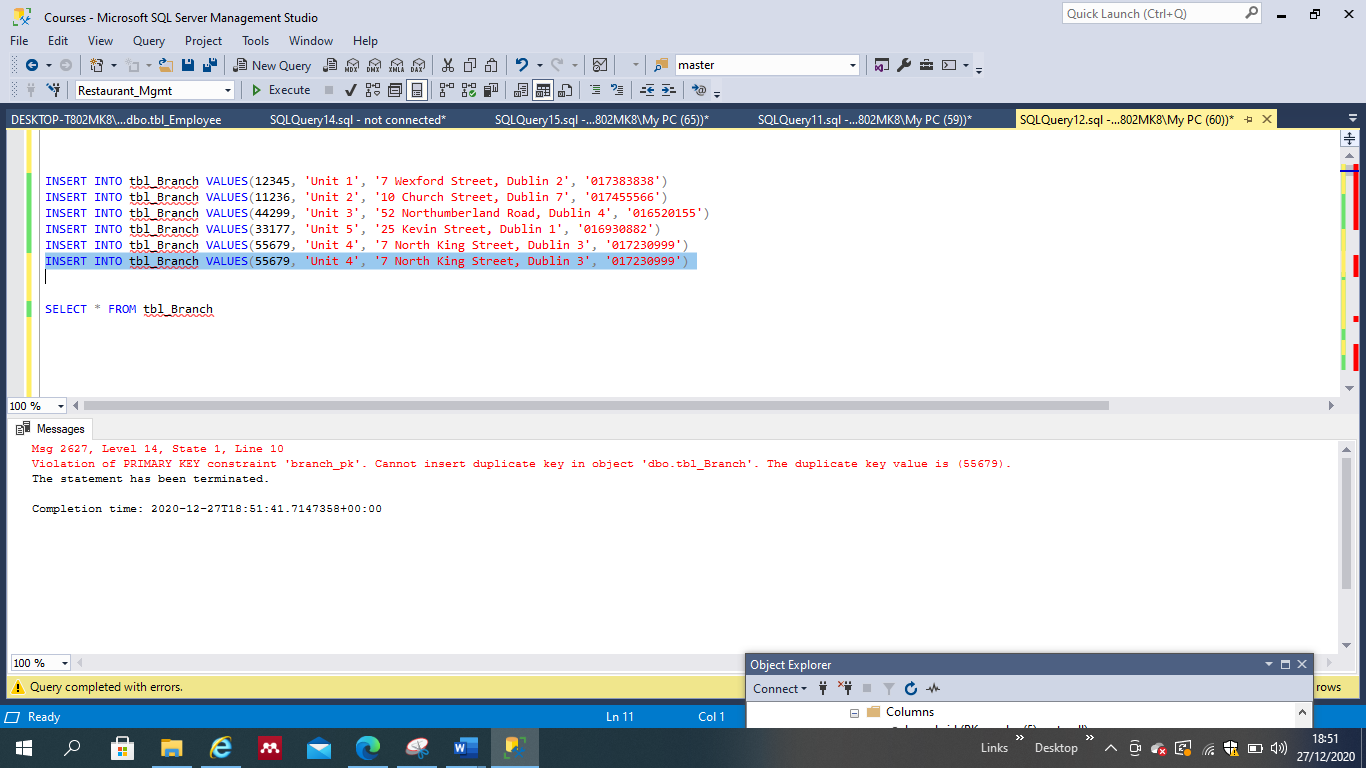
*branch\_phone varchar(10),*

*CONSTRAINT branch\_pk PRIMARY KEY(branch\_id)*

*)*

See **Figure No. ?** Branch Constraints below that shows that Primary Key in Branch Table cannot be duplicated:

**Figure No. ?** Branch Constraints.



**3.2 RULE 2. EVERY EMPLOYEE SHOULD HAVE A UNIQUE NUMBER USED AS IDENTIFIER.**

**EMPLOYEE CONSTRAINTS:**

*CREATE TABLE tbl\_Employee(*

*emp\_id varchar(4) not null,*

*branch\_id varchar(5) not null,*

*emp\_name varchar(20) not null,*

*emp\_address varchar(100) not null,*

*emp\_phone varchar(20) not null,*

*emp\_PPS varchar(15) not null,*

*emp\_email varchar(20) not null,*

*emp\_position varchar(20) not null,*

*emp\_monthly\_salary int,*

*emp\_permission\_level varchar(1) not null,*

*CONSTRAINT emp\_pk PRIMARY KEY(emp\_id),*

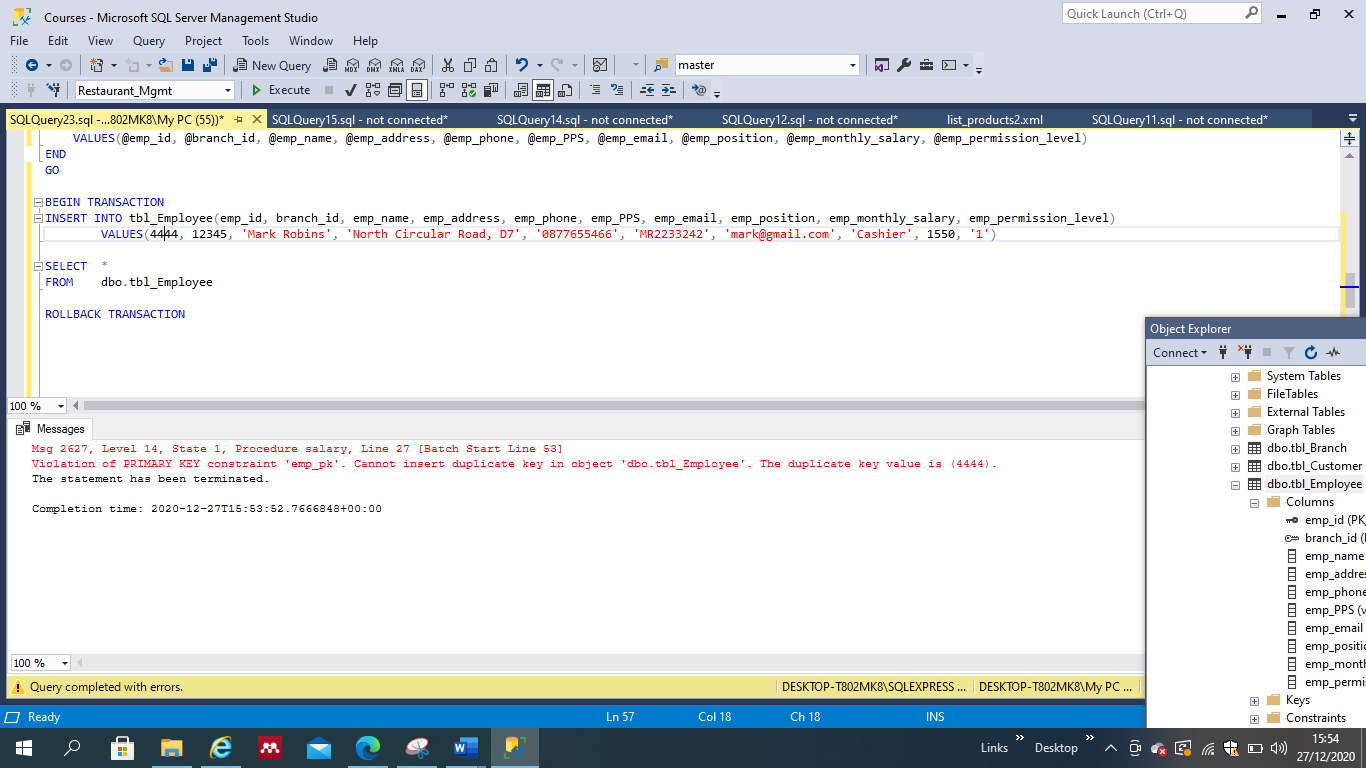
*CONSTRAINT emp\_fk FOREIGN KEY(branch\_id)*

*REFERENCES tbl\_Branch(branch\_id)*

*)*

See **figure No. ?** Employee Constraints below that shows that Primary key in Employee Table cannot be duplicated:

**Figure No. ? Employee Constraints**



**3.3 RULE 3. EVERY CUSTOMER SHOULD HAVE A UNIQUE NUMBER USED AS IDENTIFIER.**

**CONSTRAINTS:**

*CREATE TABLE tbl\_Customer(*

*customer\_id varchar(10) not null,*

*customer\_name varchar(30) not null,*

*customer\_address varchar (30) not null,*

*customer\_city varchar(20) not null,*

*customer\_phone varchar(10),*

*customer\_email varchar(30),*

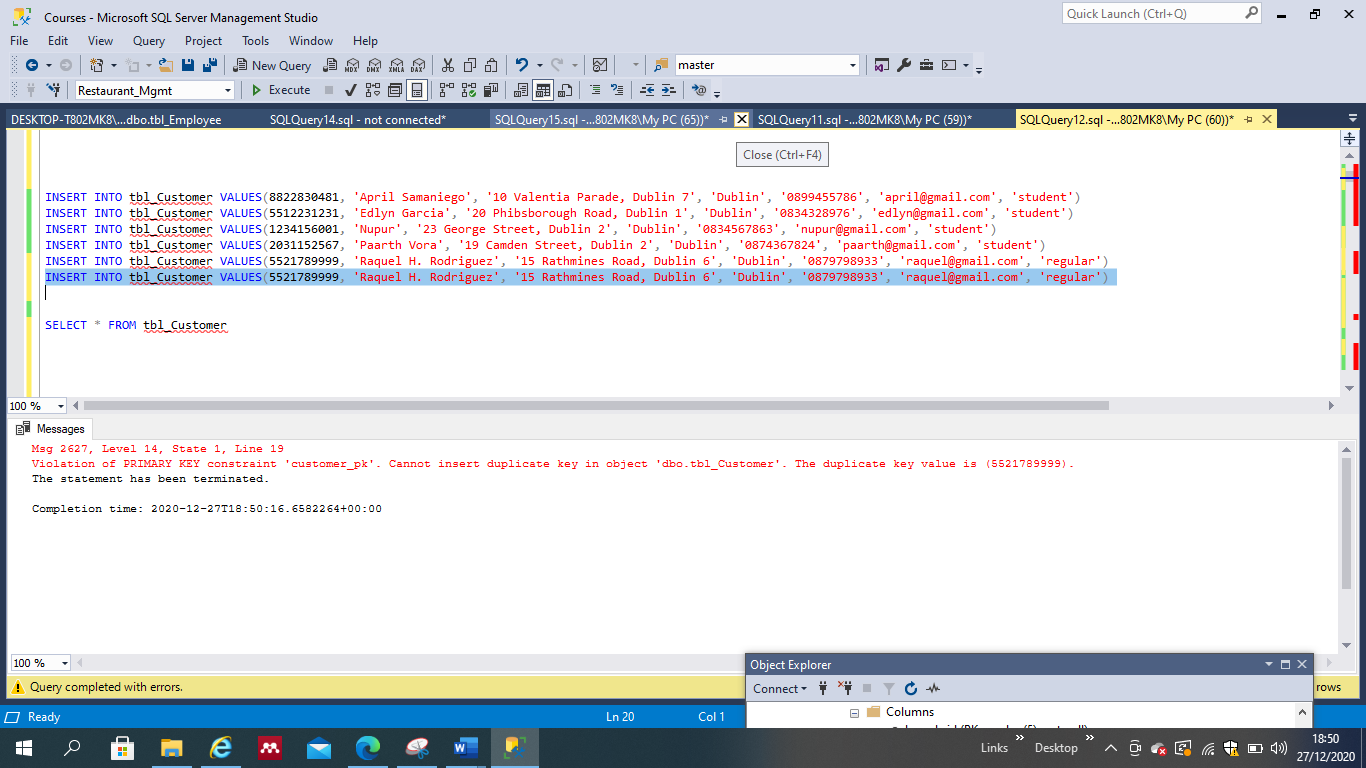
*customer\_type varchar (20),*

*CONSTRAINT customer\_pk PRIMARY KEY(customer\_id),*

*)*

See **figure No. ?** Customer Constraints below that shows that Primary key in Customer Table cannot be duplicated:

**Figure No. ?** Customer Constraints



**3.4 RULE 4. EACH PRODUCT RECORD MUST MATCH AT LEAST ONE SUPPLIER RECORD.**

**CONSTRAINTS:**

*CREATE TABLE tbl\_Product\_Ingredient(*

*prod\_id varchar(10) not null,*

*supplier\_id varchar(10) not null,*

*menu\_item\_id varchar(10) not null,*

*CONSTRAINT product\_pk PRIMARY KEY (prod\_id),*

*CONSTRAINT product\_fk1 FOREIGN KEY (supplier\_id)*

*REFERENCES tbl\_Supplier(supplier\_id),*

*CONSTRAINT product\_fk2 FOREIGN KEY(menu\_item\_id)*

*REFERENCES tbl\_MenuItem(menu\_item\_id),*

*)*

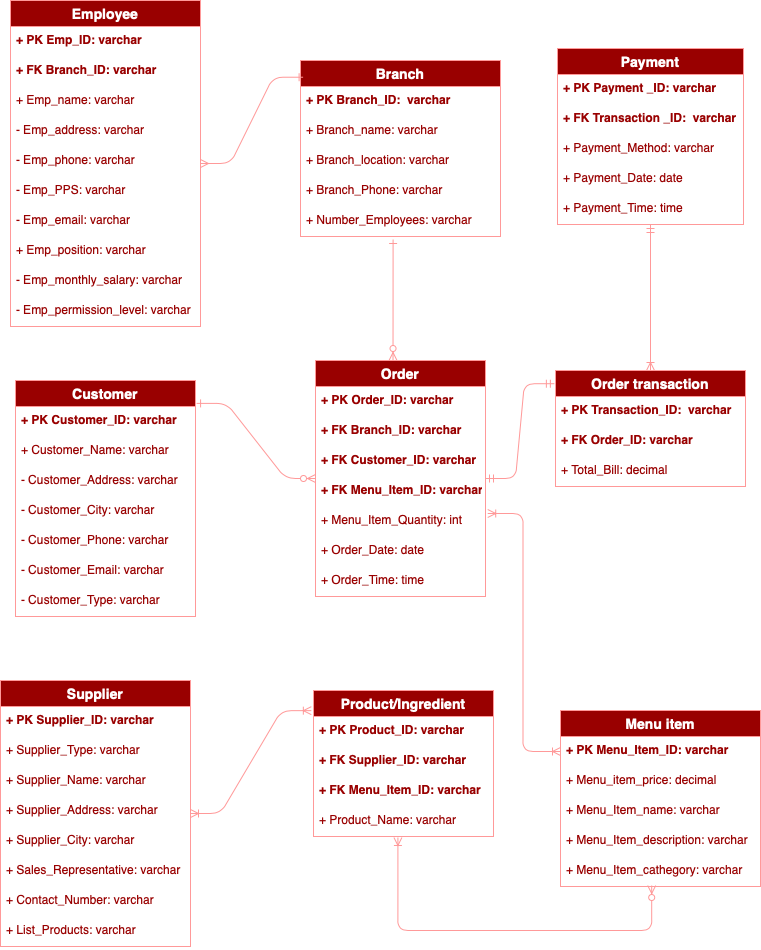
The foreign key declared in this table acts as a primary key in the Supplier table, and was used to match rows in between the Product\_Ingredient and the Supplier tables.

For the rest of the business rules, the full code and constraints is available in Appendix No. 1: Creation of tables and constraints.

In addition, a Data Diagram and a Relational Schema have been developed, in order to better illustrate the Database functionality and relationships between the entities.

See figures No. ? Data Diagram and figure No.? Relational Schema below.

**Figure No. 2** Data Diagram



Zero to many

xml

Figure No.? Relational Schema

Relational Schema for the hybrid database system (must be developed in Visio or some similar system using crow’s foot notation or UML notations). Relational schema must fulfil the requirements of 3 NF, you must provide the justification in report. Data type of each field must be shown in the diagram. It must include five substantial tables using XML data type in at least one table (or more) as appropriate. Justify the use of XML data type that makes it a hybrid database.

1. **DIAGRAM JUSTIFICATION**

A total of nine tables were created in the Database System as follows:

* **EMPLOYEE.** Employee table consists of ten fields (Employee ID, Employee\_Name, Employee\_Address, Employee\_PPS, Employee\_Phone, Employee\_email, Employee\_Position, Employee\_Permission\_level, Employee\_Monthly\_Salary and a foreign key).

**CONSTRAINTS:** Employee\_ID is the primary key which cannot be duplicated and has a relationship of many to one with Branch table, considering that a branch can have many employees but staff can be employed to work in one particular branch only; for this relationship purpose, Branch\_ID was designated as the Foreign key.

* **BRANCH.** Branch table consists of five fields (Branch\_ID, Branch\_name, Branch\_Phone Branch\_Location and Branch\_Number\_Employees)

**CONSTRAINTS:** Branch ID is the primary key that cannot be duplicated and has a relationship with Employee table previously described, and has a relationship of one to zero/many with Order Table, considering that a branch can receive many orders or none, but orders are directed to one particular branch only.

* **CUSTOMER.** Customer table consists of seven fields (Customer\_ID, Custmomer\_Name, Customer\_Address, Customer\_city, Customer\_Phone, Customer\_email, and Customer\_type).

**CONSTRAINTS:** Customer\_ID is the primary key that cannot be duplicated and has a relationship of one to zero/many with Order Table, considering that a customer can place zero or several orders but an order can be placed by one and only one customer.

* **ORDER**. Order Table consists of six fields (Order\_ID, Menu\_Item\_Quantity, Order\_Date, Order\_Time and three Foreign Keys).

**CONSTRAINTS**: Order\_ID is the primary key that cannot be duplicated. Order Table has a relationship with Branch Table and Customer Table previously described. Additionally, Order Table has a relationship of one to many to zero to many with Menu Item Table, considering that an order can contain one or many items, but an item can be part of zero or many orders.

Foreign Keys: Customer\_ID, Branch\_ID and Menu\_Item\_ID have been designated for this purpose.

* **ORDER TRANSACTION.** Order Transaction Table consists of three fields (Transaction\_ID, Total\_Bill and one Foreign Key).

**CONSTRAINTS:** Transaction\_ID is the primary key that can not be duplicated. Transaction Table has a relationship of one to one with Order Table; considering that an order can produce one Order Transaction and one Order Transaction is the result of one specific Order.

Foreign key Order\_ID as been designated for this purpose.

* **PAYMENT.** Payment Table consists of five fields (Payment\_ID, Payment\_method, Payment\_Date, Payment\_time and a Foreign Key).

**CONSTRAINTS.** Payment\_ID is the primary key that cannot be duplicated. Payment table has a relationship of one to one/many with Order Transaction, considering that Order Transaction should receive one full payment for the exact amount of the bill, but Payment can be done to different Order Transactions, i.e.: when one card holder places different orders. Transaction\_ID has been designated as the foreign key to serve this purpose.

* **MENU ITEM.** Menu Item Table consists of five fields (Menu Item\_ID, Menu\_item\_name, Menu\_item\_description, Menu\_item\_price, Menu\_item\_cathegory ).

**CONSTRAINTS.** Menu\_Item\_ID is the primary key that cannot be duplicated. It has a relationship of one/many to zero/many with Order Table, considering that Menu Items can be part of zero or many Orders, while Orders can include one or many Menu Items. Menu Item Table has a relationship of zero/many to one/many with Product\_Ingredient Table, considering that a Menu Item consists of one or many products or ingredients, and Products or Ingredients can be present in one or many Menu Items.

* **PRODUCT\_INGREDIENT.** Product\_Ingredient Table consists of four fields (Product\_ID, Product\_name and two foreign keys).

**CONSTRAINTS.** Product\_ID is the primary key that cannot be duplicated. It has a relationship with Menu Item Table previously described and a relationship of one/many to one/many with Supplier Table, considering that Products or Ingredients can be supplied by one or many suppliers, while a Supplier can provide one or many Products or Ingredients. Supplier\_ID and Menu\_Item\_ID have been designated as foreign keys for this purpose.

**SUPPLIER.** Supplier Table consists of eight fields (Supplier\_ID, Supplier\_Type, Supplier\_Name, Supplier\_Phone, Supplier\_Address, Supplier\_City, Sales\_Representative and List\_Products)

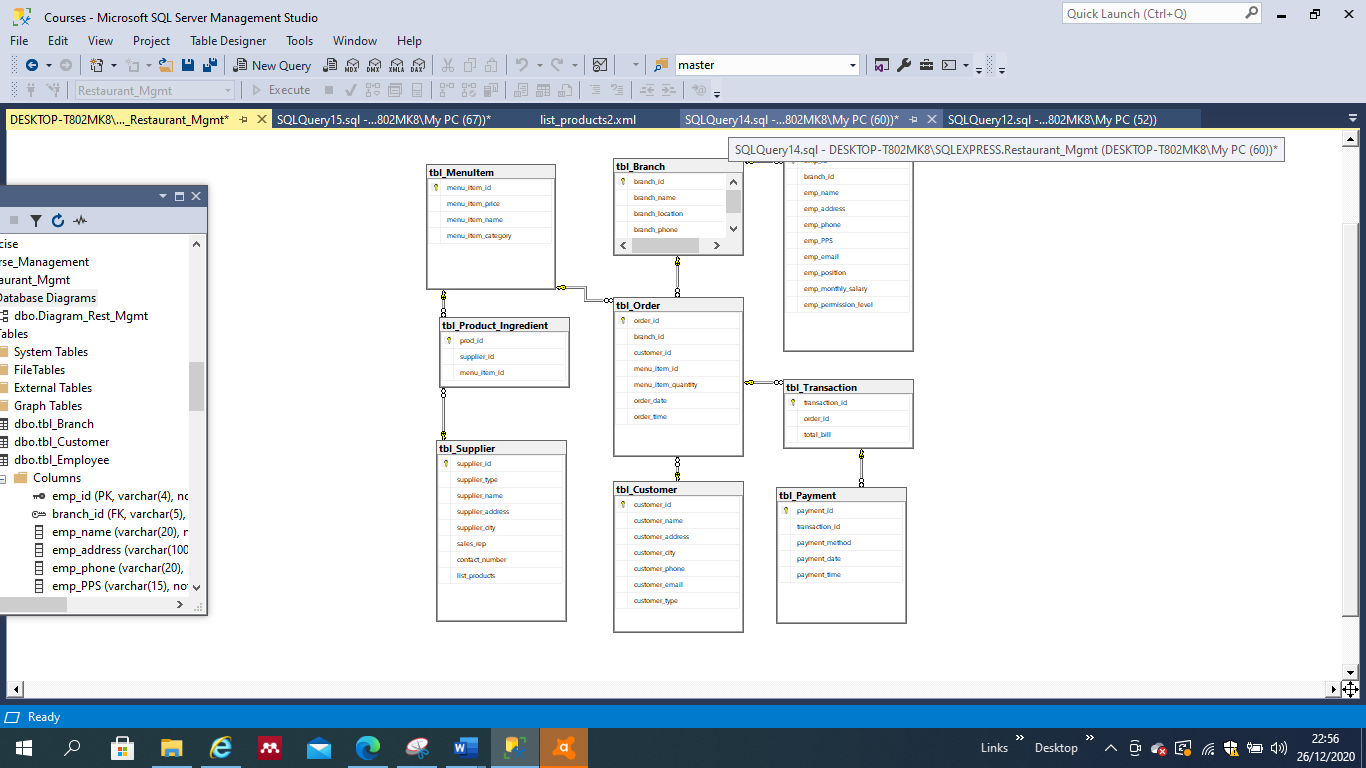
**CONSTRAINTS**. Supplier\_ID is the primary key that cannot be duplicated and has a relationship with Product\_Ingredient table previously described.

1. **XML DATA.**

Supplier\_ID table includes an XML field (List\_Products), that has been created as an XML field due to the business requirement of storage and availability of supplier data to the user; in this specific case, items available to order from each supplier were made available at the link provided in the List\_Products field, which is capable to store a collection of different values in a single field without having to design an extra set of tables for a key or value store.

Moreover, considering that in practice, the Restaurant receives lists of items by many different suppliers commonly in .xls files, it is reasonable to provide this option to the company, as new lists and data can be mapped and converted to XML files by using the developer tools provided by Microsoft Excel.

Lastly, it is possible to use XQuery to select rows that match the identifying or required criteria for managing XML data.



1. **BUSINESS REQUIREMENTS.**

The company is looking to implement tools that contribute to the performance of the company in terms of accuracy, speed of service and reliability in internal processes, such as Sales, Human Resources and Inventory Management.

The following table contains a list of the business requirements and descriptions.

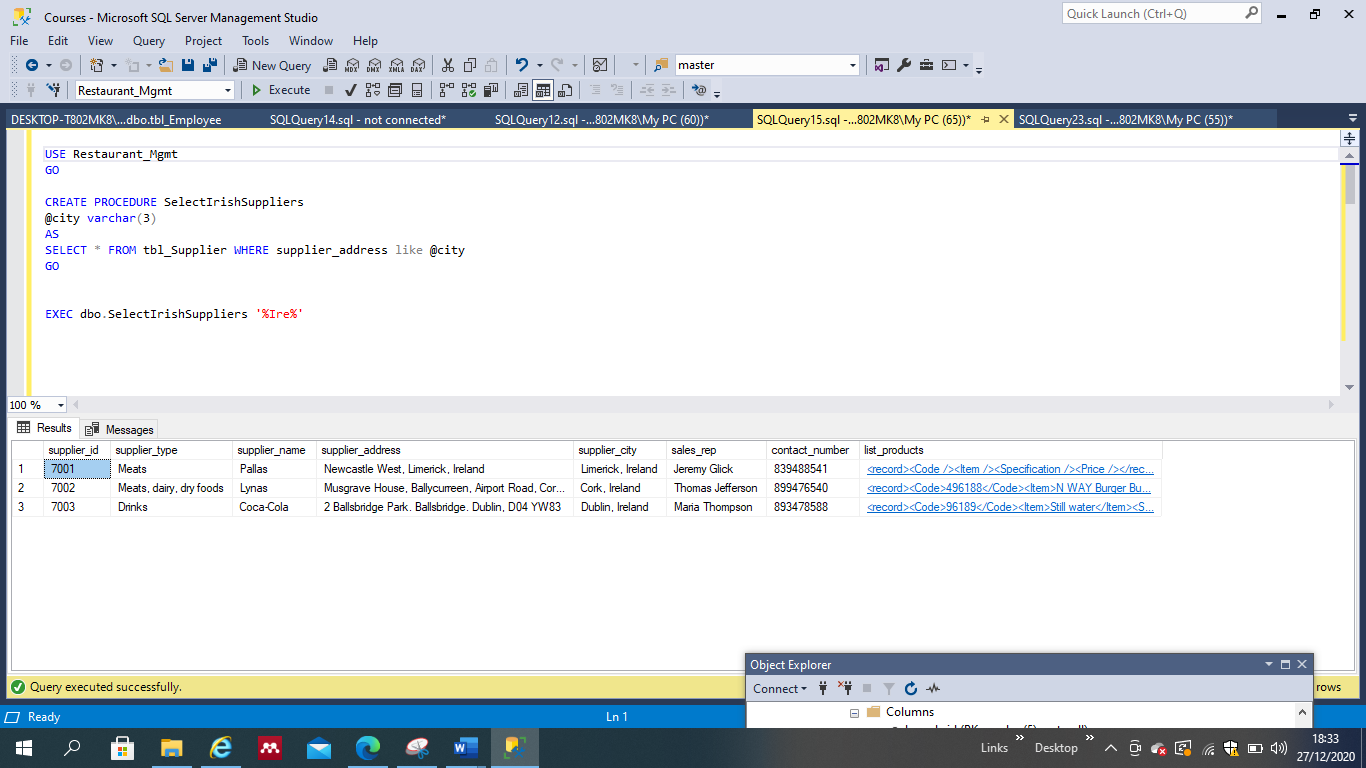
Table No. ? Business Requirements.

|  |  |
| --- | --- |
| **REQUIREMENT NO.** | **BUSINESS REQUIREMENT** |
| 1 | **Authorised suppliers should be based in Ireland.** |
| 2 | **Labour expenses and total amounts shall be made available for administrative purposes.** |
| 3 | **The user shall be able to search for suppliers lists and items contained in them.** |
| 4 | **A full list of employees of all branches shall be available.** |
| 5 | **The system shall provide a practical option to register a new employee.** |
| 6 | NEEDED |
| 7 | **The system shall be able to provide protection to confidential or sensitive information.** |

**6.1 BUSINESS REQUIREMENT NO. 1 AUTHORISED SUPPLIERS SHOULD BE BASED IN IRELAND.**

For this purpose, a stored procedure with **parameters** was created called ‘SelectIrishSuppliers’, which makes possible for the user to type in three characters contained in the word Ireland (for Irish suppliers), in this case ‘Ire’, which will be searched in the supplier\_address field and produce a result. See figure below.

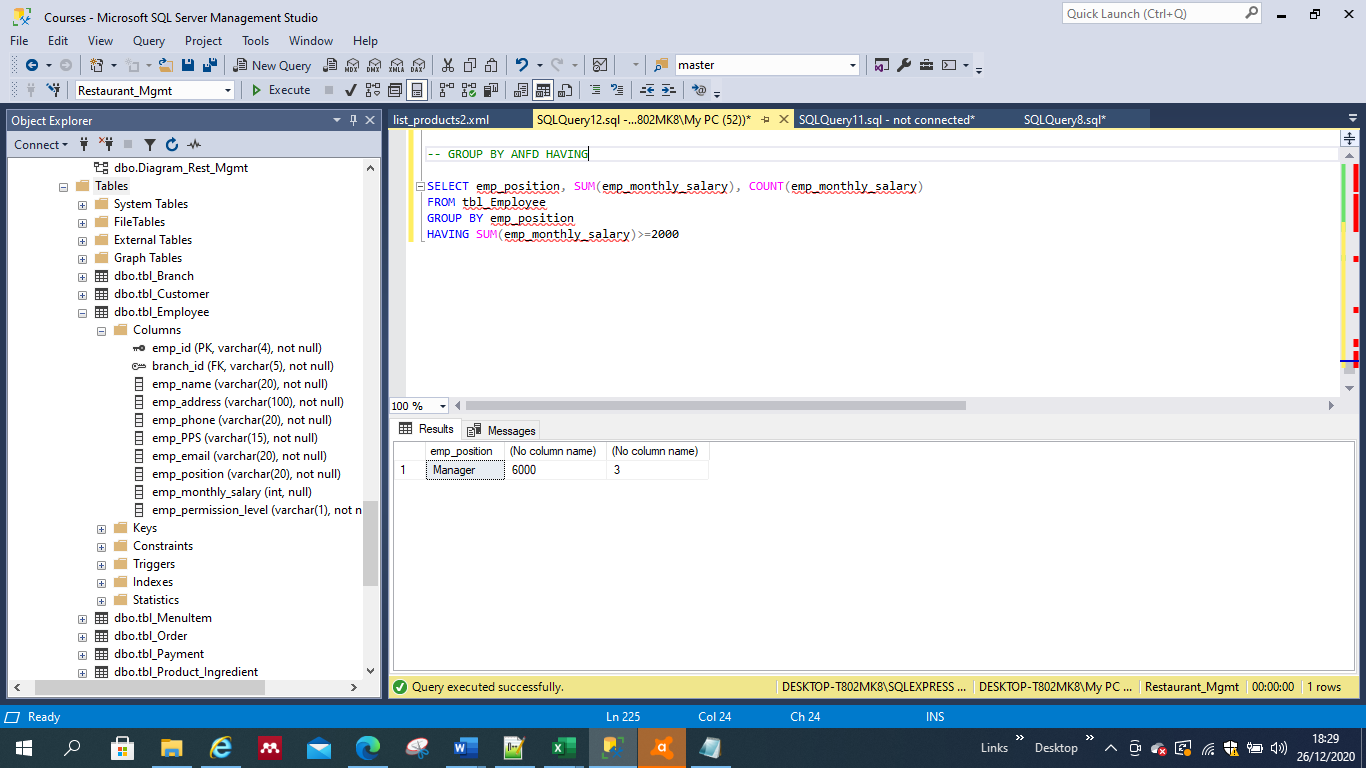
Figure No. Procedure: Select Irish Suppliers.

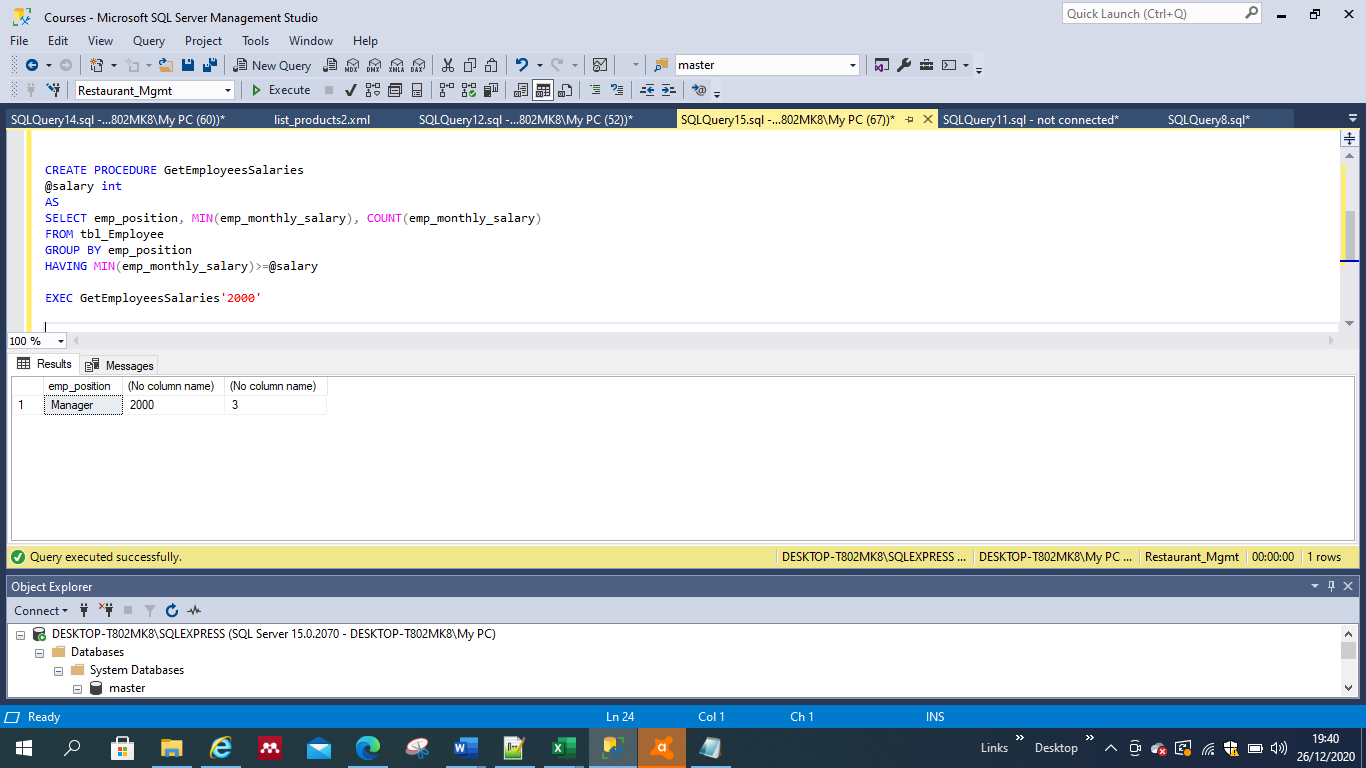


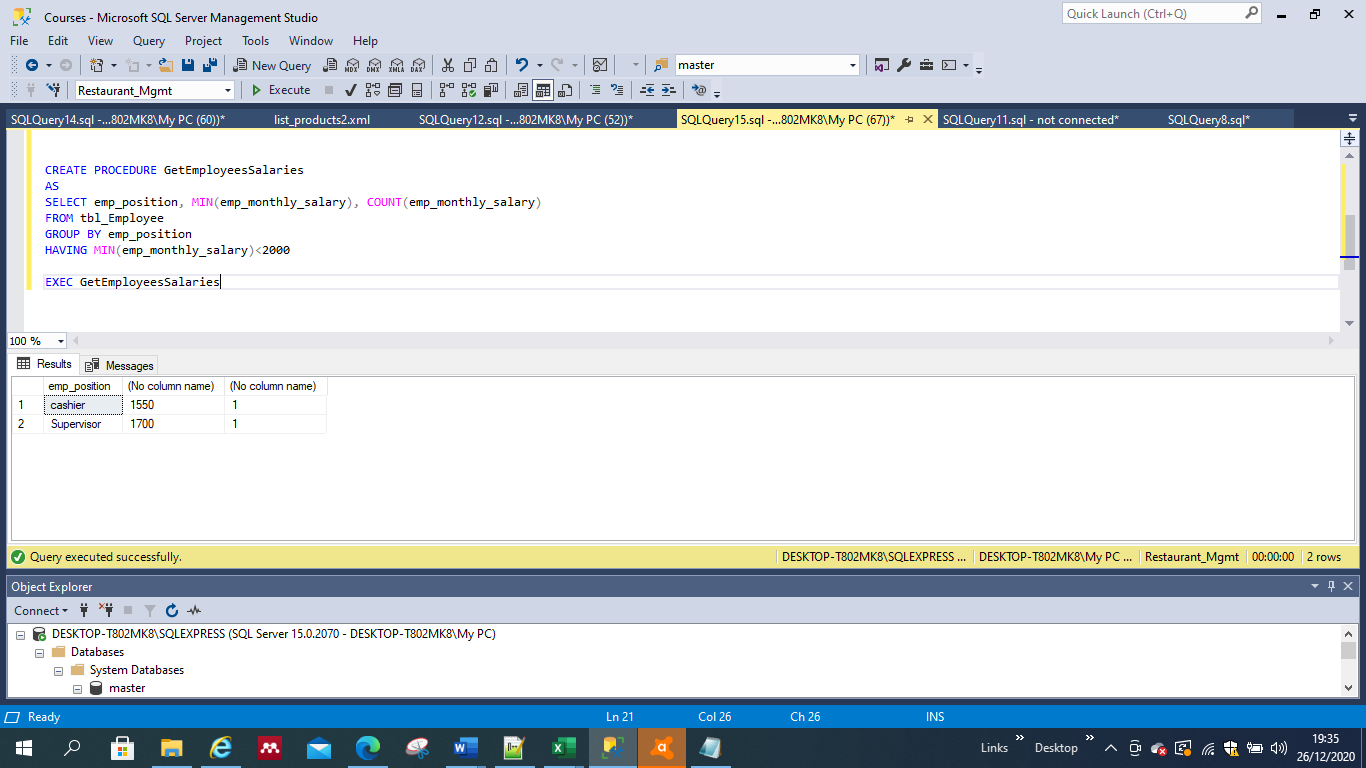
**6.2 Business Requirement No. 2. Labour expenses should be made available for administrative purposes.**

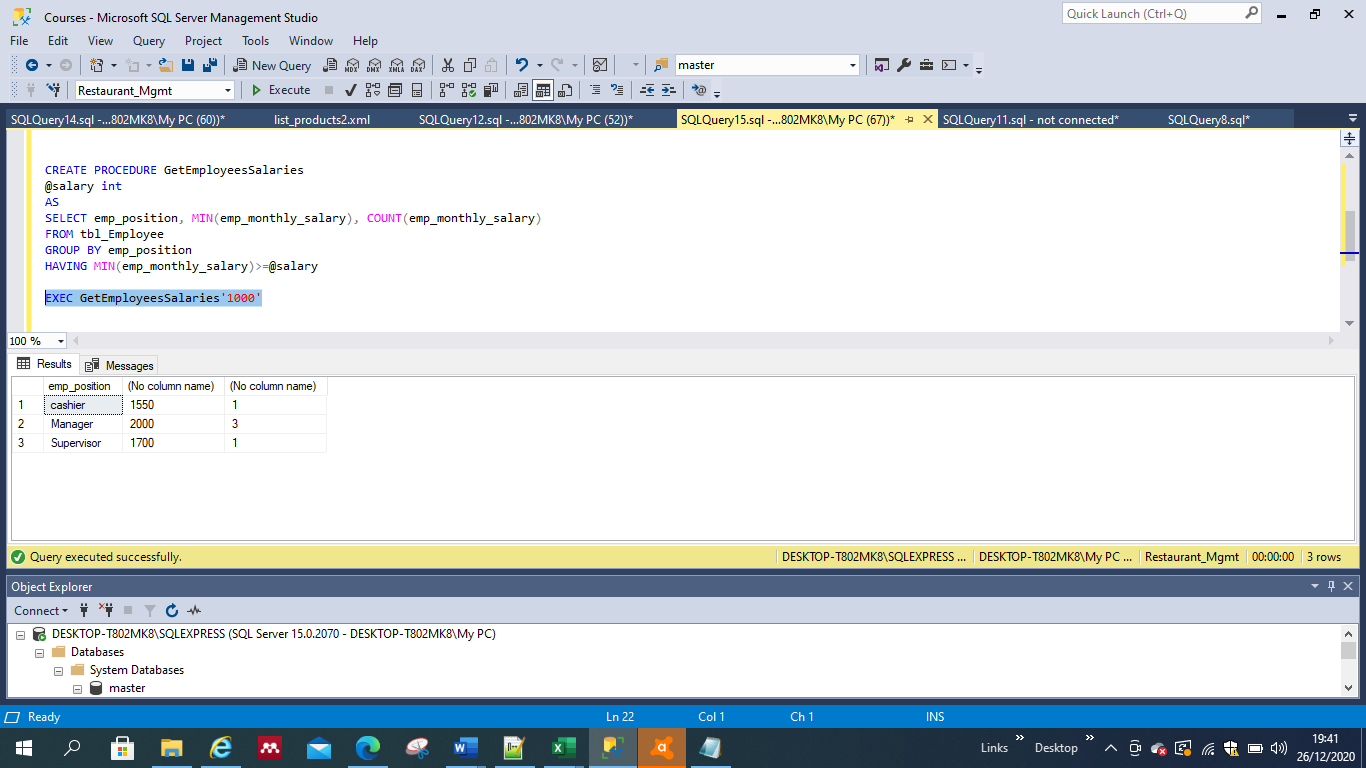
For this purpose, a stored procedure with **parameters** called ‘GetEmployeesSalaries’ was created. The functions **Group BY** and **HAVING** were included in this procedure, in order to provide a tool to get the sum of the salaries and to also filter employees’ salaries by position, or minimum wage assigned. See figure below.

**Figure No. Procedure: Employees’ Salaries.**



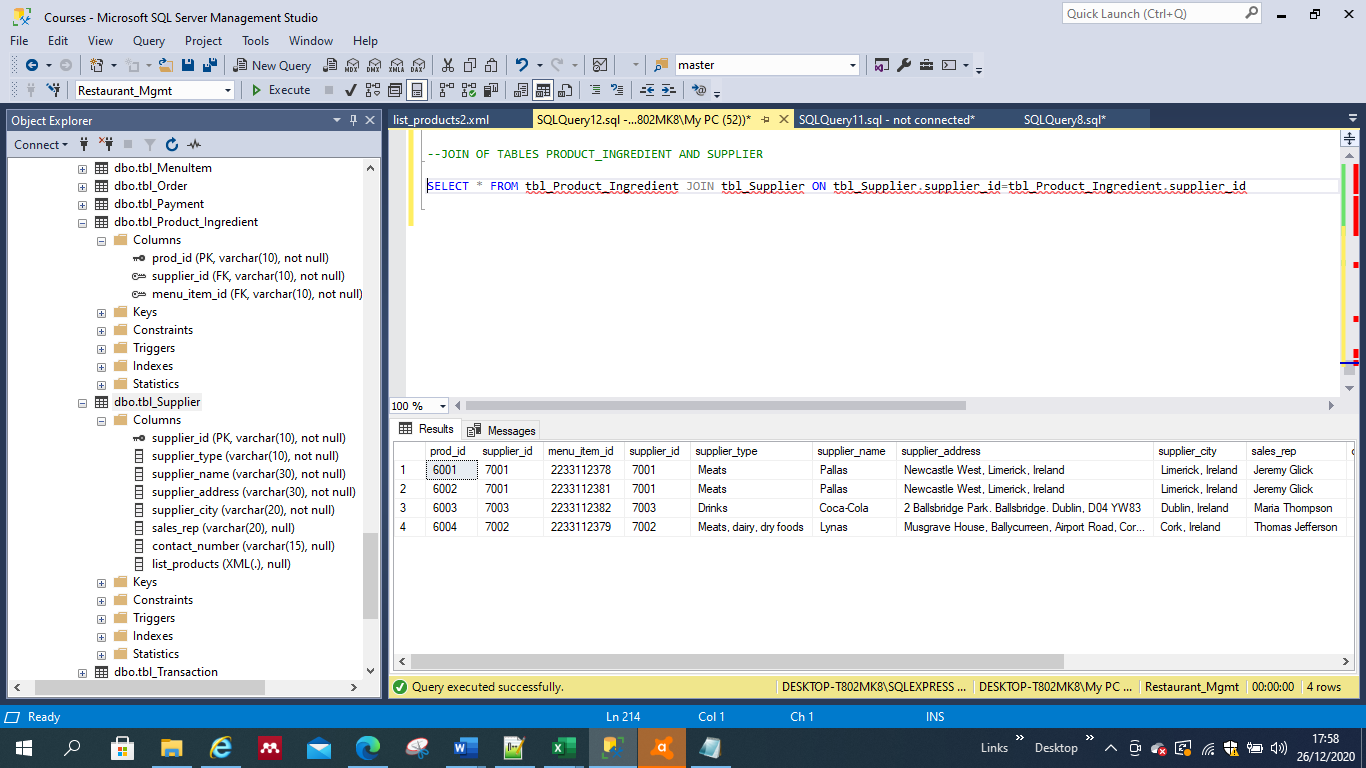






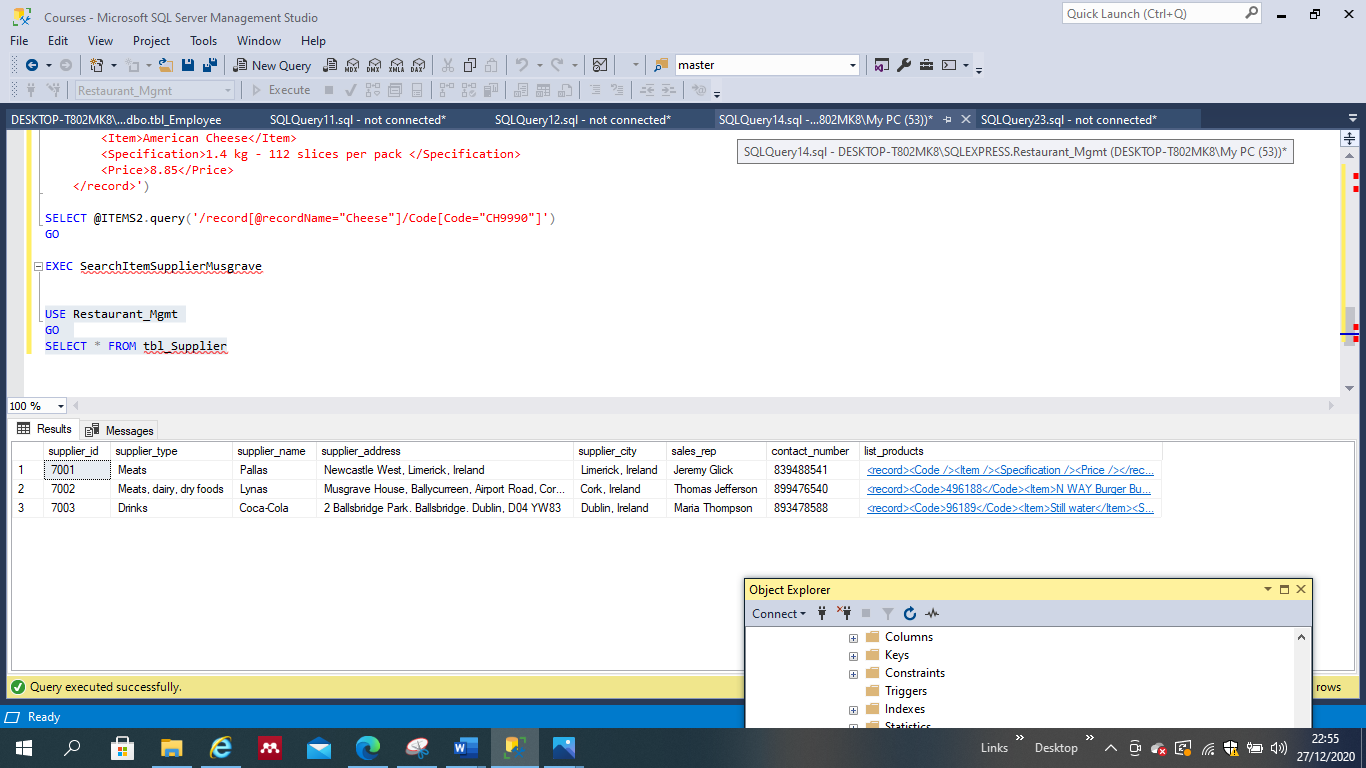
**6.3 Business Requirement No. 3. The user shall be able to search for suppliers lists and items containing in it.**

For this purpose, it is possible for the user to use JOIN, to join Supplier table and Product\_ingredient and obtain a full visualisation of all items and suppliers linked to them. See figure below:



Moreover, a field called ‘list\_products’ containing XML data was included in the Supplier Table to provide better functionality, for the reasons mentioned before. See figure below.

**Figure No. XML field.**



Lastly, stored procedures with parameters to search the XML data were created. See example below:

CREATE PROCEDURE SearchItemSupplierMusgrave

AS

DECLARE @ITEMS2 XML

SET @ITEMS2 = ('<XML DATA>')

SELECT @ITEMS2.query('/record[@recordName="Cheese"]')

GO

EXEC SearchItemSupplierMusgrave

In this case, by entering the word ‘Cheese’ and executing the procedure, the system does the search and provides all items that are identified as ‘Cheese’ from the Musgrave supplier’s list. See figures below:

Figure No. Search in XML #2.

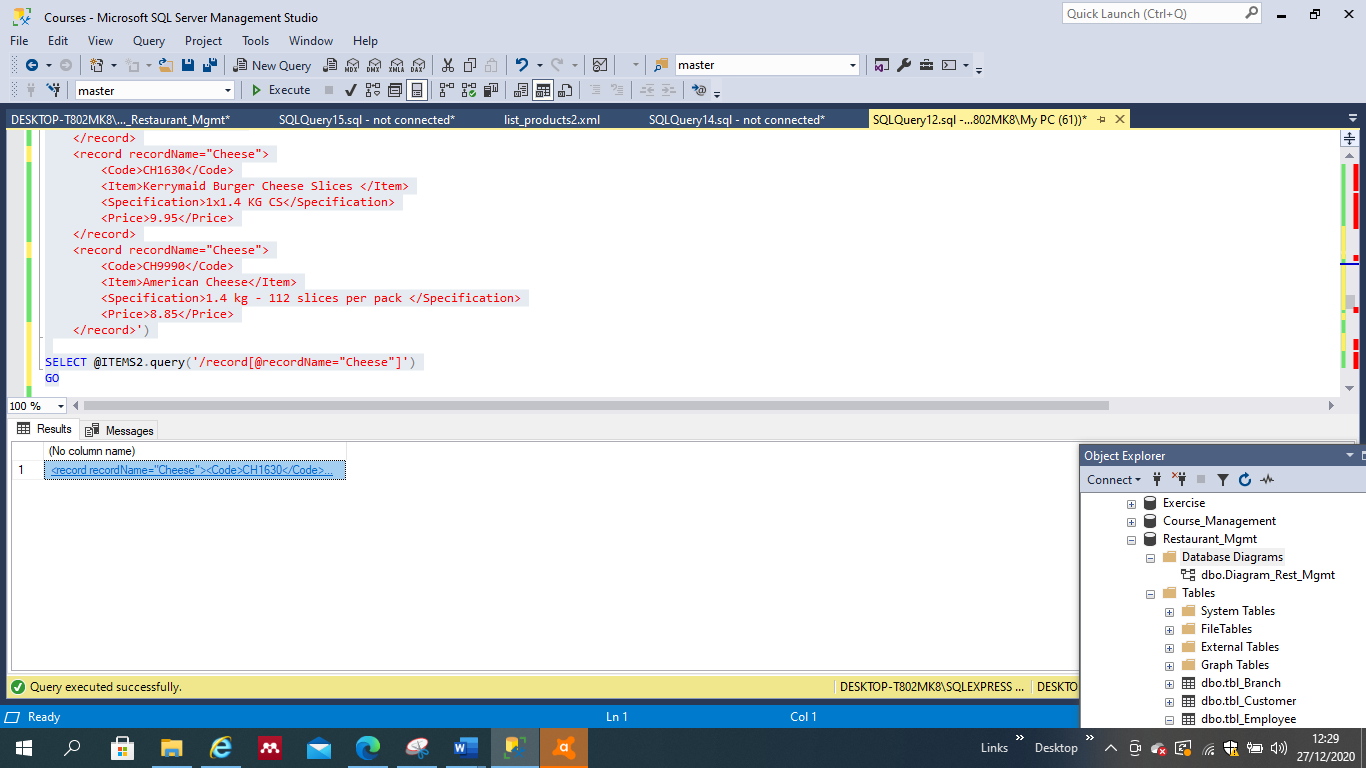


Figure No. XML results #1.

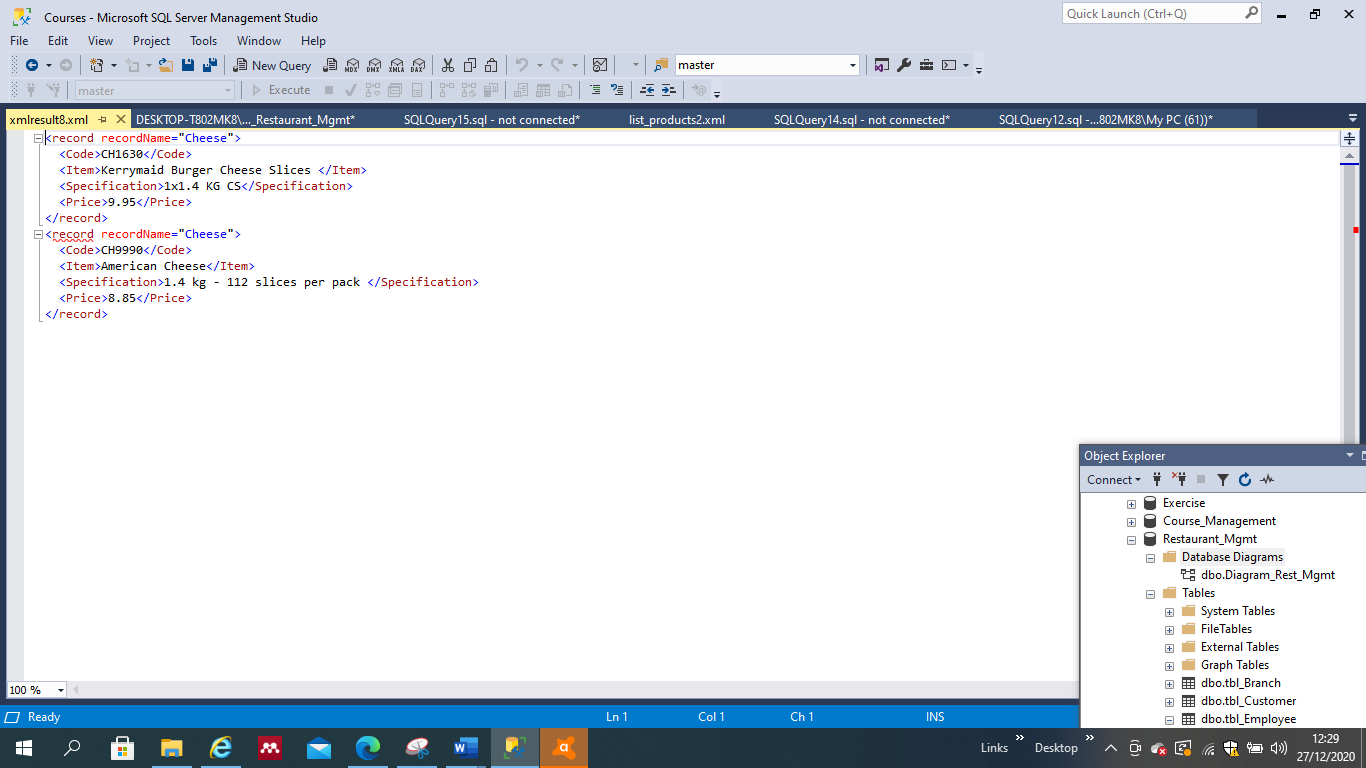


Figure No. Search in XML #1.

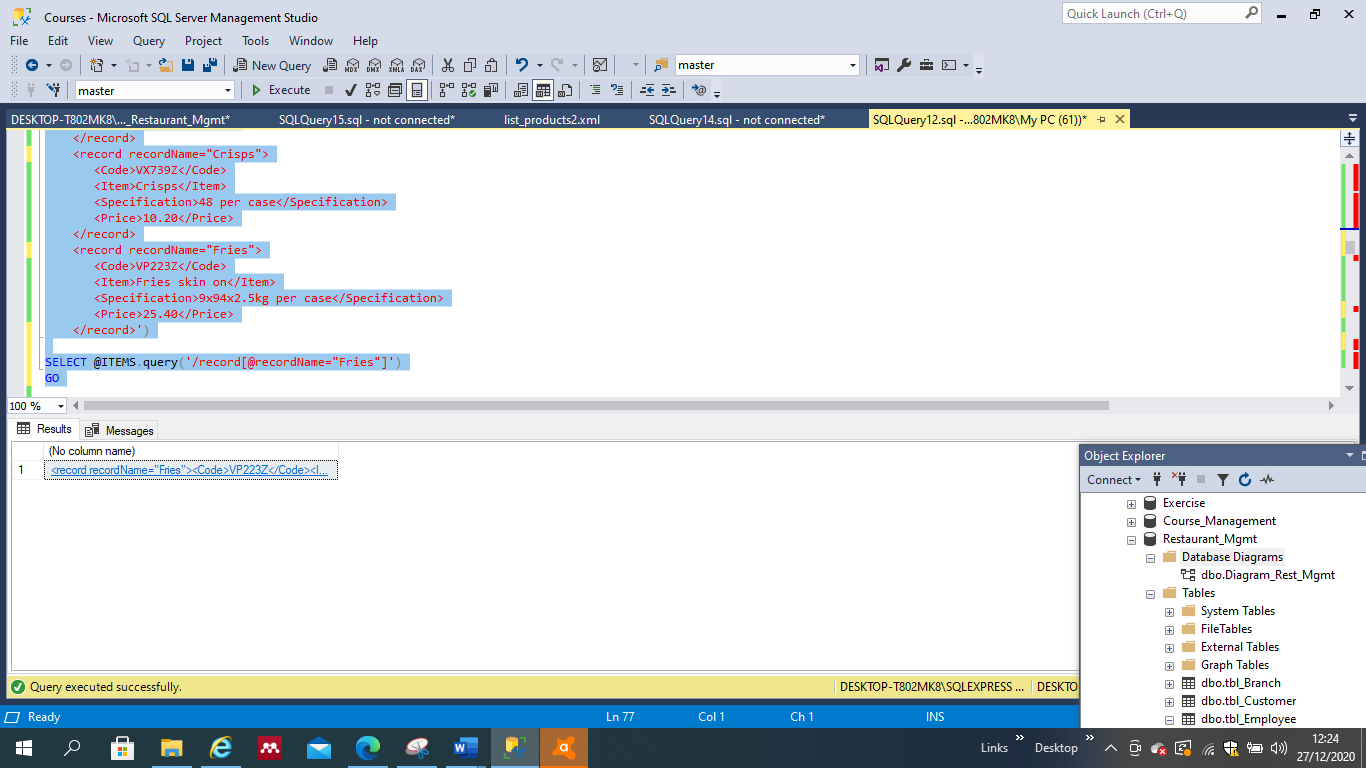
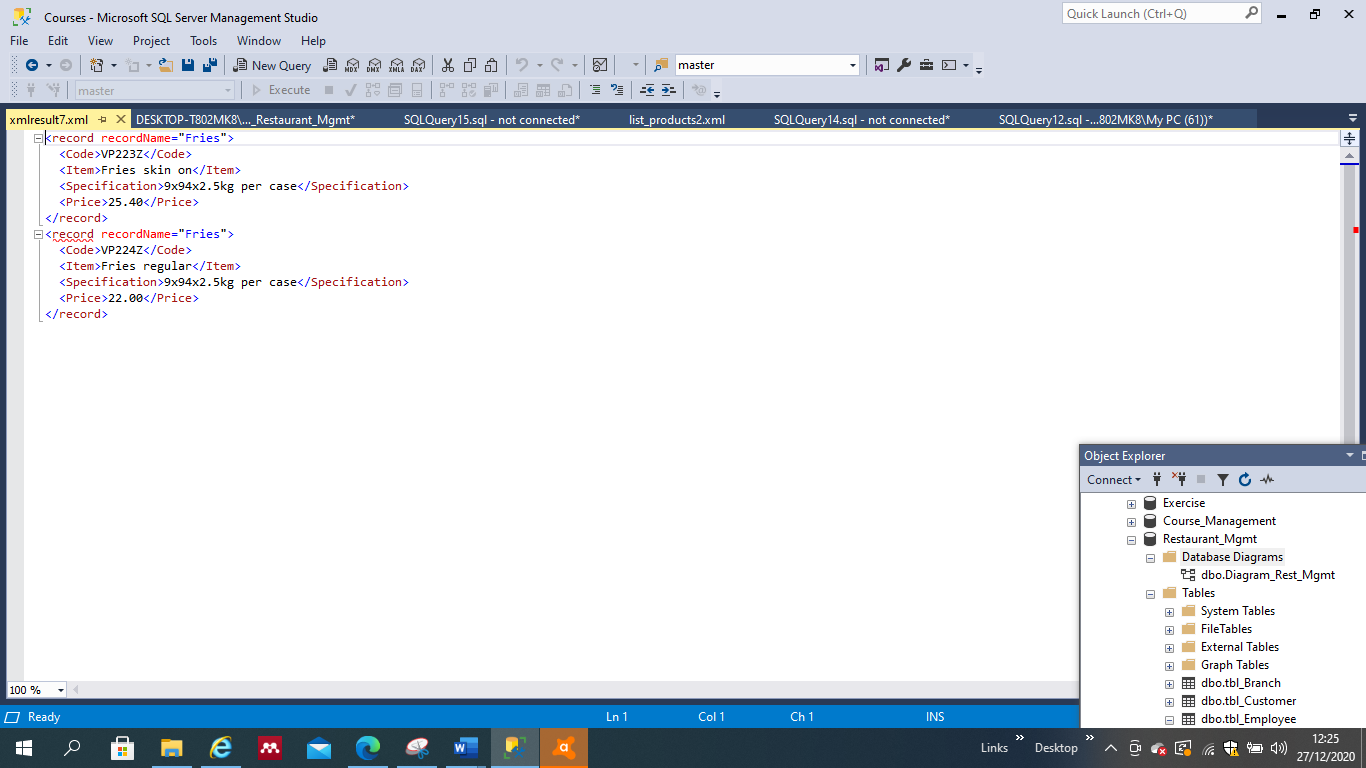


Figure No. XML results #2.



**6.4 Business Requirement No. 4. A full list of employees of all branches should be made available.**

For this purpose, a stored procedure was created called ‘GetAllEmployees’ as follows:

USE Restaurant\_Mgmt

GO

CREATE PROCEDURE GetAllEmployees

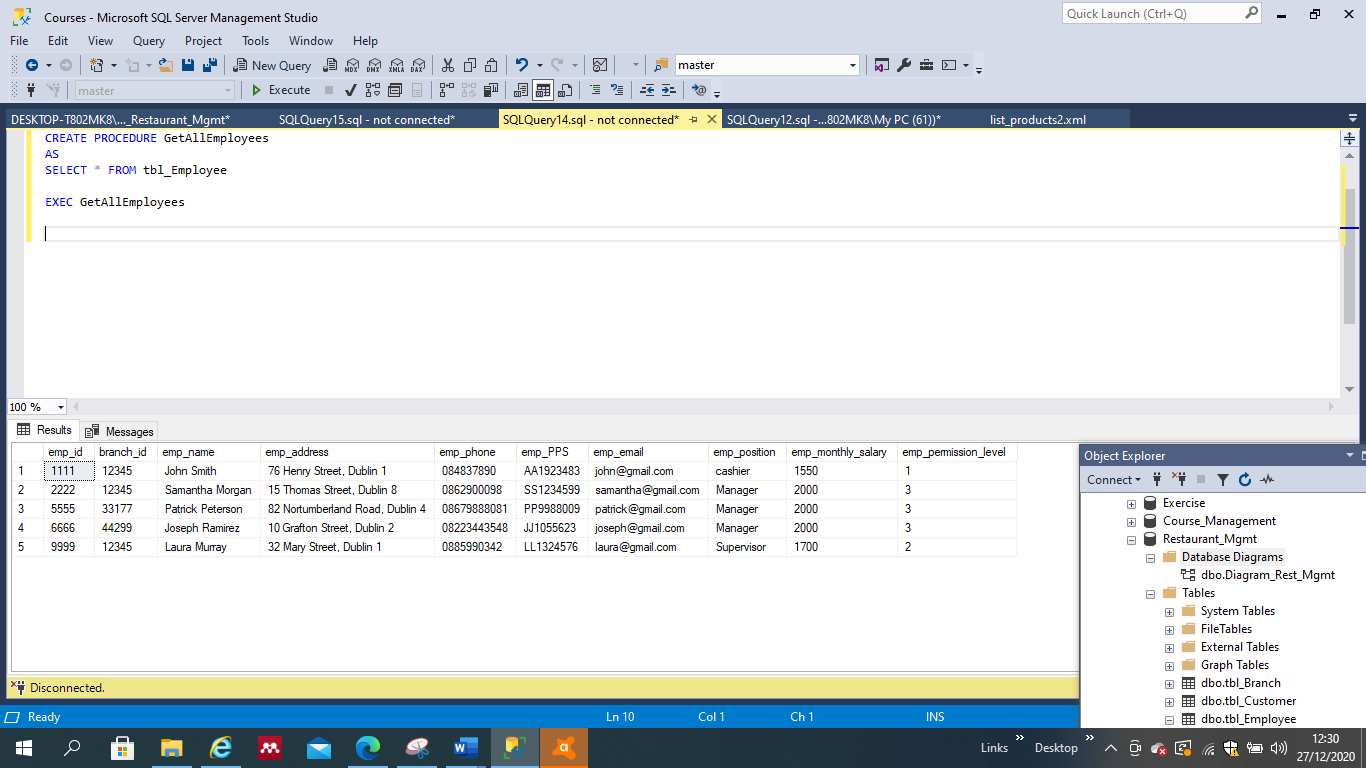
AS

SELECT \* FROM tbl\_Employee

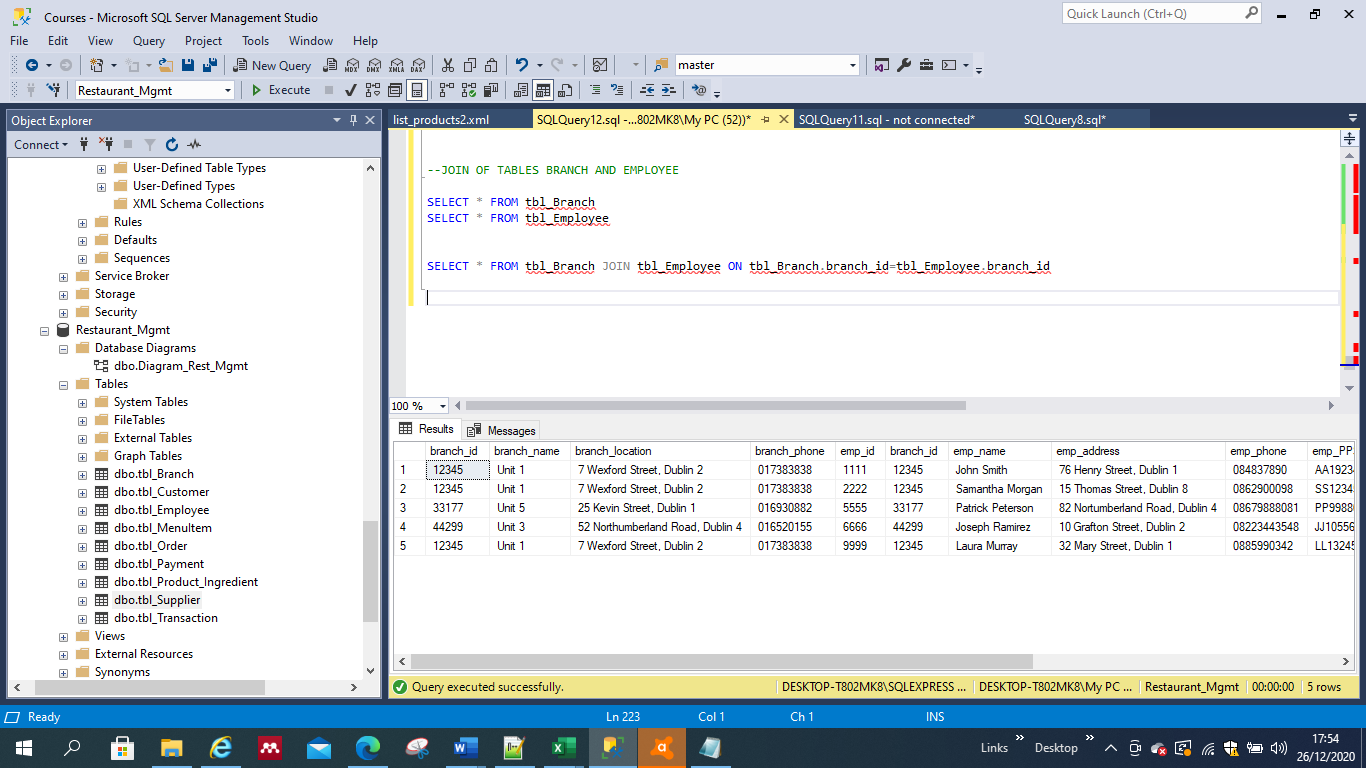
EXEC GetAllEmployees

The figure below shows the full list of employees from all branches:

Figure No. Procedure: Get All Employees.



As mentioned before, it is also possible to visualize two tables, in this case, Employee and Branch tables, by the use of JOIN as follows:



This would also give the user a full visualisation of all employees working in all branches.

**6.5 Business Requirement No. 5. The system shall provide a practical option to register a new employee.**

For this purpose, a **Trigger** called ‘new\_Employee’ was created as follows:

*CREATE TRIGGER [dbo].[new\_Employee] ON [dbo].[tbl\_Employee]*

*AFTER INSERT*

*AS*

*BEGIN*

*DECLARE @emp\_id VARCHAR(4);*

*DECLARE @branch\_id VARCHAR(5);*

*DECLARE @emp\_name VARCHAR(20);*

*DECLARE @emp\_address VARCHAR(100);*

*DECLARE @emp\_phone VARCHAR(20);*

*DECLARE @emp\_PPS VARCHAR(15);*

*DECLARE @emp\_email VARCHAR(20);*

*DECLARE @emp\_position VARCHAR(20);*

*DECLARE @emp\_monthly\_salary INT;*

*DECLARE @emp\_permission\_level VARCHAR(1);*

*SELECT @emp\_id=empList.emp\_id FROM inserted empList*

*SELECT @branch\_id=empList.branch\_id FROM inserted empList*

*SELECT @emp\_name=empList.emp\_name FROM inserted empList*

*SELECT @emp\_address=empList.emp\_address FROM inserted empList*

*SELECT @emp\_phone=empList.emp\_phone FROM inserted empList*

*SELECT @emp\_PPS=empList.emp\_PPS FROM inserted empList*

*SELECT @emp\_email=empList.emp\_email FROM inserted empList*

*SELECT @emp\_position=empList.emp\_position FROM inserted empList*

*SELECT @emp\_monthly\_salary=empList.emp\_monthly\_salary FROM inserted empList*

*SELECT @emp\_permission\_level=empList.emp\_permission\_level FROM inserted empList*

*INSERT INTO tbl\_Employee(emp\_id, branch\_id, emp\_name, emp\_address, emp\_phone, emp\_PPS, emp\_email, emp\_position, emp\_monthly\_salary, emp\_permission\_level)*

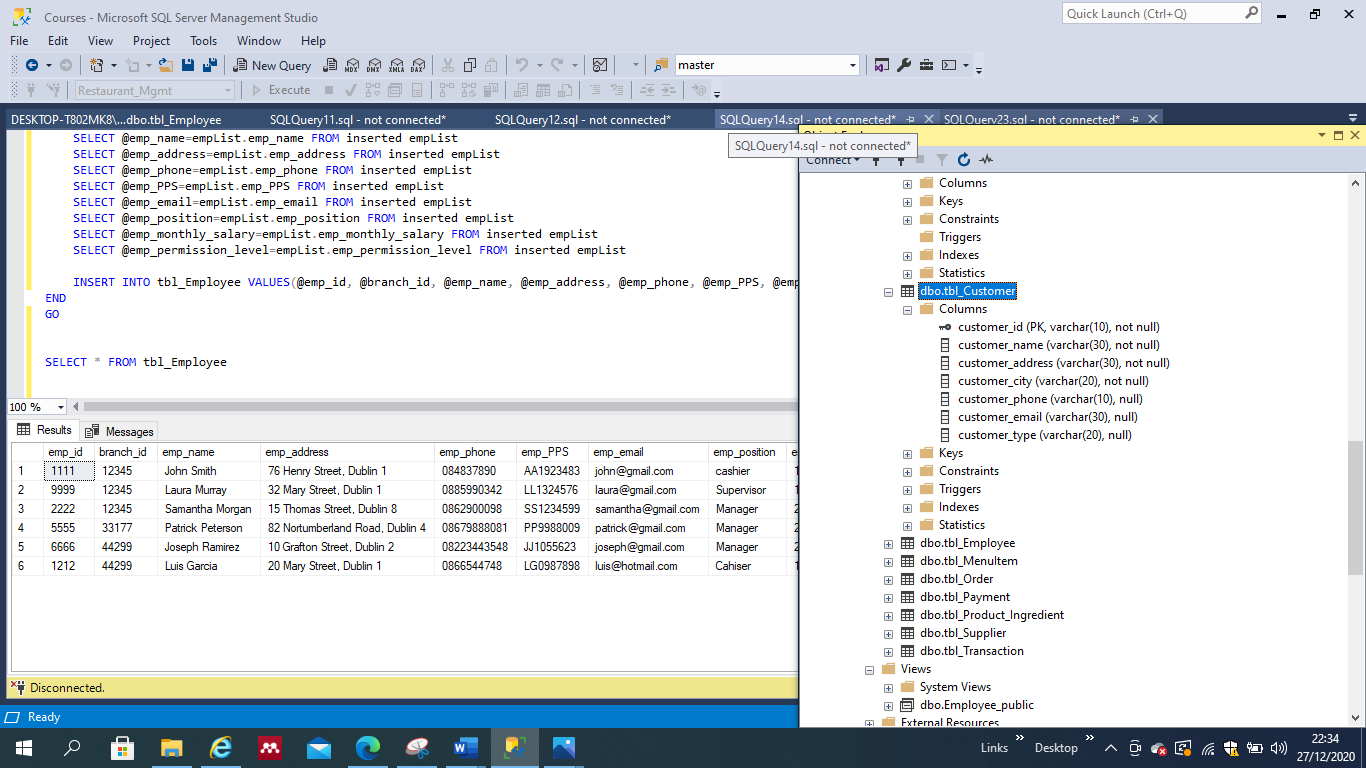
*VALUES(@emp\_id, @branch\_id, @emp\_name, @emp\_address, @emp\_phone, @emp\_PPS, @emp\_email, @emp\_position, @emp\_monthly\_salary, @emp\_permission\_level)*

*END*

*GO*

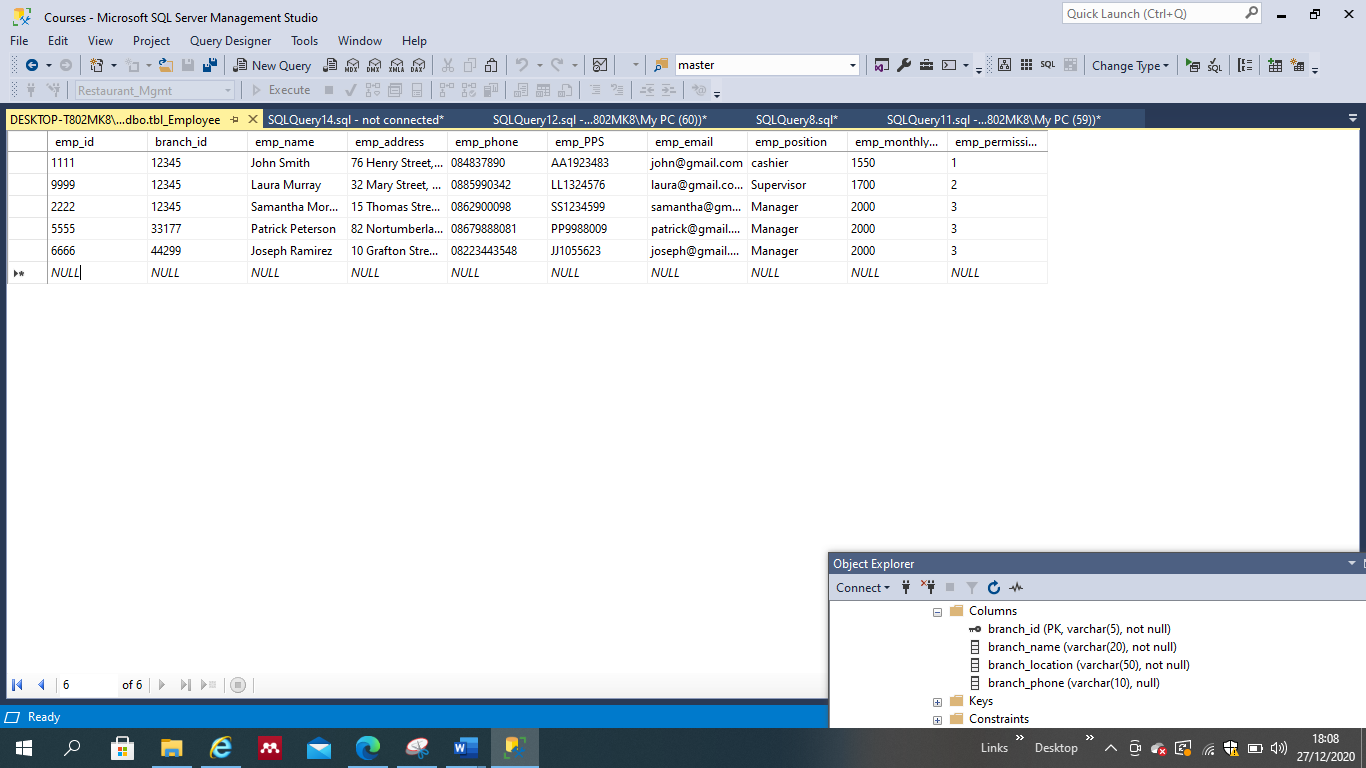
By executing this trigger and right-clicking on the Employee table, edit rows, the user is able to add a new employee record directly on the table, without having to manage SQL code:

See figure No. Edit table option



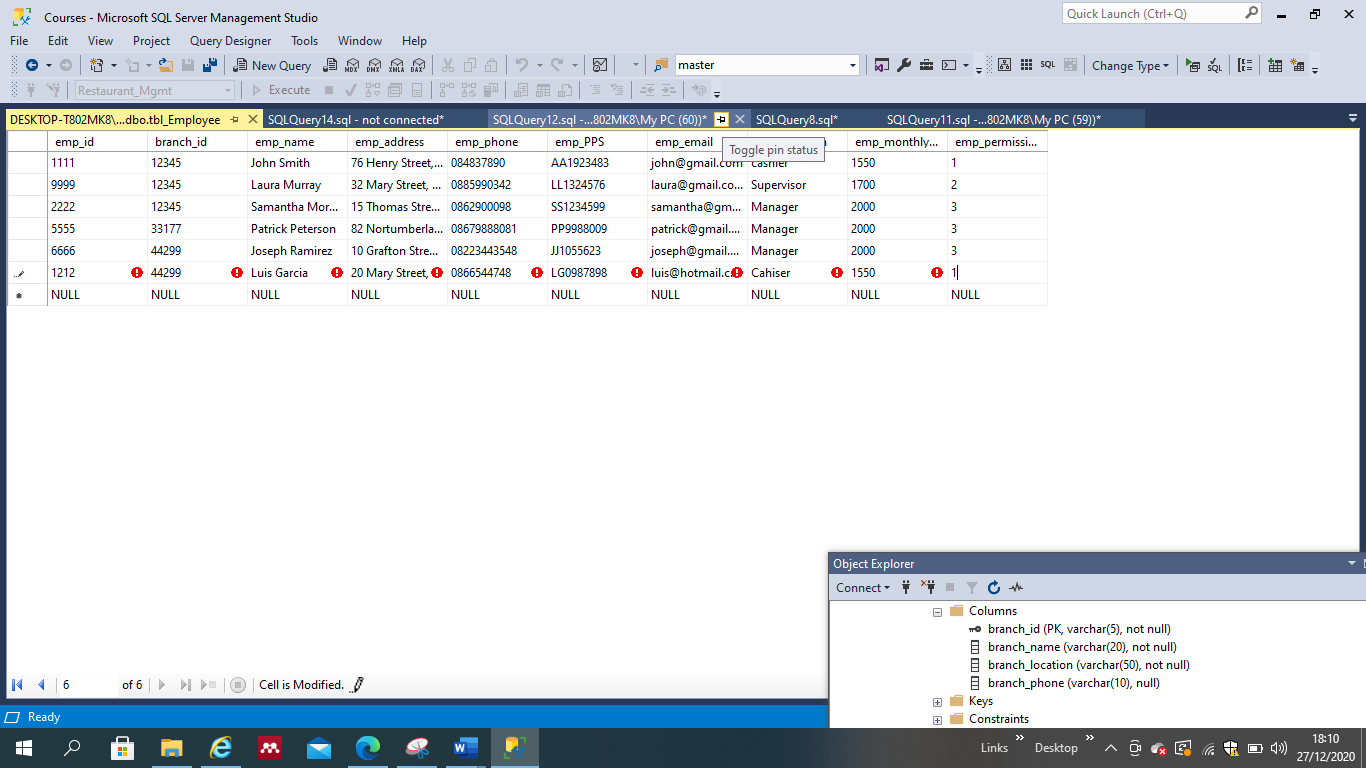
The table then will appear, allowing the user to register a new employee on the ‘NULL’ fields. See figure below.

Figure No. Add new employee.



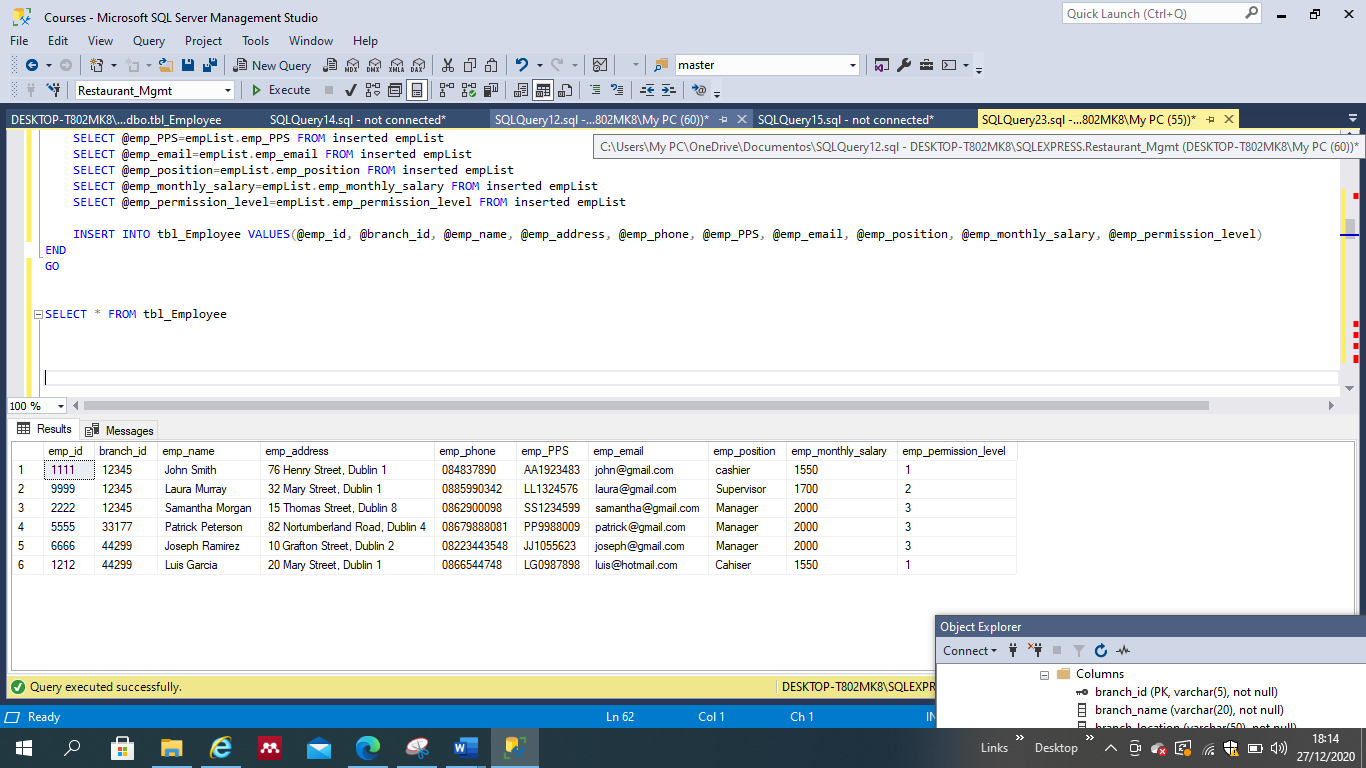
The user would fill in the camps according to data required and constraints; if the user enters a value that is not allowed, the system would prompt an error.

Figure No. Entering data.

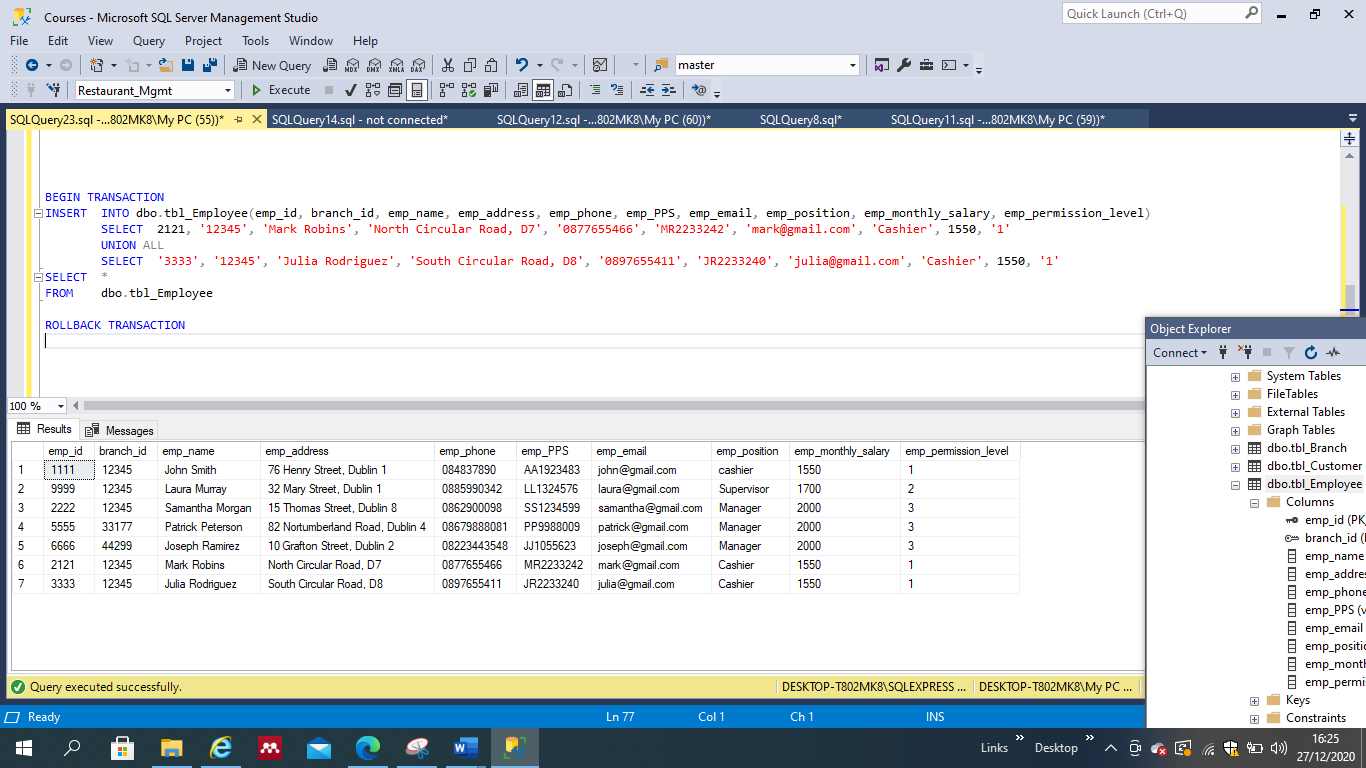


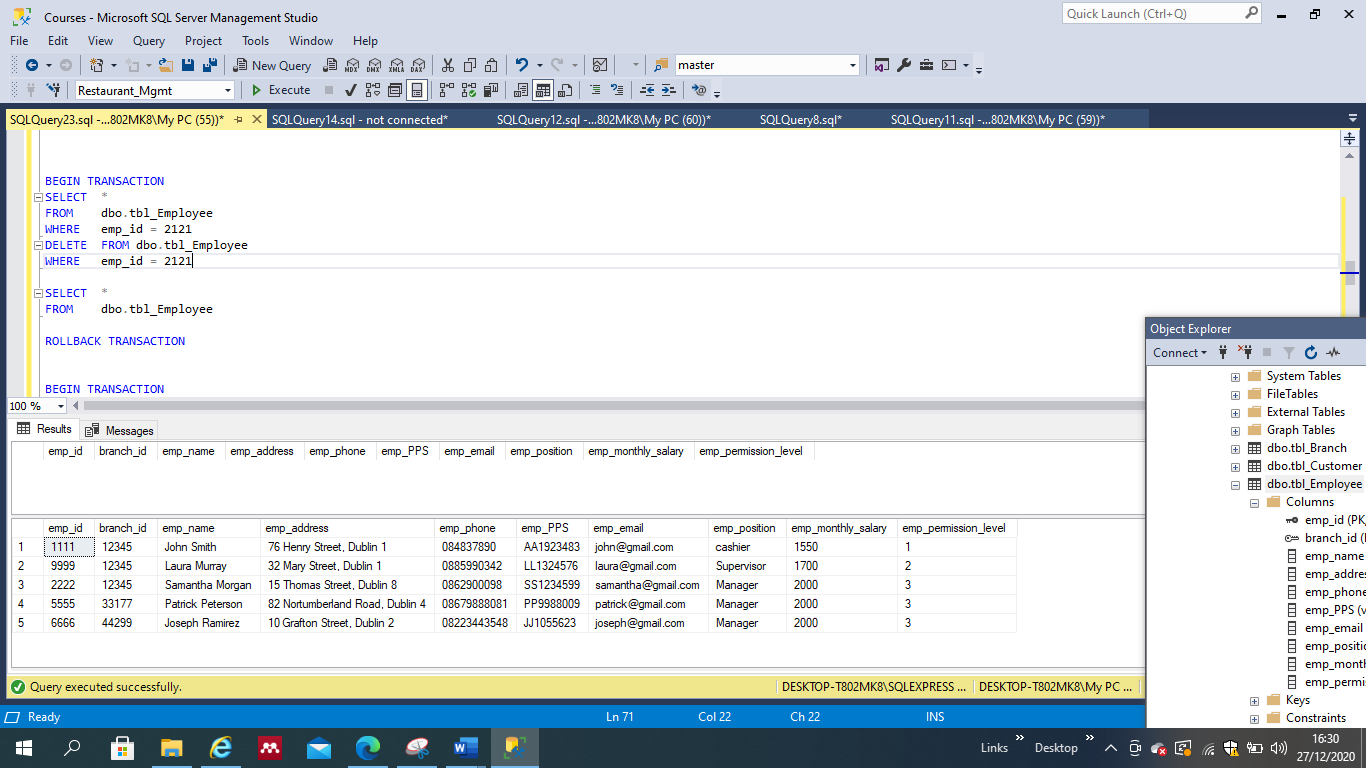
Once all data is correct, the user is able to save the date by hitting enter key. By querying a SELECT command, it is possible to verify that changes have been made on the table. See figure below.

Figure No. Table updated.



**6.6 Business Requirement No. 6. ????**

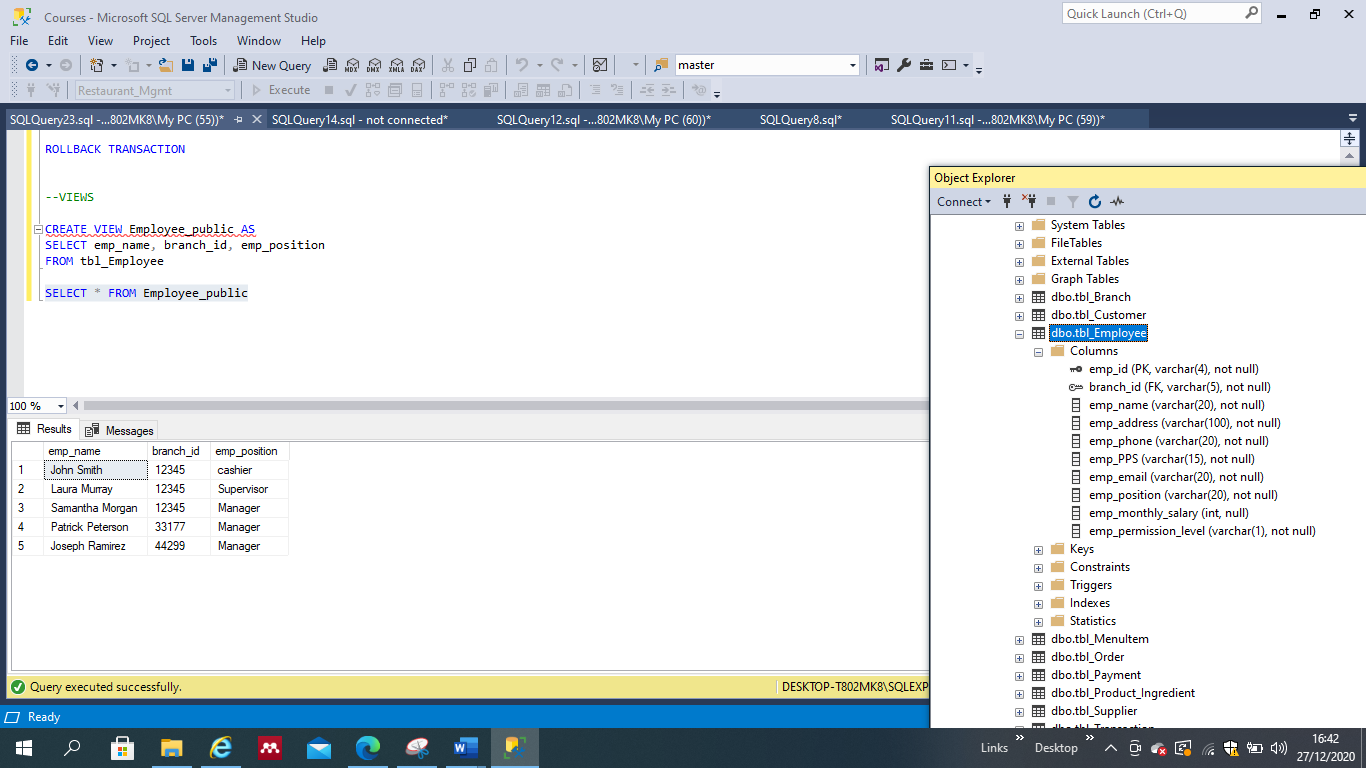




**6.7 Business Requirement No. 7. The system shall be able to provide protection to confidential or sensitive information.**

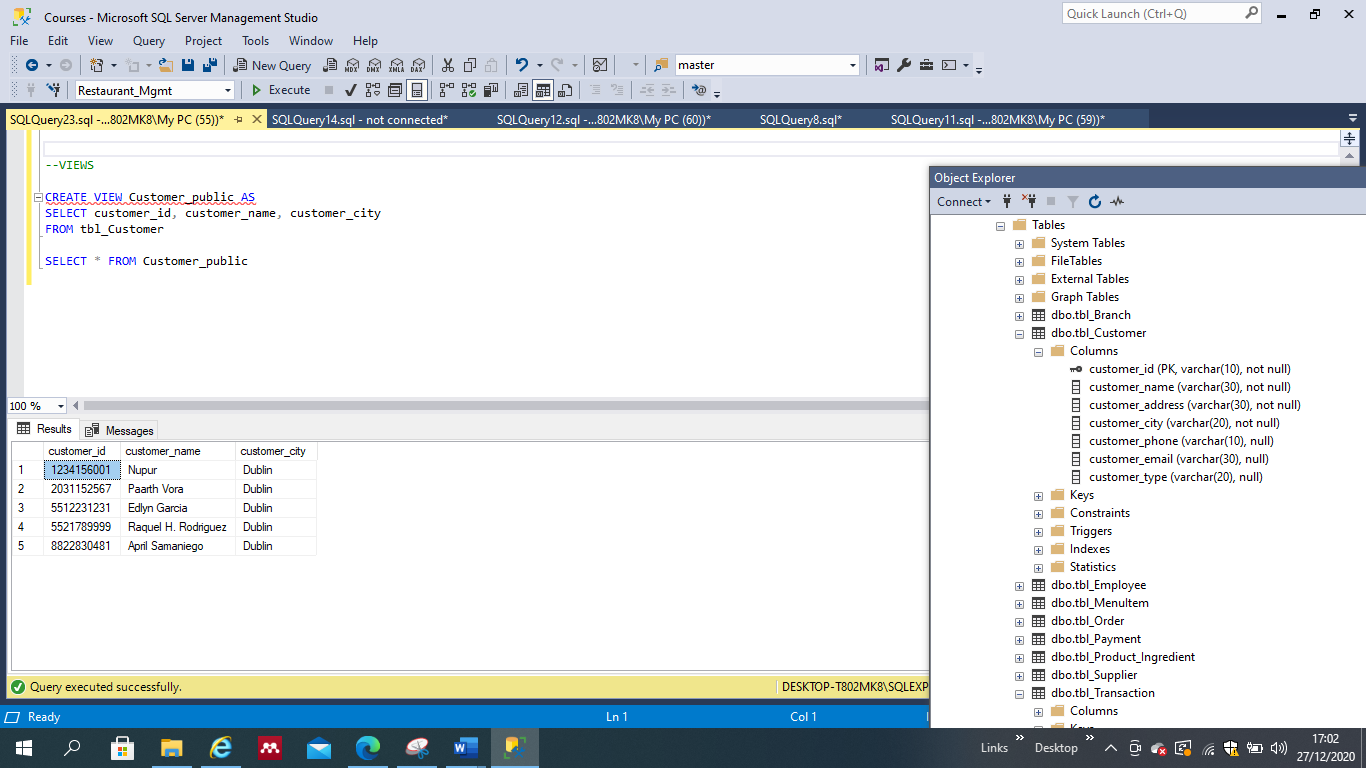
For this purpose, VIEWS were created and used to create virtual tables that shall be made available for public use. In the following example, a virtual table called “Employee\_public” was created including the fields: emp\_name, emp\_position and branch\_id only. By following this protocol, sensitive information, such as employee’s address, telephone number, salary, among others, is being kept in private. See figure below.

Figure No. Views-Employee\_public



Similarly, a view called “Customer\_public” was created to maintain customers’ sensitive information private. The virtual table created contains only customer\_id, customer\_name and customer\_city information. See figure below.

Figure No. Views-Customer\_public

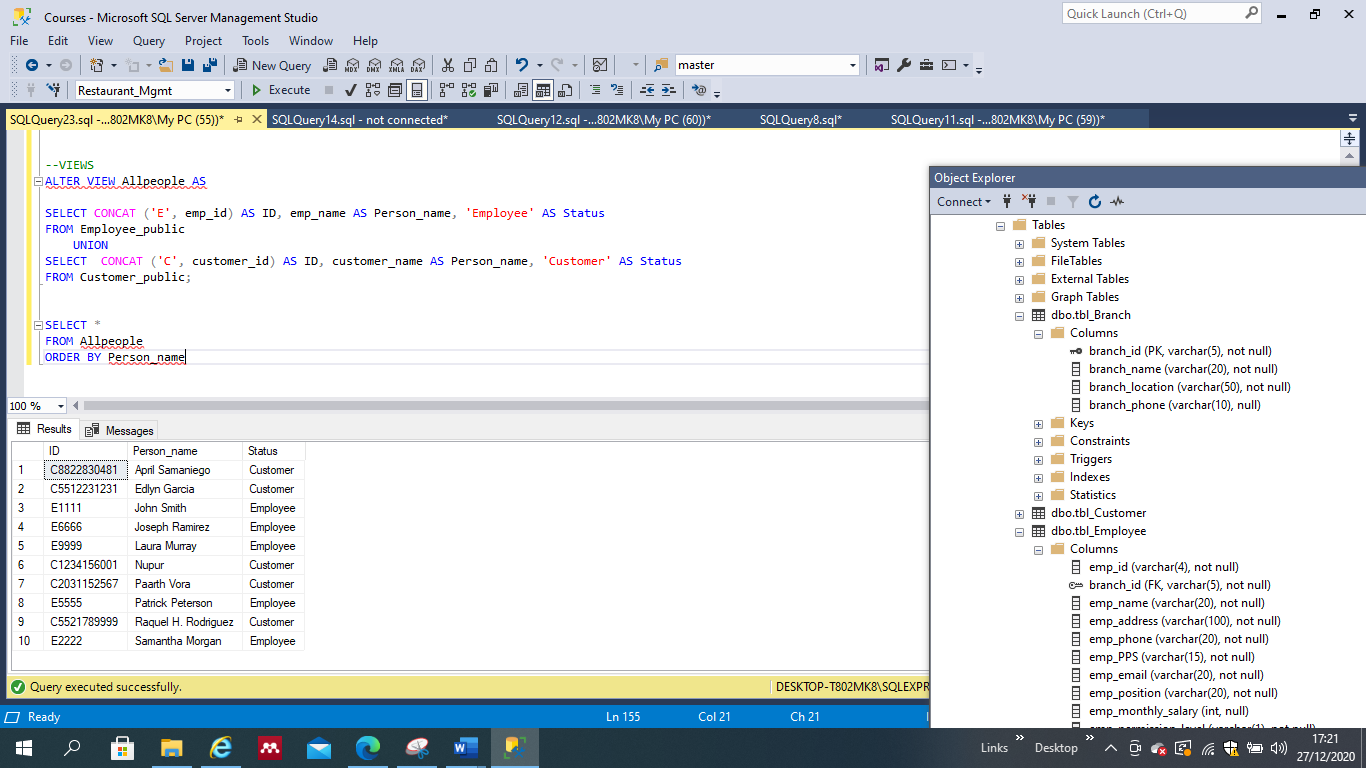


1. **BENEFITS OF VIEWS.**

One of the main benefits of views is their capability to provide Security. As seen before, by creating these virtual tables, it is possible to keep sensitive information private, and provide public access only to those tables or virtual tables with fields that do not contain sensitive information.

Another benefit is the simplicity and business logic consistency that they provide to any business.

As an example of their simplicity feature, a view called “All People” was created by the UNION of two virtual tables (Customer and Employee) that contained only fields without sensitive information, which can be made public. The Union of such tables would provide a full list of all the people that are involved in the business, either from a customer or staff perspective. An extra column was created by assigning a general identifier as ‘Person’ and an ID starting with a ‘C’ for Customers and ‘E’ for employees, to provide a differentiation between the two entites. See figure below.



This is an example of simplicity and how VIEWS can hide complexity. Instead of requiring users to query two tables or know how to apply UNION or JOIN, users can simply write a SELECT statement using this view.

Use of any other features to enhance the usability of your database system. Write a paragraph on innovation and how it is useful.

**INNOVATION**

According to McKinsey (2020), “Innovation is critical to growth, particularly as the speed of business cycles continues to increase” and most companies agree on that statement, as research shows that 84% of executives expressed their reliance on innovation for future success.

Which, combined with cloud computing and IaaS providers such as Microsoft Azure, AWS or Google Cloud

In

**CONCLUSIONS**

**BIBLIOGRAPHY**

**TO DO:**

**Relational schema**

**1 Business requirement and implementation.**

**1 Trigger and implementation.**

**Contributions: 1 EACH.**

**Conclusions.**