TECHNOLOGICAL INSTITUTE OF THE PHILIPPINES
COMPUTER ENGINEERING DEPARTMENT
Quezon City

Laboratory Manual

DATABASE MANAGEMENT SYSTEM

Engr. Royce Chua
Engr. Alonica Villanueva

Activity No. 1	
SQL Data Definition Language Commands	
Course Code: CPE011	**Program:**
Course Title: Database Management System	**Date Performed:**
Section:	**Date Submitted:**
Name:	**Instructor:**

1. Objective(s):

This activity aims to introduce the Structured Query Language (SQL) using the Data Definition Language (DDL) commands in a MySQL Database

2. Intended Learning Outcomes (ILOs):

The students should be able to:
2.1 Access a MySQL Database using Command Line Interface.
2.2 Create, Alter, and Drop a database schema using the command prompt, MySQL, and XAMPP
2.2 Create, Alter, and Drop a table and its fields in a database schema
2.3 Show the different databases, tables and their structures present in the MySQL Server
.

3. Discussion:

MySQL is a database management system that allows you to manage relational databases. It is open source software backed by Oracle. It means you can use MySQL for free. In addition, if you want, you can change its source code to suit your needs. Even though MySQL is open source software, you can buy a commercial license version from Oracle to get a premium support service.

MySQL is pretty easy to master in comparison with another database software like Oracle Database, or Microsoft SQL Server. MySQL can run on various platforms UNIX, Linux, Windows, etc. You can install it in a server or even in a desktop. In addition, MySQL is reliable, scalable, and fast.

We can interact with an SQL Database such as MySQL using various SQL commands that are grouped into 2 main categories: **Data Definition Language (DDL)**, and the **Data Manipulation Language (DML)**. Other categories of commands are used for more advanced use-cases such as user permissions, roles, and transactions.

The **Data Definition Language (DDL)** allows you to create and define the structure of your database and your tables. Moreover, it allows you to alter the structure of your database and also to delete/drop them if necessary. The SQL DDL Commands that are commonly used in database programming are: **CREATE**, **DROP**, and **ALTER.**

In this activity you will use SQL DDL Commands to create a database and a database table, then alter the structure of a table, and finally perform cleanup by dropping the tables and the database itself. To interact with a database, you will first need to connect to a MySQL Database Server.

A sample of the command to create a database is shown below:

```
MariaDB [(none)]> CREATE DATABASE db1;
Query OK, 1 row affected (0.07 sec)

MariaDB [(none)]>
```

For creating tables, you will need to be familiar with the various data types needed:

Data Types

A data type or simply type is a classification identifying one of various types of data.

In MySQL, there are three main types of data: Text, Number and Date/Time. This table shows some of the data types available in MySQL although there are many more variations besides the ones in this discussion.

Data Type	Description
CHARACTER(n)	Character string. Fixed-length n
VARCHAR(n) or CHARACTER VARYING(n)	Character string. Variable length. Maximum length n
BINARY(n)	Binary string. Fixed-length n
BOOLEAN	Stores TRUE or FALSE values
VARBINARY(n) or BINARY VARYING(n)	Binary string. Variable length. Maximum length n
INTEGER(p)	Integer numerical (no decimal). Precision p
SMALLINT	Integer numerical (no decimal). Precision 5
INTEGER	Integer numerical (no decimal). Precision 10
BIGINT	Integer numerical (no decimal). Precision 19
DECIMAL(p,s)	Exact numerical, precision p, scale s. Example: decimal(5,2) is a number that has 3 digits before the decimal and 2 digits after the decimal
NUMERIC(p,s)	Exact numerical, precision p, scale s. (Same as DECIMAL)
FLOAT(p)	Approximate numerical, mantissa precision p. A floating number in base 10 exponential notation. The size argument for this type consists of a single number specifying the minimum precision.
REAL	Approximate numerical, mantissa precision 7
FLOAT	Approximate numerical, mantissa precision 16
DOUBLE PRECISION	Approximate numerical, mantissa precision 16
DATE	Stores year, month, and day values
TIME	Stores hour, minute, and second values
TIMESTAMP	Stores year, month, day, hour, minute, and second values
INTERVAL	Composed of a number of integer fields, representing a period of time, depending on the type of interval
ARRAY	A set-length and ordered collection of elements
MULTISET	A variable-length and unordered collection of elements
XML	Stores XML data

You will need to select the right data type for each field in your database table. More information about each data type is available on the MySQL official documentation: https://dev.mysql.com/doc/refman/5.7/en/data-types.html

A sample of the command to create a database is shown below:

```
MariaDB [(none)]> use db1;
Database changed
MariaDB [db1]> CREATE TABLE students(first_name CHAR(50), last_name CHAR(50),
    ->                               age INT, date_lastenrolled DATE, is_enrolled BOOLEAN);
Query OK, 0 rows affected (0.91 sec)

MariaDB [db1]>
```

XAMPP which stands for (**X** – Cross Platform, **A** – Apache, **M** – MariaDB, **P** – PHP, **P** – PERL) is a cross-platform, open-source package of various software which includes a variant of MySQL DB called MariaDB to help you easily get started with databases and using various SQL commands. We will be using XAMPP to access a MySQL database in this activity.

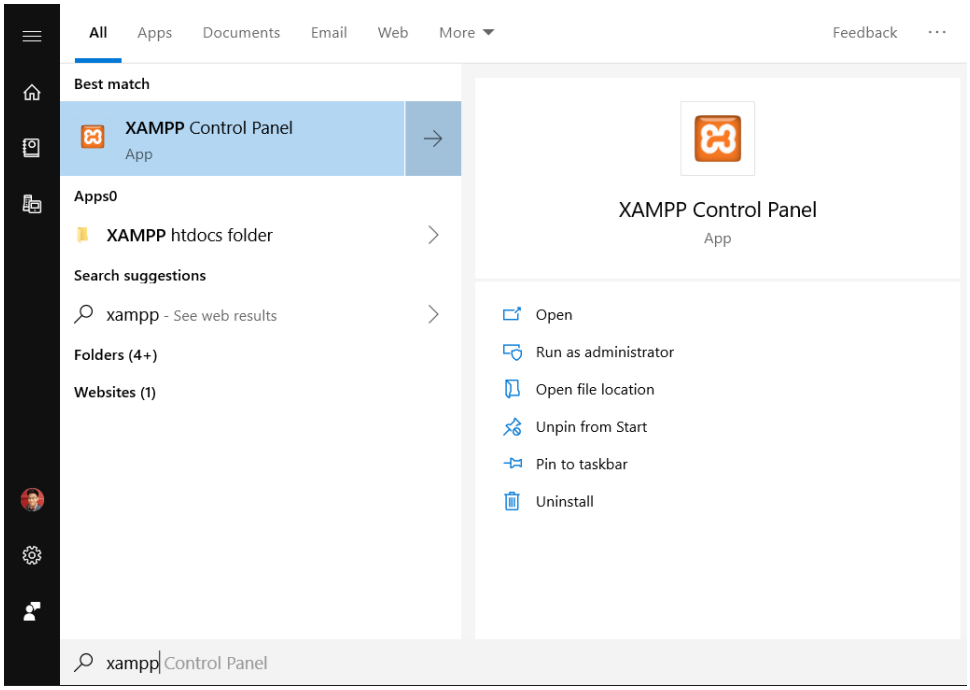
4. Materials and Equipment:

Desktop Computer
Windows Operating System
XAMPP Application

5. Procedure:

Starting MySQL using XAMPP

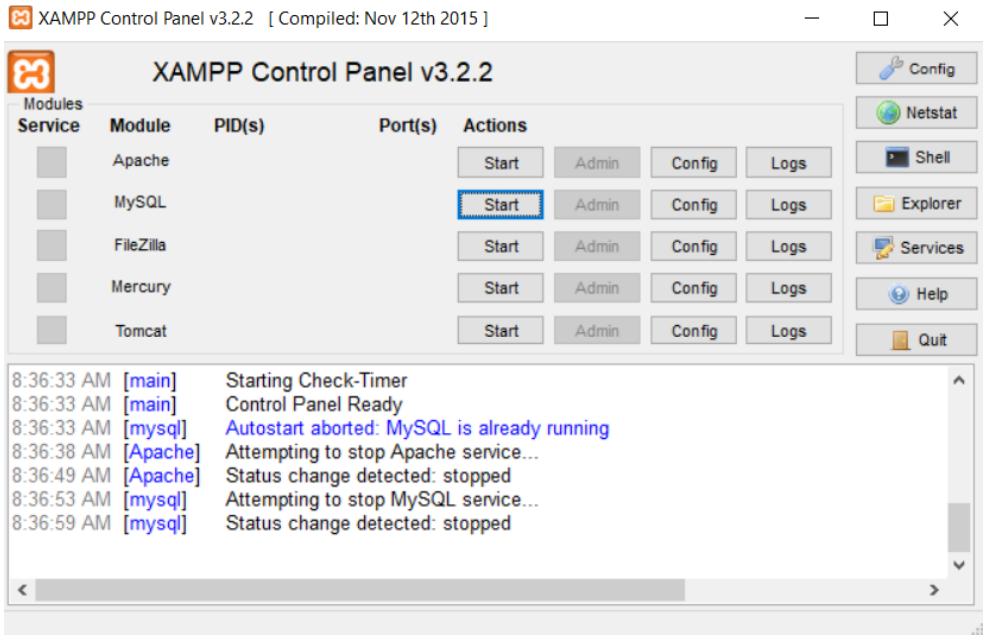
1. Select the XAMPP Control Panel Application from the Desktop or Start Menu



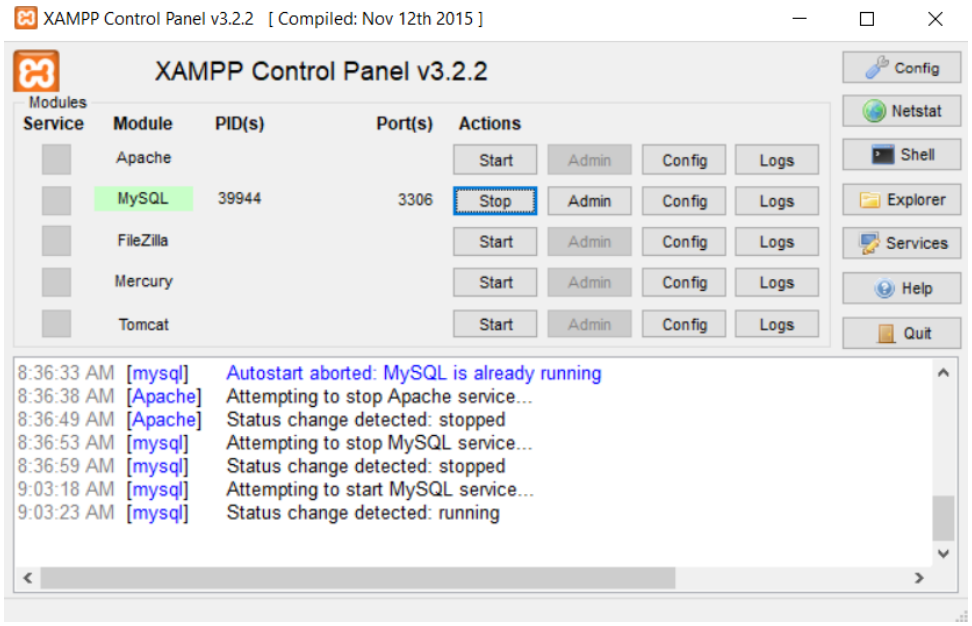
Accessing **XAMPP** is the same for both Windows 10 and lower Windows Operating System versions. It can be searched easily using the Windows Search field.

2. On the dialog box, press *Start* button located on the same row as the MySQL Label.

   Make sure that the MySQL label has changed its background color to green and the Start button has changed to Stop. Additional log information will display the Status as running.



After pressing start, the Xampp control panel should look similar to the image below



3. Close the Dialog Window.

Accessing the MySQL Application using Command Line Interface

1. Run the Command Line Interface (cmd) through the start menu or desktop.

2. Type the following command to navigate to the MySQL Folder:

cd c://xampp/mysql/bin

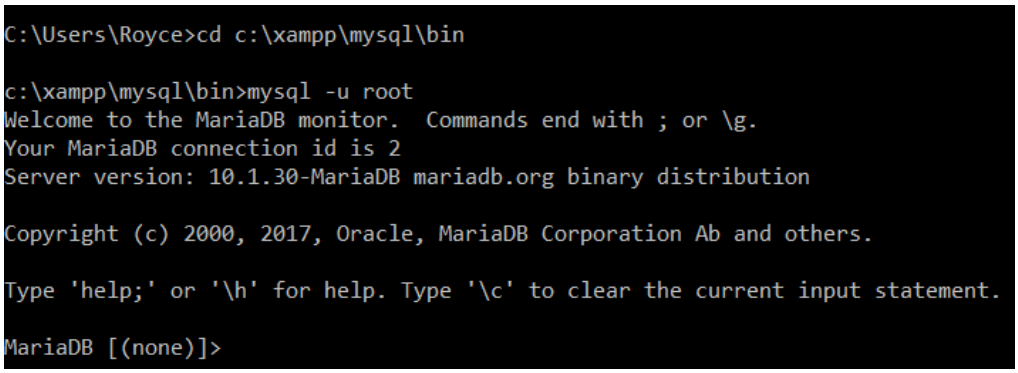
The *cd* command sets the current directory of the command line allowing you to access certain executable files and folders within that folder.

3. Type the following command to connect to the MySQL database server.

mysql.exe -u root

The command runs the program mysql.exe with an additional argument -u which means user and root meaning the root user. The MySQL database comes with a default user called root with no initial password.

4. The output should look similar to the image below:



Navigating through the MySQL Database

1. Type the following command, show the screenshot of the output with your observation below it in **Section 6 Database Output**.

   - **SHOW DATABASES;**

Creating a Database

1. Type the following command to create a MySQL Database. Note: You will use this database for activity 1.

   - **CREATE DATABASE vehiclesdb;**

2. Run the ***show databases*** command again in the command line. Show the screenshot of the output with your observation below it in Section 6 Database Output.

Deleting a Database

1. To delete an existing database, use the command:

   - **DROP DATABASE vehiclesdb;**
     Note: You will have to recreate vehiclesdb for the next part of the activity.

2. Run the ***show databases*** command again in the command line. Show the screenshot of the output with your observation below it in Section 6 Database Output.

Using a Database

1. Before you are able to create a table in database, you will need to select the database you want to use. You can use the command use.

   - **USE vehiclesdb;**

2. Show the screenshot of the output with your observation below it in the Data and Results section.

Creating a Table

1. Once the database has been selected, you can create a table with the following attributes using the following command:

   - **CREATE TABLE *vehicles(car_maker CHAR(20), car_model CHAR(20),number_of_doors INT);***

2. Show the screenshot of the output with your observation below it in the Data and Results section.

3. To view the tables created on the database, use the command below. Show the screenshot of the output with your observation below it in Section 6 Database Output.

   - **SHOW TABLES;**

4. To view the structure of a particular table, use the command below. Show the screenshot of the output with your observation below it in Section 6 Database Output.

   - **DESCRIBE vehicles;**

Altering the table structure

1. To change or alter the structure of an existing database, follow the sequence of commands as an example. Show the screenshot of the output of each bullet with your observation below it in Section 6 Database Output.

   - **ALTER TABLE vehicles *RENAME* vehicle;**
   - **DESCRIBE vehicle;**
   - **ALTER TABLE vehicle MODIFY number_of_doors INT(2);**
   - **DESCRIBE vehicle;**
   - **ALTER TABLE vehicle ADD year_model DATE;**
   - **DESCRIBE vehicle;**
   - **ALTER TABLE vehicle CHANGE year_model year_released DATE;**
   - **ALTER TABLE vehicle DROP year_model;**
   - **DESCRIBE vehicle;**
   - **ALTER TABLE vehicle *RENAME* vehicles;**

The table structure can be modified in the following but not limited to: **Modifying** field structure like its Data Type, **Adding** new columns, and **Renaming** the table and existing field structures.

Deleting a table

1. To delete an existing table from the database, use the command:

   - **DROP TABLE vehicles;**

6. Database Output:

Copy screenshot(s) of your output with observations after completing the procedures provided in Part 5.

7. Supplementary Activity:

Tasks

1. Create a new database **driversdb_<LASTNAME>.** Show the new list of databases on the MySQL monitor.
2. Create a table named drivers with the following attributes:
    - ID
    - First Name
    - Last Name
    - Age
    - Address
    - Vehicles
    - Years driving
    - Status

      Note: (Active or Not Active)
3. Create another tabled named vehicles with the following attributes:
    - Car Maker
    - Car Model
    - Number of Doors
4. Display all the structure for each table. Place a screenshot of your tables in the initial output.
5. Add a new column on drivers called drivers_license with data type int.
6. Change the column of drivers_license data type to a CHAR(13)
7. Remove the field vehicles from the drivers table.
8. Display all the structure for each table. Place a screenshot of your tables in the final output.
   Note: Do not drop the database or any tables.
9. Cite the commands used for this activity and its output per number using the table below:

Tasks	Syntax	Actual SQL Command Used
Create a new database		
Create a table named drivers		
Create a table named vehicles		
Add a new column drivers_license		
Change the data type of drivers_license		
Remove the field vehicles from the drivers table		

Initial Output
<INSERT SCREENSHOT HERE>

Final Output

<INSERT SCREENSHOT HERE>

Questions:

1. **Try and run your SQL Commands in opposite case or in mixed cases (ex. SHOW TABLES – show tables). What is the output and what is the feature of SQL that you can see in doing this task?**

   _____

   _____

2. **Attempt creating a database with a similar name, but with varying capitalization. What is the result? Why?**

   _____

   _____

3. **Attempt creating a table with two similar names. What is the result? Why?**

   _____

   _____

4. **Try creating two fields in table with two similar names with same then with different data types. What are the results? Why?**

   _____

   _____

5. **From the output of Question #4. How does MySQL perform the check to determine the output the result in Question#4? Does its data type matter?**

   _____

   _____

6. **Why do you think the keywords in SQL commands written in the instructions of this manual and online references are capitalized? Does it affect the query in anyway? Why?**

   _____

   _____

8. Conclusion:

9. Assessment Rubric:

Activity No. 2	
SQL Data Manipulation Language Commands	
Course Code: CPE011	**Program:**
Course Title: Database Management System	**Date Performed:**
Section:	**Date Submitted:**
Name:	**Instructor:**

1. Objective(s):

This activity aims to introduce the Structured Query Language (SQL) using the Data Manipulation Language (DML) commands in a MySQL Database

2. Intended Learning Outcomes (ILOs):

The students should be able to:
2.1 Use the command line interface to perform SQL Data Manipulation Commands.
2.2 Insert data to a MySQL Database
2.3 Update the contents of a database table
2.4 Delete the contents of database table

3. Discussion:

Recall that in a relational database, we organize our data in terms of columns and rows or fields and tuples. In the previous activity, we have learned to create a database and its schema by create tables in it through the SQL Data Definition Language (DDL) commands of MySQL.

In order to insert or modify any data in the table as well as to read/retrieve values that you inserted to the database, you will need to use the SQL Data Manipulation Language also called DML. This activity will cover the four fundamental SQL DML commands SELECT, INSERT, UPDATE, and DELETE which is essential in any database connected system.

Some of the SQL DML Commands will require additional keywords or clauses and inputs in order for you to get your desired output.

Inserting Data to a MySQL table

To insert data into MySQL table, you would need to use SQL INSERT INTO command. Here is generic SQL syntax of INSERT INTO command to insert data into MySQL table:

```
INSERT INTO table_name ( field1, field2,...fieldN )
    VALUES
    ( value1, value2,...valueN );
```

Note that the field1, field2 are attributes in the table. The value1 and value2 are the instances of the attributes. This indicates that the fields must follow the same order as values.

An alternative way to insert data without using field names is by using this syntax:

```
INSERT INTO table_name
    VALUES
    ( value1, value2,...valueN );
```

This shows that the data does not necessarily require an indication of the respective attributes. This command inserts data to whichever column is present in the database, as long as the data type matches.

Viewing the data present in a database

The SELECT statement is used to select data from a database. The result is stored in a result table, called the result-set. To select all the data from a particular table, the following syntax can be used:

SELECT * FROM table_name;

The following syntax, on the other hand, selects only the specific columns of the database.

SELECT column_name,column_name FROM table_name;

The SQL UPDATE Statement

The UPDATE statement is used to update existing records in a table. The syntax for this command is as follows:

```
UPDATE table_name
SET column1=value1,column2=value2,...
WHERE some_column=some_value;
```

SQL DELETE Statement

The DELETE statement is used to delete records in a table. The syntax is as follows:

```
DELETE FROM table_name
WHERE some_column=some_value;
```

We will again connect to the MySQL Database through the Command Prompt and access the mysql.exe program located in the XAMPP folder. If you don't recall how to perform this task, refer to the discussion in activity 1.

4. Materials and Equipment:

Desktop Computer
Windows Operating System
XAMPP Application

5. Procedure

Database Schema

The image below shows the tables in the database that will be used in this activity. You may use your previous database from activity 1 or recreate the database to match the structure below.

```
MariaDB [db1]> describe vehicles;
+-----------------+----------+------+-----+---------+-------+
| Field           | Type     | Null | Key | Default | Extra |
+-----------------+----------+------+-----+---------+-------+
| car_maker       | char(20) | YES  |     | NULL    |       |
| car_model       | char(20) | YES  |     | NULL    |       |
| number_of_doors | int(11)  | YES  |     | NULL    |       |
+-----------------+----------+------+-----+---------+-------+
3 rows in set (0.10 sec)

MariaDB [db1]> describe drivers;
+---------------+------------+------+-----+---------+-------+
| Field         | Type       | Null | Key | Default | Extra |
+---------------+------------+------+-----+---------+-------+
| id            | int(11)    | YES  |     | NULL    |       |
| first_name    | char(50)   | YES  |     | NULL    |       |
| last_name     | char(50)   | YES  |     | NULL    |       |
| age           | int(3)     | YES  |     | NULL    |       |
| address       | char(100)  | YES  |     | NULL    |       |
| years_driving | int(3)     | YES  |     | NULL    |       |
| status        | tinyint(1) | YES  |     | NULL    |       |
+---------------+------------+------+-----+---------+-------+
7 rows in set (0.01 sec)

MariaDB [db1]>
```

Part I

1. Use the syntax from the discussion to populate each table with 5 entries each.
2. Display all the values in each table.
3. Display only the ID, First Name, and Last name of the drivers.
4. Display the car makers and the car models in the vehicles table.
5. Show the screenshot of each set of results with your observation below each in Section 6 Database Output.

Part II

1. Use the UPDATE command to change a row value (ex. First Name, age) in each table.
2. Display the values in each table.
3. Show the screenshot of the newly updated tables with your observation below each in Section 6 Database Output.

Part III

1. Delete one row from each table.
2. Show the screenshot of the newly updated tables with your observation below each in Section 6 Database Output.

Note: Do not drop the database or any tables.

Part IV

1. Insert two rows to the drivers table with only age and address.
2. Run the command ***SELECT first_name, last_name FROM drivers;***
3. Run the command ***DELETE FROM drivers WHERE first_name=NULL;***
   Note: Observe the rows affected if there were rows deleted.
4. Run the command ***DELETE FROM drivers WHERE first_name is NULL;***
5. Run the command ***SELECT first_name, last_name FROM drivers;***
6. Add a new column in drivers called license_number with data type char(13)
7. Run the command ***DELETE FROM drivers WHERE license_number is NULL;***
8. Update one of the rows in the drivers table with a license_number (ex. "N014-123-M01")

9. Run the command ***SELECT * FROM drivers;***
10. Take a screenshot of the code snippets of resulting tables from SELECT commands with your observation below each in Section 6 Database Output.

6. Database Output:

Copy screenshot(s) of your output with observations after completing the procedures provided in Part 5.

7. Supplementary Activity

Questions

1. **What happens if you insert an exact duplicate of any record that is already in your current database table?**
   Ex. > INSERT INTO drivers VALUES(1,"James","Nicolas",20,"5 Peace St. QC",3,true);
   > INSERT INTO drivers VALUES(1,"James","Nichelistine",20,"5 Peace St. QC",3,true)
   You can try the result. Is the output allowable in a real environment? Why do you think the output was what you saw?

   _____
   _____
   _____

2. **Given that you ran your SQL Command in the example in Question 1.**

id	first_name	last_name	age	address	years_driving	status
1	Mary	Nichelistine	20	5 Peace St. QC	3	1
2	Roger	Michael	45	8 Orange St. Marikina	10	0
1	James	Nichelistine	20	5 Peace St. QC	3	1
1	James	Nichelistine	20	5 Peace St. QC	3	1

   What happens if you try to delete one of the rows with the redundant id? (Ex. DELETE FROM drivers WHERE id=1;) What do you think should the solution be against this kind of problem?

   _____
   _____
   _____

3. **In Part IV of the Procedure section, what do you think is the difference between using an equals sign and the is keyword when checking for NULL?**

   _____
   _____
   _____

   Additional Reference: https://dev.mysql.com/doc/refman/8.0/en/working-with-null.html

4. **Following from the part IV tasks where you've added a new column called license_number. What would happen if you inserted "N014-123-M012XD3"? Why do you think that was the output?**

   _____
   _____
   _____

5. **Why is it necessary to introduce a WHERE clause when updating and deleting data?**

   _____

_____ _____
8. Conclusion
9. Assessment Rubric

Activity No. 3	
Primary and Foreign Keys	
Course Code: CPE011	**Program:**
Course Title: Database Management System	**Date Performed:**
Section:	**Date Submitted:**
Name:	**Instructor:**

1. Objective(s):

This activity aims to discuss the purpose of database tables with the Primary Key Constraint, the creation of databases and alteration of the existing structure to utilize the primary key functions.

2. Intended Learning Outcomes (ILOs):

The students should be able to:
2.1 Understand and discuss the purpose of primary and foreign keys in a database structure.
2.2 Create or Alter a database tables to include Primary and Foreign Keys.

3. Discussion:

In the previous activity, we have seen that our initial database design did not account for maintaining the uniqueness of each row so that we were able to insert an exact duplicate of one row to a new row and we saw that deleting one of them results in the deletion of the redundant tables which means data could be lost permanently. This can be solved by using a primary key.

Primary Keys

A primary key is a column or a set of columns that uniquely identifies each row in the table. You must follow the rules below when you define a primary key for a table:

- A primary key must contain unique values. If the primary key consists of multiple columns, the combination of values in these columns must be unique.

- A primary key column cannot contain NULL values. It means that you have to declare the primary key column with the NOT NULL attribute. If you don't, MySQL will force the primary key column as NOT NULL implicitly.

- A table has only one primary key. Also, a primary key column often has the AUTO_INCREMENT attribute that generates a unique sequence for the key automatically. The primary key of the next row is greater than the previous one.

To insert a primary key, one can add the constraint during table creation. To do this, the following syntax can be used.

```
CREATE TABLE Persons
(
        P_Id int NOT NULL,
        LastNamevarchar(255) NOT NULL,
        FirstNamevarchar(255),
        Address varchar(255),
        City varchar(255),
        PRIMARY KEY (P_Id)
)
```

Note that P_Id is the primary key. Another method in adding a primary key is using the alter table command. The following syntax can be used.

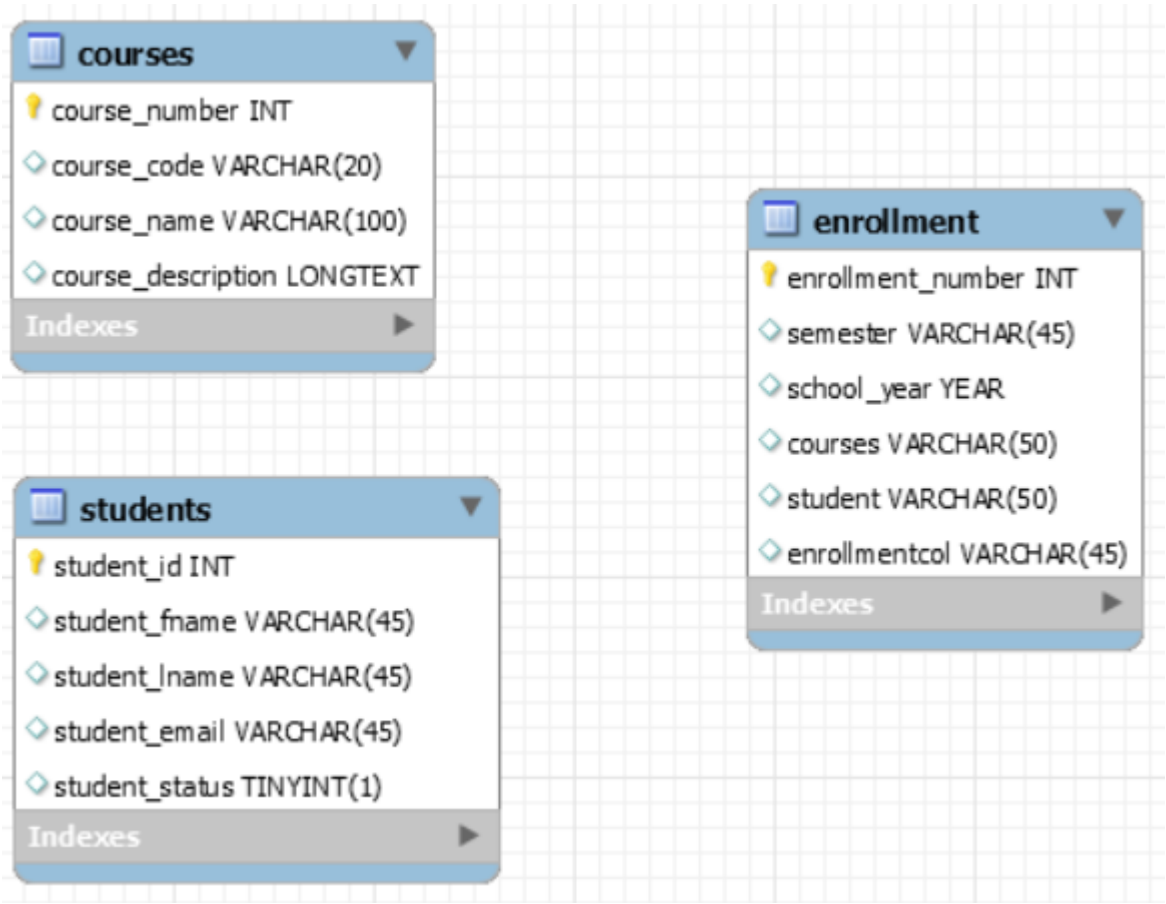
```
ALTER TABLE Persons
ADD PRIMARY KEY (P_Id);
```

To drop a PRIMARY KEY constraint, use the following SQL Command:

ALTER TABLE Persons
DROP PRIMARY KEY

Foreign Keys
A foreign key is a field in a table that matches/links another field of another table. A foreign key places constraint on data in the related tables, which enables MySQL to maintain referential integrity.

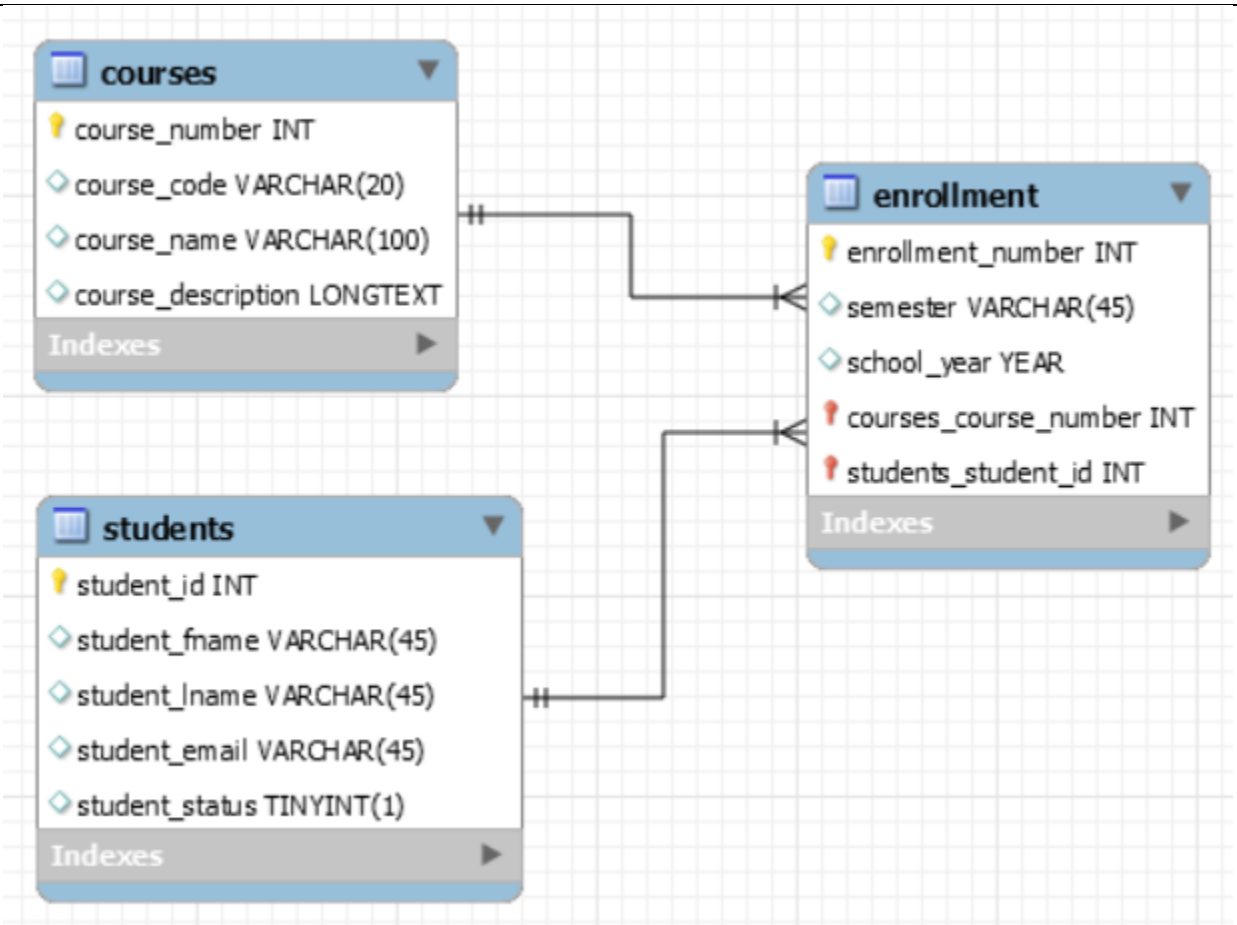
Consider the following database diagram:



Observe that each table now has its own Primary Key which eliminates redundant records in that table and keeps each record unique, If we populate each table we will have no redundant records as the PRIMARY KEY constraint will prevent this.

A new problem can be observed when we take a look at the enrollment table. It has a student name and course code. If we tried inserting a record that does not in either students or courses it will still be accepted. You may try implementing the schema in your database. The problem encountered with this type of database design is a violation of the Referential Integrity constraint. The accuracy and the consistency of data cannot be validated with this database design.

Adding a Foreign Key constraint resolves this problem. We will build a relationship – connect enrollment with students. There are three types of relationship between tables that they can have: One to one, One to Many, and Many to Many. MySQL does not have an option to specify this. It depends on your implementation of the primary keys and foreign keys.



We have now established a relationship between the two tables through foreign keys (denoted by the red bulbs) on the enrollment table. MySQL will now deny any insertion of data on the course and student field that is not existing in courses and students.

To create a foreign key constraint during table creation process, note that the besides the key itself, a constraint is also created. The syntax for creating a foreign key follows this syntax:

```
FOREIGN KEY constraint_name(column)
REFERENCES reference_table(reference_pkColumn)
```

Where constraint name is the name of the foreign key constraint that was added to that column. Make sure to always put a constraint name otherwise MySQL will automatically generate one.

```
CREATE TABLE enrollment
(
        enrollment_number INT,
        student INT,
        course INT,
        semester VARCHAR(20),
        PRIMARY KEY(enrollment_number),
        FOREIGN KEY fk_student(student) REFERENCES students(student_id,)
        FOREIGN KEY fk_course(course) REFERENCES courses(course_number)
);
```

Observe that we need to create the fields first before assigning them the Primary Key and Foreign Key constraint.

To create a FOREIGN KEY constraint when the enrollment table or when you've already created a table. You can use the following command:

ALTER TABLE enrollment
ADD FOREIGN KEY fk_student(student) REFERENCES students(student_id);

Adding Foreign keys using the ALTER command is one at a time. You cannot add multiple foreign keys in one statement. To drop a foreign key, use the following command syntax:

ALTER TABLE enrollment DROP FOREIGN KEY constraint_name;
ALTER TABLE enrollment DROP KEY constraint_name;

You need to first drop the constraint/rule foreign key then drop the actual key/index.

4. Materials and Equipment:

Desktop Computer
Windows Operating System
XAMPP Application

5. Procedure

1. Create a database studentdb_<LASTNAME>;
2. Create a table named students with attributes:
    - ID(Primary Key)
    - First Name
    - Last Name
    - Program
3. Create another table named Courses with attributes
    - Course Code(Primary Key)
    - Course Description
    - Department(Not Null)
4. Create a third table named Sections with Attributes
    - Student Number(References from students)
    - Course Code(References from courses)
    - Section (Primary Key)
    - Class Code
5. Create a fourth table named classrooms with attributes
    - Room Number (3210, 5204)
    - Time (ex. 730, 830, 1230, 1330)
    - Duration
    - Course Code (Reference from courses)
    - Section (Reference from sections)
6. Place the screenshot of the table structure of each table in Section 6.

7. Try inserting a value into classrooms. What is the result? _____
8. Try inserting a value into sections. What is the result? _____
9. Populate the students and courses table with 5 records each.
10. Take screenshots of your table values for each table write your observation below the image.
11. Insert records into sections with a student number based from existing student IDs and with a course code based from existing course codes records. What is the result? _____
12. Insert records into sections with a student number that does not exist student table ID but has a course code that exists in the courses table. What is the result? _____
13. Populate the sections table with at least 5 records.
14. Take a screenshot of the values in sections table. Write your observations below the image.
15. Insert values into classrooms with the same records (ex. 3210,730,2,'CPE011','CPE21FB1') two times. What was the result? _____
16. Delete all the values in classroom.
17. Implement a primary key in classrooms. Use techniques you've learned previously to add a classroom ID.
18. Try inserting two identical records into classroom again. What is the result? _____
19. Delete all the values in classroom.
20. Run this command: ***ALTER TABLE classrooms ADD UNIQUE KEY uk_roomtime (room_number,time);***
21. Try inserting two identical records into classrooms again. What is the result? _____
22. Populate classroom with at least 5 records.
23. Take screenshots of the values in classrooms. Write observations below the image.

6. Database Output

Copy screenshot(s) of your output with observations after completing the procedures provided in Part 5.

7. Supplementary Activity:

Questions

1. **Explain the results of step 7 and 8 in Procedure. Why did they give the outputs that they did?**

   _____
   _____
   _____

2. **Explain the results of step 11 and 12 in Procedure. Why did they give the outputs that they did?**

   _____
   _____
   _____

3. **Why is it necessary to implement a primary key and foreign key for a database?**

   _____
   _____
   _____

4. **Cite your own real-life situation where a primary key and foreign key in a database is necessary. Explain.**

   _____
   _____
   _____

5. **Why did implementing the primary key in classrooms fail to solve the redundancy problem?**

   _____
   _____
   _____

6. **What do you think is the difference between the Primary Key and the Unique Key?**

   _____
   _____
   _____

8. Conclusion

9. Assessment Rubric

Activity No. 4	
Advanced SELECT commands	
Course Code: CPE011	**Program:**
Course Title: Database Management System	**Date Performed:**
Section:	**Date Submitted:**
Name:	**Instructor:**

1. Objective(s):

This activity aims to introduce the advanced techniques for selecting records in a MySQL Database using the SQL SELECT Command.

2. Intended Learning Outcomes (ILOs):

The students should be able to:
2.1 Select data from a database based on a specific criterion
2.2 View data using select statements partnered with wildcards
2.2 Select data from a database in a specific order
2.3 View data using select statements partnered with wildcards
.

3. Discussion:

The SELECT Statement's basic syntax allows you to retrieve rows either all or only specific columns. In this activity you will see the different additional features that you can use with the SELECT statement through SQL Clauses and a special COUNT() function.

1. WHERE Clause

In computing and database technology, conditional statements, conditional expressions and conditional constructs are features of a programming language, which perform different computations or actions depending on whether a programmer-specified Boolean condition evaluates to true or false. Apart from the case of branch predication, this is always achieved by selectively altering the control flow based on some condition.

The SQL WHERE clause is used to specify a condition while fetching the data from single table or joining with multiple tables.

If the given condition is satisfied then only it returns specific value from the table. You would use WHERE clause to filter the records and fetching only necessary records.

The WHERE clause is not only used in SELECT statement, but it is also used in UPDATE, DELETE statement, etc., which we would examine in subsequent chapters.

```
SELECT column_name,column_name
FROM table_name
WHERE column_name operator value;
```

SQL requires single quotes around text values (most database systems will also allow double quotes). However, numeric fields should not be enclosed in quotes:

```
SELECT * FROM CustomerWHERE CustomerID=1;
```

Operators in The WHERE Clause The following operators can be used in the WHERE clause:

Operator	Description
=	Equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

An example of an application for conditional statements are as follows:

SELECT * FROM Customer WHERE CustomerID> 1;

The LIKE operator
The LIKE operator is used in a WHERE clause to search for a specified pattern in a column. The syntax is as follows.

SELECT column_name(s)
FROM table_name
WHERE column_name LIKE pattern;

Using Wildcards
In using the LIKE pattern, wildcard characters are used with the SQL LIKE operator. SQL wildcards are used to search for data within a table. With SQL, the wildcards are:

Wildcard	Description
%	A substitute for zero or more characters
_	A substitute for a single character
[*charlist*]	Sets and ranges of characters to match
[^*charlist*] or [!*charlist*]	Matches only a character NOT specified within the brackets

Using the SQL % Wildcard
The % wildcard symbol is used to substitute any character, with any count. The following SQL statement selects all data from the student table with a program starting with "elec":

SELECT * FROM student
WHERE program LIKE 'elec%';

Any number of wildcards can be used in a single query. In the following example, two % wildcards are used. To search for data with an 'er' in between, use the following syntax.

SELECT * FROM student
WHERE program LIKE '%er%';

Using the SQL _ Wildcard
The _ wildcard is used to substitute any single character. The following SQL statement selects all customers with a City starting with any character, followed by "erlin":

```
SELECT * FROM Customers
WHERE City LIKE '_erlin';
```

The following SQL statement selects all customers with a City starting with "L", followed by any character, followed by "n", followed by any character, followed by "on":

```
SELECT * FROM Customers
WHERE City LIKE 'L_n_on';
```

Using the SQL [charlist] Wildcard
The [charlist] sets and ranges of characters to match The following SQL statement selects all customers with a City starting with "b", "s", or "p":

```
SELECT * FROM Customers
WHERE City LIKE '[bsp]%';
```

Ranges can also be utilized for the charlist wildcard. These ranges can be indicated using a hypen(-).The following SQL statement selects all customers with a City starting with "a", "b", or "c":

```
SELECT * FROM Customers
WHERE City LIKE '[a-c]%';
```

The NOT symbol, represented by an exclamation point, can be used alongside the charlist wildcard to indicate a query that does not contain any of the indicated. The following SQL statement selects all customers with a City NOT starting with "b", "s", or "p":

```
SELECT * FROM Customers
WHERE City LIKE '[!bsp]%';
```

2. ORDER BY Clause

Displaying data in a particular order
The ORDER BY keyword is used to sort the result-set by one or more columns. The ORDER BY keyword sorts the records in ascending order by default. To sort the records in a descending order, you can use the DESC keyword. The syntax for selecting based on a single column:

```
SELECT column_name, column_name
FROM table_name
ORDER BY column_name ASC|DESC;
```

For selecting data and ordering based on multiple columns, the syntax can be used:

```
SELECT column_name, column_name
FROM table_name
ORDER BY column_name ASC|DESC, column_name ASC|DESC;
```

3. LIMIT Clause

Limiting only a specific number of results
MySQL provides a LIMIT clause that is used to specify the number of records to return. The LIMIT clause makes it easy to code multi page results or pagination with SQL, and is very useful on large tables. Returning a large number of records can impact on performance. The syntax is as follows:

SELECT * FROM table LIMIT 30;

4. COUNT FUNCTION()

Counting Data
The COUNT() function returns the number of rows that matches a specified criteria. The syntax is as follows:

SELECT COUNT(column_name) FROM table_name;

Note that the count command can utilize conditions similar to the where clause. This can be seen in the following syntax below.

SELECT COUNT(CustomerID) AS OrdersFromCustomerID FROM Orders WHERE CustomerID=7;

Note that the AS command is used to create a result column that is named OrdersFromCustomerID. This name will be used as an identifier for the count value.

To count only data with unique values, use the following query:

SELECT COUNT(DISTINCT column_name) FROM table_name;

Note that the distinct values are case sensitive.

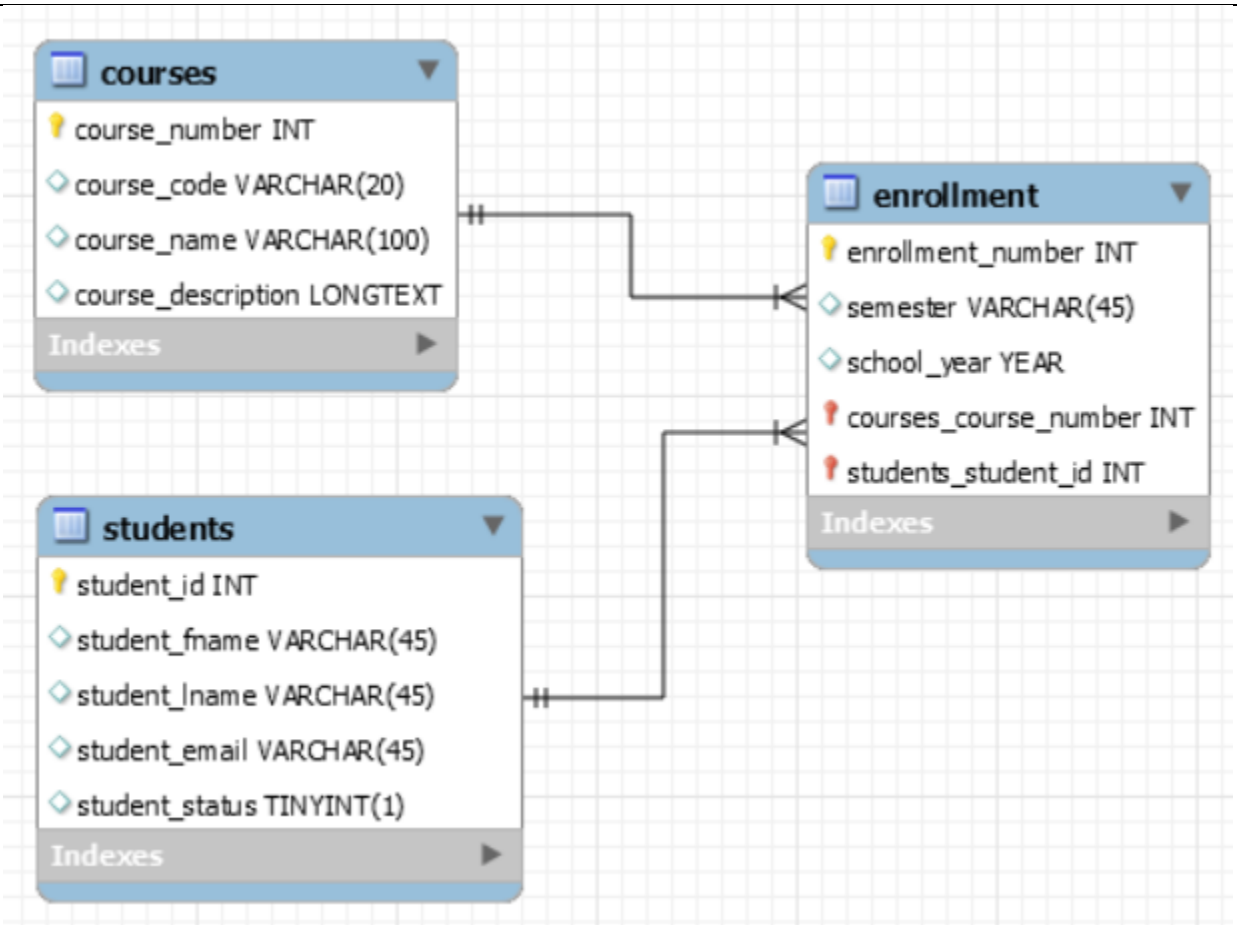
4. Materials and Equipment:

Desktop Computer
Windows Operating System
XAMPP Application

5. Procedure

Database Schema
The database that we'll be using for this activity is the simple enrollment database from the previous activity discussion. We will perform SQL commands that allow us to retrieve data based on specific requirements/conditions.

The image below shows the database schema.



Tasks

1. Create a database named **enrollmentdb_<LASTNAME>**
   Note: Use the techniques you've learned in the previous activity to implement a database that handles redundancy.
2. Populate the database to have at least 10 records each.
   Side activity:
   You can load pre-made values by loading data from CSV instead of manually inserting them.
   1. Copy the *students.csv*, *courses.csv* to your computer from the FTP or Canvas Instructure group. For the enrollment.csv, you will be creating your based on the pattern of the two files.
      **Take note of the order in which the table columns were created and the order of values in your csv files.

   2. Follow the pattern of the following command to load each csv files to the respective tables.

```
LOAD DATA INFILE 'C:/Users/Royce/Desktop/students.csv'
INTO TABLE students
FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

3. Take screenshots of the values in each table. Write descriptions for each image, explain the table and its values.
4. For the succeeding steps take screenshots of each record displayed by the SELECT commands. Write your observations under each image.

5. Select the row in course that contains the course code cpe011 along with its course description.
6. Select the row of student number 7.
7. Select the rows in students table whose student email ends in yahoo.com
8. Select the rows in students table whose first name starts with d or D.
9. Select the rows in students table who are not currently active.
10. Select the first five rows in the courses table.
11. Display the rows/course(s) which includes the words programming in courses table.
12. Display the row/course which teaches VHDL.
13. Display the courses that were enrolled during school year 2019.
14. Display the student records in alphabetical order.
15. Count how many students are currently not active. What is the answer? _____

6. Database Output

Copy screenshot(s) of your output with observations after completing the procedures provided in Part 5.

7. Supplementary Activity:

Questions

1. **Why do you think the SQL WHERE clause exists? Based from the activity, explain its importance.**

   _____
   _____
   _____

2. **Where do we use SQL wildcards? Cite a use-case not based from the activity.**

   _____
   _____
   _____

3. **What is the purpose of LIMIT clauses in database queries?**

   _____
   _____
   _____

4. **Can ORDER clause be used for multiple columns? How?**

   _____
   _____
   _____

5. **What particular situation can the COUNT() function be used? Use an example not based in the activity.**

   _____
   _____
   _____

8. Conclusion

9. Assessment Rubric

Activity No. 5	
Advanced Database Programming	
Course Code: CPE011	**Program:**
Course Title: Database Management System	**Date Performed:**
Section:	**Date Submitted:**
Name:	**Instructor:**

1. Objective(s):

This activity aims to introduce the Structured Query Language (SQL) using the Data Definition Language (DDL) commands in a MySQL Database

2. Intended Learning Outcomes (ILOs):

The students should be able to:
2.1 Use SQL Built-in functions in an SQL Script
2.2 Use JOIN SQL Commands to perform complex queries in databases
2.2 Create subqueries and correlated queries
.

3. Discussion:

As your databases becomes more complex and more relations between entities are created, we would need to implement queries that can retrieve records from multiple tables and display them in one single query. Some queries may require outputs from nested queries also called subqueries in order to get more specific results. Some tasks such as date generation, and summing quantities of products can also be automated through the use of SQL built-in functions similar to a programming language built-in function.

SQL Function

SQL Functions

An SQL function is similar to a function in programming language both in syntax and in its general use. A function in SQL can be defined with the following syntax function_name(). The opening and closing parentheses is the main indicator that it is a function. There are many built-in SQL Functions available. Some of the possible SQL functions are given below:

SQL Function	Description
COUNT()	Returns the number of the rows in a table.
SUM()	Returns the sum of a set of values. The SUM function ignores NULL values. If no matching row found, the SUM function returns a NULL value.
AVG()	Calculates the average value of a set of values. It ignores NULL values in the calculation.
MAX()	Returns the maximum value in a set of values.
MIN()	Returns the minimum value in a set of values.
ROUND(number, decimals)	Rounds a number to a specified number of decimal places.
ABS()	Return the absolute value of a number
CURRENT_DATE()	Returns the current date
NOW()	Returns the current date and time

More functions can be found through the official documentation: https://dev.mysql.com/doc/refman/5.6/en/functions.html

Sample use:

```
MariaDB [db1_demo]> INSERT INTO accounts(first_name, last_name, email_address, username, date_created) VALUES ('Royce','
Chua','royce123@ymail.com','royce123',CURRENT_DATE());
Query OK, 1 row affected (0.16 sec)

MariaDB [db1_demo]> SELECT * FROM accounts;
+------------+-----------+-------------------+----------+--------------+----+
| first_name | last_name | email_address     | username | date_created | id |
+------------+-----------+-------------------+----------+--------------+----+
| Royce      | Chua      | royce123@ymail.com | royce123 | 2019-06-24   |  2 |
+------------+-----------+-------------------+----------+--------------+----+
1 row in set (0.00 sec)
```

A date value was inserted without explicitly stating the actual date today. We can use NOW() if the column was of DATETIME data type.

SQL JOINS

The SQL Joins clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

The syntax to be used for joins is at follows.

```
SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate
FROM Orders
INNER JOIN Customers
ON Orders.CustomerID=Customers.CustomerID;
```

Consider the following two tables, (a) CUSTOMERS table is as follows:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

(b) Another table is ORDERS as follows:

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08	3	3000
100	2009-10-08	3	1500
101	2009-11-20	2	1560
103	2008-05-20	4	2060

Now, let us join these two tables in our SELECT statement as follows:

```
SELECT ID, NAME, AGE, AMOUNT
FROM CUSTOMERS, ORDERS
WHERE CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

This would produce the following result:

ID	NAME	AGE	AMOUNT
3	Kaushik	23	3000
3	Kaushik	23	1500
2	Khilan	25	1560
4	Chaitali	25	2060

Here, it is noticeable that the join is performed in the WHERE clause. Several operators can be used to join tables, such as =, <, >, <>, <=, >=, !=, BETWEEN, LIKE, and NOT; they can all be used to join tables. However, the most common operator is the equal symbol.

4. Materials and Equipment:

Desktop Computer
Windows Operating System
XAMPP Application

5. Procedure

1. CREATE DATABASE myDatabase5_<LASTNAME>.
2. Create three tables named Item_List, Nanays,Kaloys and Supplier.

Item List		
PID	**Product**	**supID**
p01	Beef Steak	mla2
p02	Beef Cutlet	mla1
p03	Pork Steak	mla1
p04	Chicken Wings	mla4
p05	Ground Pork	mla6
p06	Chicken Liver	mla7
p07	Beef Liver	mla1
p08	Chicken Feet	mla6
p09	Chicken Skin	mla5
p10	Whole Pork	mla5
p11	Whole Chicken	mla1
p12	Whole Cow	mla2
p13	Young Pork	mla6
p14	One Day Old	mla4
p15	Beef Franks	mla2

Kaloys Diner	
PID	**QTY**
p01	1
p04	5
p06	4
p10	4
p12	1
p14	6

Nanay's Eatery	
PID	**QTY**
p04	5
p09	4
p13	3

SupplierList	
supID	**Location**
mla1	Navotas
mla2	Navotas
mla3	Marikina
mla4	Marikina
mla5	Paranaque
mla6	Paranaque
mla7	Quezon City
mla8	Quezon City
mla9	Zapote
mla10	Zapote

3. Perform the following tasks using JOIN commands.

- List of Product ID's, Product Names and Quantities That Kaloys dinner need
- Combined List of products that Both Customers need, including quantites
- List of suppliers that Kaloy's need and their location
- List of Suppliers that Nanay's need to contact and their location
- Determine which Products are not being used by Nanays.
- Determine which Products are not being used by Kaloy

6. Database Output
7. Supplementary Activity:
8. Conclusion
9. Assessment Rubric

Activity No. 6	
Introduction to SQL Server Environment	
Course Code: CPE011	**Program:**
Course Title: Database Management System	**Date Performed:**
Section:	**Date Submitted:**
Name:	**Instructor:**

1. Objective(s):

This activity aims to introduce the environment of the SQL Server database engine and perform basic database tasks.

2. Intended Learning Outcomes (ILOs):

The students should be able to:
2.1 Configure the services and protocol of an instance in SQL Server using SQL Server Configuration Manager.
2.2 Create, modify and delete database and table objects using SQL Server Management Studio.
2.3 Create, modify and delete relationship between table objects using Database Diagram.

3. Discussion:

Microsoft SQL Server is a relational database management system developed by Microsoft. The primary function of the database server such as SQL Server is the storage and retrieval of data requested by other software applications. The software applications and the SQL server may run either on the same computer or on another computer across a network.

There are several editions of Microsoft SQL Server.

1. **Enterprise**
   It includes core database engine and add-on services to create and manage SQL Server cluster. It can manage databases as large as 524 petabytes and address 12 terabytes of memory and supports 640 logical processors (CPU cores
2. **Standard**
   It includes the core database engine with stand-alone devices. It only supports fewer active instances and does not include some high-availability functions.
3. **Web**
   SQL Server Web Edition is a low-TCO option for Web hosting.
4. **Business Intelligence**
   It focused on Self Service and Corporate Business Intelligence. It also includes the Standard Edition capabilities and Business Intelligence tools
5. **Express**
   A free edition of SQL Server that includes the core database engine. There are no limitations on the number of databases or users supported. However, it is limited to one processor, 1GB memory and 10GB database files. . The first is SQL Server Express with Tools, which includes SQL Server Management Studio Basic. SQL Server Express with Advanced Services adds full-text search capability and reporting services.

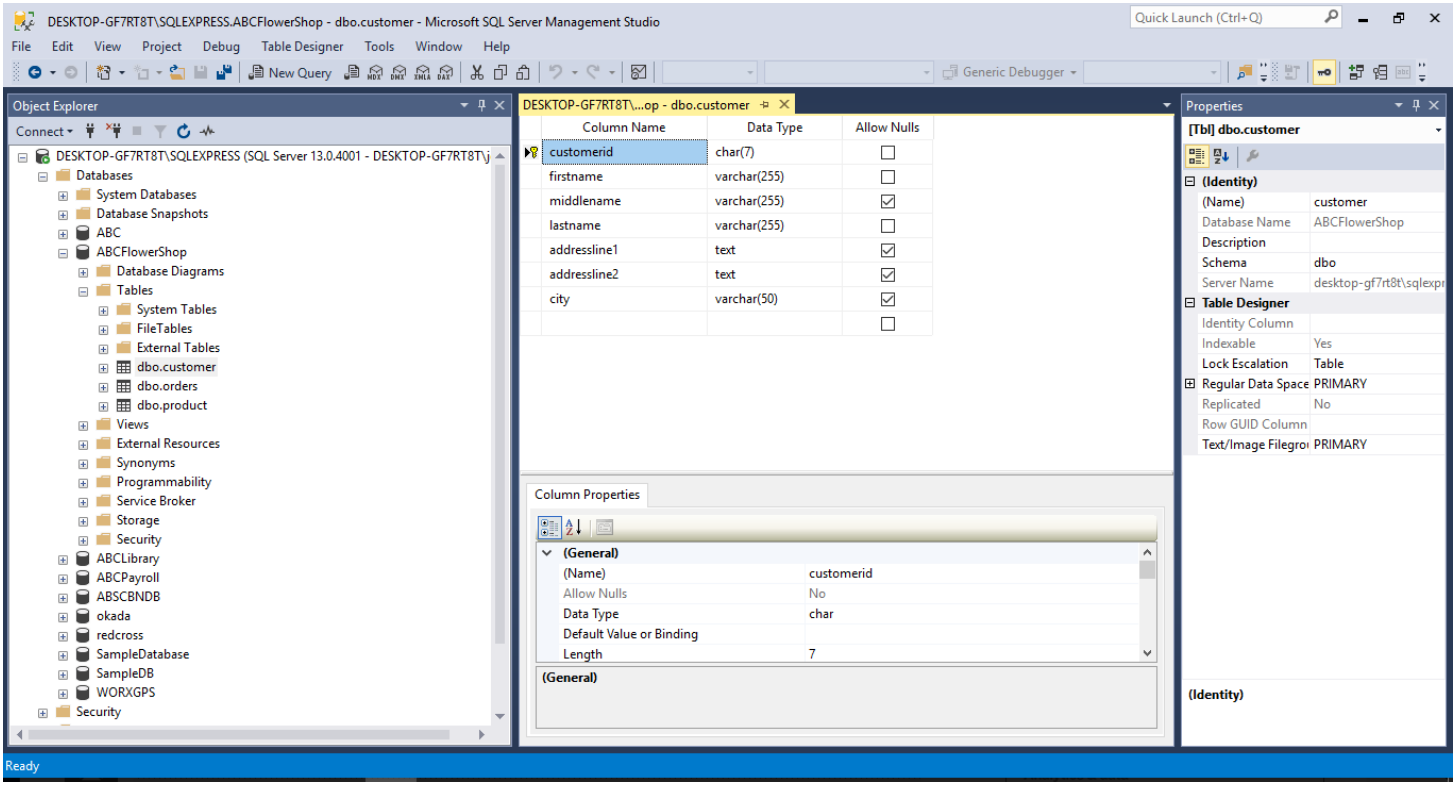
General Architecture

1. Client − Where the request initiated.
2. Query − SQL query which is high level language.
3. Logical Units − Keywords, expressions and operators, etc.
4. N/W Packets − Network related code.
5. Protocols − In SQL Server we have 4 protocols.
   - Shared memory (for local connections and troubleshooting purpose).
   - Named pipes (for connections which are in LAN connectivity).

- TCP/IP (for connections which are in WAN connectivity).

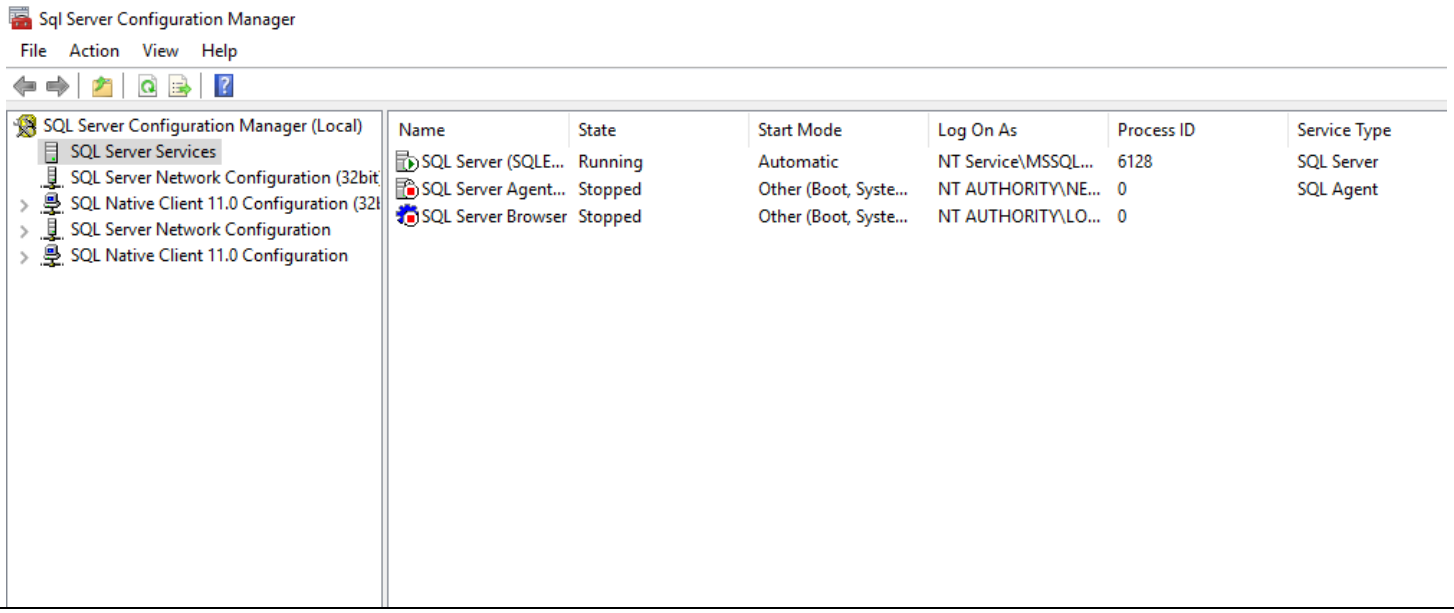
SQL Server Management Studio

It is a client tool that allows the user to connect and manage SQL Server from a graphical interface instead of Command Line Interface (CLI)



SQL Server Configuration Manager

It is a tool to manage the services associated with SQL Server, to configure the network protocols used by SQL Server, and to manage the network connectivity configuration from SQL Server client computers.



4. Resources:

Personal Computer with installed SQL Server

5. Procedure:

Manage Services using SQL Server Configuration Manager

To Automatically Start a Service

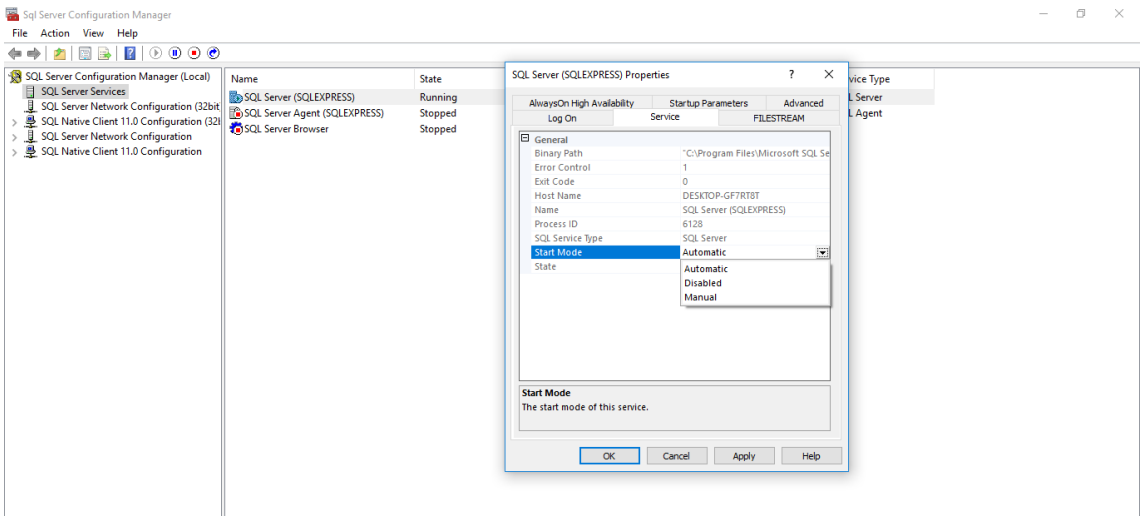
Step 1: On the **Start** menu, point to **All Programs**, point to **Microsoft SQL Server 2008 R2**, point to **Configuration Tools**, and then click **SQL Server Configuration Manager**.

Step 2: In **SQL Server Configuration Manager**, expand **Services**, and then click **SQL Server**.

Step 3: In the details pane, right-click the name of the instance you want to start automatically, and then click **Properties**.

Step 4: In the **SQL Server <***instancename***> Properties** dialog box, set **Start Mode** to **Automatic**.

Step 5: Click **OK**, and then close SQL Server Configuration Manager.

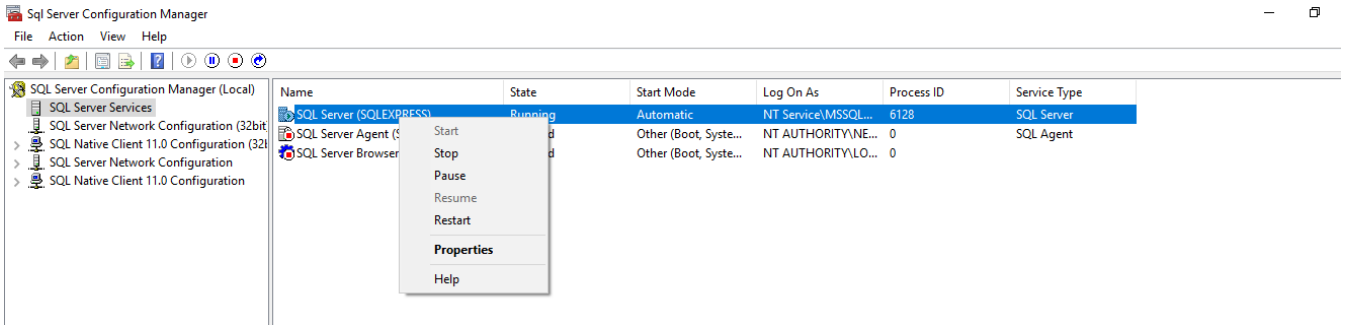


To Start, Stop, Pause or Resume Service

Step 1: On the **Start** menu, point to **All Programs**, point to **Microsoft SQL Server 2008 R2**, point to **Configuration Tools**, and then click **SQL Server Configuration Manager**.

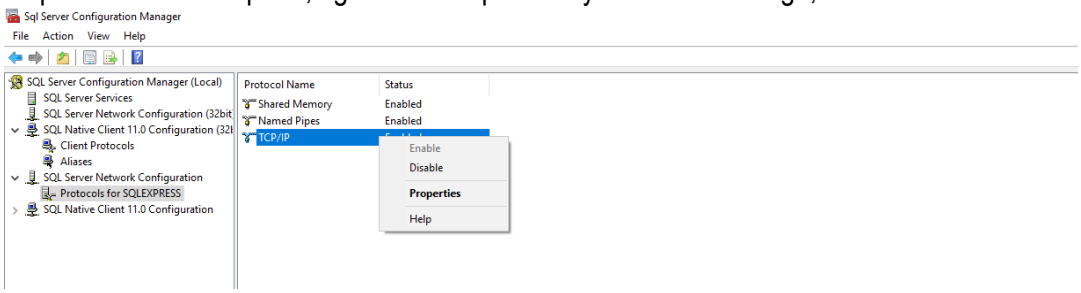
Step 2: In **SQL Server Configuration Manager**, expand **Services**, and then click **SQL Server**.

Step 3: In the details pane, right-click the name of the instance you want to start automatically, and then choose either **Start, Stop, Pause or Resume**.

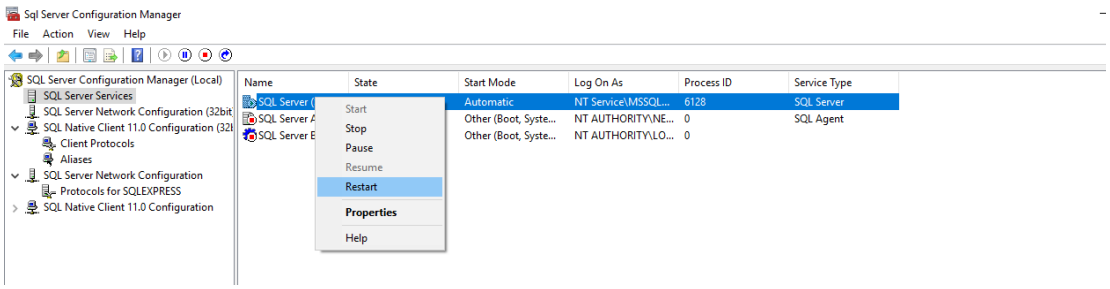


Enable or Disable a Server Network Protocol

Step 1: In SQL Server Configuration Manager, in the console pane, expand **SQL Server Network Configuration**.
Step 2: In the console pane, click **Protocols for** <*instance name*>.
Step 3: In the details pane, right-click the protocol you want to change, and then click **Enable** or **Disable**.

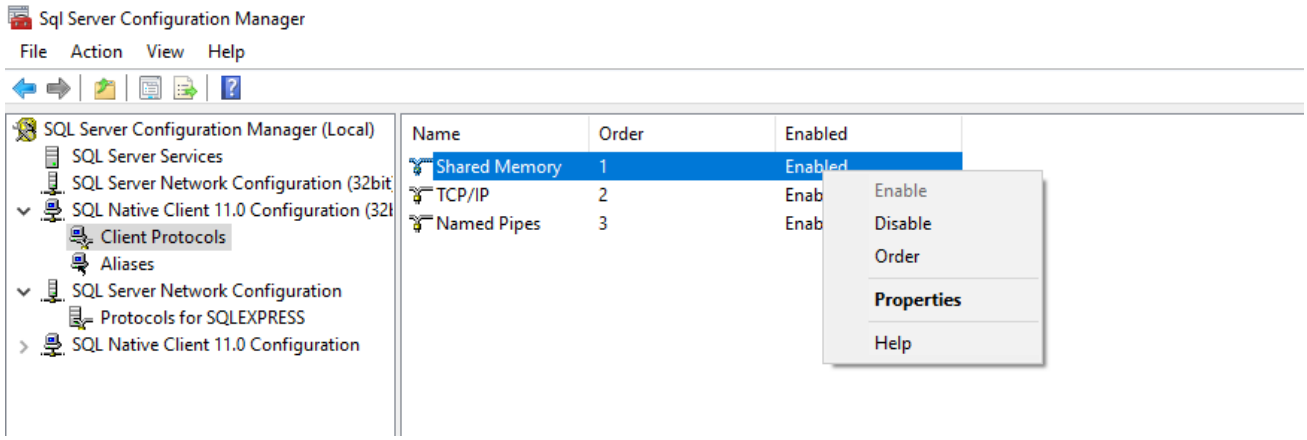


Step 4: In the console pane, click **SQL Server Services**.
Step 5: In the details pane, right-click **SQL Server (**<*instance name*>**)**, and then click **Restart**, to stop and restart the SQL Server service.



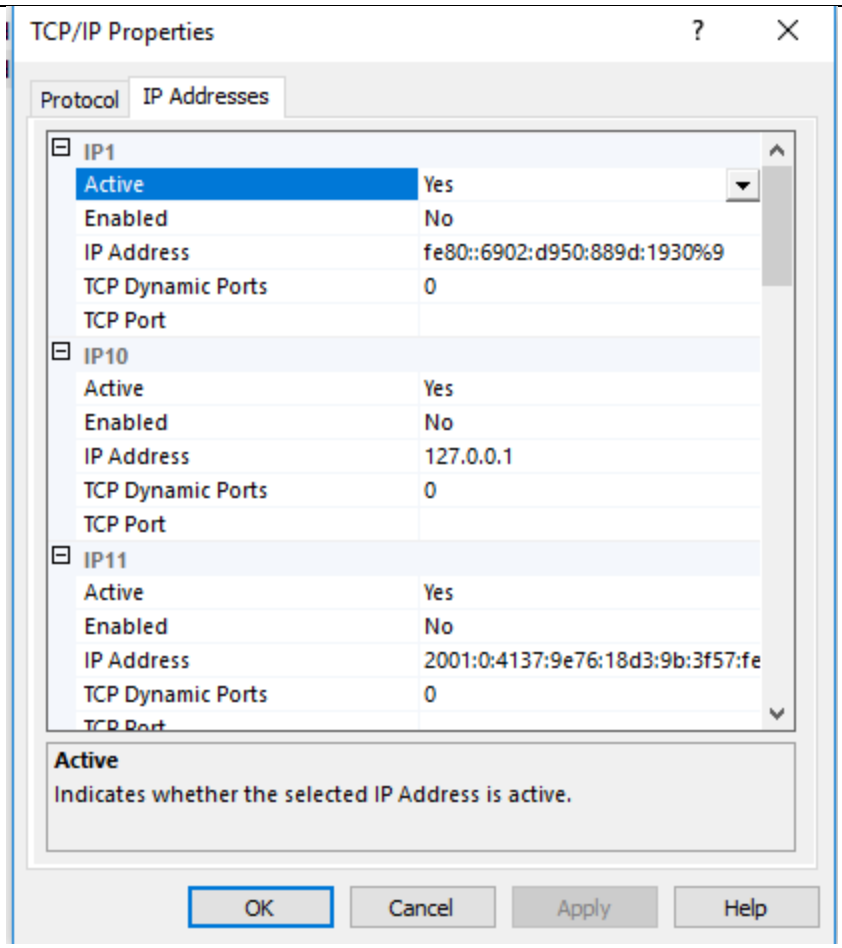
Configure Client Protocols

Step 1: In SQL Server Configuration Manager, expand **SQL Server Native Client Configuration**, right-click **Client Protocols**, and then click **Properties**.
Step 2: Click a protocol in the **Disabled Protocols** box, and then click **Enable**, to enable a protocol.
Step 3: Click a protocol in the **Enabled Protocols** box, and then click **Disable**, to disable a protocol.

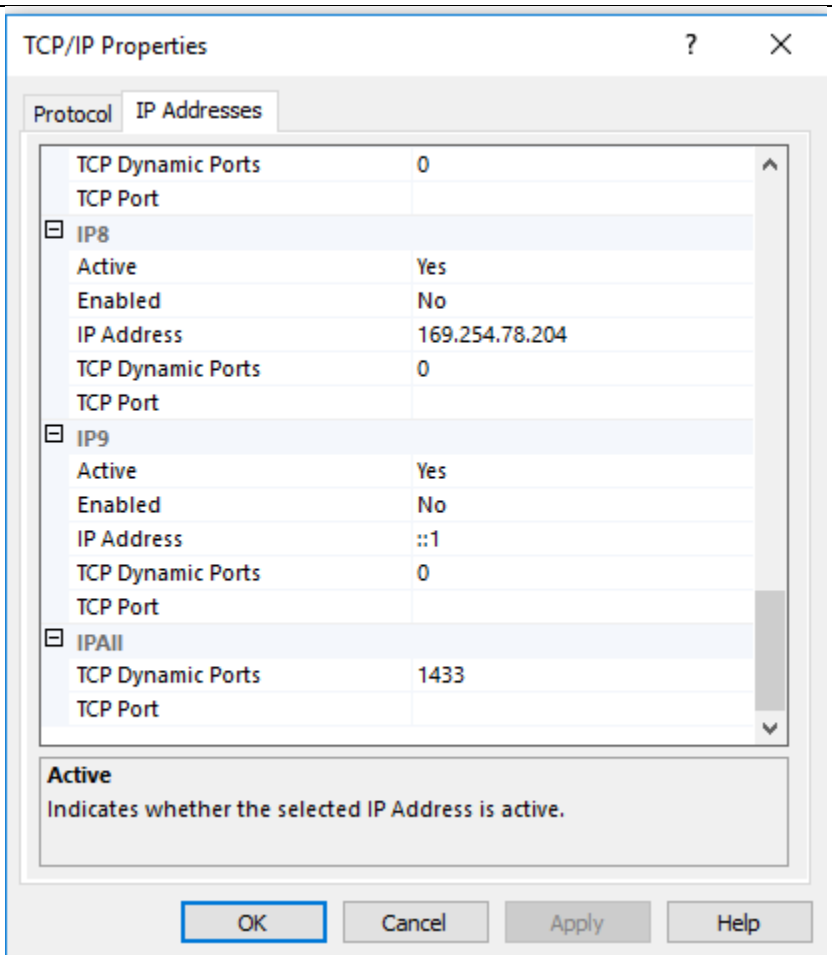


Configure a Server to Listen on a Specific TCP Port

Step 1: In SQL Server Configuration Manager, in the console pane, expand **SQL Server Network Configuration**, expand **Protocols for <instance name>**, and then double-click **TCP/IP**.
Step 2: In the **TCP/IP Properties** dialog box, on the **IP Addresses** tab, several IP addresses appear in the format **IP1**, **IP2**, up to **IPAll**. One of these is for the IP address of the loopback adapter, 127.0.0.1. Additional IP addresses appear for each IP Address on the computer. Right-click each addresses, and then click **Properties** to identify the IP address that you want to configure.



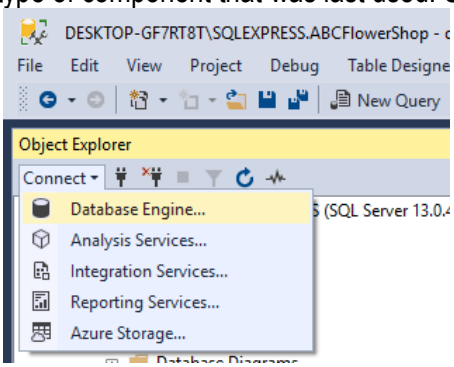
Step 3: If the **TCP Dynamic Ports** dialog box contains **0**, indicating the Database Engine is listening on dynamic ports, delete the 0.In the **IP***n* **Properties** area box, in the **TCP Port** box, type the port number you want this IP address to listen on, and then click **OK**.



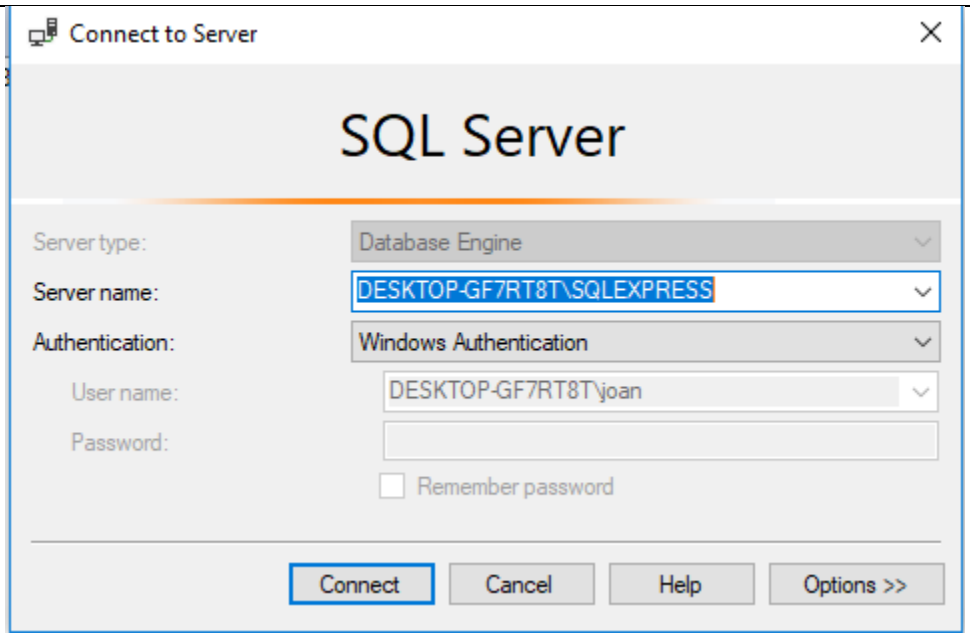
Step 4: In the console pane, click **SQL Server Services**.
Step 5: In the details pane, right-click **SQL Server (**<instance name>**)** and then click **Restart**, to stop and restart SQL Server.

Connect to the Database Engine using SQL Server Management Studio

Step 1: In Management Studio, on the **File** menu, click **Connect Object Explorer**. The **Connect to Server** dialog box opens. The **Server type** box displays the type of component that was last used. Select **Database Engine**.



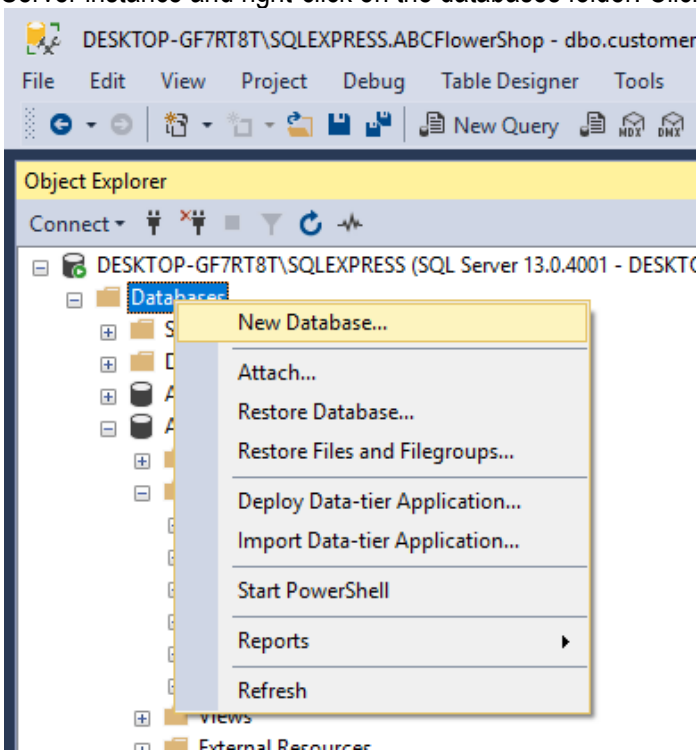
Step 2: In the Server name box, type the name of the instance of the Database Engine. For the default instance of SQL Server, the server name is the computer name. For a named instance of SQL Server, the server name is the <computer_name>\<instance_name>.



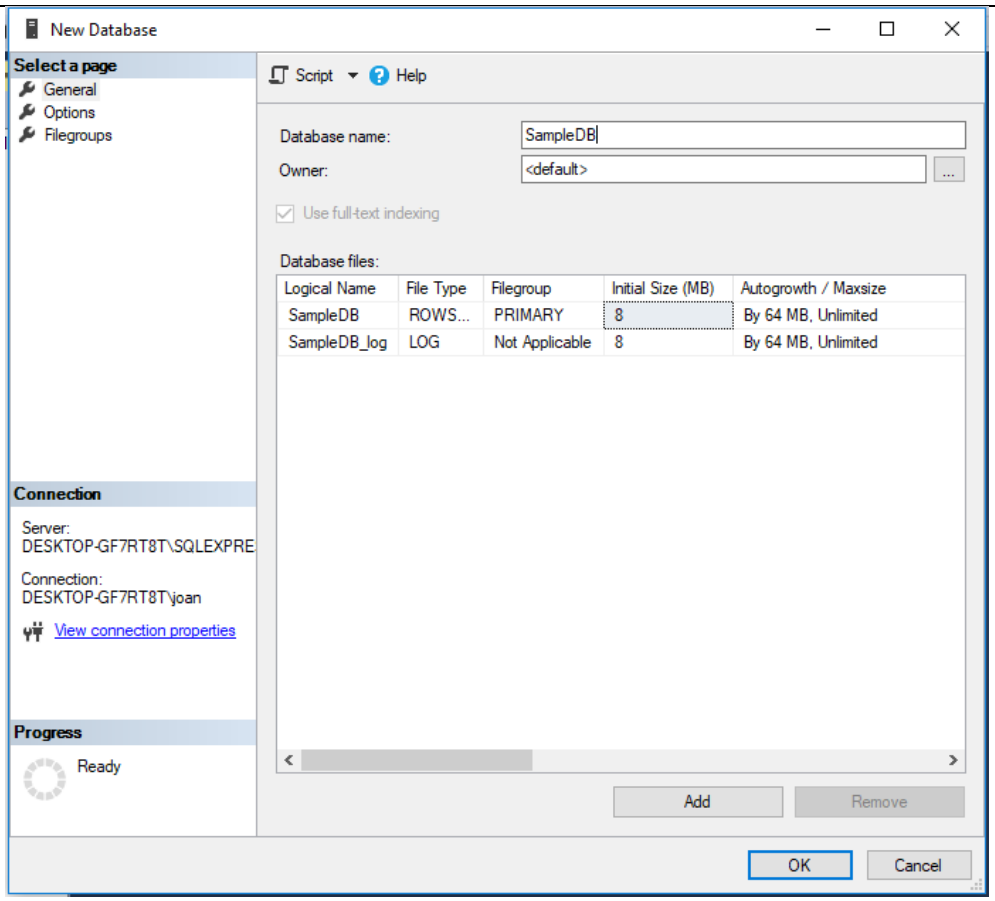
Step 3: Click Connect.

Create Database using SQL Server Management Studio

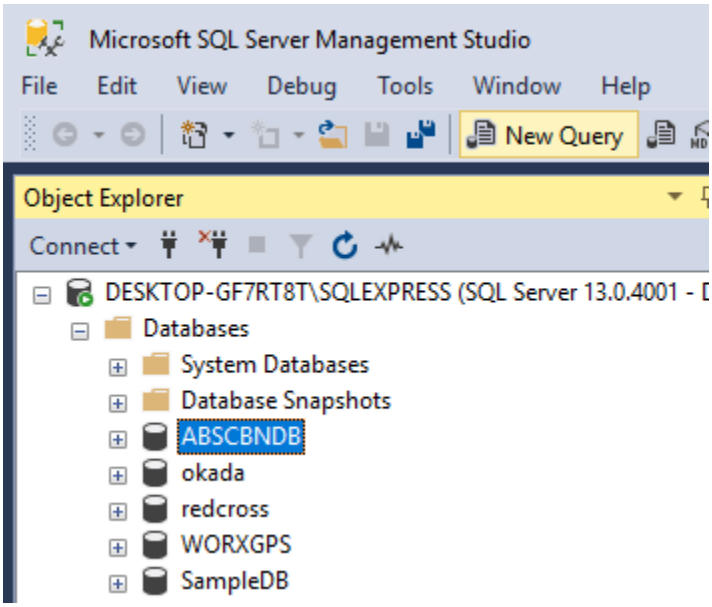
Step 1: Connect to SQL Server instance and right-click on the databases folder. Click on new database.



Step 2: Enter the database name field with the database name and click OK.

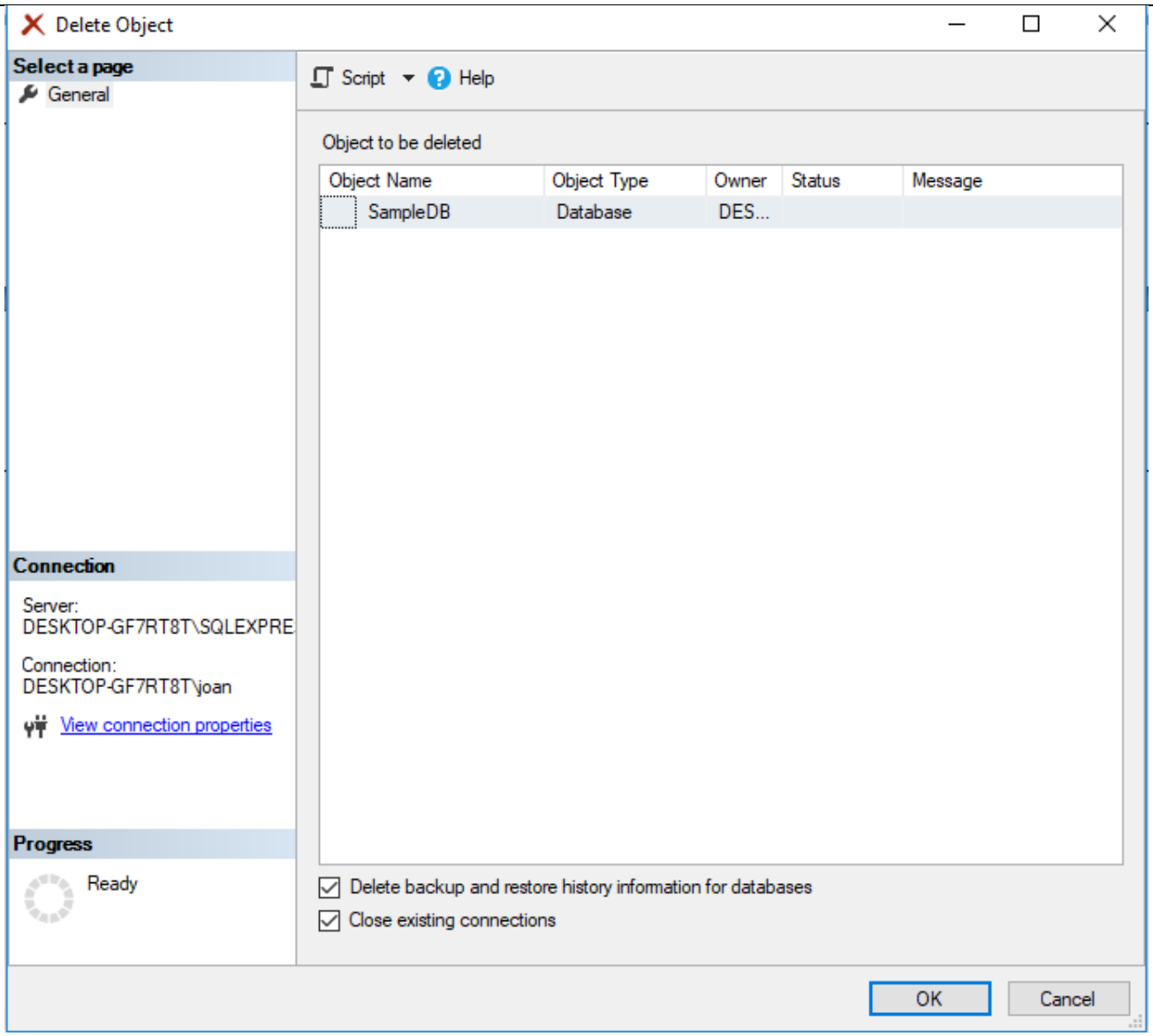


SampleDB database will be created as shown in the following screenshot.



Drop Database using SQL Server Management Studio

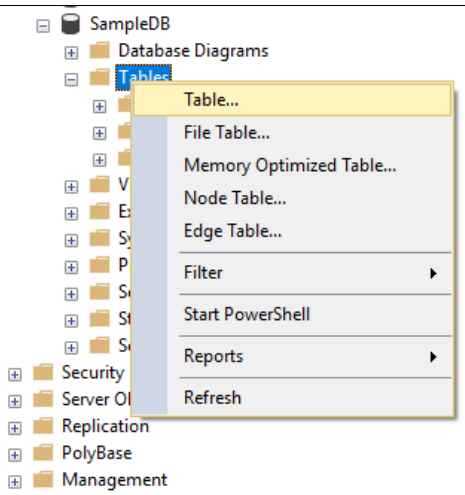
Step 1: Connect to SQL Server and right-click the database you want to remove. Click Delete command. The Delete Object window will appear as shown in the following screenshot.



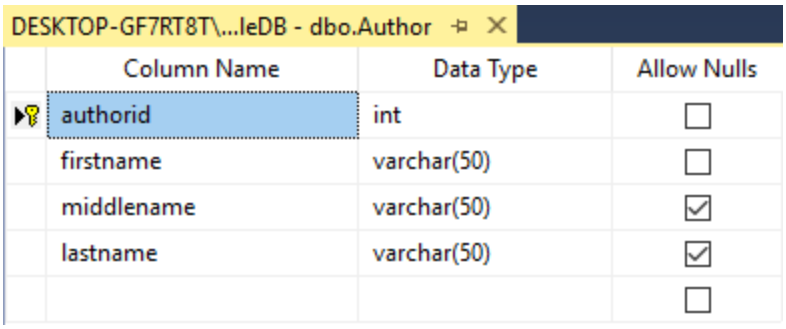
Step 3: Click OK to delete the database

Create Tables using Table Designer

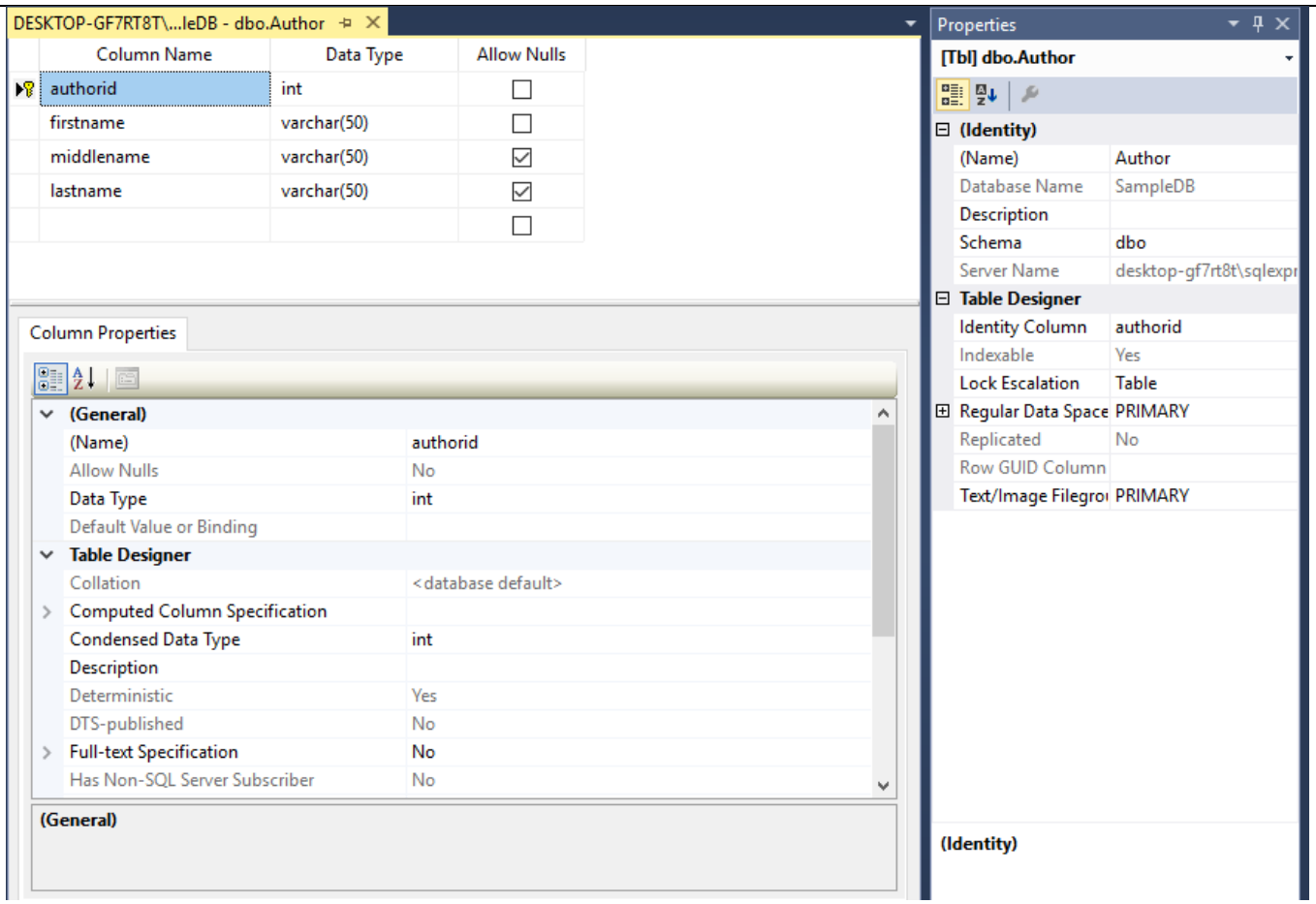
Step 1: In **Object Explorer**, connect to the instance of Database Engine that contains the database to be modified.
Step 2: In **Object Explorer**, expand the **Databases** node and then expand the database that will contain the new table.
Step 3: In Object Explorer, right-click the **Tables** node of your database and then click **Table**.



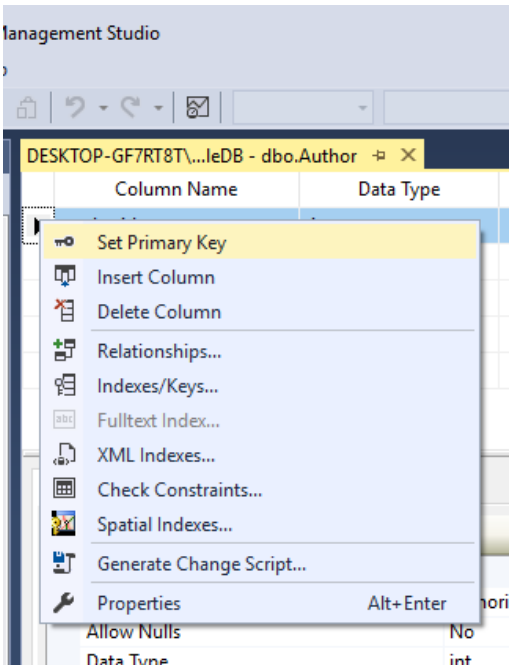
Step 4: Type column names, choose data types, and choose whether to allow nulls for each column as shown in the following screenshot:



Step 5: To specify more properties for a column, such as identity or computed column values, click the column and in the column properties tab, choose the appropriate properties.

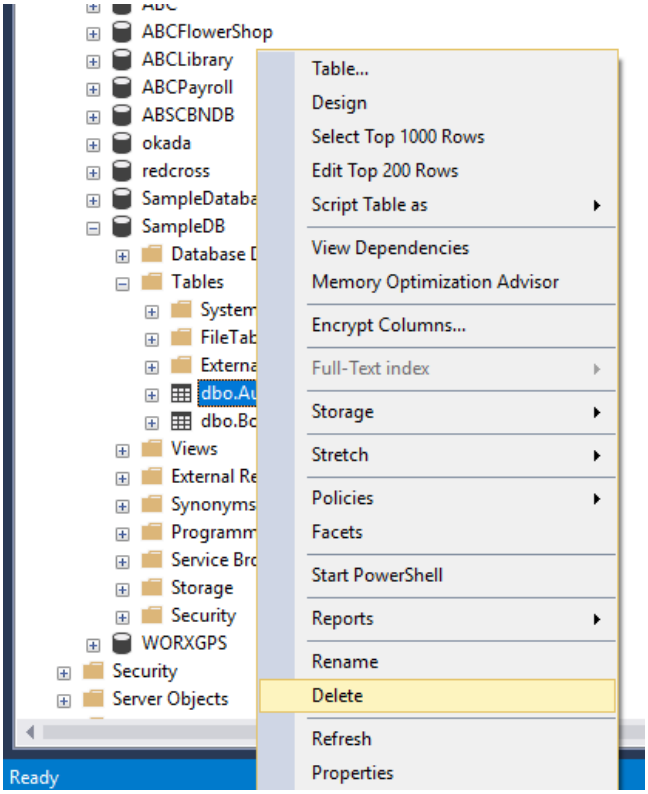


Step 6: To specify a column as a primary key, right-click the column and select **Set Primary Key**.

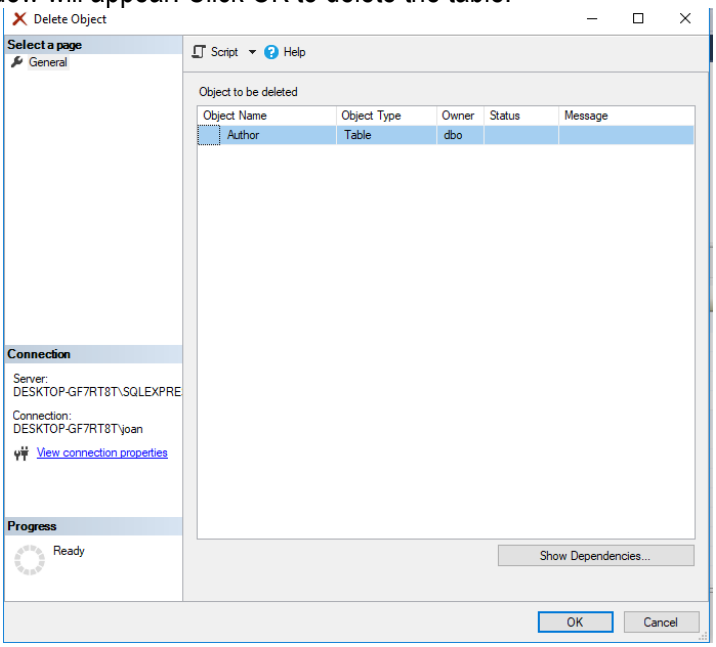


Delete Database using SQL Server Management Studio

Step 1: In Object Explorer, select the table you want to delete.
Step 2: Right-click the table and choose **Delete** from the shortcut menu.



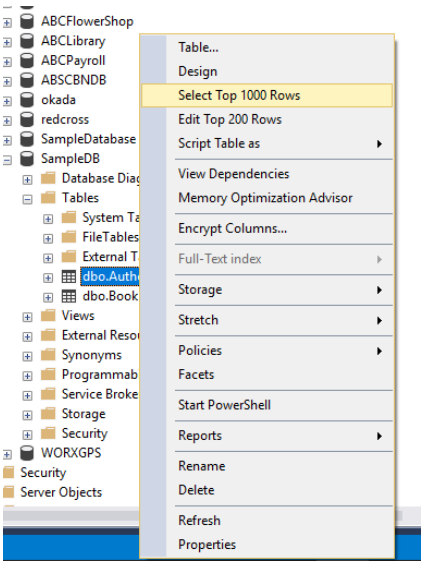
Step 3: The Delete Object window will appear. Click OK to delete the table.



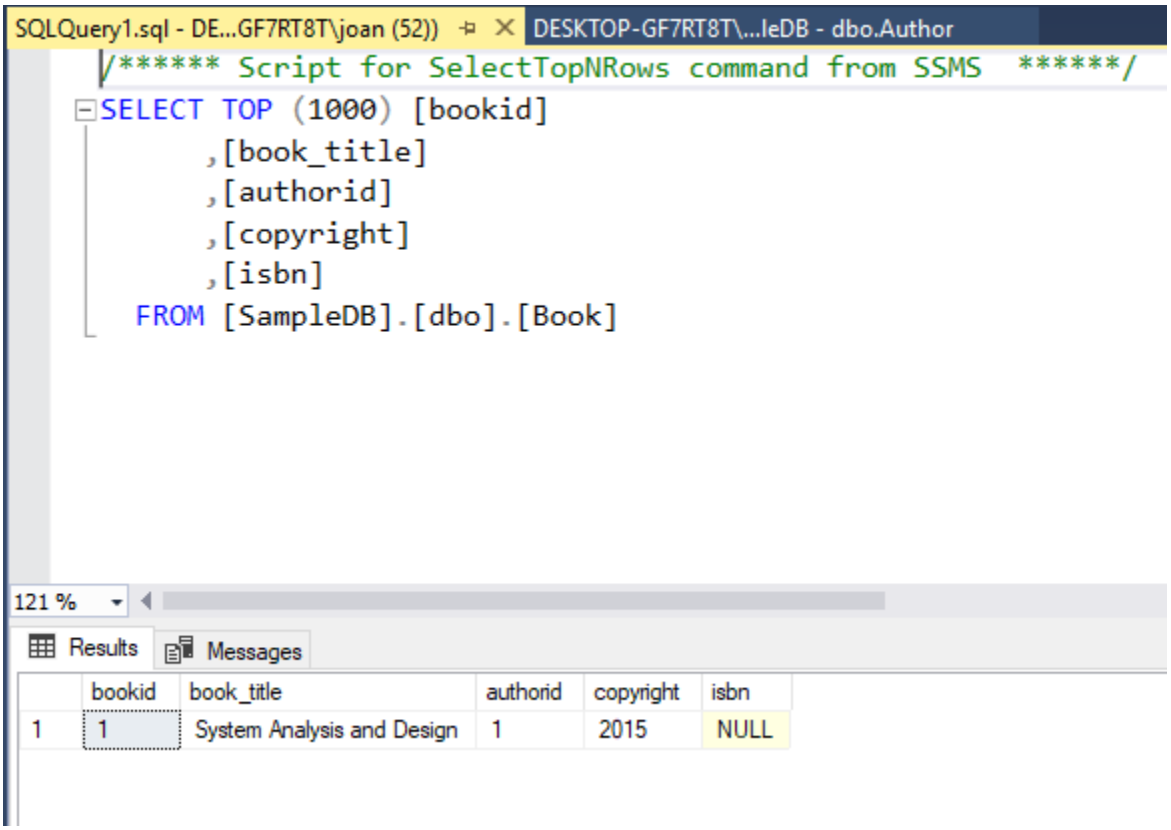
View and Edit Data in a Table using SQL Server Management Studio

1. View Data

Step 1: In Object Explorer, expand the database and choose the table to view.
Step 2: Right-click the table and select **Select Top 1000 Rows**.

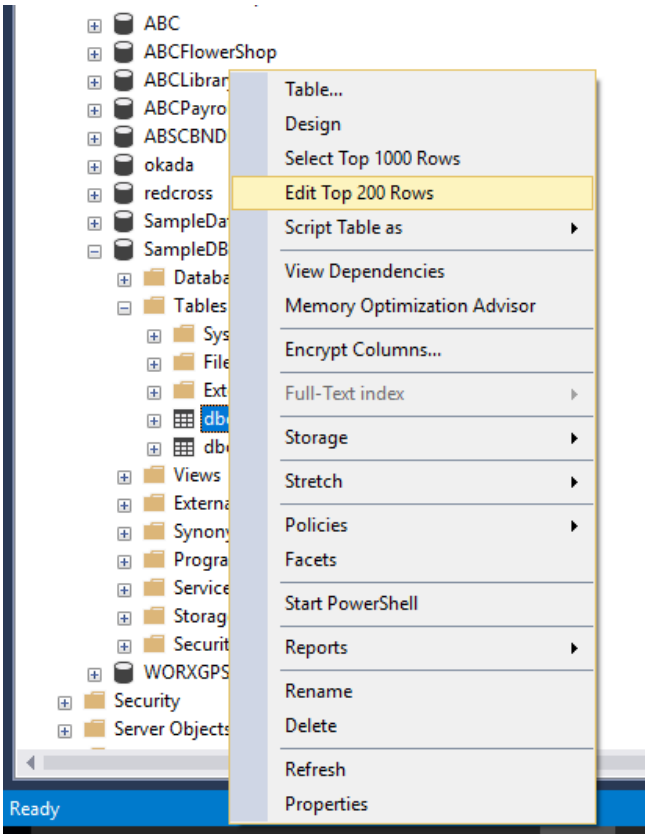


The Top 1000 rows will be selected from the table.



2. Edit Data

Step 1: In Object Explorer, expand the database and choose the table to view.
Step 2: Right-click the table and select Edit Top 200 Rows.



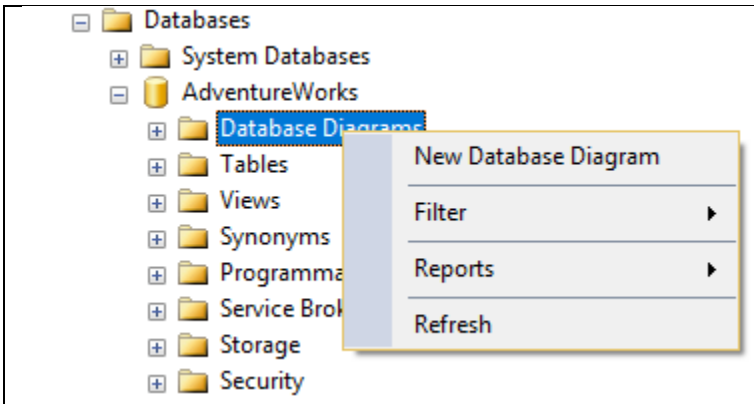
Step 3: Insert or modify the content of the given table.

DESKTOP-GF7RT8T\...leDB - dbo.Author | DESKTOP-GF7RT8T\...leDB - dbo.Author

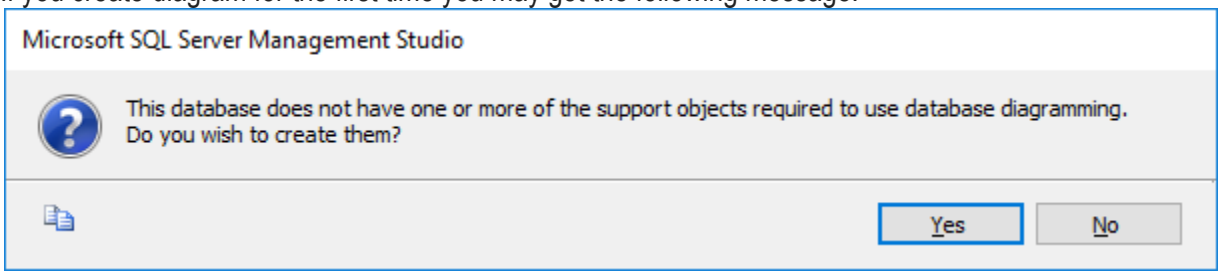
	authorid	firstname	middlename	lastname
*	NULL	NULL	NULL	NULL

Creating Relationship using Database Diagram with Microsoft SQL Server Management Studio

Step 1: To create the new database diagram, right click on **Database Diagrams** and click on **New Database Diagram**.

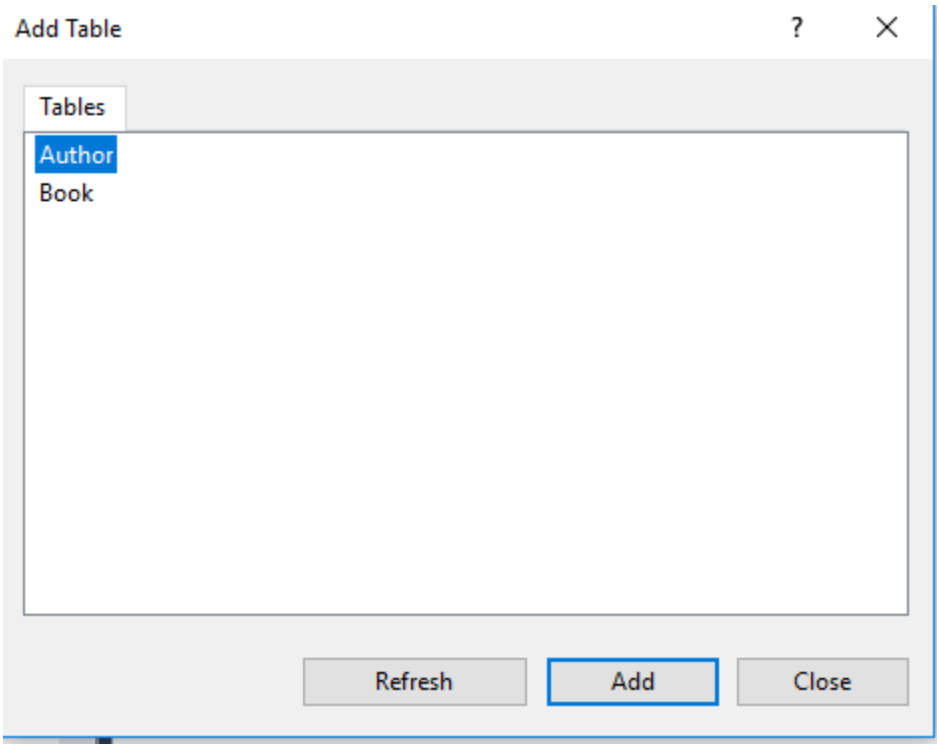


If you create diagram for the first time you may get the following message:

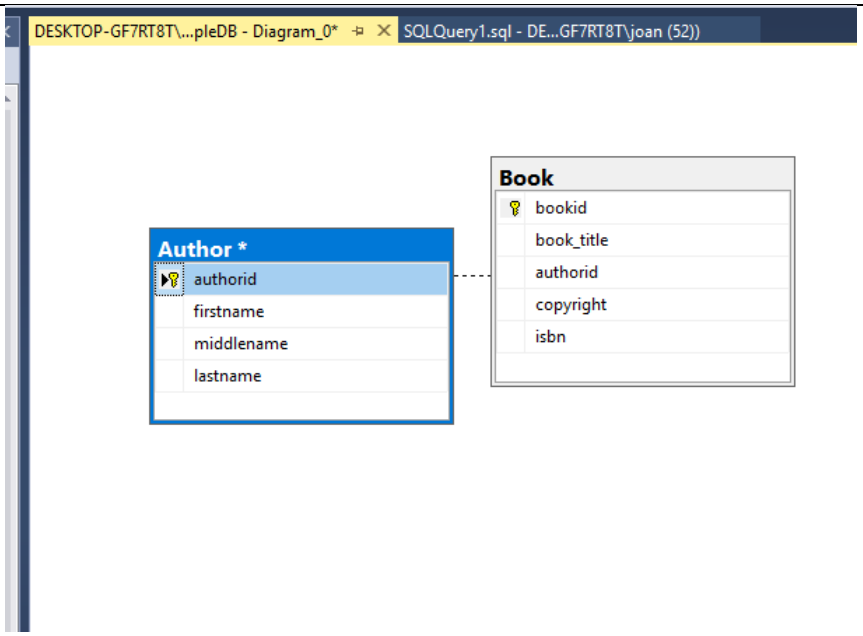


The SSMS requires some system procedures and a table that are not created with the database.

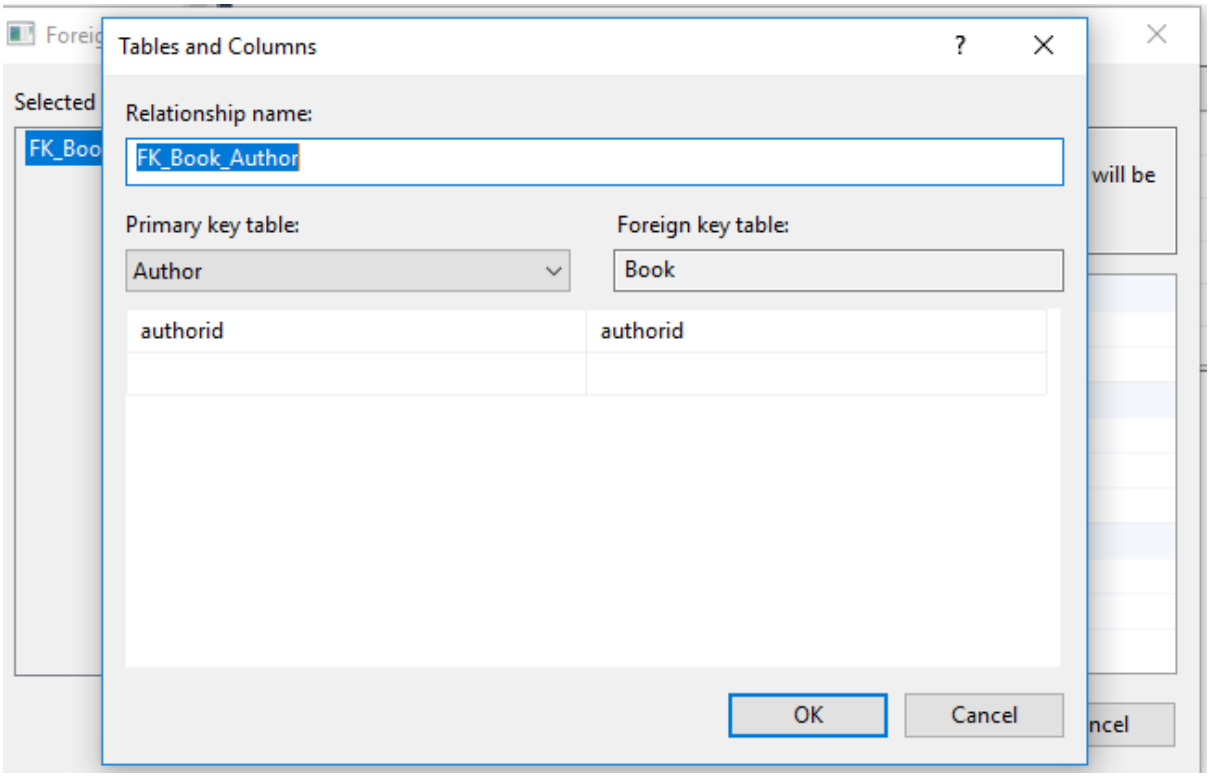
Step 2: Add the following tables to the diagram. Select the following tables and then click Add. Click Close to close the Add Table dialog box.



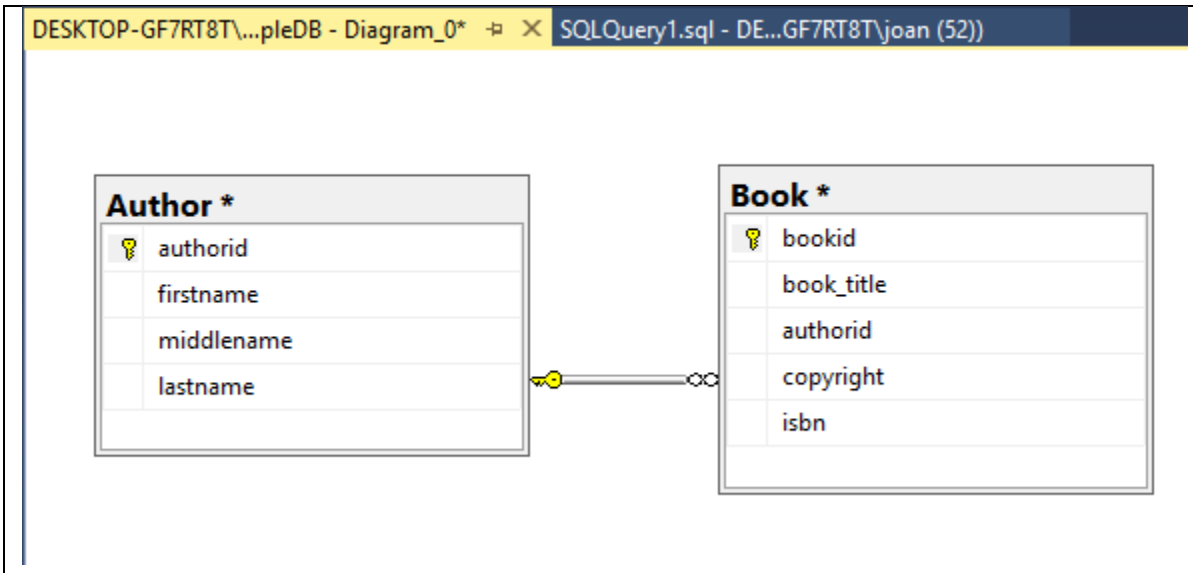
Step 3: Drag the column field of a table related to the column field of another table.



Step 4: The Tables and Columns relationship window will appear. Check the primary key and foreign key table, as well as the relationship name. Click OK to create relationship.



The relationship will appears in the database diagram window after clicking OK.



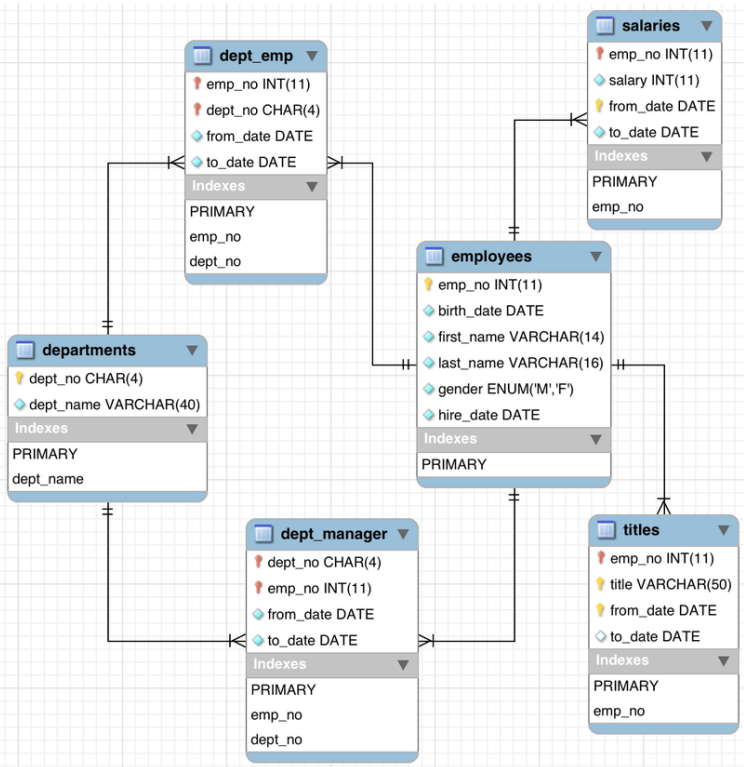
6. Database Output

Copy screenshot(s) of your output after completing the procedures provided in Part 5.

7. Supplementary Activity

Do the following tasks and copy screenshot(s) of your output.

1. Create a database ABC_Employees.
2. Create the following tables in the ABC_Employees database.



3. Create relationships using the database diagram.
4. Insert 10 Employees with their salary, department and their manager.
5. Select and view the data in the tables of ABC_EmployeesDB database.

8. Conclusion

9. Assessment (Rubric for Laboratory Performance):

Activity No. 7	
Administering Databases	
Course Code: CPE011	**Program:**
Course Title: Database Management System	**Date Performed:**
Section:	**Date Submitted:**
Name:	**Instructor:**

1. Objective(s):

This activity aims to perform basic administration of databases.

2. Intended Learning Outcomes (ILOs):

The students should be able to:
2.1 Attach and detach database.
2.2 Import and export data to other format.
2.3 Create a backup and restore the database.

3. Discussion :

Database administration refers to the whole set of activities performed by a database administrator to ensure that a database is always available as needed. These also include the several basic database administration tasks such as copying or moving of database using attach and detach method, importing and exporting of data and creating backup of database for restoration.

Attaching and Detaching Database

The data and transaction log files of a database can be detached and then reattached to the same or another instance of SQL Server. Detaching and attaching a database is useful to change the database to a different instance of SQL Server on the same computer or to move the database.

Detaching a database temporarily removes or making it offline from the instance of the SQL Server. However, it leaves the database in the database files (MDF and NDF). These two files can be used to attach the database to any instance of SQL Server.

Attaching a database is a method that is used after detaching the SQL Server database. It is used to make the database available again to a certain instance of SQL Server. The data files must be available when attaching the data files.

Importing and Exporting Database

SQL Server Import and Export Wizard is a simple way to copy data from a source to a destination. It allows easily import or export of SQL Database from any of the following data sources.

- Microsoft Excel
- Microsoft Access
- Flat files
- Another SQL Server database

Backing up and Restoring Databases

Backing up the SQL Server database is essential for protecting data. A full database backup backs up the whole database. This includes part of the transaction log so that the full database can be recovered after a full database backup is restored. Full database backups represent the database at the time the backup finished.

A differential backup is based on the most recent, previous full data backup. A differential backup captures only the data that has changed since that full backup.

To recover a SQL Server database from a failure, a database administrator has to restore a set of SQL Server backups in a logically correct and meaningful restore sequence. SQL Server restore and recovery supports restoring data from backups of a whole database, a data file, or a data page

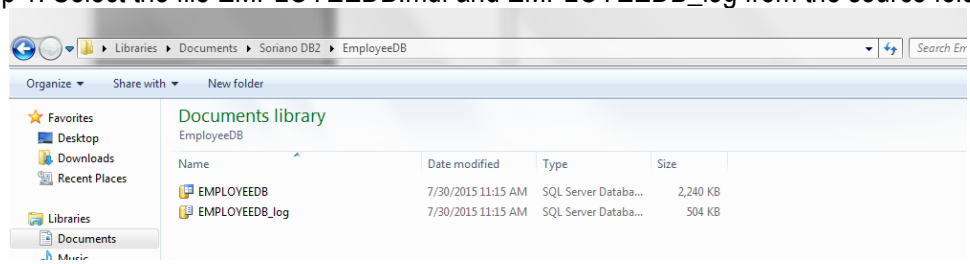
4. Resources:

Personal Computer with installed SQL Server
EmployeeDB and EmployeeDB_log file.

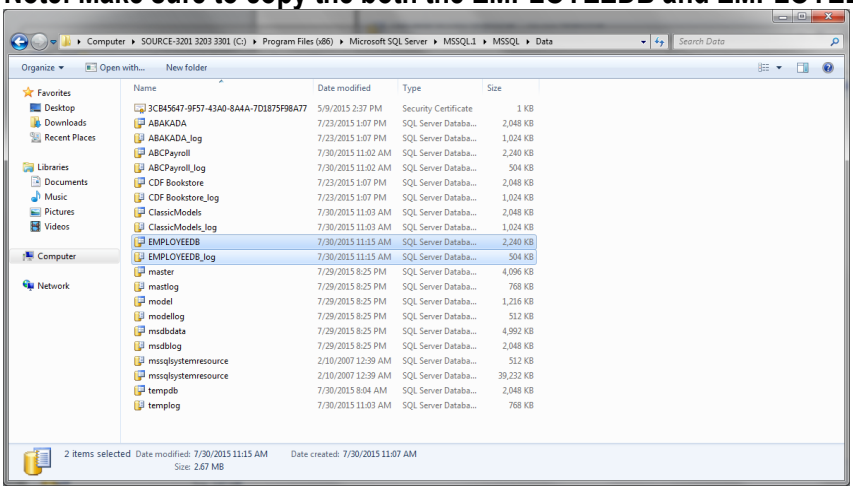
5. Procedure:

To Attach a Database

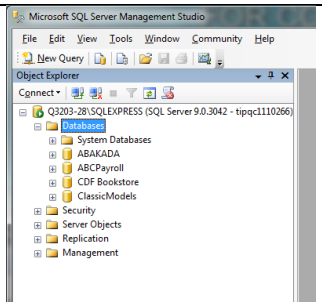
Step 1. Select the file EMPLOYEEDB.mdf and EMPLOYEEDB_log from the source folder.



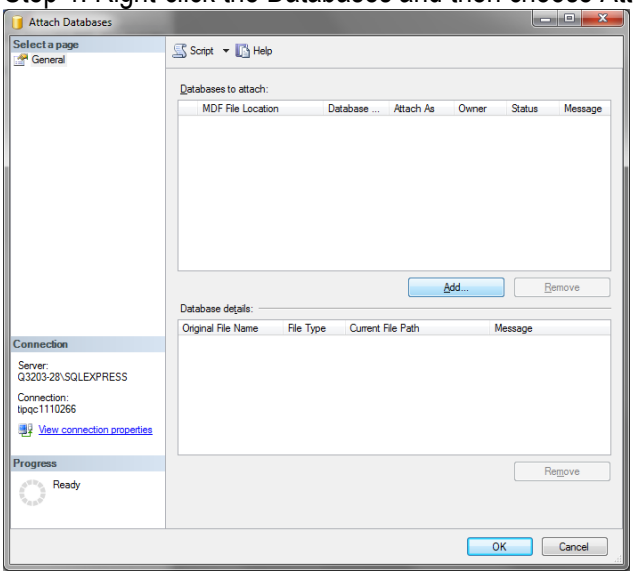
Step 2. Copy and paste the two files to **C:\Program Files (x86)\Microsoft SQL Server\MSSQL.1\MSSQL\Data**
Note: Make sure to copy the both the EMPLOYEEDB and EMPLOYEDB_log.



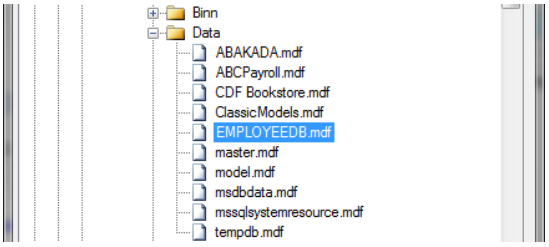
Step 3. Connect to the SQL Server database engine. Choose Databases.



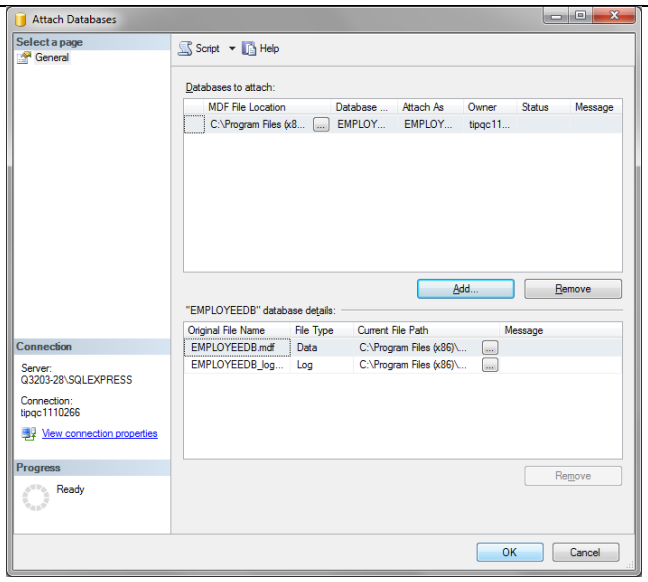
Step 4. Right-click the Databases and then choose Attach. The Attach Database window appears.



Step 5. Click the Add button. The window will display the available databases of the SQL Server that can be found in **C:\Program Files (x86)\Microsoft SQL Server\MSSQL.1\MSSQL\Data. Data. S**elect the EMPLOYEEDB.mdf. Click OK.



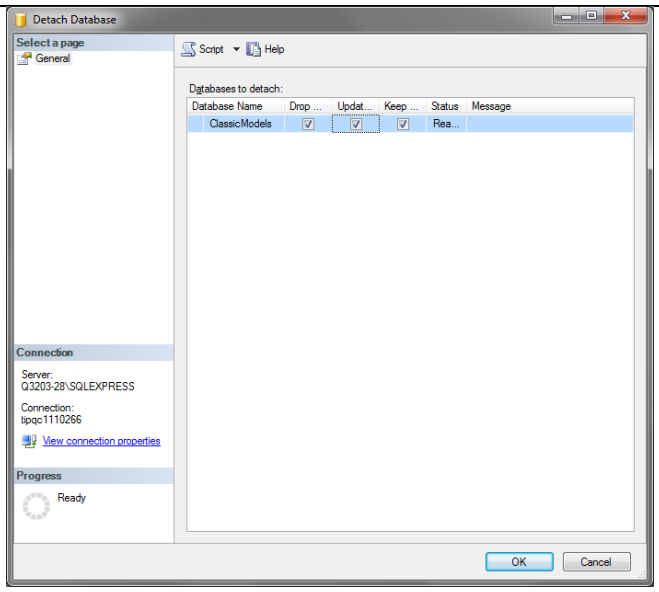
Note: The Attach Database window displays the mdf and log files of the Employee database.



To Detach a Database

Step 1.In SQL Server Management Studio, connect to an instance of the SQL Server Database Engine. Expand **Databases**, select the name of the user database you want to detach.

Step 2. Right-click the database name, point to **Tasks**, and then click**Detach**. The **Detach Database** dialog box appears. Check the boxes Drop Connections, Update Statistics and Keep Full-Text Catalogs**. Click OK.**



Note:

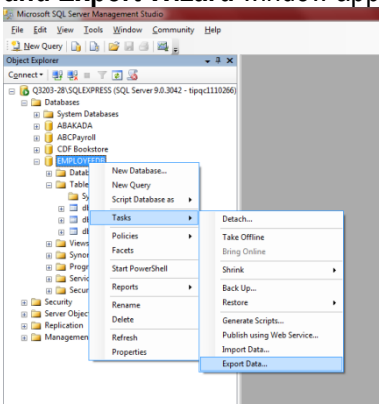
You cannot detach a database with active connections.

The newly detached database will remain visible in the Databases **node of Object Explorer until the view is refreshed. You can refresh the view at any time: Click in the Object Explorer pane, and from the menu bar select** View **and then** Refresh**.**

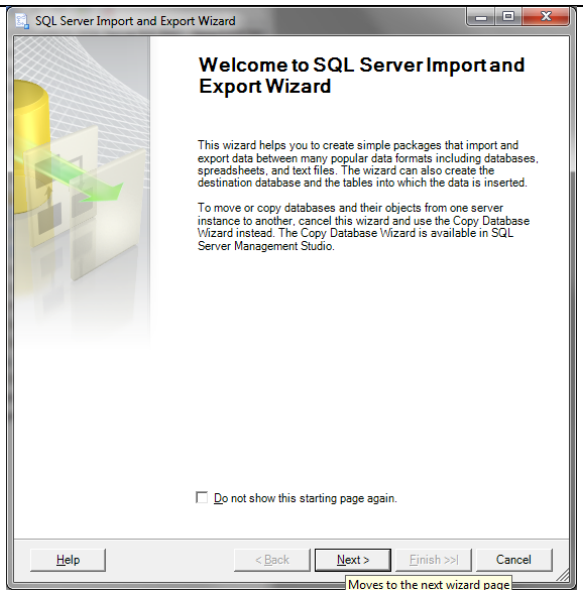
To Export Data using SQL Server Import and Export Wizard

A. Export Employee table of EmployeeDB to Employee.xls (excel file).

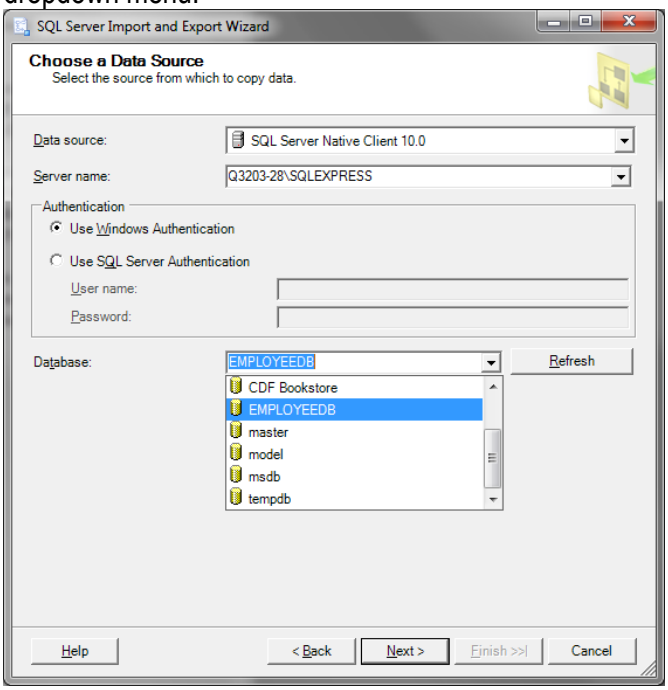
Step 1. In SQL Server Management Studio, connect to an instance of the SQL Server Database Engine. Expand **Databases**.
Step 2. Right-click the database name, point to **Tasks**, and then click**Export Data**. The **SQL Server Import and Export Wizard** window appears.



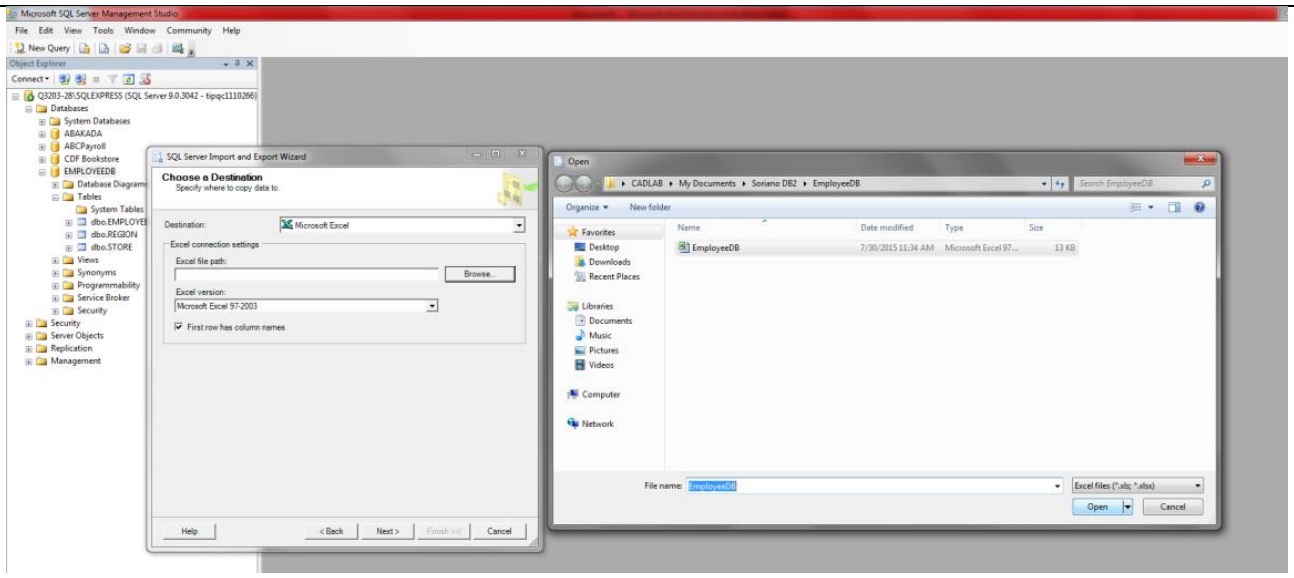
Step 3. Click the Next button on the Import and Export Wizard

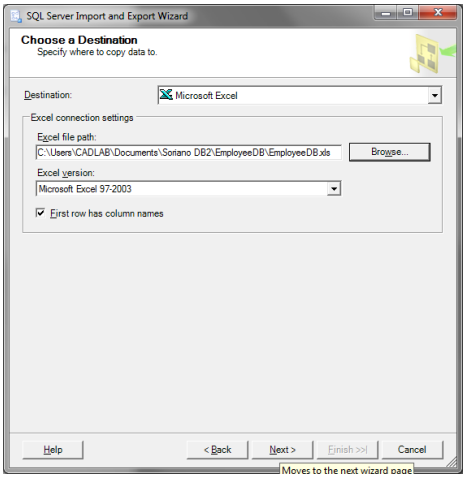


Step 4. Select Data source and choose SQL Server Native Client 10.0. Select EMPLOYEEDB on Database dropdown menu.

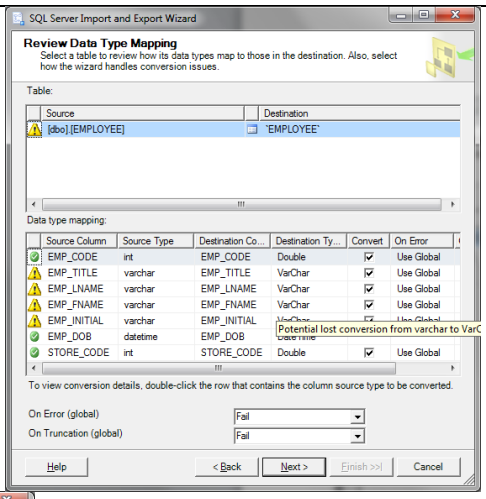


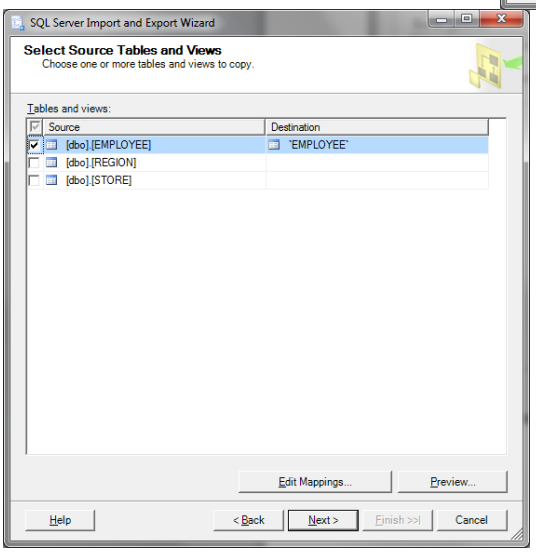
Step 5. Choose Microsoft Excel in the destination. Create a filename and click Open.



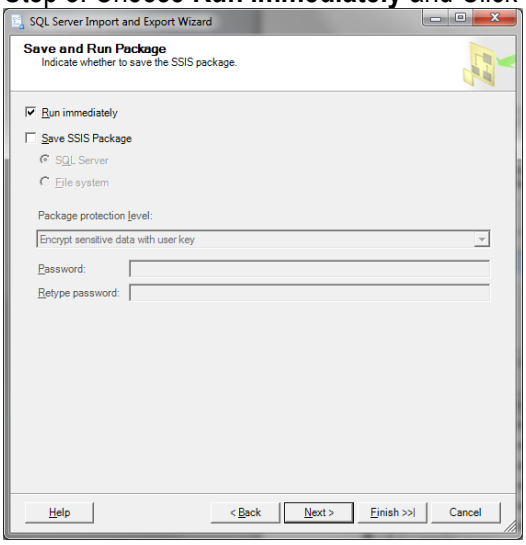


Step 6: Check the EMPLOYEE as source then click Next button.

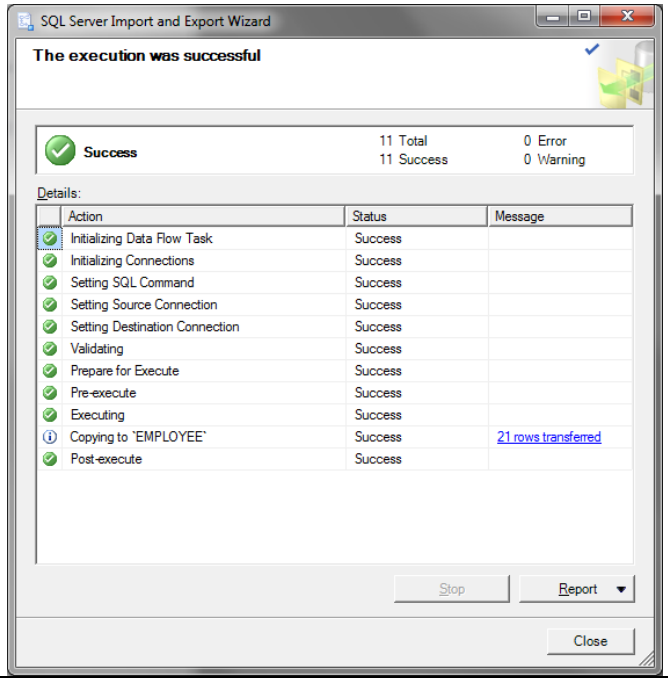




Step 8: Choose **Run immediately** and Click **NEXT.**

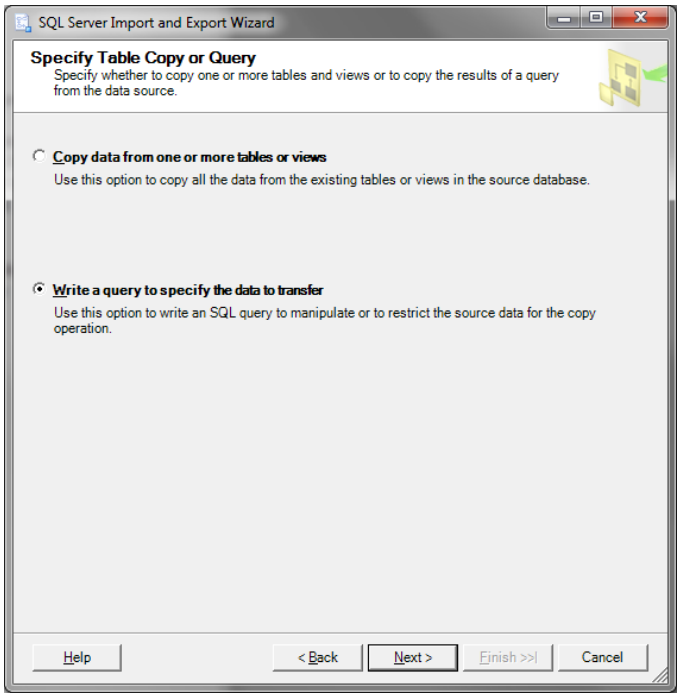


.

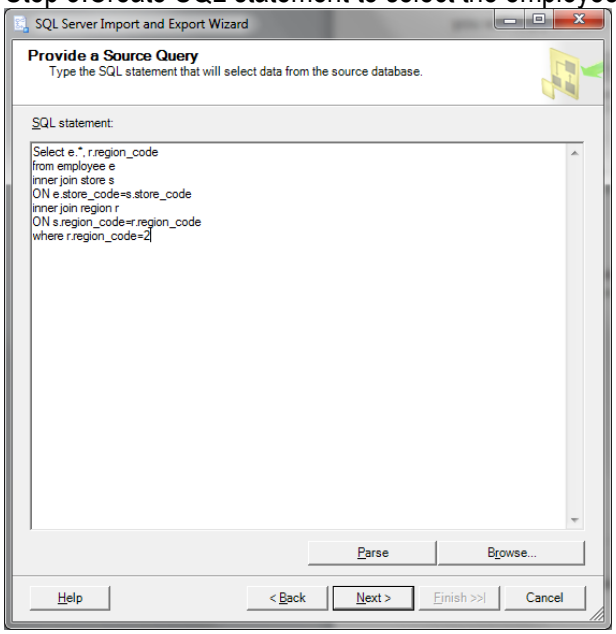


B. Export the employees list of east region to EmployeeEast worksheet of Employee excel file.

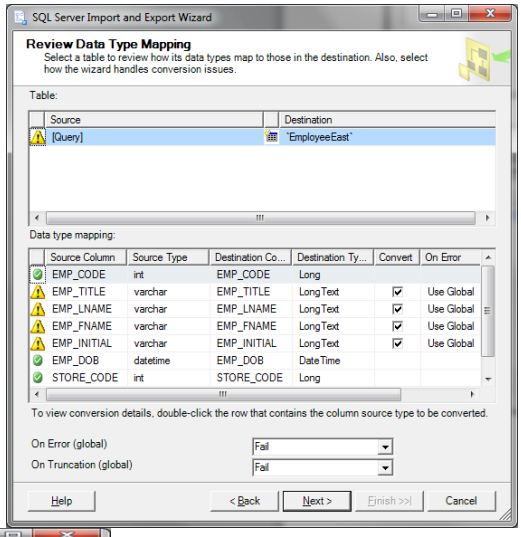
Step 1. Repeat **STEP 1 to STEP 5 to export data to excel file (Part A)**
Step 2. Choose **Write a query to specify the data to transfer**on the**Specify Table Copy or Query** window**.** Click **Next button.**

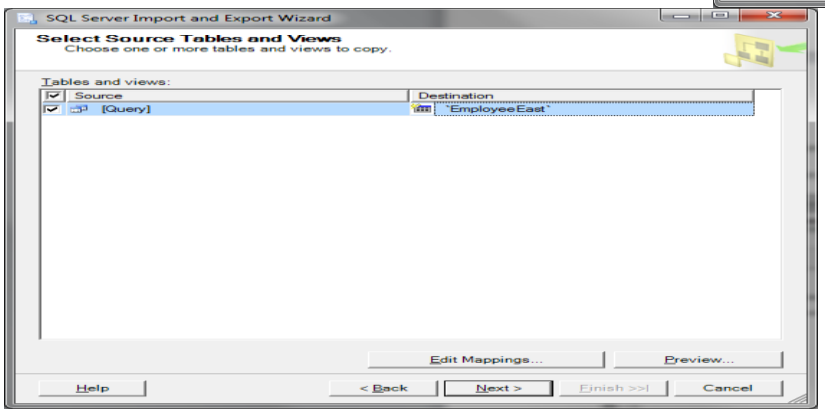


Step 3.Create SQL statement to select the employees on the East Region area only.

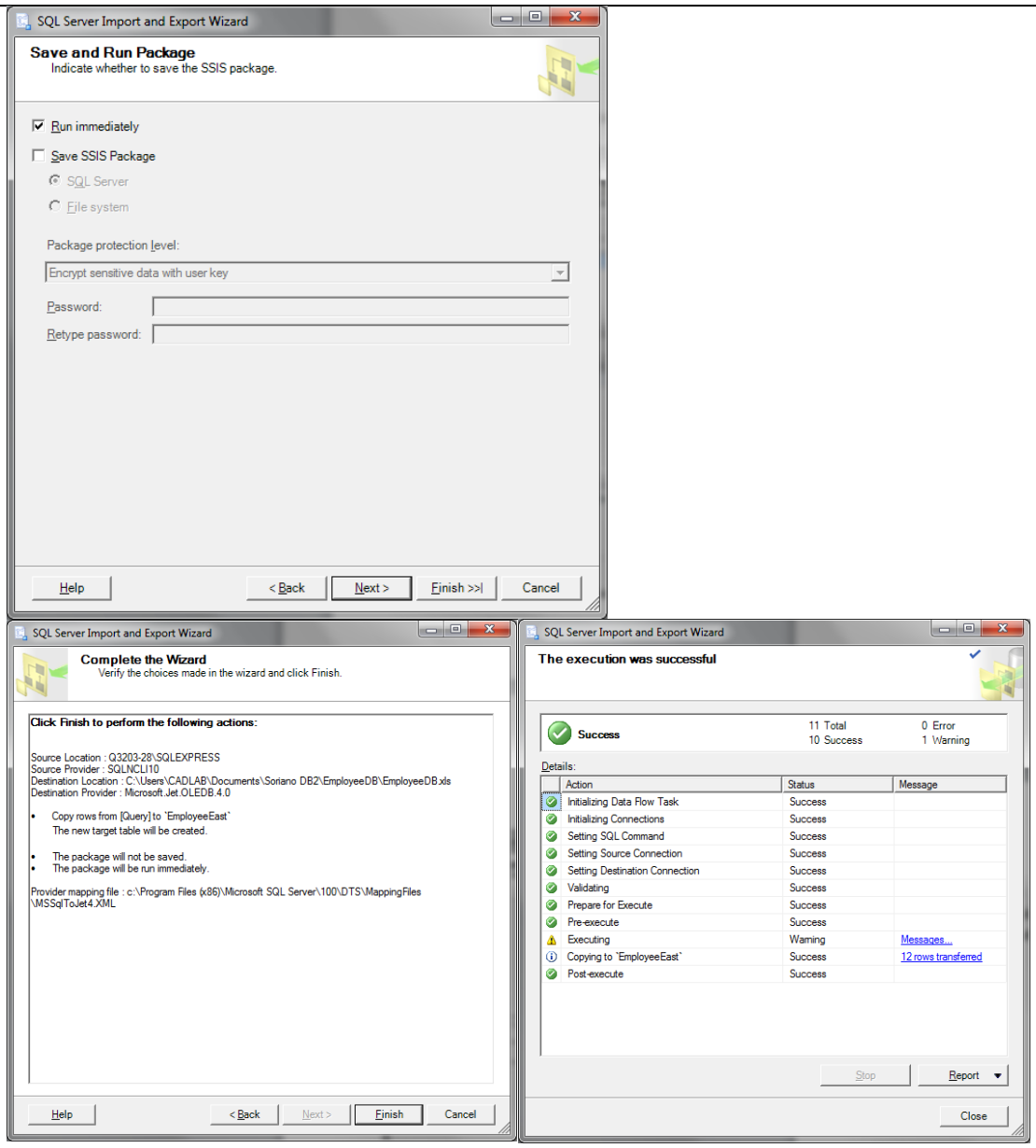


Step 4.Type **EmployeeEast** on the **Destination**column and click the **Next** button.

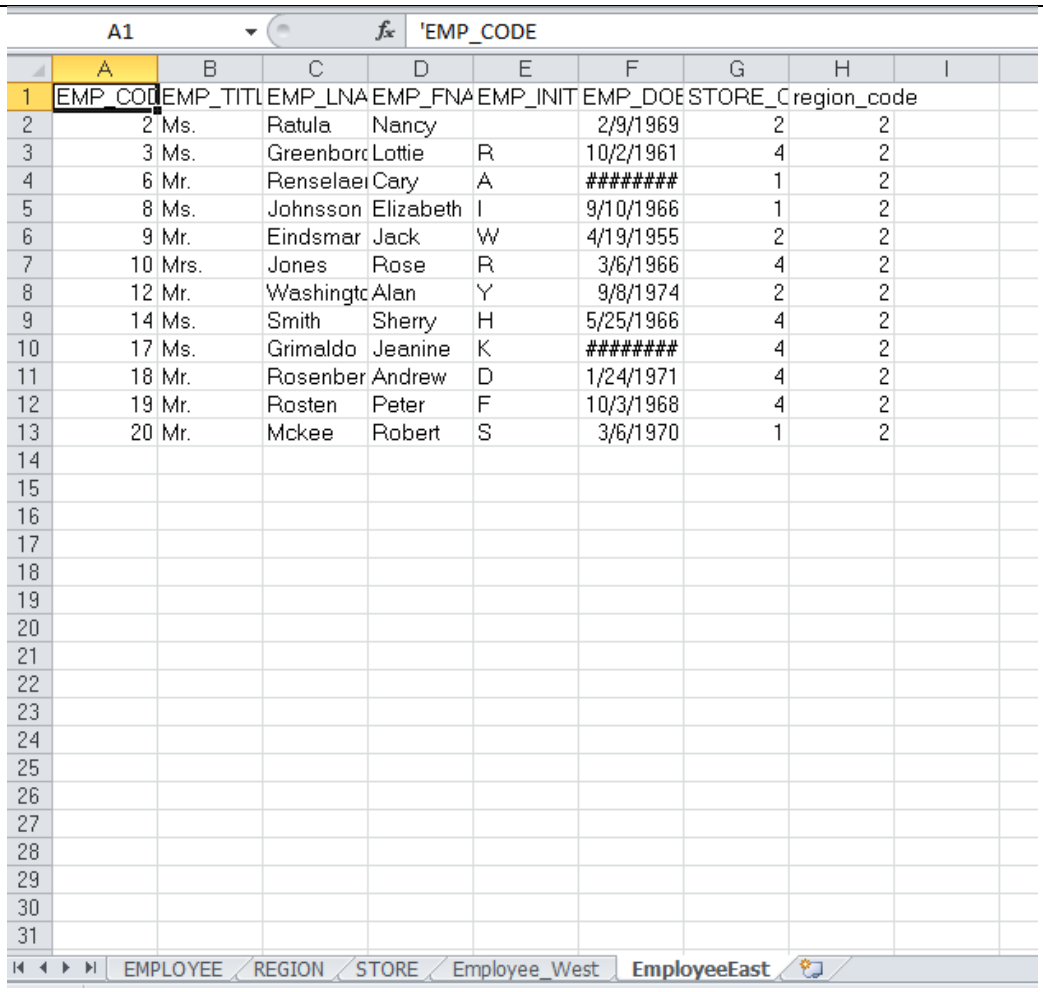




Step 5.Choose**Run immediately** and then click the **Next button.**



Step 7: Open the .xls file where you save the exported file from the SQL Server.



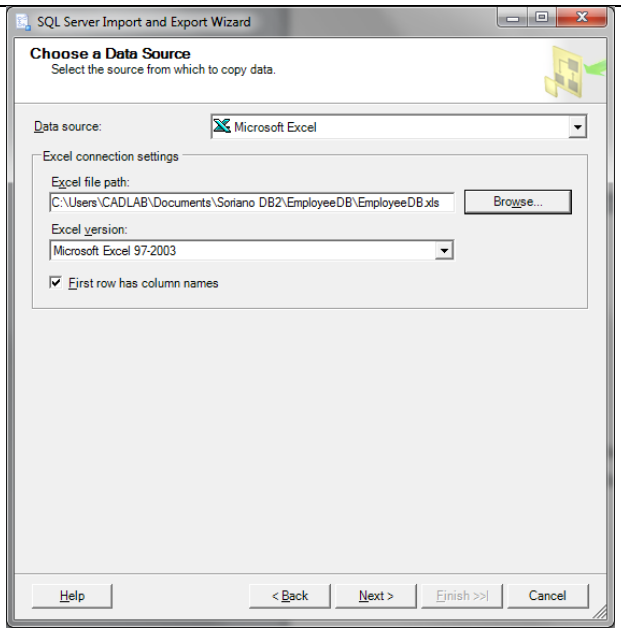
	A	B	C	D	E	F	G	H	I
1	EMP_COD	EMP_TITL	EMP_LNA	EMP_FNA	EMP_INIT	EMP_DOE	STORE_C	region_code	
2	2	Ms.	Ratula	Nancy		2/9/1969	2	2	
3	3	Ms.	Greenbor	Lottie	R	10/2/1961	4	2	
4	6	Mr.	Renselaer	Cary	A	########	1	2	
5	8	Ms.	Johnsson	Elizabeth	I	9/10/1966	1	2	
6	9	Mr.	Eindsmar	Jack	W	4/19/1955	2	2	
7	10	Mrs.	Jones	Rose	R	3/6/1966	4	2	
8	12	Mr.	Washingtc	Alan	Y	9/8/1974	2	2	
9	14	Ms.	Smith	Sherry	H	5/25/1966	4	2	
10	17	Ms.	Grimaldo	Jeanine	K	########	4	2	
11	18	Mr.	Rosenber	Andrew	D	1/24/1971	4	2	
12	19	Mr.	Rosten	Peter	F	10/3/1968	4	2	
13	20	Mr.	Mckee	Robert	S	3/6/1970	1	2	

To Import Data using SQL Server Import and Export Wizard

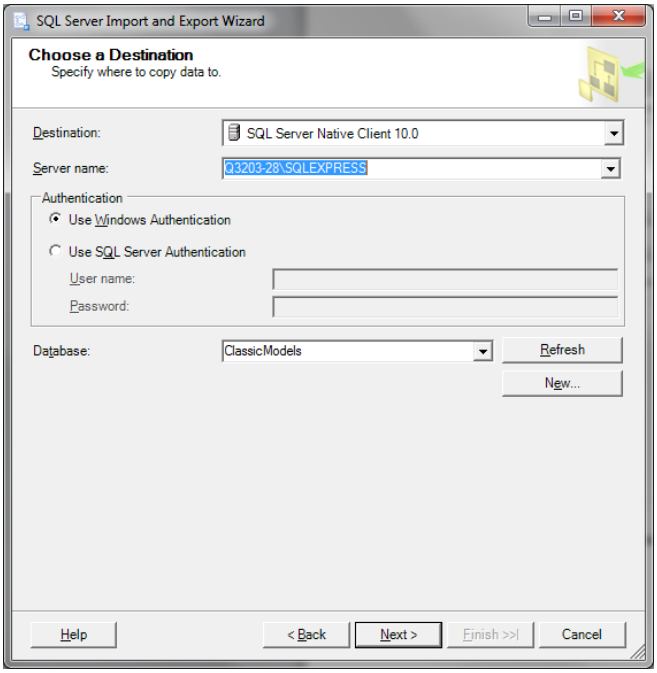
Step 1. In SQL Server Management Studio, connect to an instance of the SQL Server Database Engine. Expand **Databases**.

Step 2. Right-click the database name, point to **Tasks**, and then click**Imprt Data**. The **SQL Server Import and Export Wizard** window appears.

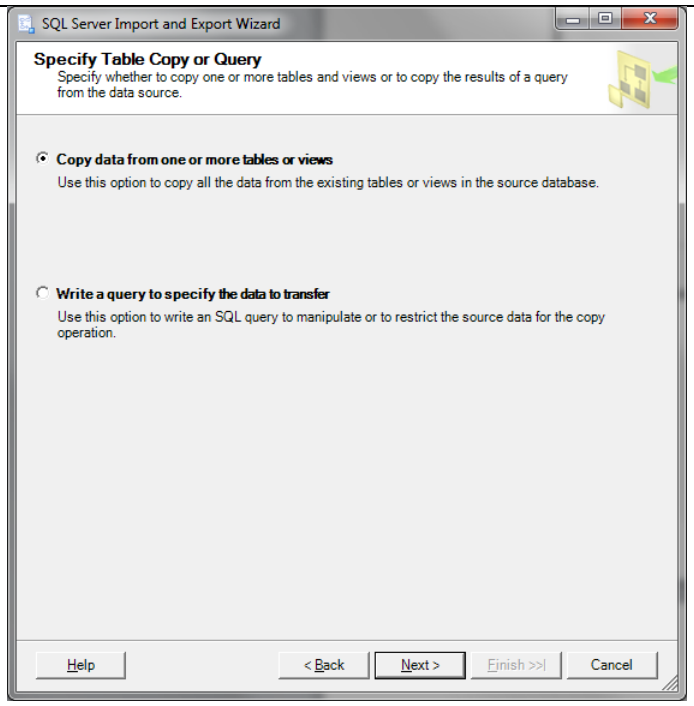
Step 3.Change the Data Source as **Microsoft Excel** choose the Excel file path **EmployeeDB.xls** click **next.**



Step 4. Change the destination SQL Server Native Client 10.0 and Choose ClassicModels in database dropdown menu. Click Next button

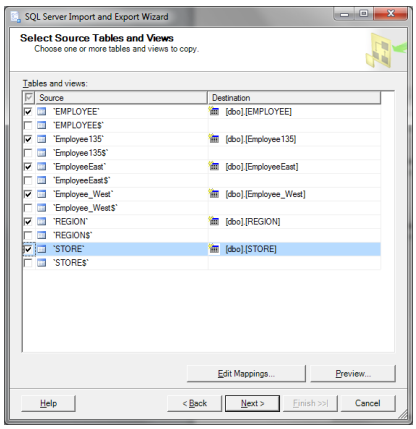


Step 5. Choose Copy data from one or more tables or views

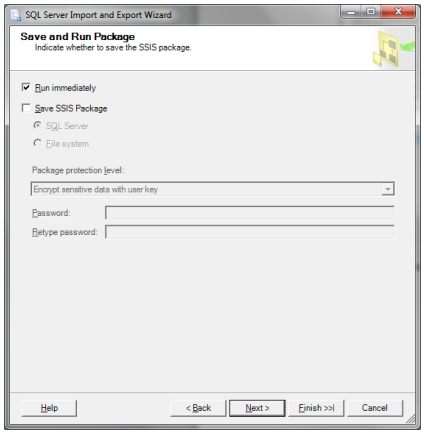


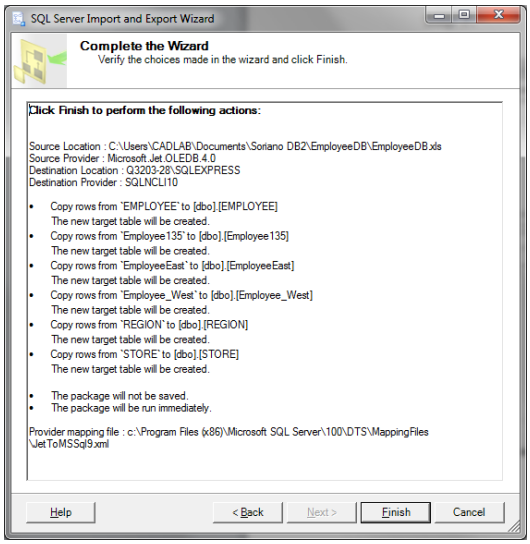
Step 6: Choose the source and destination table.

Note:
The dollar sign ($) on the name of the source table indicates an Excel worksheet. (A named range in Excel is represented by its name alone.)
The starburst on the destination table icon indicates that the wizard is going to create a new destination table.

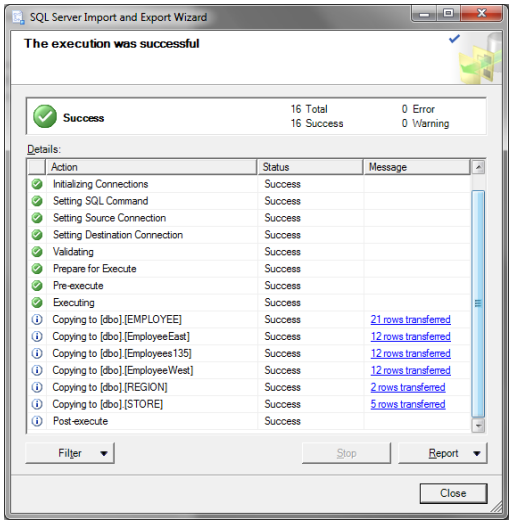


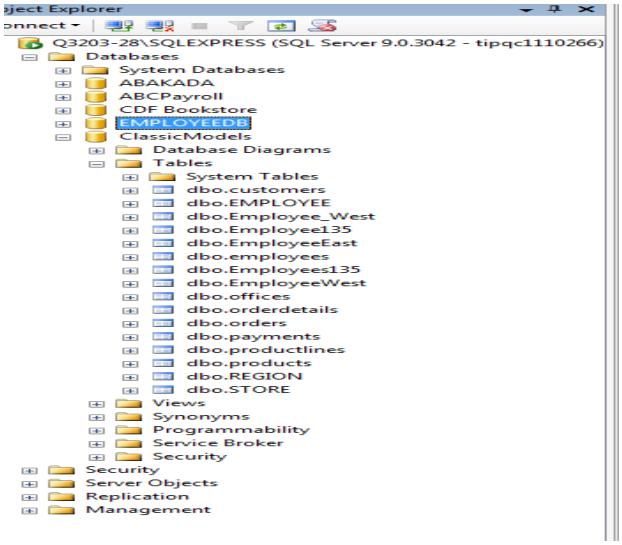
Step 7. Choose **Run immediately** and Click **Next button.** The **Complete the Wizard appears** and a summary of what the wizard is going to do. Click **Finish** to run the import-export operation.





Step 8. The results displays on the final page of the wizard. The highlighted line indicates that the wizard copied your data successfully.



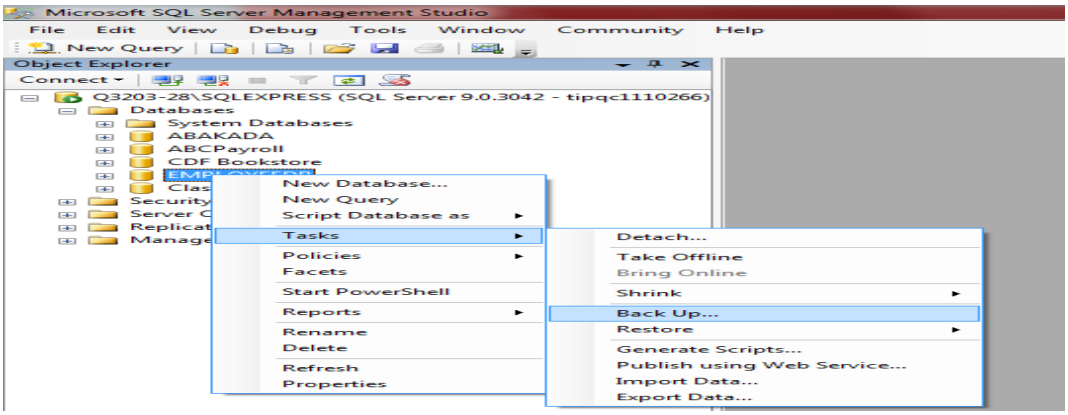


To Back up a Database using Full Backup

Step 1. After connecting to the appropriate instance of the Microsoft SQL Server Database Engine, in **Object Explorer**, click the server name to expand the server tree.

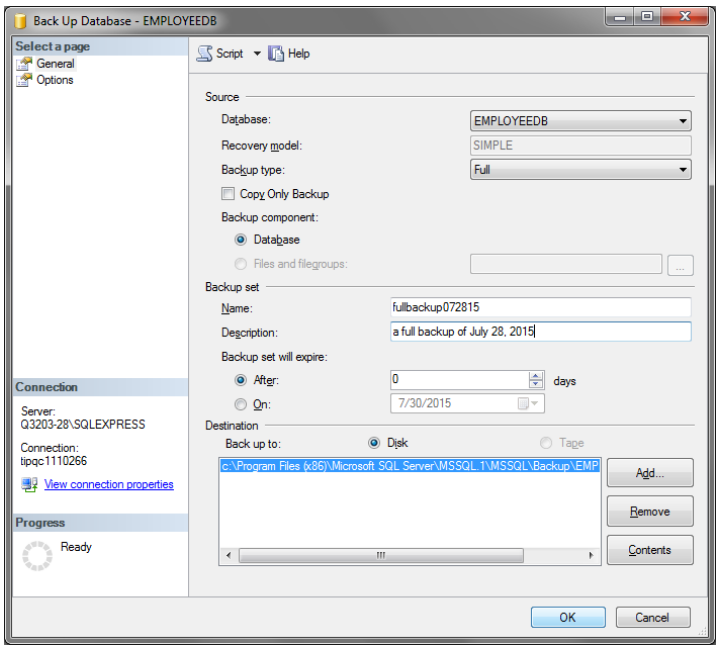
Step 2. Expand **Databases**, and select a user database to backup.

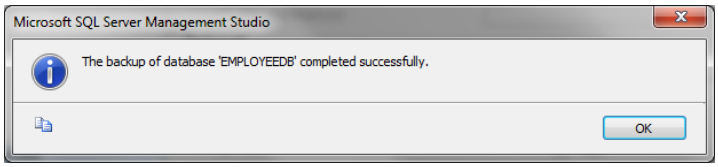
Step 3. Right-click the database, point to **Tasks**, and then click**Back Up**. The **Back Up Database** dialog box appears.

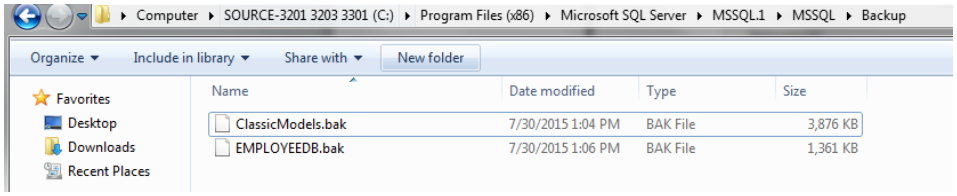


In the **Database** drop-down list, verify the database name. In the **Backup type** drop-down list, select. For **Backup component**, select the **Database** radio button.

In the **Destination** section, use the **Back up to** drop-down list to select the backup destination. Click **Add** to add additional backup objects and/or destinations.



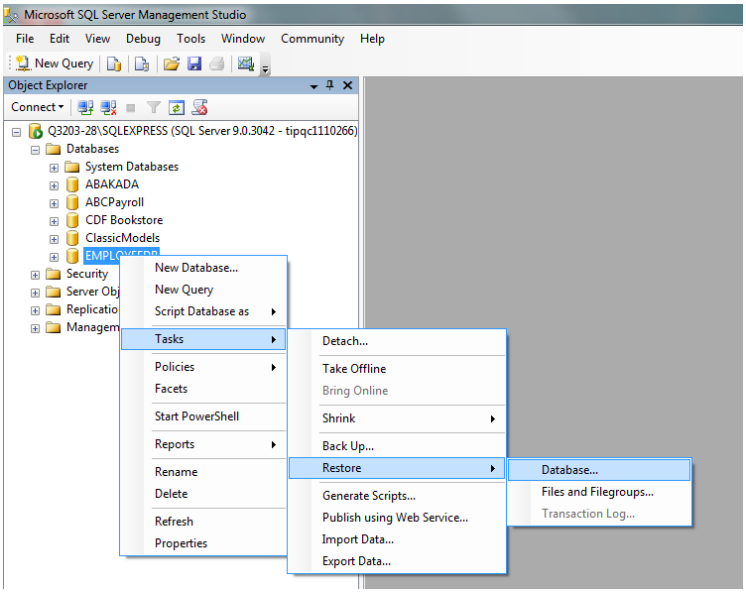


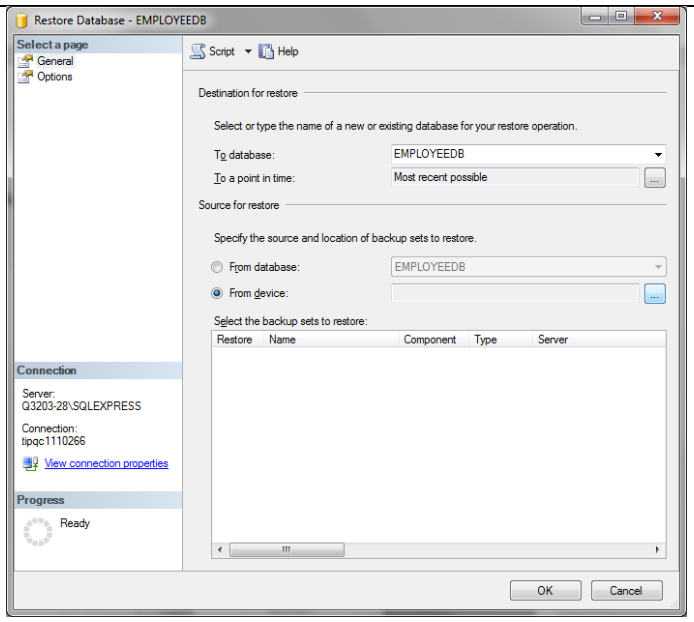


To Restore a Database using Full Backup

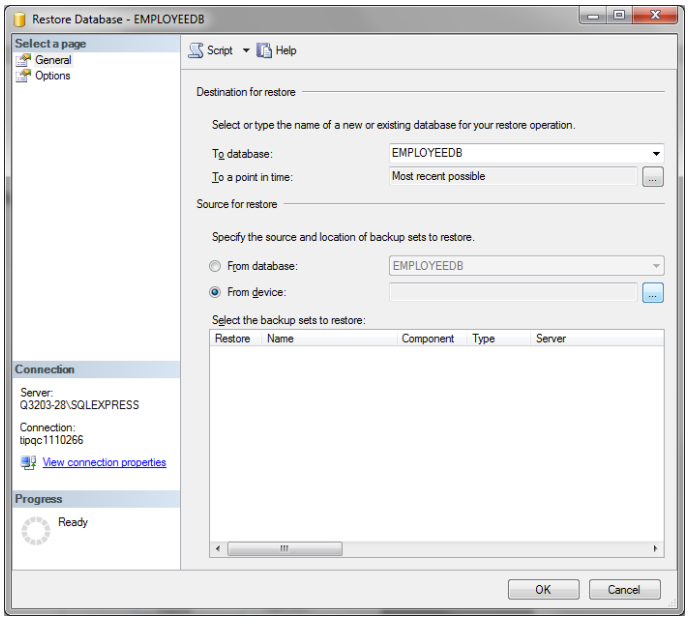
Step 1. After you connect to the appropriate instance of the Microsoft SQL Server Database Engine, in Object Explorer, click the server name to expand the server tree.

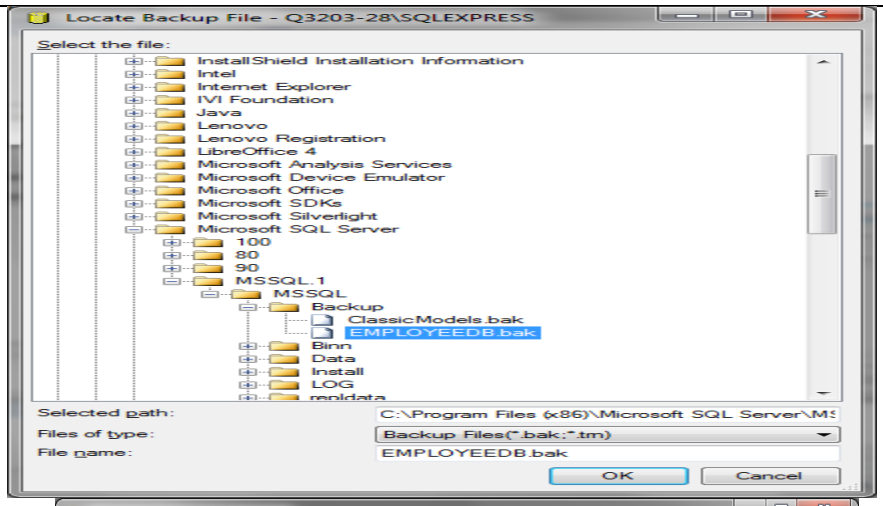
Step 2. Expand **Databases**. Select a user database to restore. Right-click the database, point to **Tasks**, and then click**Restore**. Click **Database**, which opens the **Restore Database** dialog box. On the **General** page, the name of the restoring database appears in the **To database** list box. To create a new database, enter its name in the list box. In the **To a point in time** text box, either retain the default (**Most recent possible**) or select a specific date and time by clicking the browse button, which opens the **Point in Time Restore** dialog box.

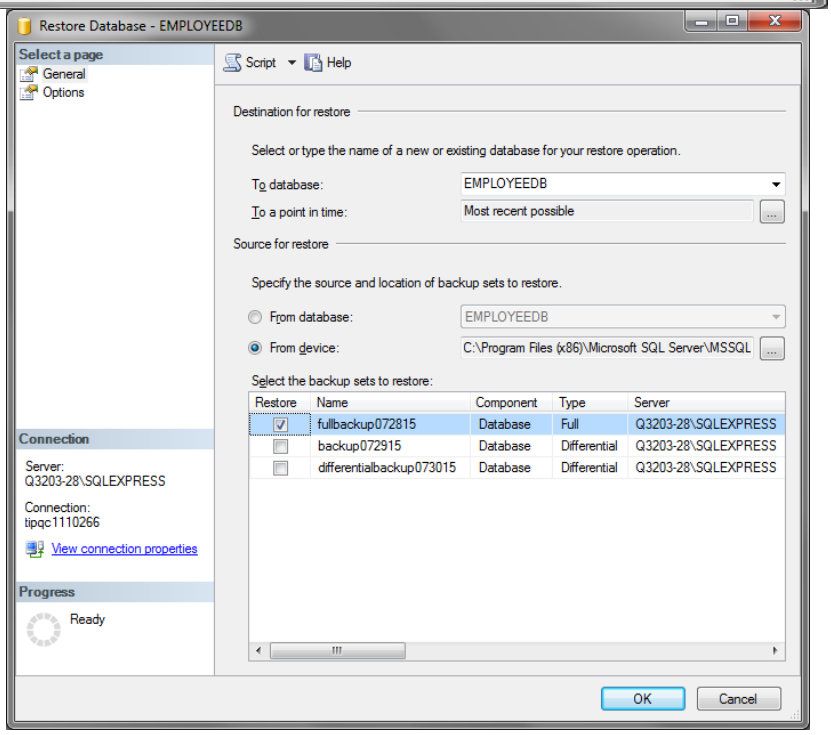


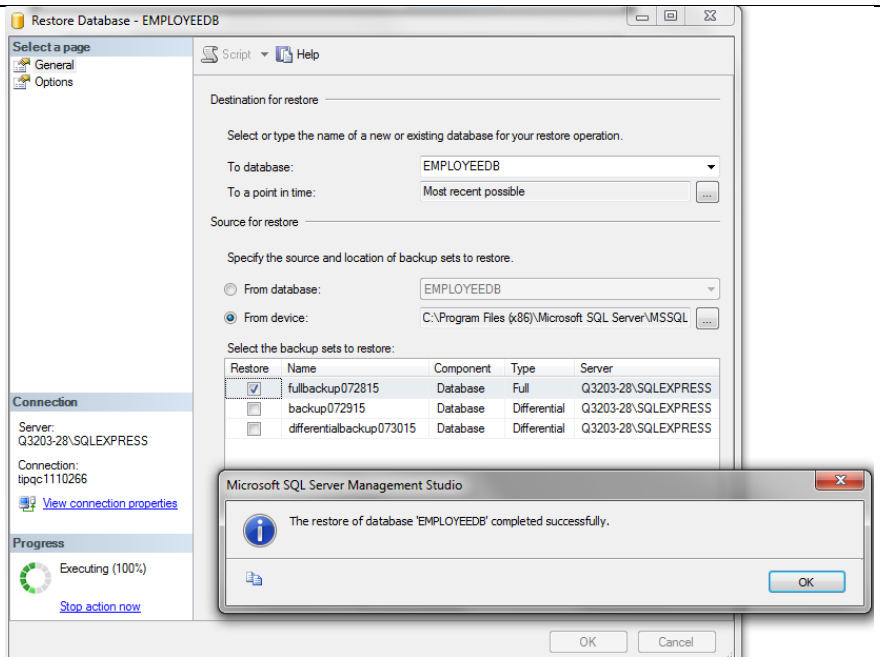


Step 3. Specify the source and location of the backup sets to restore. Click the browse button, which opens the **Specify Backup** dialog box. In the **Backup media** list box, select one of the listed device types. To select one or more devices for the **Backup location** list box, click **Add**. In the **Select the backup sets to restore** grid, select the backups to restore. This grid displays the backups available for the specified location. By default, a recovery plan is suggested. Click OK button to restore the database.




Locate Backup File - Q3203-28\SQLEXPRESS
Select the file:
InstallShield Installation Information
Intel
Internet Explorer
IVI Foundation
Java
Lenovo
Lenovo Registration
LibreOffice 4
Microsoft Analysis Services
Microsoft Device Emulator
Microsoft Office
Microsoft SDKs
Microsoft Silverlight
Microsoft SQL Server
100
80
90
MSSQL.1
MSSQL
Backup
ClassicModels.bak
EMPLOYEEDB.bak
Binn
Data
Install
LOG
Selected path:
C:\Program Files (x86)\Microsoft SQL Server\MS
Files of type:
Backup Files(*.bak;*.tm)
File name:
EMPLOYEEDB.bak
OK
Cancel


Restore Database - EMPLOYEEDB
Select a page
General
Options
Script
Help
Destination for restore
Select or type the name of a new or existing database for your restore operation.
To database:
EMPLOYEEDB
To a point in time:
Most recent possible
Source for restore
Specify the source and location of backup sets to restore.
From database:
EMPLOYEEDB
From device:
C:\Program Files (x86)\Microsoft SQL Server\MSSQL
Select the backup sets to restore:
Restore
Name
Component
Type
Server
fullbackup072815
Database
Full
Q3203-28\SQLEXPRESS
backup072915
Database
Differential
Q3203-28\SQLEXPRESS
differentialbackup073015
Database
Differential
Q3203-28\SQLEXPRESS
Connection
Server:
Q3203-28\SQLEXPRESS
Connection:
tipqc1110266
View connection properties
Progress
Ready
OK
Cancel



6. Database Output

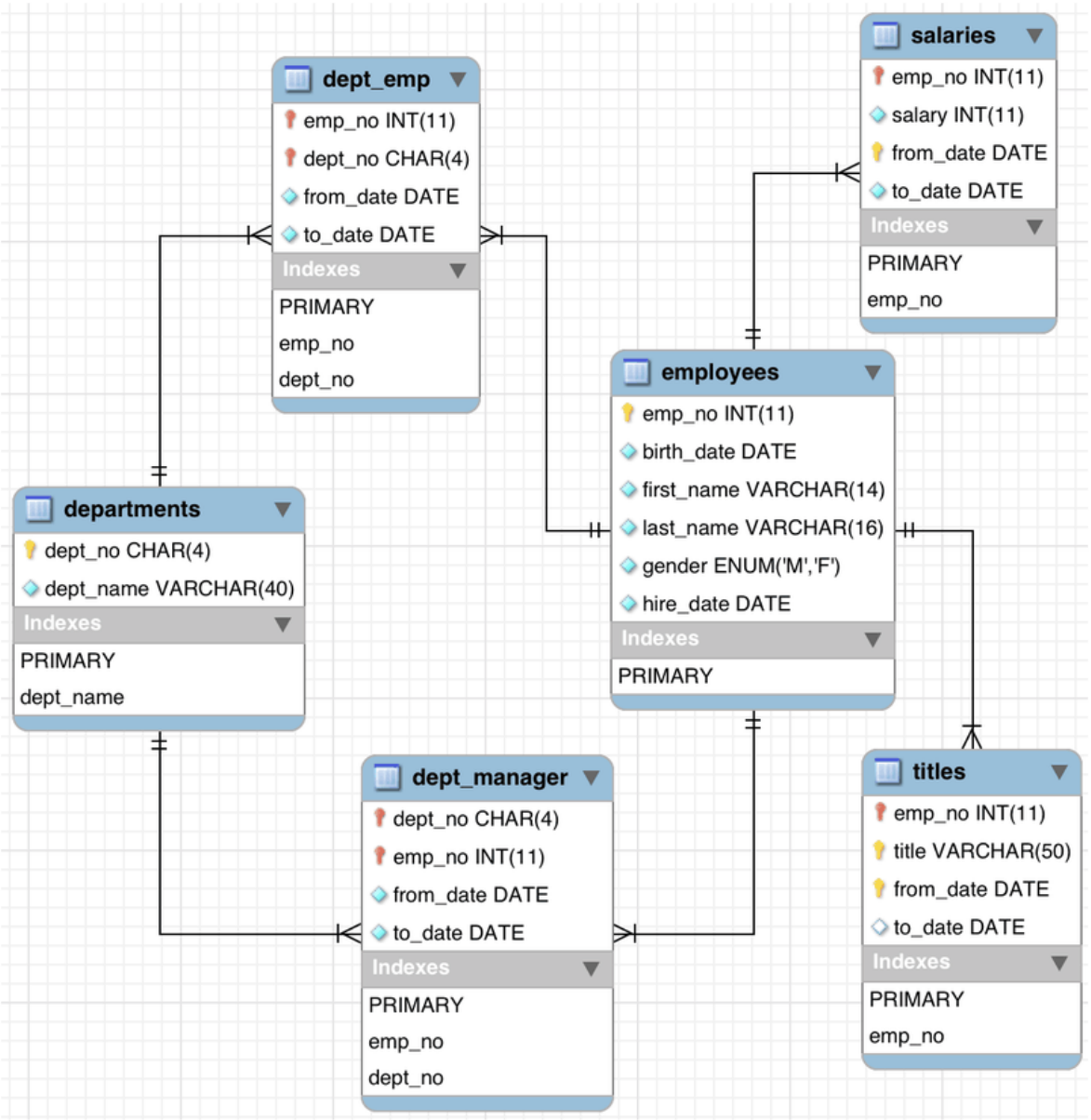
Copy screenshot(s) of your database after completing the procedures provided in Part 5.

A. Attach Database
B. Detach Database
C. Export Data
D. Import Data
E. Backup Database

7. Supplementary Activity

Do the following tasks and copy screenshot(s) of your output.

1. Create Employees_ABC database using the given structure. Change the data type according to the data type of SQL server.



2. Create a copy of Employees_ABC database and save the mdf and log files to EMPLOYEES folder.
3. Create an excel file named as EMPLOYEES_ABC. Create a department worksheet. Input the following information to the department worksheet.

Department No	Department Name
1	Administration
2	Accounting
3	Manufacturing
4	Finance
5	Information Technology
6	Maintenance

4. Import the department worksheet to the department table of Employees_ABC database.
5. Export the department table to Employees_CDF database.
6. Insert twenty (10 F and 10 M) employees information to the employees table of Employees_ABC database.

7. Export only 10 female employees information to employee_female worksheet of EMPLOYEES_ABC excel file.
8. Create a full backup of Employees_ABC database. Choose your own backup name.
9. Create a database Employees_Happy. Restore Employees_Happy database using the full backup created on step 8.

8. Conclusion

9. Assessment (Rubric for Laboratory Performance):

Activity No. 8	
Securing Databases	
Course Code: CPE011	**Program:**
Course Title: Database Management System	**Date Performed:**
Section:	**Date Submitted:**
Name:	**Instructor:**

1. Objective(s):

This activity aims to secure databases using different methodologies.

2. Intended Learning Outcomes (ILOs):

The students should be able to:
2.1 Configure authentication and authorization of database users.
2.2Assign appropriate server and database roles to the users.

3. Discussion :

Database security uses a broad range of security controls to protect databases from illegitimate use and malicious threats and attacks that can compromise the confidentiality, integrity and availability of the database. Database security is generally planned, implemented and maintained by a database administrator and or other information security professional. These include restricting unauthorized access and use by implementing strong and multifactor access and data management controls.

SQL Server provides a security architecture that is designed to allow database administrators and developers to create secure database applications and counter threats.The SQL Server security framework manages access to securable entities through authentication and authorization.

Authentication is the process of logging on to SQL Server by which a principal requests access by submitting credentials that the server evaluates. Authentication establishes the identity of the user or process being authenticated.

SQL Server supports two authentication modes, Windows authentication mode and mixed mode.

- Windows authentication is the default, and is often referred to as integrated security because this SQL Server security model is tightly integrated with Windows. Specific Windows user and group accounts are trusted to log in to SQL Server. Windows users who have already been authenticated do not have to present additional credentials.
- Mixed mode supports authentication both by Windows and by SQL Server. User name and password pairs are maintained within SQL Server.

Authorization is the process of determining which securable resources a principal can access, and which operations are allowed for those resources. *Principals* are entities that can request SQL Server resources.Every principal has a security identifier (SID).

SQL Server-level principals

- SQL Server authentication Login
- Windows authentication login for a Windows user
- Windows authentication login for a Windows group
- Server Role

Database-level principals

- Database User
- Database Role
- Application Role

Permissions in the Database Engine are managed at the server level through logins and server roles, and at the database level through database users and database roles. To easily manage the permissions in your databases, SQL Server provides several *roles* which are security principals that group other principals. Database-level roles are database-wide in their permissions scope.

There are two types of database-level roles: *fixed-database roles* that are predefined in the database and *user-defined database roles* that you can create.

Fixed-database roles are defined at the database level and exist in each database. Members of the **db_owner** database role can manage fixed-database role membership.

Fixed-Database role name	Description
db_owner	Members can perform all configuration and maintenance activities on the database, and can also drop the database in SQL Server.
db_securityadmin	Members can modify role membership and manage permissions.
db_accessadmin	Members can add or remove access to the database for Windows logins, Windows groups, and SQL Server logins.
db_backupoperator	Members can back up the database.
db_ddladmin	Members can run any Data Definition Language (DDL) command in a database.
db_datawriter	Members can add, delete, or change data in all user tables.
db_datareader	Members can read all data from all user tables.
db_denydatawriter	Members cannot add, modify, or delete any data in the user tables within a database.
db_denydatareader	Members cannot read any data in the user tables within a database.

QL Server provides server-level roles to help you manage the permissions on a server. These roles are security principals that group other principals.Server-level roles are server-wide in their permissions scope.Fixed server roles are provided for convenience and backward compatibility. Assign more specific permissions whenever possible.

SQL Server provides nine fixed server roles. The permissions that are granted to the fixed server roles (except **public**) cannot be changed.

Fixed server-level role	Description
sysadmin	Members can perform any activity in the server.
serveradmin	Members can change server-wide configuration options and shut down the server.
securityadmin	Members manage logins and their properties. They can GRANT, DENY, and REVOKE server-level permissions. They can also GRANT, DENY, and REVOKE database-level permissions if they have access to a database. Additionally, they can reset passwords for SQL Server logins. **IMPORTANT:** The ability to grant access to the Database Engine and to configure user permissions allows the security admin to assign most server permissions. The **securityadmin** role should be treated as equivalent to the **sysadmin** role.
processadmin	Members can end processes that are running in an instance of SQL Server.
setupadmin	Members can add and remove linked servers by using Transact-SQL statements. (**sysadmin** membership is needed when using Management Studio.)
bulkadmin	Members can run the BULK INSERT statement.

diskadmin	The **diskadmin** fixed server role is used for managing disk files.
dbcreator	Members can create, alter, drop, and restore any database.
public	Every SQL Server login belongs to the **public** server role. When a server principal has not been granted or denied specific permissions on a securable object, the user inherits the permissions granted to public on that object. Only assign public permissions on any object when you want the object to be available to all users. You cannot change membership in public. **Note:public** is implemented differently than other roles, and permissions can be granted, denied, or revoked from the public fixed server roles.

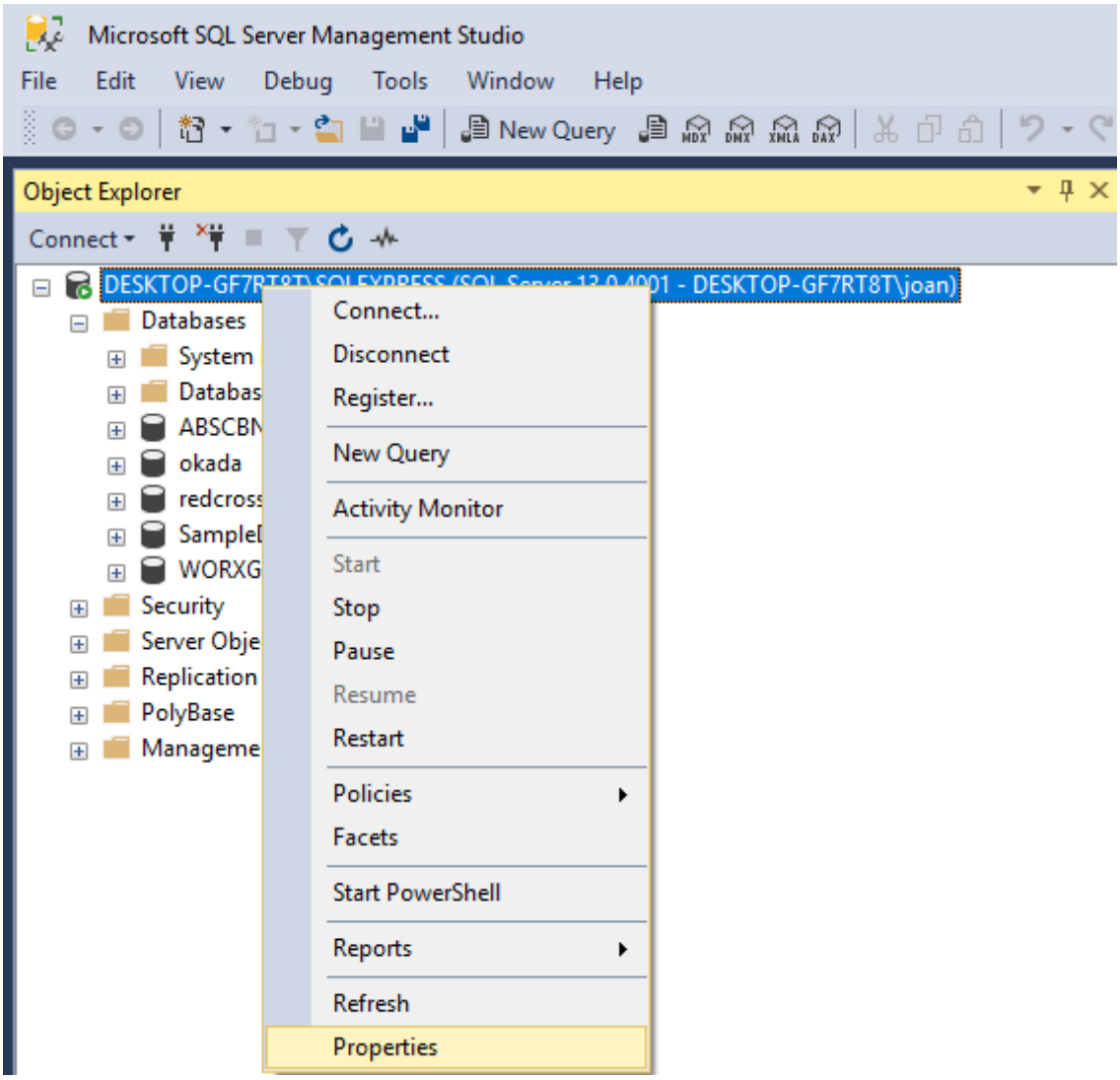
4. Resources:

Personal Computer with installed SQL Server

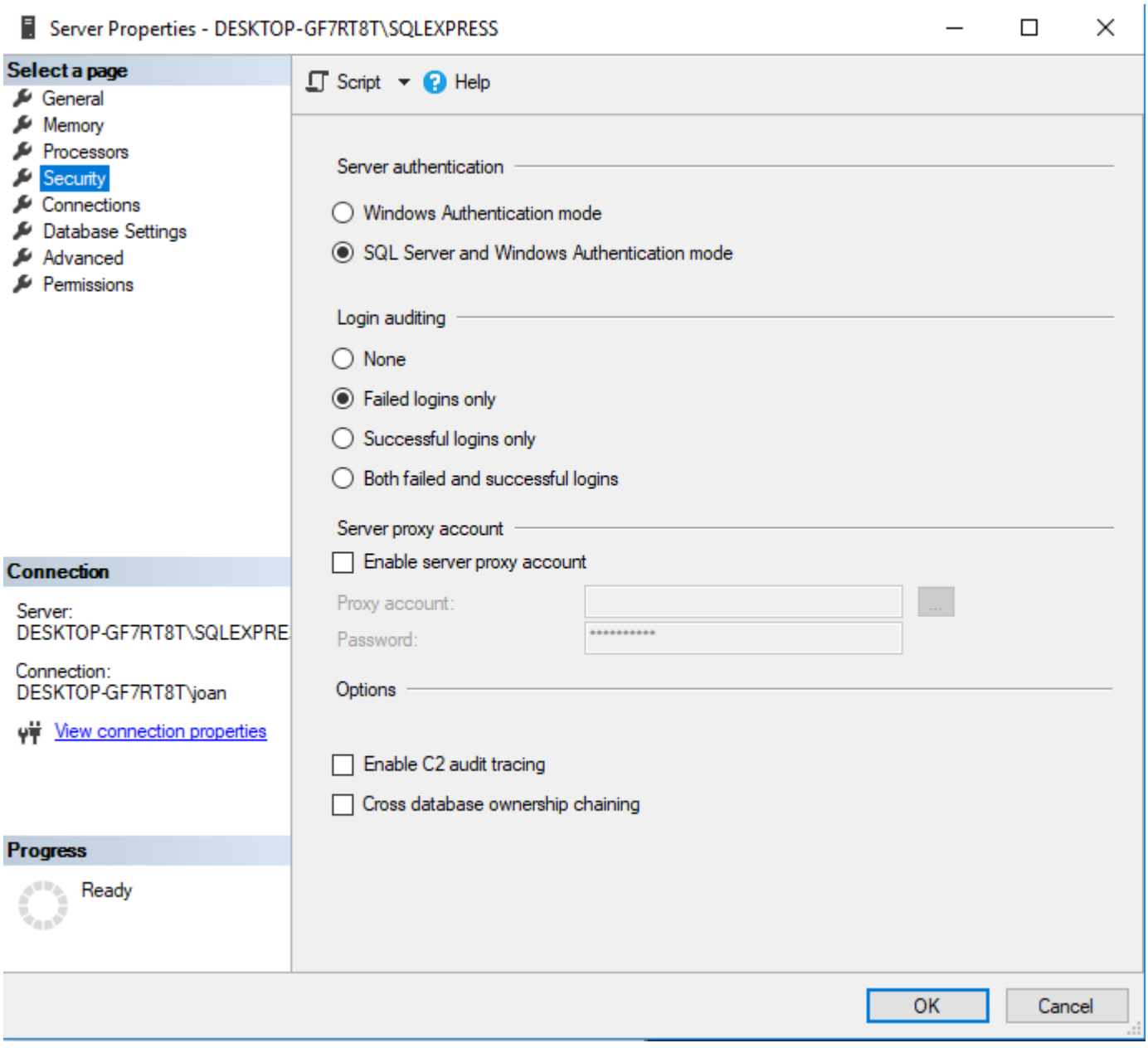
5. Procedure:

To change security authentication mode

1. In SQL Server Management Studio Object Explorer, right-click the server, and then click **Properties**.



2. On the **Security** page, under **Server authentication**, select the new server authentication mode, and then click **OK**.

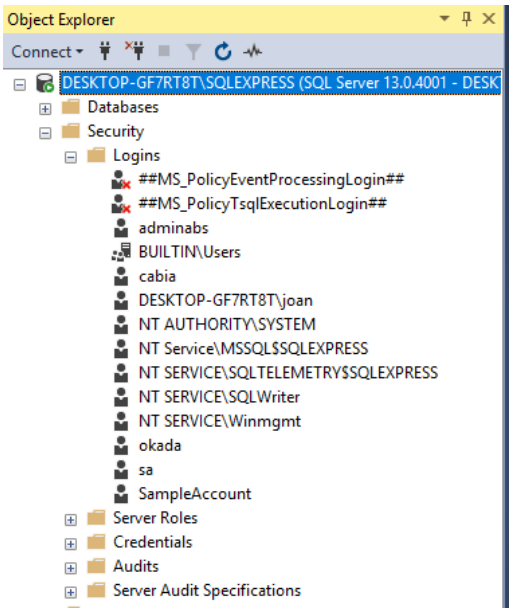


3. In the SQL Server Management Studio dialog box, click **OK** to acknowledge the requirement to restart SQL Server.
4. In Object Explorer, right-click your server, and then click **Restart**. If SQL Server Agent is running, it must also be restarted.

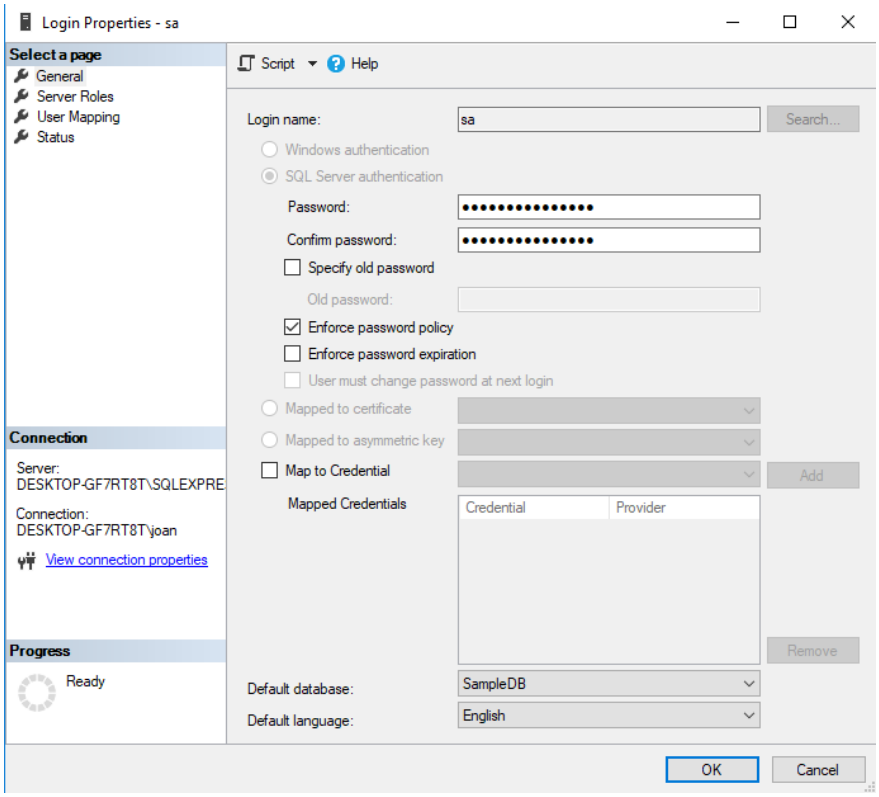
Note: It is recommended using Windows authentication wherever possible. Windows authentication uses a series of encrypted messages to authenticate users in SQL Server. When SQL Server logins are used, SQL Server login names and passwords are passed across the network, which makes them less secure.

To enable the sa login

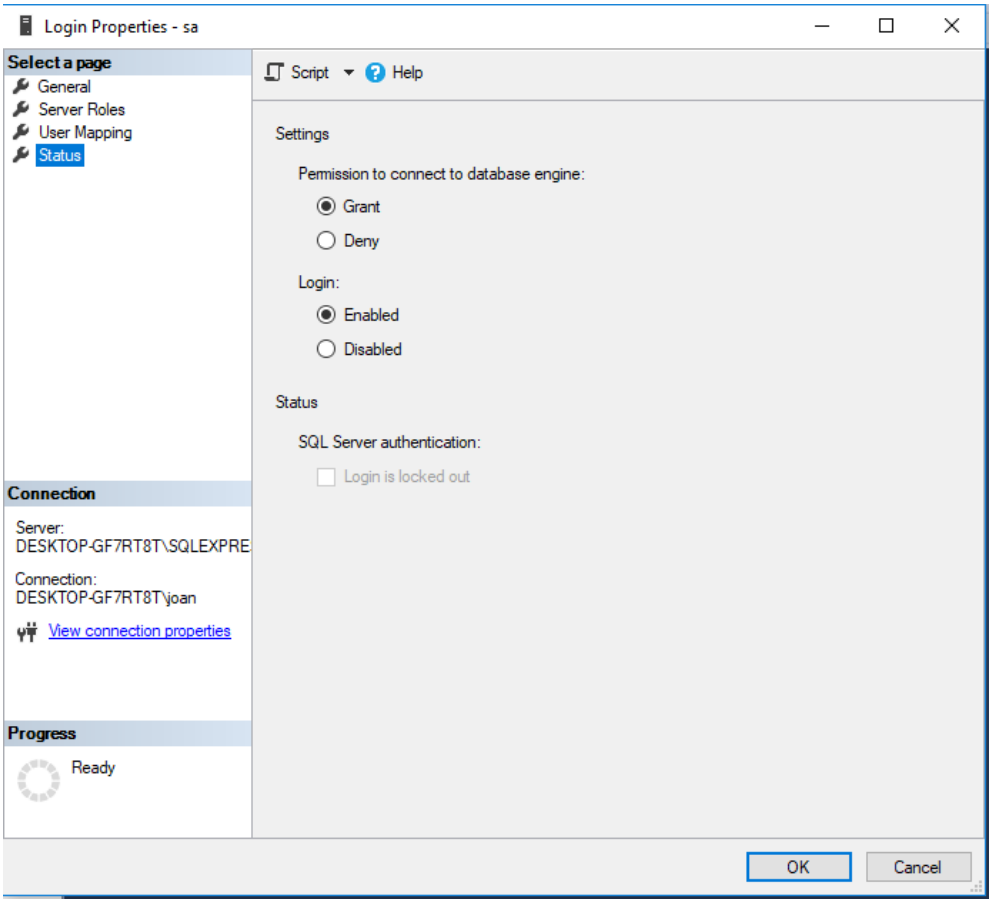
Step 1. In Object Explorer, expand **Security**, expand Logins, right-click **sa**, and then click **Properties**.



Step 2. On the **General** page, you might have to create and confirm a password for the login.

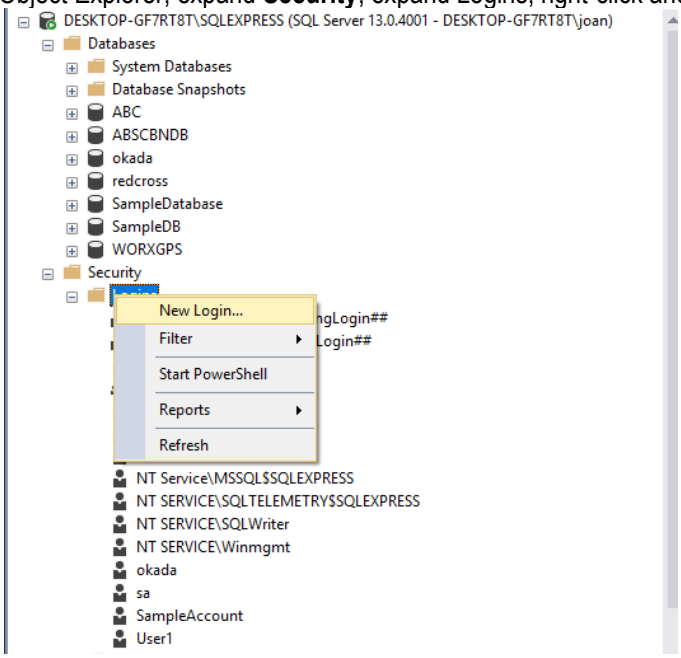


Step 3. On the **Status** page, in the **Login** section, click **Enabled**, and then click **OK**.



To create a user login

Step 1. Object Explorer, expand **Security**, expand Logins, right-click and then choose new Login.



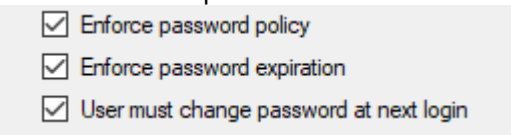
Step 2. On the **General** page, type the login name and the authentication. Type the password.

Note; If you select SQL server authentication, you can apply any of the three options for the password.

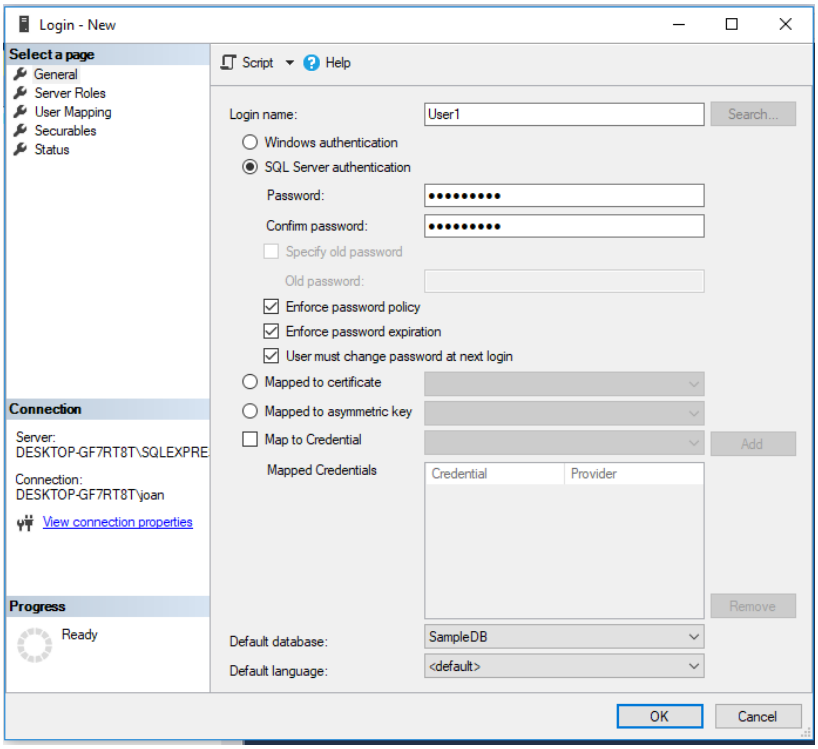
Choose a default database and language. Click OK.

For this activity, use the following:

Login name: User1
Password: user123456
SQL Server Authentication
Check the three options under SQL server authentication

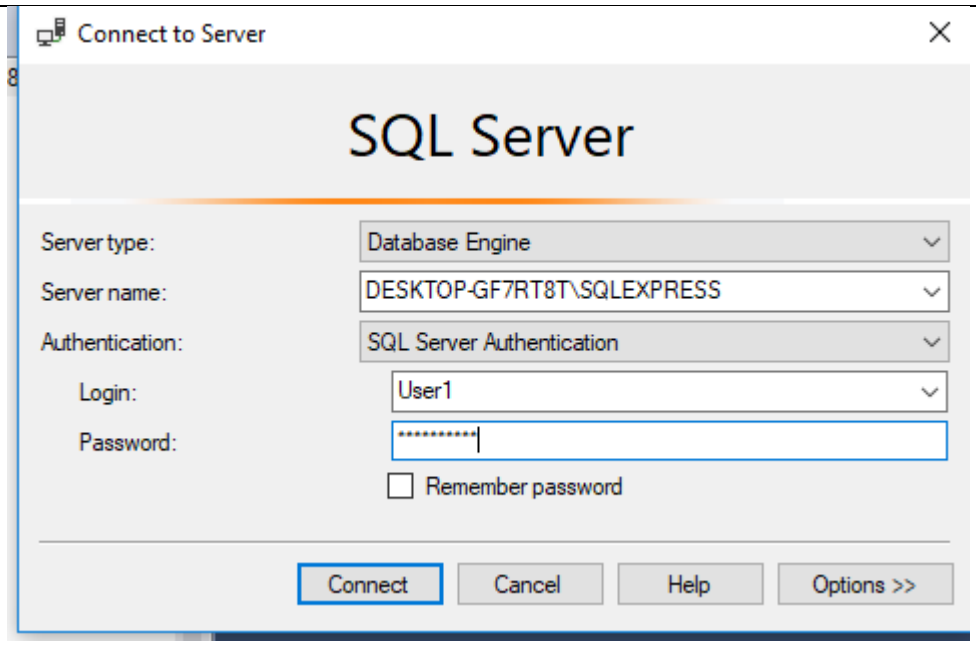


Choose master as default database.



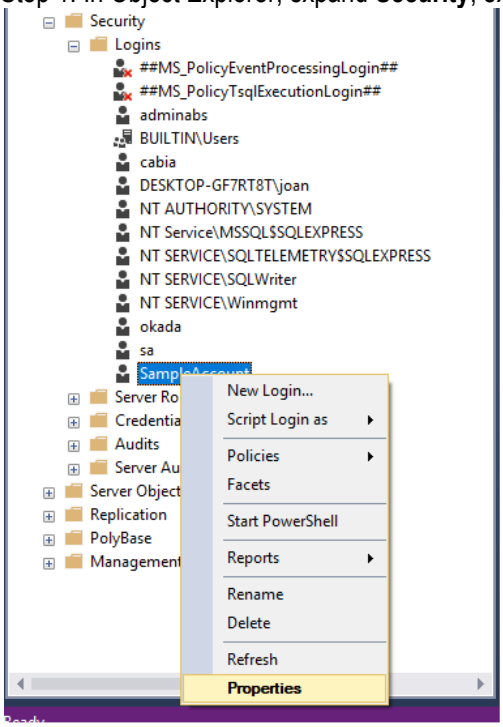
Have you successfully connected to the database engine?(Yes / No). _____
If No, Explain the procedures to troubleshoot.
_____

Step 3. Disconnect and reconnect the SQL server database engine to test the User1 user login.



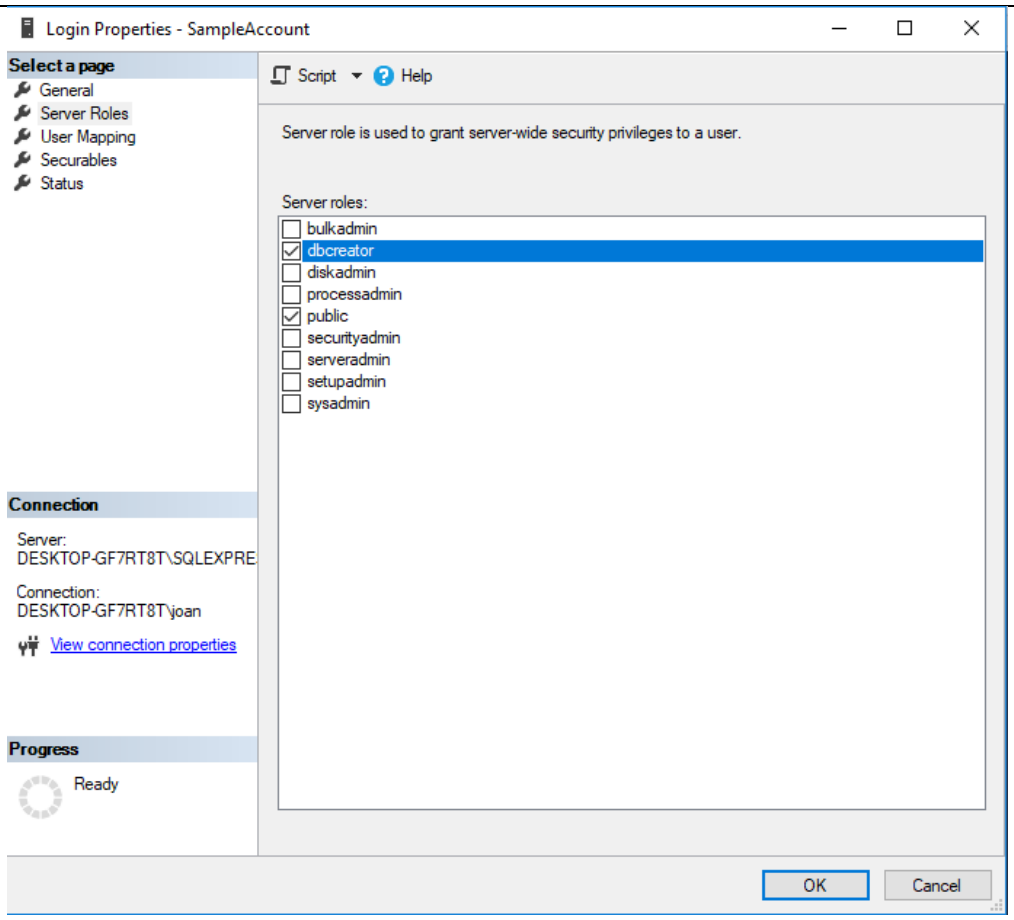
To change the server role of a user

Step 1. In Object Explorer, expand **Security**, expand Logins, right-click a desired user, and then click **Properties**.

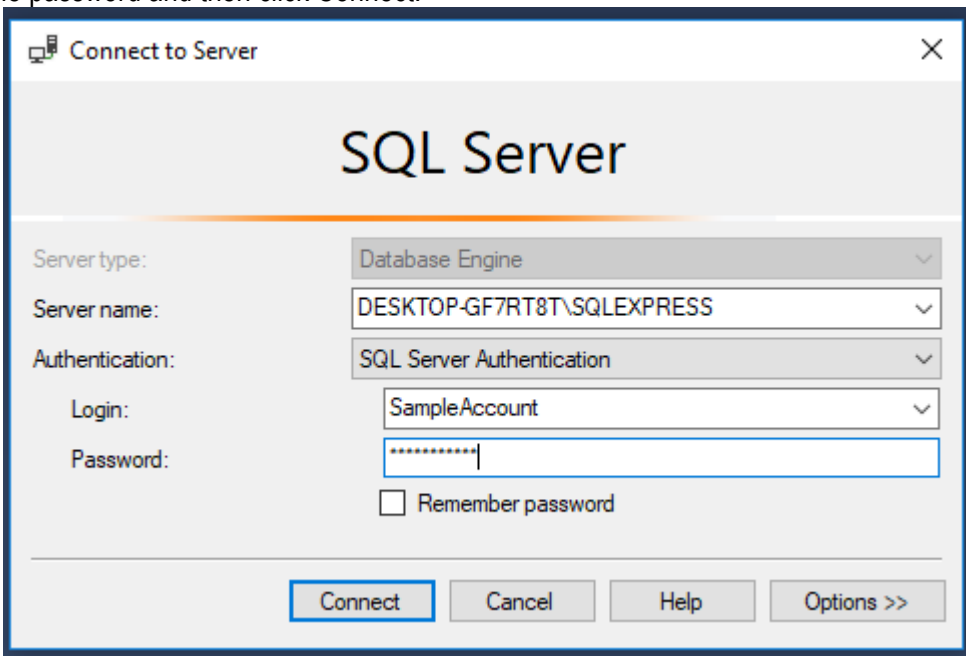


Step 2. On the **Server Roles** page, check the appropriate server role according to the permission allowed to the user and then click OK.

Use dbcreator for this activity.



Step 3. Disconnect and reconnect to the SQL server. Choose the user account you previously changed as login. Type the password and then click Connect.



Step 4. Create a new database SampleDB.

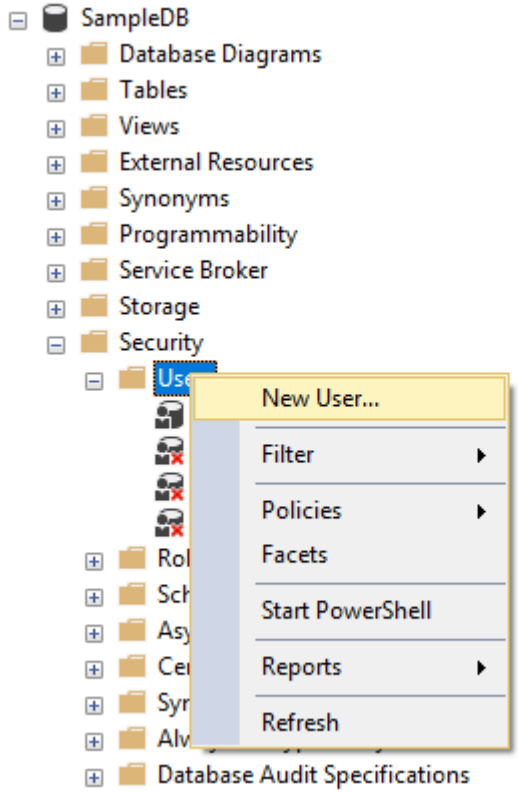
Have you successfully created a database? (Yes / No). _____

If Yes, explain why. _____

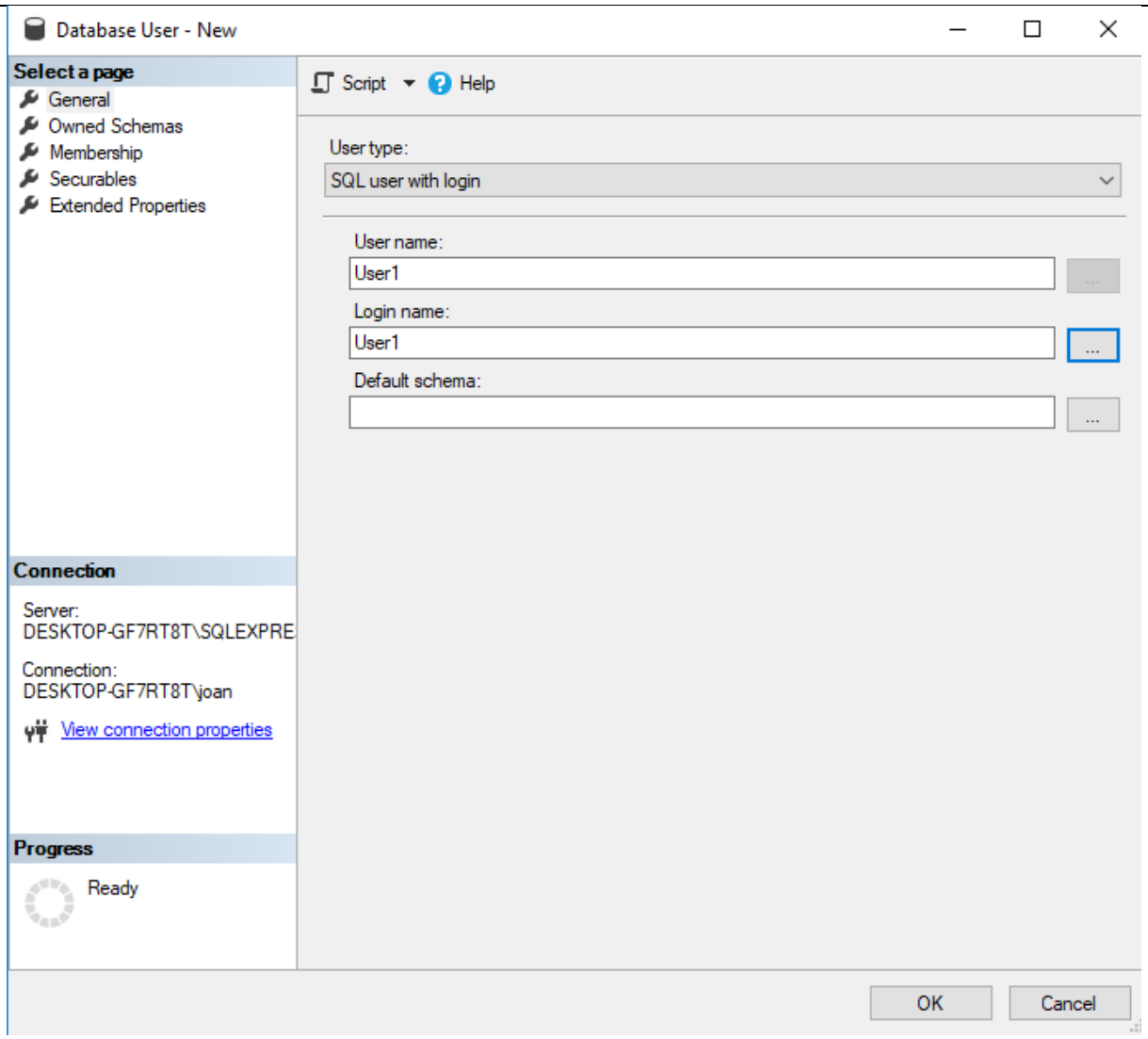
Step 5. Create a new user login.
Have you successfully created a user login? (Yes / No). _____
If No, explain why. _____

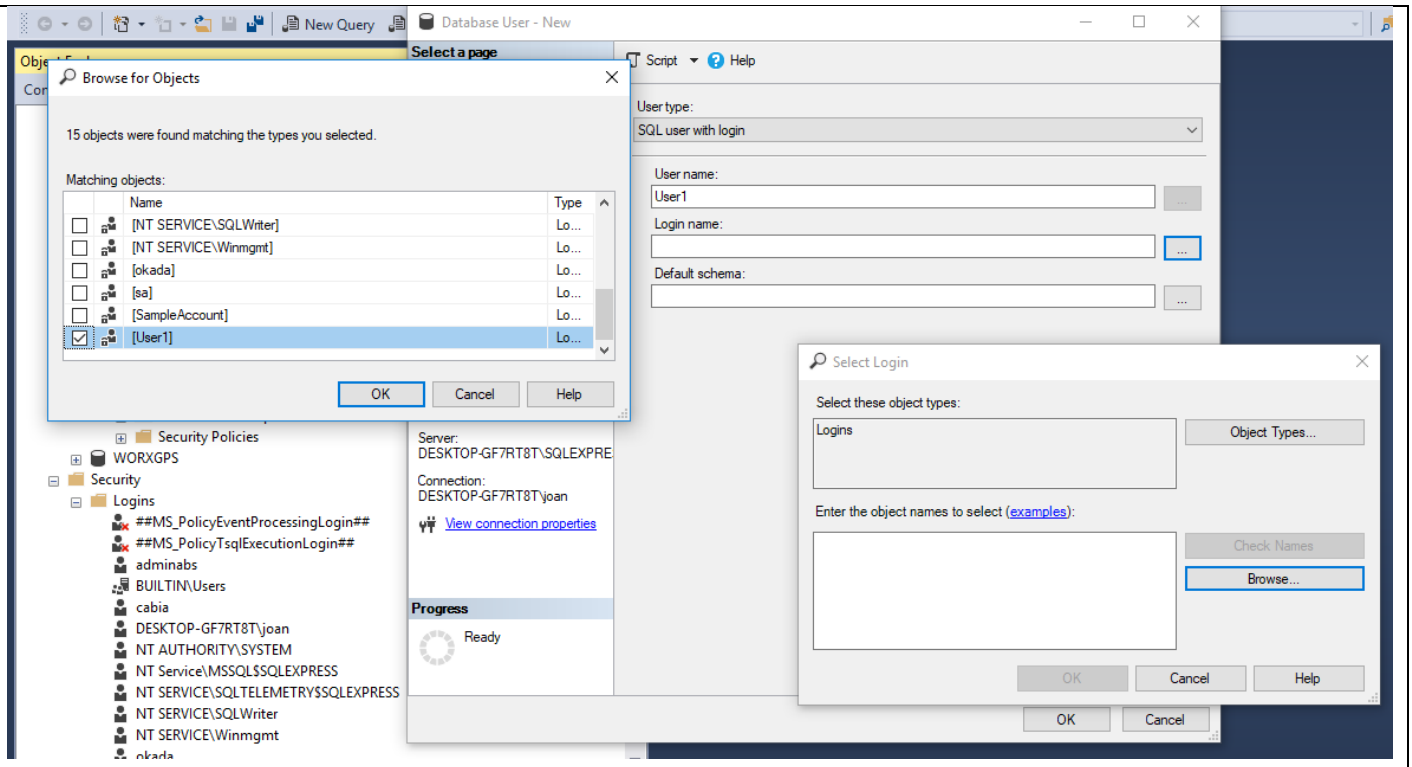
To create a database user

Step 1. In Object Explorer, expand the Databases folder.
Step 2. Expand the database in which to create the new database user.
Step 3. Right-click the Security folder, point to New and Select User



Step 4. On the General page of the Database User- New window, choose one of the user type options from the User type list.
For this activity, use the following options:
User type: SQL Server with login
User name: User1
Login name: User1

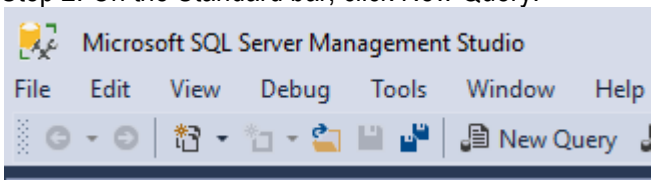

Database User - New
Select a page
General
Owned Schemas
Membership
Securables
Extended Properties
Script
Help
User type:
SQL user with login
User name:
User1
Login name:
User1
Default schema:
Connection
Server:
DESKTOP-GF7RT8T\SQLEXPRE
Connection:
DESKTOP-GF7RT8T\joan
View connection properties
Progress
Ready
OK
Cancel



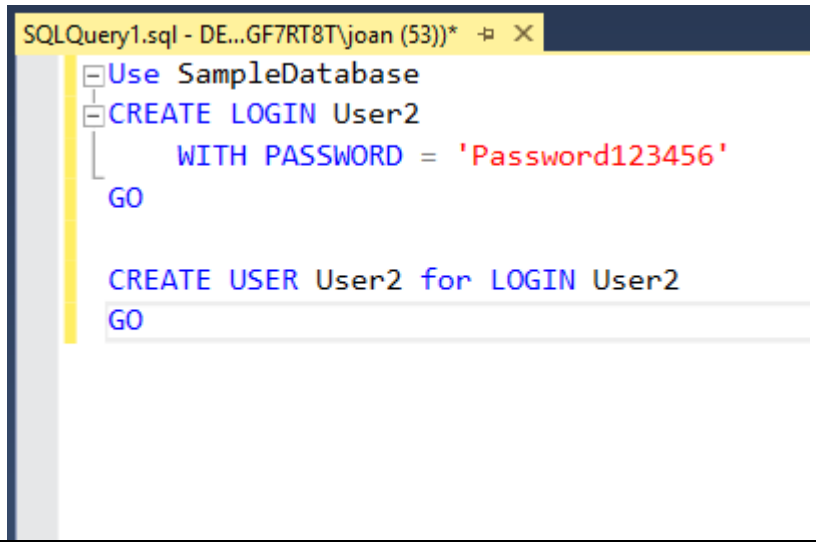
To create a user using T-SQL

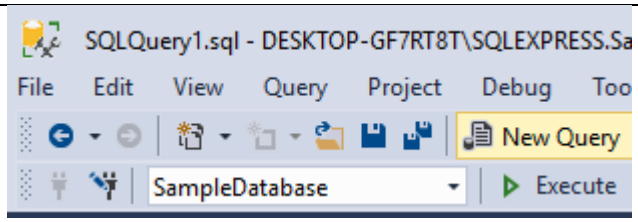
Step 1. In the Object Explorer, connect to an instance of Database Engine.

Step 2. On the Standard bar, click New Query.

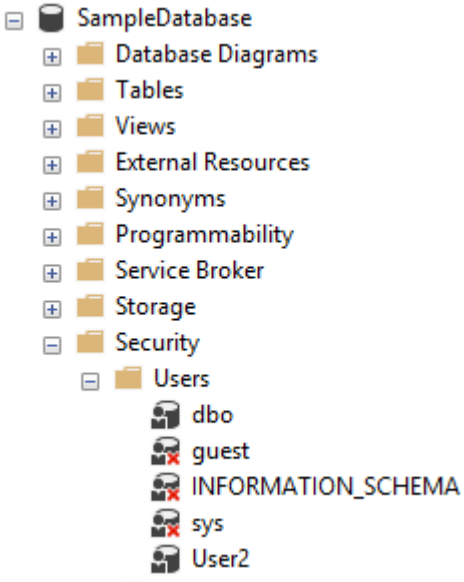


Step 3. Copy and paste the following example into the query window and click execute.





Step 4. Verify the database user and login. In Object Explorer, expand SampleDatabase. Choose Security and expand Users. Check if User2 exists.



6. Database Output

Copy screenshot(s) of your database after completing the procedures provided in Part 5.

7. Supplementary Activity

Do the following tasks and copy screenshot(s) of your output.

1. Create a desired SQL user with login. Make sure that the user login can create another database user and login. Choose a strong password.
2. Connect to the database engine using the created user login.
3. Create a database CarinderiaDB.
4. Create a user peter that can only create, modify and delete database.
5. Connect using user peter and delete CarinderiaDB.
6. Disable the user sa login.
7. Change the authentication to Windows only.
8. Connect to the database engine using the Windows authentication.
9. Create a desired SQL user with login using T-SQL.

8. Conclusion

9. Assessment (Rubric for Laboratory Performance):

Activity No. 9	
Protecting Data by Encryption	
Course Code: CPE011	**Program:**
Course Title: Database Management System	**Date Performed:**
Section:	**Date Submitted:**
Name:	**Instructor:**

1. Objective(s):

This activity aims to implement encryption to protect data.

2. Intended Learning Outcomes (ILOs):

The students should be able to:
2.1. Demonstrate encryption and decryption of a column or row level data.
2.2. Create SQL server master key to perform encryption

3. Discussion :

Encryption is the process of obfuscating data by the use of a key or password. This can make the data useless without the corresponding decryption key or password. Encryption does not solve access control problems. However, it enhances security by limiting data loss even if access controls are bypassed.

SQL Server uses encryption keys to help secure data, credentials, and connection information that is stored in a server database. SQL Server has two kinds of keys: *symmetric* and *asymmetric*. Symmetric keys use the same password to encrypt and decrypt data. Asymmetric keys use one password to encrypt data (called the *public* key) and another to decrypt data (called the *private* key).

In SQL Server, encryption keys include a combination of public , private, and symmetric keys that are used to protect sensitive data. The symmetric key is created during SQL Server initialization when you first start the SQL Server instance. The key is used by SQL Server to encrypt sensitive data that is stored in SQL Server. Public and private keys are created by the operating system and they are used to protect the symmetric key. A public and private key pair is created for each SQL Server instance that stores sensitive data in a database.

There are two different kinds of encryptions available in SQL Server:

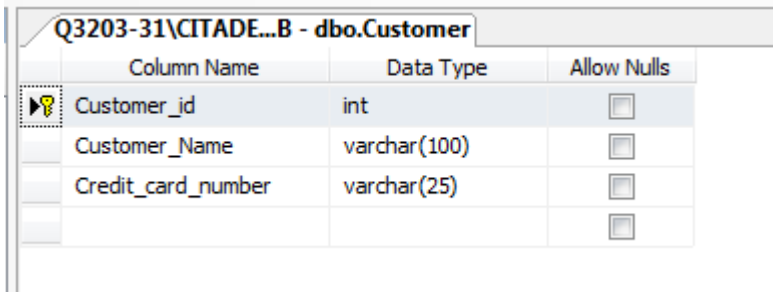
- **Database Level** – This level secures all the data in a database. However, every time data is written or read from database, the whole database needs to be decrypted. This is a very resource-intensive process and not a practical solution.
- **Column (or Row) Level** – This level of encryption is the most preferred method. Here, only columns containing important data should be encrypted; this will result in lower CPU load compared with the whole database level encryption. If a column is used as a primary key or used in comparison clauses (WHERE clauses, JOIN conditions) the database will have to decrypt the whole column to perform operations involving those columns.

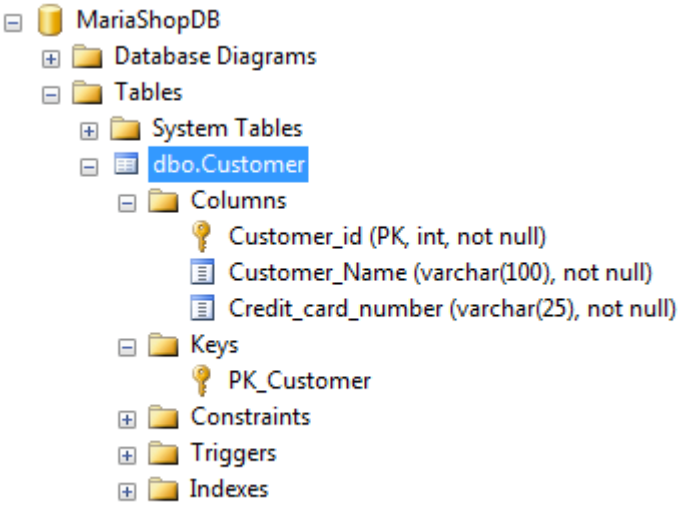
4. Resources:

Personal Computer with installed SQL Server

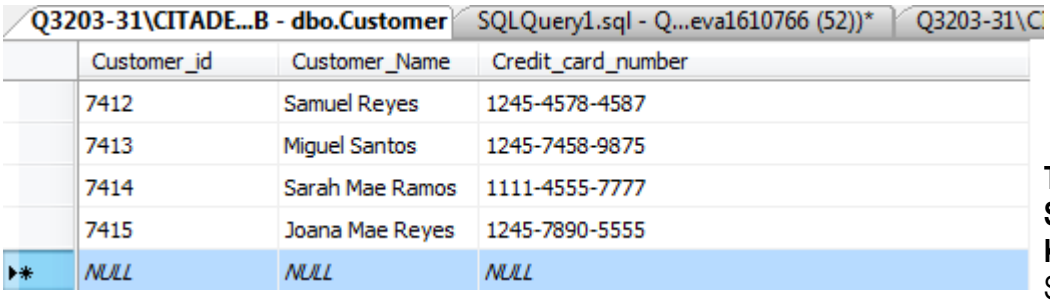
5. Procedure:

Step 1. Create MariaShopDB database.



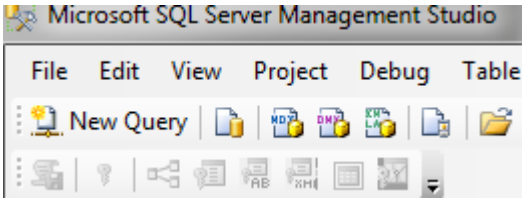


Step 2. Input the following data in the Customer table.

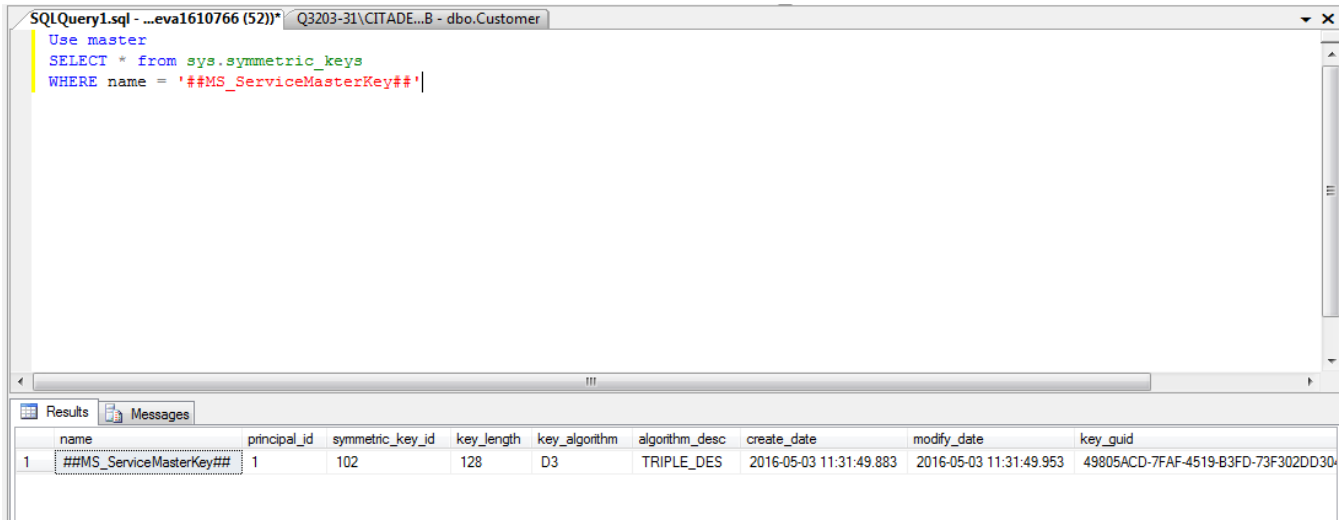


To check SQL Server Master Key

Step 1. In **Object Explorer**, connect to an instance of Database Engine. On the Standard bar, click **New Query**.



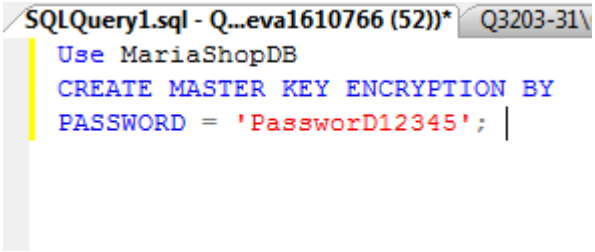
Step 2. Copy and paste the following example into the query window and click **Execute**.



To create SQL Server Database Master Key

Step 1. In **Object Explorer**, connect to an instance of Database Engine. On the Standard bar, click **New Query**.

Step 2. Copy and paste the following example into the query window and click **Execute**.



To encrypt a column of data using simple symmetric encryption

Step 1. In **Object Explorer**, connect to an instance of Database Engine. On the Standard bar, click **New Query**.

Step 2. Copy and paste the following example into the query window and click **Execute.**

```
USE MariaShopDB;

GO

CREATE CERTIFICATE Sales18
   WITH SUBJECT = 'Customer Credit Card Numbers';
GO

CREATE SYMMETRIC KEY CreditCards_Key18
    WITH ALGORITHM = AES_256
    ENCRYPTION BY CERTIFICATE Sales18;
GO


ALTER TABLE Customer
    ADD CardNumber_Encrypted varbinary(128);
GO

-- Open the symmetric key with which to encrypt the data.
OPEN SYMMETRIC KEY CreditCards_Key18
   DECRYPTION BY CERTIFICATE Sales18;

-- Encrypt the value in column Credit_card_number using the
-- symmetric key CreditCards_Key1718.
-- Save the result in column CardNumber_Encrypted.
UPDATE Customer
SET CardNumber_Encrypted = EncryptByKey(Key_GUID('CreditCards_Key18')
    , Credit_card_number, 1, HashBytes('SHA1', CONVERT( varbinary
    , Customer_id)));
GO
Select * from Customer
GO
```

Results | Messages

	Customer_id	Customer_Name	Credit_card_number	CardNumber_Encrypted
1	7412	Samuel Reyes	1245-4578-4587	0x00E32D8E436E5F4A83F053C1B3B27FA901000000A78BF06...
2	7413	Miguel Santos	1245-7458-9875	0x00E32D8E436E5F4A83F053C1B3B27FA90100000015C86F7...
3	7414	Sarah Mae Ramos	1111-4555-7777	0x00E32D8E436E5F4A83F053C1B3B27FA9010000003A66FB...
4	7415	Joana Mae Reyes	1245-7890-5555	0x00E32D8E436E5F4A83F053C1B3B27FA901000000FB36AF3...

To decrypt a column of data using simple symmetric encryption

Step 1. In **Object Explorer**, connect to an instance of Database Engine. On the Standard bar, click **New Query**.

Step 2. Copy and paste the following example into the query window and click **Execute.**

SQLQuery5.sql - Q...eva1610766 (58))* | SQLQuery3.sql - Q...ueva1610766 (54)) | SQLQuery2.sql - Q...ueva161076

```
USE MariaShopDB;

GO


-- Open the symmetric key with which to encrypt the data.
OPEN SYMMETRIC KEY CreditCards_Key18
   DECRYPTION BY CERTIFICATE Sales18;

SELECT Customer_id, CardNumber_Encrypted
 AS 'Encrypted card number', CONVERT(varchar,
 DecryptByKey(CardNumber_Encrypted, 1 ,
  HashBytes('SHA1', CONVERT(varbinary, Customer_id))))
  AS 'Decrypted card number' FROM Customer;
```

Results | Messages

	Customer_id	Encrypted card number	Decrypted card number
1	7412	0x00E32D8E436E5F4A83F053C1B3B27FA901000000A78BF06...	1245-4578-4587
2	7413	0x00E32D8E436E5F4A83F053C1B3B27FA90100000015C86F7...	1245-7458-9875
3	7414	0x00E32D8E436E5F4A83F053C1B3B27FA9010000003A66FB...	1111-4555-7777
4	7415	0x00E32D8E436E5F4A83F053C1B3B27FA901000000FB36AF3...	1245-7890-5555

6. Database Output

Copy screenshot(s) of your database after completing the procedures provided in Part 5.

7. Supplementary Activity

Do the following tasks and copy screenshot(s) of your output.

1. Modify Customer table of MariaShopDB. Add SSS number . Use varchar(50) as data type.
2. Encrypt the SSS number column.
3. Decrypt the SSS number column.

8. Conclusion

9. Assessment (Rubric for Laboratory Performance):

Activity No. 10	
Transact SQL	
Course Code: CPE011	**Program:**
Course Title: Database Management System	**Date Performed:**
Section:	**Date Submitted:**
Name:	**Instructor:**

1. Objective(s):

This activity aims to design and implement Transact SQL to the databases

2. Intended Learning Outcomes (ILOs):

The students should be able to:
2.1 Develop transact SQ L to create database, tables and view objects..
2.2 Develop transact SQL to manipulate data and control user privilege.
2.2 Execute and implement transact SQL

3. Discussion :

Transact-SQL (T-SQL) is Microsoft's and extension to the SQL (Structured Query Language). Transact-SQL is central to using Microsoft SQL Server and used to interact with relational databases. T-SQL expands on the SQL standard to include procedural programming, local variables, various support functions for string processing, date processing, mathematics, etc. and changes to the DELETE and UPDATE statements. All applications that communicate with an instance of SQL Server do so by sending Transact-SQL statements to the server, regardless of the user interface of the application.

There are different types of statements or activities that can be performed I in Transact SQL.

- Data Definition Language (DDL) is used to define data structures. Use these statements to create, alter, or drop data structures in an instance of SQL Server.
- Data Manipulation Language (DML) is used to retrieve and work with data. Use these statements to add, modify, query, or remove data from a SQL Server database.
- Data Control Language is used to create roles, permissions, and referential integrity as well it is used to control access to database objects.

  Examples: GRANT, REVOKE statements

  GRANT – allows users to read/write on certain database objects

  REVOKE – keeps users from read/write permission on database objects

4. Resources:

Personal Computer with installed SQL Server

5. Procedure:

Data Definition Language using T-SQL

Step 1: In a Query Editor window, type but do not execute the following code:

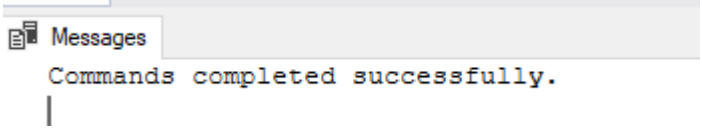
```
if exists (select name from sys.databases where name='ABCPayroll')
drop database ABCPayroll
go
create database ABCPayroll
go
use ABCPayroll
go

if exists(select name from sys.tables where name='department')
drop table department
go
create table department
(
 departmentid int primary key,
 department varchar(50) not null
)

if exists (select name from sys.tables where name='employeeinfo')
drop table employeeinfo
 go
create table employeeinfo
(
 employeeid char(7) not null primary key,
 firstname varchar(50) not null,
 middlename varchar(50),
 lastname varchar(50) not null,
 dateemployment datetime not null,
 departmentid int not null


)
```

Note :

The keyword GO separates statements when more than one statement is submitted in a single batch. GO is optional when the batch contains only one statement.

Step 2: Save the T-SQL file as createABCPayroll.sql.

Step 3: In Query Editor, press **F5** to execute the statement.



The ABCPayroll database and its tables will be displayed in the object explorer.

Data Control Language using T-SQL

Step 1: Create the following database user in ABCPayroll database.

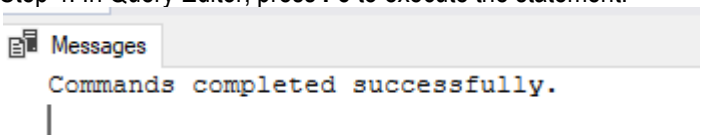
- employee12345
- admin12345

Step 2: In a Query Editor window, type but do not execute the following code:

```
use ABCPayroll
grant select on employeeinfo to employee12345
grant insert on employeeinfo to employee12345
grant update on employeeinfo to employee12345
deny  delete on employeeinfo to employee12345
deny  delete  on department to employee12345
go
grant select, insert, update, delete on employeeinfo to admin12345
grant select, insert, update, delete on department to admin12345

go
revoke update on employeeinfo to employee12345
```

Step 3: Save the T-SQL file as securityABCPayroll.sql

Step 4: In Query Editor, press **F5** to execute the statement.

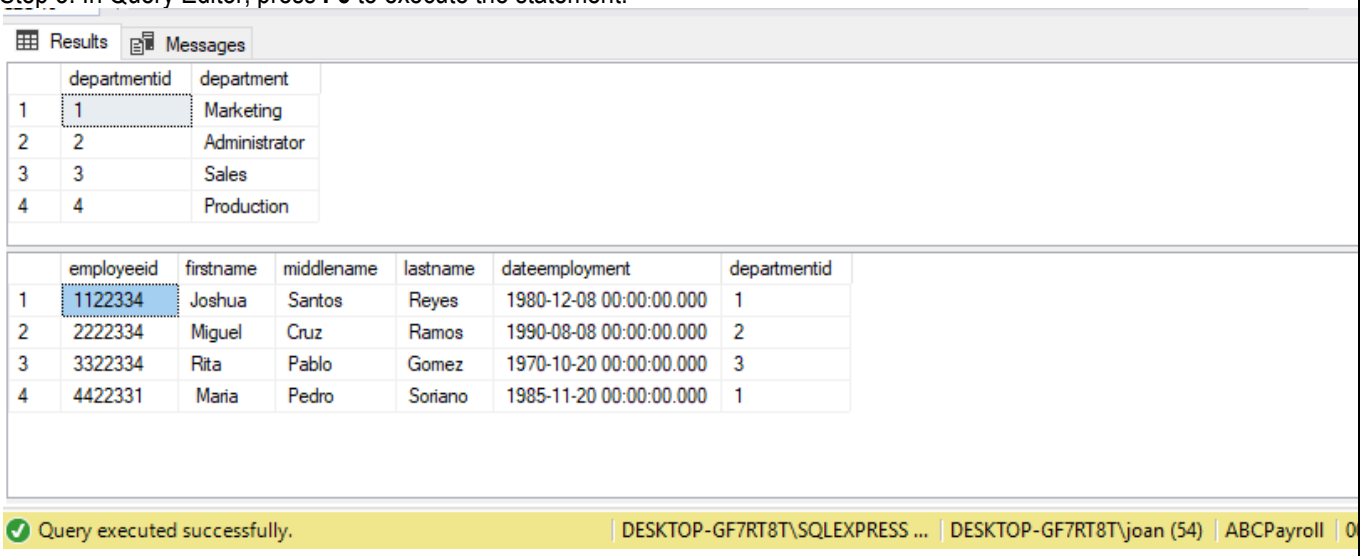


Data Manipulation Language using T-SQL

Step 1: In a Query Editor window, type but do not execute the following code:

```
use ABCPayroll
if exists(select name from sys.tables where name='department')
delete from department
go
insert department values(1,'Marketing')
insert department values(2,'Administrator')
insert department values(3,'Sales')
insert department values(4,'Production')
go
select * from department
go
if exists (select name from sys.tables where name='employeeinfo')
delete from employeeinfo
insert employeeinfo values(1122334, 'Joshua', 'Santos', 'Reyes', '1980-12-08', 1)
insert employeeinfo values(4422331, ' Maria', 'Pedro', 'Soriano', '1985-11-20', 1)
insert employeeinfo values(2222334, 'Miguel', 'Cruz', 'Ramos', '1990-08-08', 2)
insert employeeinfo values(3322334, 'Rita', 'Pablo', 'Gomez', '1970-10-20', 3)
go
select * from employeeinfo
```

Step 2: Save the T-SQL file as dataABCPayroll.sql
Step 3: In Query Editor, press **F5** to execute the statement.



Create View using T-SQL

Step 1: Execute the following statement to create a very simple view that executes a select statement, and returns the information of the employees under Marketing department.

```
CREATE VIEW vwEmployeeMarketing AS
SELECT *
FROM employeeinfo
WHERE departmentid=1
```

Step 2: Test the view. Views are treated just like tables. Use a SELECT statement to access a view

```
select * from vwEmployeeMarketing
```

121 %

Results | Messages

	employeeid	firstname	middlename	lastname	dateemployment	departmentid
1	1122334	Joshua	Santos	Reyes	1980-12-08 00:00:00.000	1
2	4422331	Maria	Pedro	Soriano	1985-11-20 00:00:00.000	1

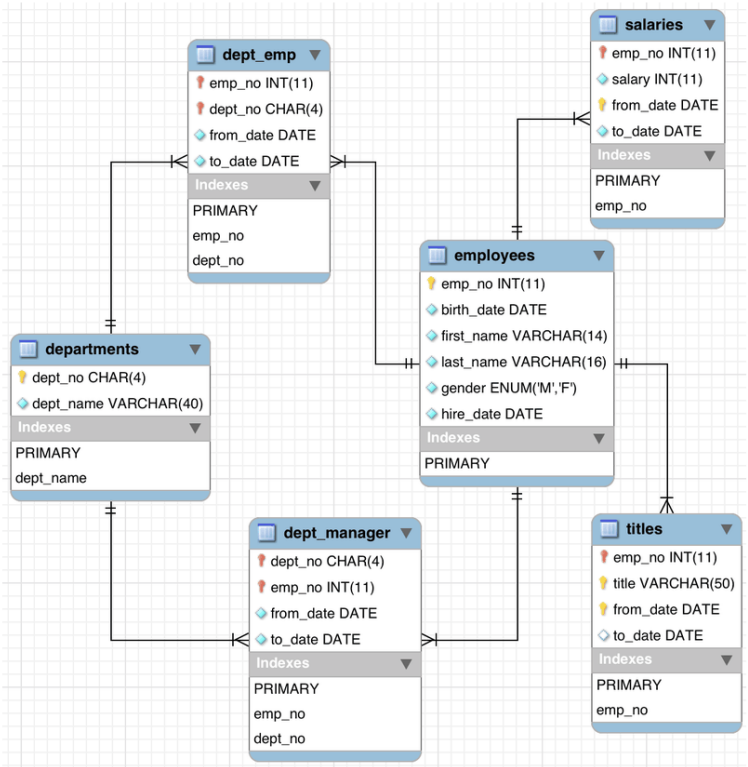
6. Database Output

Copy screenshot(s) of your database after completing the procedures provided in Part 5.

1. Data Definition Language
2. Data Control Language
3. Data Manipulation Language
4. View

7. Supplementary Activity

1. Create and execute Employees.sql to develop the Employees database. Use the given ERD to create Employees.sql



2. Create and execute securityEmployees.sql to assign the following users to perform specific operations
   a. tipqcfaculty – select and insert only
   b. tipqcstudent – select only
   c. tipqcadmin – select, delete, insert and update
3. Create and execute insertdataEmployees.sql to insert 20 employees (10 Female and 10 Male employees).
4. Create and execute viewEmpInfoFemale.sql to create a view of the female employees.

8. Conclusion

9. Assessment (Rubric for Laboratory Performance):

Activity No. 11	
Stored Procedures	
Course Code: CPE011	**Program:**
Course Title: Database Management System	**Date Performed:**
Section:	**Date Submitted:**
Name:	**Instructor:**

1. Objective(s):

This activity aims to create and execute stored procedures in databases

2. Intended Learning Outcomes (ILOs):

The students should be able to:
2.1 Create stored procedures using Management Studio and T-SQL.
2.2Apply input parameters and return output data in stored procedures.
2.3 Implement and execute stored procedures

3. Discussion :

A stored procedure in SQL Server is a group of one or more Transact-SQL statements. Stored procedures are similar to procedures in other programming languages in that they can:

- Accept input parameters and return multiple values in the form of output parameters to the calling procedure or batch.
- Contain programming statements that perform operations in the database, including calling other procedures.
- Return a status value to a calling procedure or batch to indicate success or failure (and the reason for failure).

Benefits of Using Stored Procedures

- Reduced server/client network traffic
- Stronger security
- Procedures can be encrypted, helping to obfuscate the source code
- Reuse of code
- Easier maintenance
- Improved performance

Specify Parameters

By specifying procedure parameters, calling programs are able to pass values into the body of the procedure. Those values can be used for a variety of purposes during procedure execution. Procedure parameters can also return values to the calling program if the parameter is marked as an OUTPUT parameter.

A procedure can have a maximum of 2100 parameters; each assigned a name, data type, and direction. Optionally, parameters can be assigned default values.

4. Resources:

Personal Computer with installed SQL Server

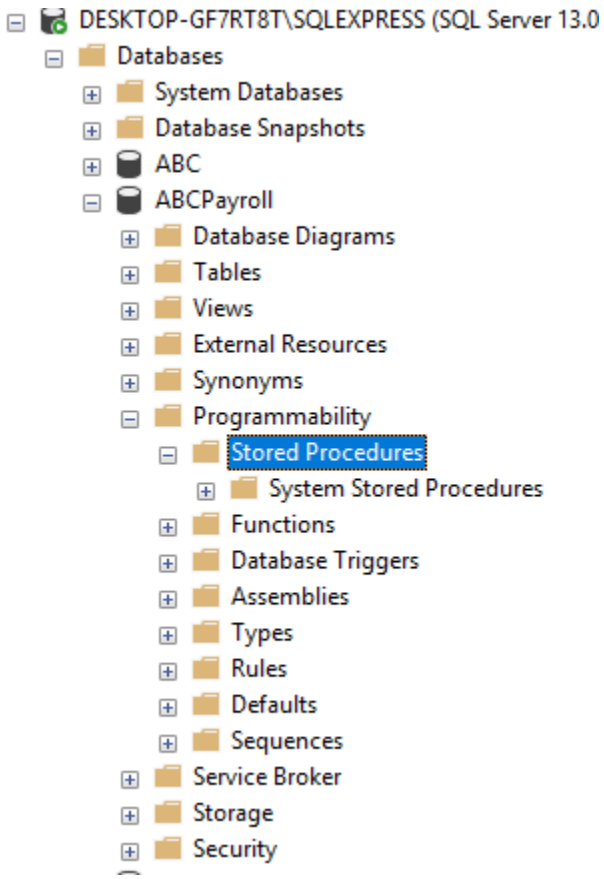
5. Procedure:

Create a Stored Procedure

A. Using SQL Management Studio

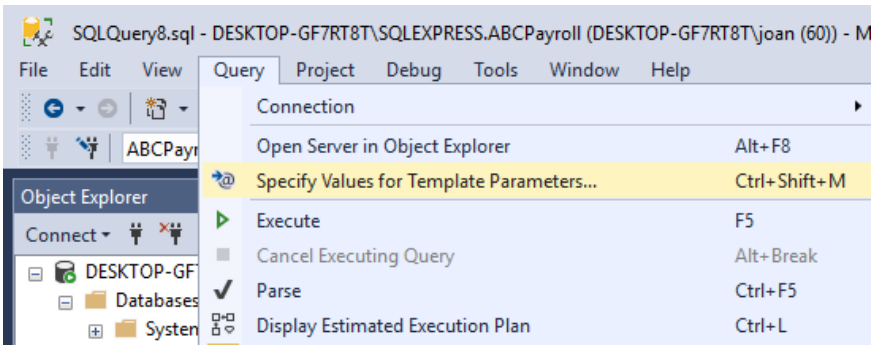
Step 1: In **Object Explorer**, connect to an instance of Database Engine and then expand that instance.

Step 2: Expand **Databases**, expand the **ABCPayroll** database, and then expand **Programmability**.

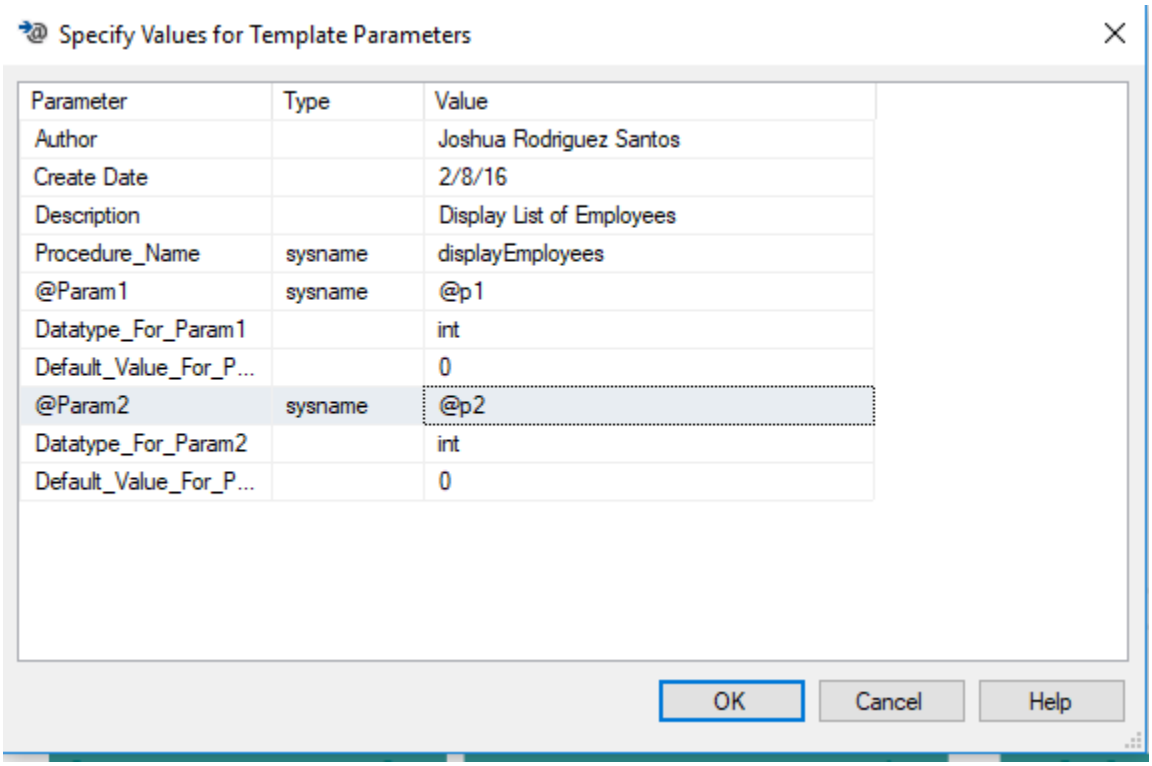


Step 3: Right-click **Stored Procedures**, and then click **New Stored Procedure**.

Step 4: On the **Query** menu, click **Specify Values for Template Parameters**.



In the **Specify Values for Template Parameters** dialog box, enter the following values for the parameters shown.

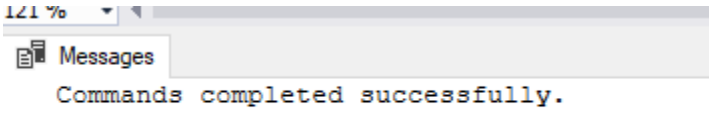


Step 5: Click **OK**.

Step 6: Modify the displayEmployees stored procedure.

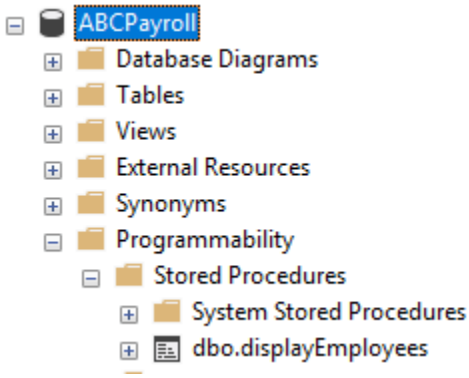
```
-- ================================================
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:      Joshua Rodriguez Santos
-- Create date: 2/8/16
-- Description: Display List of Employees
-- =============================================
CREATE PROCEDURE displayEmployees
    -- Add the parameters for the stored procedure here

AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for procedure here
    SELECT * from employeeinfo
END
GO
```

Step 7: Press F5 or Click Execute to save the stored procedure.



Note:

Refresh the database to view the newly created stored procedure.



B. Using T-SQL

Step 1: In **Object Explorer**, connect to an instance of Database Engine.

Step 2: From the **File** menu, click **New Query**.

Step 3: Copy and paste the following example into the query window and click **Execute**. This example creates the same stored procedure as above using a different procedure name.

```
USE ABCPayroll;
GO
CREATE PROCEDURE uspDisplayEmployees

AS

    SET NOCOUNT ON;
    SELECT * from employeeinfo
GO
```

Execute a Stored Procedure

Step 1: In **Object Explorer**, connect to an instance of Database Engine.

Step 2: From the **File** menu, click **New Query**.

Step 3: :Copy and paste the following example into the query window and click **Execute.** This example execute the stored procedure uspDisplayEmployees.

```
USE ABCPayroll
GO
EXEC dbo.uspDisplayEmployees
```

Results | Messages

employeeid	firstname	middlename	lastname	dateemployment	departmentid
1122334	Joshua	Santos	Reyes	1980-12-08 00:00:00.000	1
2222334	Miguel	Cruz	Ramos	1990-08-08 00:00:00.000	2
3322334	Rita	Pablo	Gomez	1970-10-20 00:00:00.000	3
4422331	Maria	Pedro	Soriano	1985-11-20 00:00:00.000	1

Modify a Stored Procedure

Step 1: In Object Explorer, connect to an instance of Database Engine and then expand that instance.

Step 2: Expand **Databases**, expand the database in which the procedure belongs, and then expand **Programmability**.

Step 3: Expand **Stored Procedures**, right-click the uspDisplayEmployeesprocedure to modify, and then click **Modify**.

Step 4: Modify the text of the stored procedure.

```
USE [ABCPayroll]
GO
/****** Object:  StoredProcedure [dbo].[uspDisplayEmployees]    Script
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[uspDisplayEmployees]
        @employeeid char(7)
AS

    SET NOCOUNT ON;
    SELECT * from employeeinfo where employeeid=@employeeid
```

Step 5: Save the modifications to the procedure definition, on the **Query** menu, click **Execute**.

Step 6: Create a new query to test the uspDisplayEmployees stored procedure.

```
exec dbo.uspDisplayEmployees @employeeid=1122334
```

121 %

Results | Messages

	employeeid	firstname	middlename	lastname	dateemployment	departmentid
1	1122334	Joshua	Santos	Reyes	1980-12-08 00:00:00.000	1

Specify Parameters

Step 1: In **Object Explorer**, connect to an instance of Database Engine.

Step 2: From the **File** menu, click **New Query**.

Step 3: Copy and paste the following example into the query window and click **Execute**. This example creates the uspDisplayEmployeeperDepartment stored procedure. The uspDisplayEmployeeperDepartment used departmentid as input parameter.

```
USE [ABCPayroll]
GO

CREATE PROCEDURE uspDisplayEmployeeperDepartment
        @departmentid int
AS

    SET NOCOUNT ON;
    SELECT e.employeeid as 'employee id', CONCAT(e.firstname,' ',
    e.middlename,' ', e.lastname) as 'employee name', d.department
    from employeeinfo e
    inner join department d
    on e.departmentid = d.departmentid
    where e.departmentid = @departmentid
```

Step 5: Create a new query to test the uspDisplayEmployeeperDepartment stored procedure.

```
use ABCPayroll
exec dbo.uspDisplayEmployeeperDepartment @departmentid=1
```

121 %

Results | Messages

	employee id	employee name	department
1	1122334	Joshua Santos Reyes	Marketing
2	4422331	Maria Pedro Soriano	Marketing

```
use ABCPayroll
exec dbo.uspDisplayEmployeeperDepartment @departmentid=2
```

121 %

Results | Messages

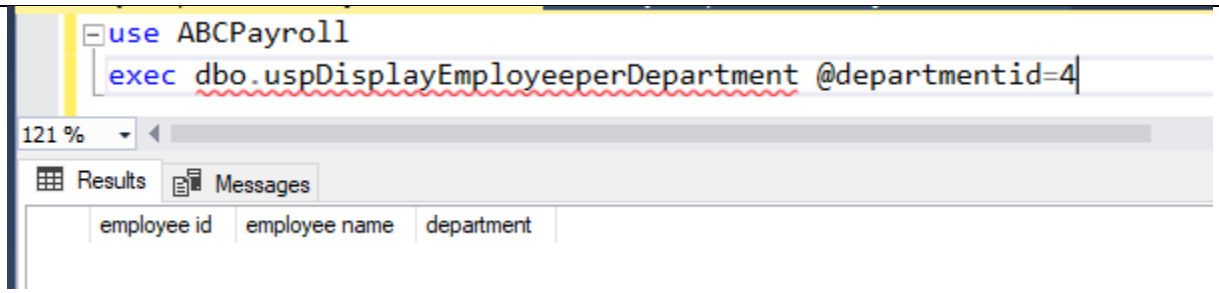
	employee id	employee name	department
1	2222334	Miguel Cruz Ramos	Administrator

```
use ABCPayroll
exec dbo.uspDisplayEmployeeperDepartment @departmentid=3
```

121 %

Results | Messages

	employee id	employee name	department
1	3322334	Rita Pablo Gomez	Sales



Return a Data

Step 1: In **Object Explorer**, connect to an instance of Database Engine.

Step 2: From the **File** menu, click **New Query**.

Step 3: Copy and paste the following example into the query window and click **Execute**. This example creates the uspCountTotalEmployeesperDepartment stored procedure. The uspCountTotalEmployeesperDepartment used total variable as output.

```
USE [ABCPayroll]
GO

CREATE PROCEDURE dbo.uspCountTotalEmployeesperDepartment
         @departmentid int
AS

    SET NOCOUNT ON;
    DECLARE @total int
    SELECT @total = count(*) from employeeinfo
    where departmentid = @departmentid

    IF @total=0
        BEGIN
            PRINT 'There is no existing employee'
            RETURN
        END
    ELSE
        BEGIN
           SELECT * from employeeinfo where departmentid = @departmentid
           PRINT 'Total No. of Employees:' +
           convert(varchar(10),@total)
         END
```

Step 5: Create a new query to test the uspCountTotalEmployeesperDepartmentstored procedure.

```
use ABCPayroll
exec dbo.uspCountTotalEmployeesperDepartment @departmentid=1
```

21 %

Results | Messages

	employeeid	firstname	middlename	lastname	dateemployment	departmentid
1	1122334	Joshua	Santos	Reyes	1980-12-08 00:00:00.000	1
2	4422331	Maria	Pedro	Soriano	1985-11-20 00:00:00.000	1

```
use ABCPayroll
exec dbo.uspCountTotalEmployeesperDepartment @departmentid=4
```

Messages

```
There is no existing employee
```

```
use ABCPayroll
exec dbo.uspCountTotalEmployeesperDepartment @departmentid=1
```

%

Results | Messages

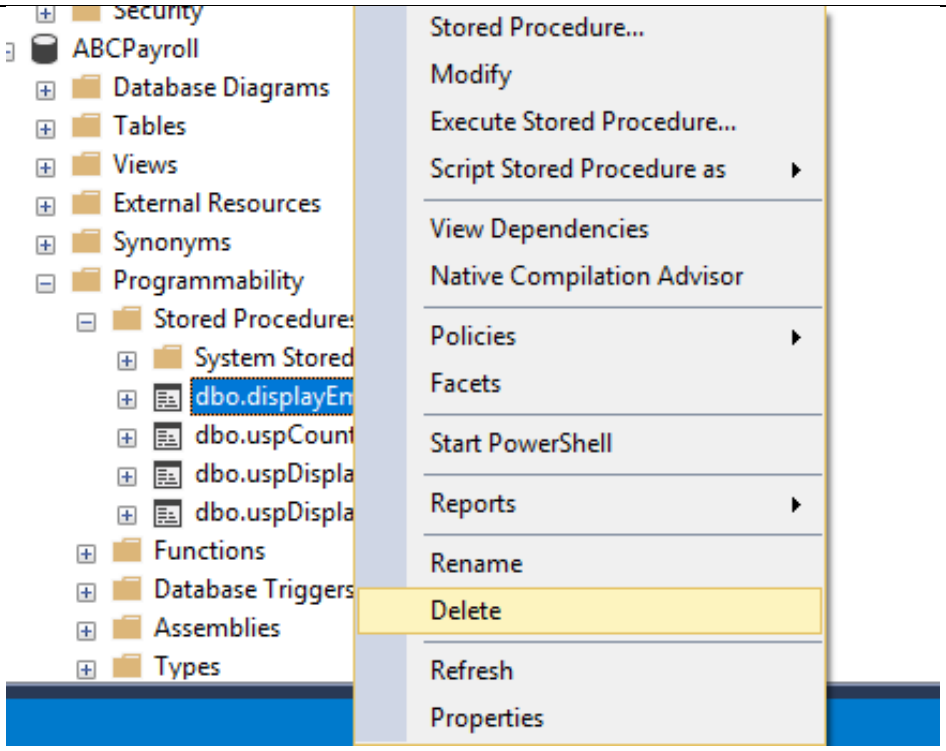
```
Total No. of Employees:2
```

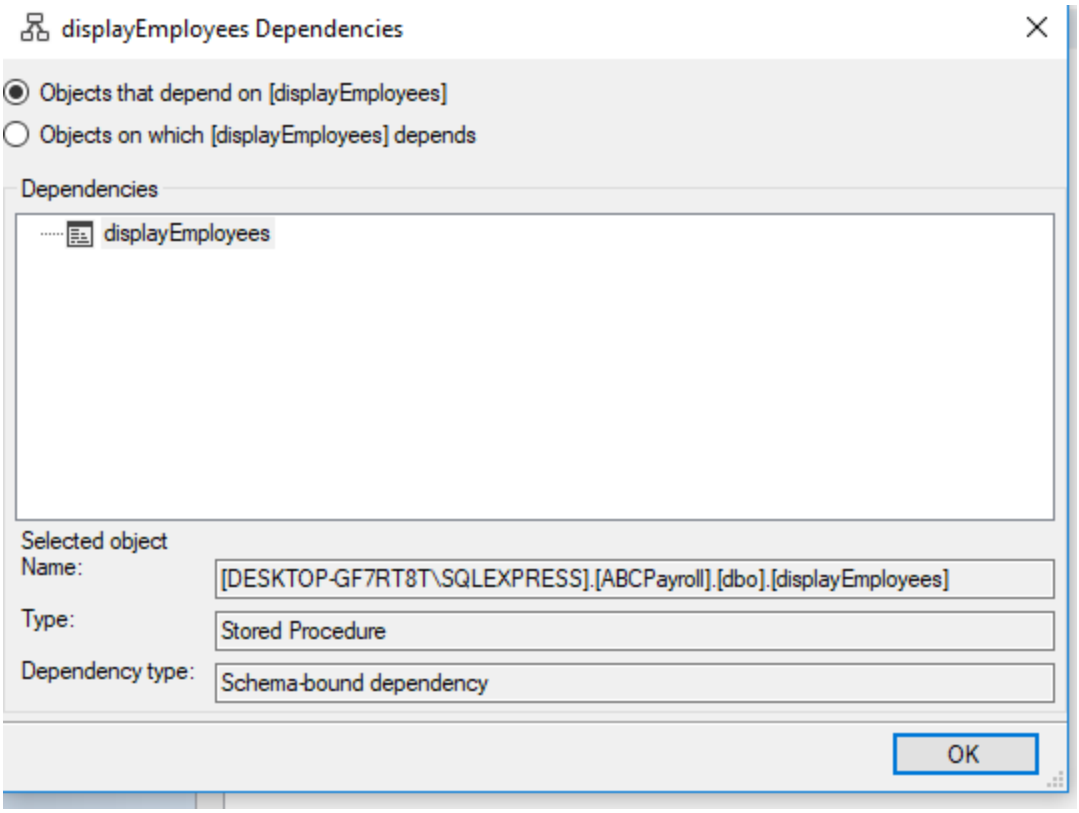
Delete a Stored Procedure

Step 1: In Object Explorer, connect to an instance of Database Engine and then expand that instance.

Step 2: Expand **Databases**, expand the database in which the procedure belongs, and then expand **Programmability**.

Step 3: Expand **Stored Procedures**, right-click the procedure to remove, and then click **Delete**.





Step 4. Confirm the correct procedure is selected, and then click **OK**.

6. Database Output

Copy screenshot(s) of your output after completing the procedures provided in Part 5.

F. Create a Stored Procedure
G. Execute a Stored Procedure
H. Modify a Stored Procedure
I. Specify a Parameter
J. Return Data
K. Delete a Stored Procedure

7. Supplementary Activity

Do the following tasks and copy screenshot(s) of your output.

Table name: TRUCK
Primary key: TRUCK_NUM
Foreign key: BASE_CODE, TYPE_CODE

TRUCK_NUM	BASE_CODE	TYPE_CODE	TRUCK_MILES	TRUCK_BUY_DATE	TRUCK_SERIAL_NUM
1001	501	1	32123.5	23-Sep-07	AA-322-12212-W11
1002	502	1	76984.3	05-Feb-06	AC-342-22134-Q23
1003	501	2	12346.6	11-Nov-06	AC-445-78656-Z99
1004		1	2894.3	06-Jan-07	WQ-112-23144-T34
1005	503	2	45673.1	01-Mar-06	FR-998-32245-W12
1006	501	2	193245.7	15-Jul-03	AD-456-00845-R45
1007	502	3	32012.3	17-Oct-04	AA-341-96573-Z84
1008	502	3	44213.6	07-Aug-05	DR-559-22189-D33
1009	503	2	10932.9	12-Feb-08	DE-887-98455-E94

Table name: BASE
Primary key: BASE_CODE
Foreign key: none

BASE_CODE	BASE_CITY	BASE_STATE	BASE_AREA_CODE	BASE_PHONE	BASE_MANAGER
501	Murfreesboro	TN	615	123-4567	Andrea D. Gallager
502	Lexington	KY	568	234-5678	George H. Delarosa
503	Cape Girardeau	MO	456	345-6789	Maria J. Talndo
504	Dalton	GA	901	456-7890	Peter F. McAvee

Table name: TYPE
Primary key: TYPE_CODE
Foreign key: none

TYPE_CODE	TYPE_DESCRIPTION
1	Single box, double-axle
2	Single box, single-axle
3	Tandem trailer, single-axle

1. Create a script to create the Trucking database and the following tables. Use the appropriate data types and assign the primary keys and foreign keys.
2. Create a script to insert the given values using the Trucking database.
3. Create and execute a stored procedure to display the total number of truck per base. Display the truck number, base code, and base manager. Arrange the list from highest to lowest.
4. Create and execute a stored procedure to display the total number of truck per type code. Display the truck number, type code and type description. Use TYPE_CODE as input parameter.
5. Create and execute a stored procedure to display the truck number, base city, base manager and type description.
6. Create and execute a stored procedure to return the total number of truck purchased per month. Arrange the list from highest to lowest number of truck. Print the total number of truck and the corresponding month. Use month as input parameter and total as output.

7. Create and execute a stored procedure to display the truck with truck miles ranging from 5000 to 40000. Arrange the list from highest to lowest.
8. Create and execute a stored procedure to display the truck number, truck_buy_date and type description that purchased for a specific year. Use year as input parameter.

8. Conclusion

9. Assessment (Rubric for Laboratory Performance):

Activity No. 12	
User-Defined Functions	
Course Code: CPE011	**Program:**
Course Title: Database Management System	**Date Performed:**
Section:	**Date Submitted:**
Name:	**Instructor:**

1. Objective(s):

This activity aims to create and implement user-defined functions in databases

2. Intended Learning Outcomes (ILOs):

The students should be able to:
2.1 Create different types of user-defined functions
2.2Implement and execute user-defined functions in a database.

3. Discussion :

Like functions in programming languages, SQL Server user-defined functions are routines that accept parameters, perform an action, such as a complex calculation, and return the result of that action as a value. The return value can either be a single scalar value or a result set.

Benefits

- ✓ They allow modular programming.
- ✓ They allow faster execution.
- ✓ They can reduce network traffic.

Types of User-defined Functions

1. **Scalar Function**

   User-defined scalar functions return a single data value of the type defined in the RETURNS clause. For an inline scalar function, there is no function body; the scalar value is the result of a single statement. For a multistatement scalar function, the function body, defined in a BEGIN...END block, contains a series of Transact-SQL statements that return the single value. The return type can be any data type except **text**, **ntext**, **image**, **cursor**, and **timestamp**.

2. **Table-Valued Functions**

   User-defined table-valued functions return a **table** data type. For an inline table-valued function, there is no function body; the table is the result set of a single SELECT statement.

3. **System Functions**

   SQL Server provides many system functions that you can use to perform a variety of operations. They cannot be modified.

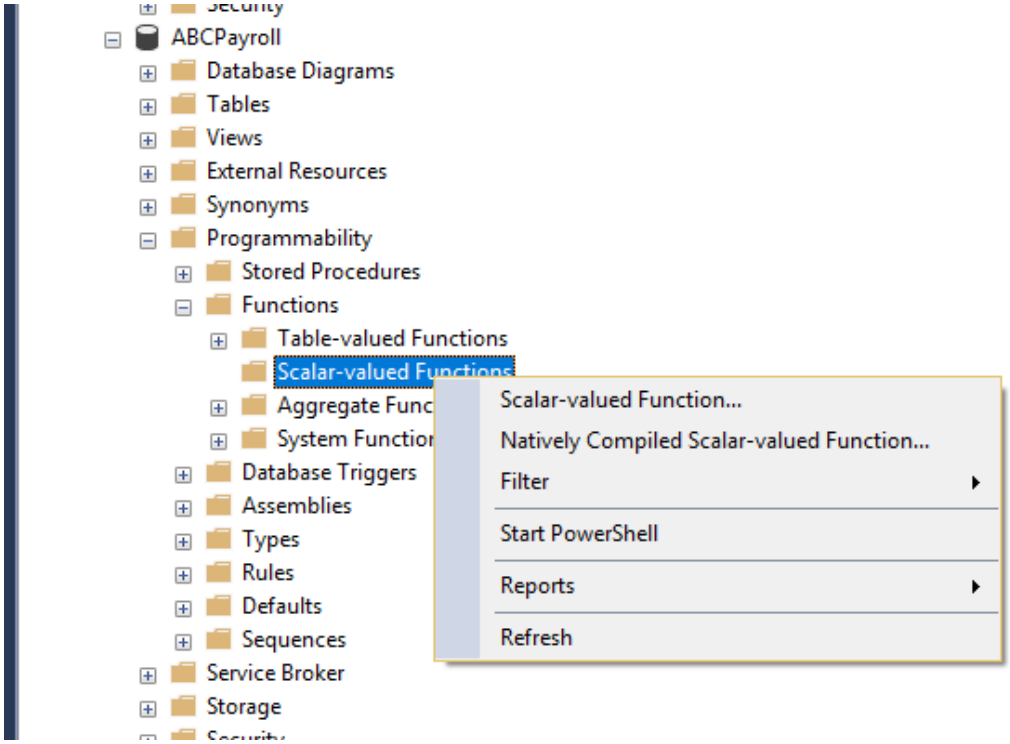
4. Resources:

Personal Computer with installed SQL Server
ABCPayroll Database

5. Procedure:

Scalar Functions

Step 1: In Object Explorer, connect to an instance of Database Engine and then expand that instance.
Step 2: Expand Databases, expand the ABCPayroll database, and then expand Programmability
Step 3: Choose**Functions**, and then right-click **Scalar-valued Functions**. Choose **Scalar-valued Function.**



Step 4: Modify the function using the given screenshot. Type your name as author and the creation date.

```
GO
-- ============================================
-- Author:      <Author,,Name>
-- Create date: <Create Date, ,>
-- Description: <count the total no. of employees by department>
-- ============================================
CREATE FUNCTION ufnCountTotalEmployeesbyDept
(
    -- Add the parameters for the function here
    @departmentid int
)
RETURNS int
AS
BEGIN
    -- Declare the return variable here
    DECLARE @total int

    -- Add the T-SQL statements to compute the return value here
    Select @total = count(*) from employeeinfo where departmentid = @departmentid

    -- Return the result of the function
    RETURN @total

END
GO
```

Step 5: Click execute or press F5 to save the stored function.
Step 6: Test the ufnCountTotalEmployeesbyDept stored function. Create a new query and type the given statement. Click Execute.

```
use ABCPayroll
DECLARE @total int
exec @total = dbo.ufnCountTotalEmployeesbyDept @departmentid = 1
PRINT 'Total no. of employees: ' + convert(varchar(10), @total)
```

121 %

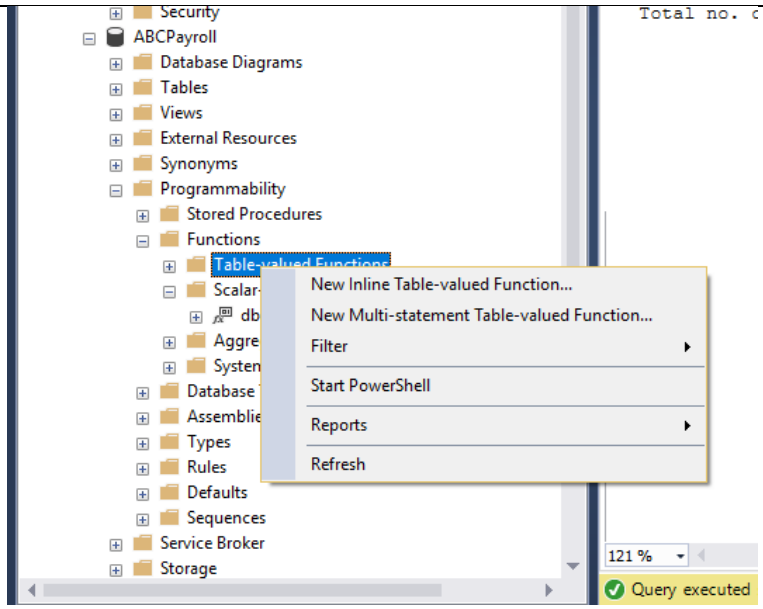
Messages

```
Total no. of employees: 2
```

Table-valued Functions

A. Inline Table-valued Function

Step 1: In Object Explorer, connect to an instance of Database Engine and then expand that instance.
Step 2: Expand Databases, expand the ABCPayroll database, and then expand Programmability
Step 3: Choose**Functions**, and then right-click **Table-valued Functions**. Choose **New Inline Table-valued Function**



Step 4: Modify the function using the given screenshot. Type your name as author and the creation date.

```
-- ============================================
-- Author:      <Author,,Name>
-- Create date: <Create Date,,>
-- Description: <Display Employee Information>
-- ============================================
CREATE FUNCTION ufnDisplayEmployeeInfo
(
    -- Add the parameters for the function here
    @employeeid char(7)
)
RETURNS TABLE
AS
RETURN
(
    -- Add the SELECT statement with parameter references here
    SELECT e.employeeid as 'employee id', CONCAT(e.firstname,' ',
    e.middlename,' ', e.lastname) as 'employee name', d.department
    from employeeinfo e
    inner join department d
    on e.departmentid = d.departmentid
    where e.employeeid = @employeeid
)
GO
```

Step 5: Click execute or press F5 to save the stored function.
Step 6: Test the ufnDisplayEmployeeInfo stored function. Create a new query and type the given statement. Click Execute.

```
use ABCPayroll
select * from ufnDisplayEmployeeInfo(1122334)
```

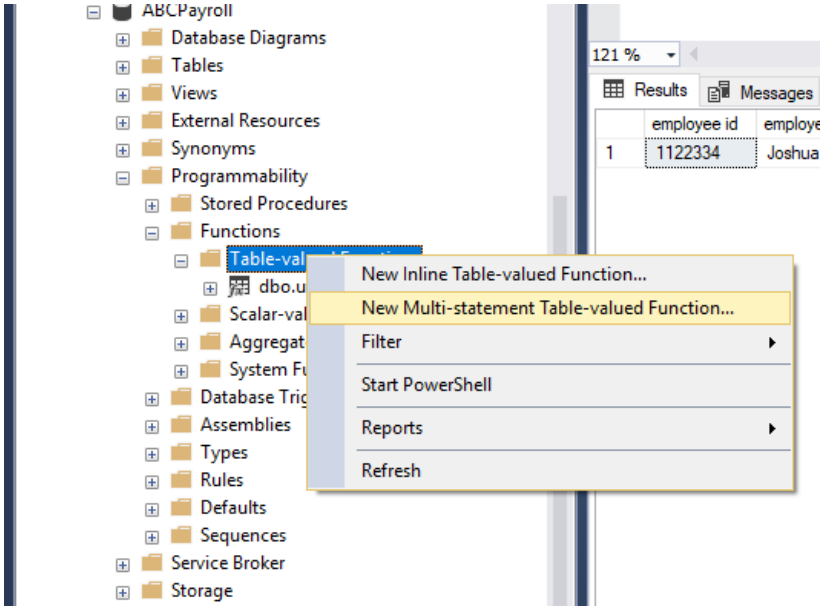
121 %

Results | Messages

	employee id	employee name	department
1	1122334	Joshua Santos Reyes	Marketing

B. Multi-statement Table-valued Function

Step 1: In Object Explorer, connect to an instance of Database Engine and then expand that instance.
Step 2: Expand Databases, expand the ABCPayroll database, and then expand Programmability
Step 3: Choose**Functions**, and then right-click **Table-valued Functions**. Choose **New Multi-statement Table-valued Function**



Step 4: Modify the function using the given screenshot. Type your name as author and the creation date.

```
-- ============================================
-- Author:      <Author,,Name>
-- Create date: <Create Date,,>
-- Description: <Description,,>
-- ============================================
CREATE FUNCTION ufnDisplayEmpServiceYears
(
    -- Add the parameters for the function here
    @employeeid char(7)
)
RETURNS
@findEmpServiceYears TABLE
(
    -- Add the column definitions for the TABLE variable here
    employeeid char(7),
    employeename varchar(255),
    serviceyear int
)
AS
BEGIN
    -- Fill the table variable with the rows for your result set
          DECLARE @employeename varchar(255)
        DECLARE @serviceyear int, @currentyear int, @yearemployed int
        SELECT @employeename = CONCAT(firstname,' ',
        middlename,' ', lastname) from employeeinfo
        where employeeid = @employeeid
        SET @currentyear = year(getdate())
        SELECT @yearemployed = year(dateemployment)
        from employeeinfo where employeeid = @employeeid
        SET @serviceyear = @currentyear - @yearemployed
        INSERT @findEmpServiceYears(employeeid,employeename,serviceyear)
        VALUES(@employeeid,@employeename,@serviceyear)

    RETURN
END
GO
```

Step 5: Click execute or press F5 to save the stored function.
Step 6: Test the ufnDisplayEmpServiceYears stored function. Create a new query and type the given statement. Click Execute.

```
select * from dbo.ufnDisplayEmpServiceYears (1122334)
```

121 %

Results | Messages

	employeeid	employeename	serviceyear
1	1122334	Joshua Santos Reyes	38

6. Database Output

Copy screenshot(s) of your output after completing the procedures provided in Part 5.

1. **Scalar Functions**
2. **Table-valued functions**
    a. **Inline Table-valued Function**
    b. **Multi-statement Table-valued Function**

7. Supplementary Activity

Do the following tasks and copy screenshot(s) of your output.

Table name: TRUCK
Primary key: TRUCK_NUM
Foreign key: BASE_CODE, TYPE_CODE

TRUCK_NUM	BASE_CODE	TYPE_CODE	TRUCK_MILES	TRUCK_BUY_DATE	TRUCK_SERIAL_NUM
1001	501	1	32123.5	23-Sep-07	AA-322-12212-W11
1002	502	1	76984.3	05-Feb-06	AC-342-22134-Q23
1003	501	2	12346.6	11-Nov-06	AC-445-78656-Z99
1004		1	2894.3	06-Jan-07	WQ-112-23144-T34
1005	503	2	45673.1	01-Mar-06	FR-998-32245-W12
1006	501	2	193245.7	15-Jul-03	AD-456-00845-R45
1007	502	3	32012.3	17-Oct-04	AA-341-96573-Z84
1008	502	3	44213.6	07-Aug-05	DR-559-22189-D33
1009	503	2	10932.9	12-Feb-08	DE-887-98455-E94

Table name: BASE
Primary key: BASE_CODE
Foreign key: none

BASE_CODE	BASE_CITY	BASE_STATE	BASE_AREA_CODE	BASE_PHONE	BASE_MANAGER
501	Murfreesboro	TN	615	123-4567	Andrea D. Gallager
502	Lexington	KY	568	234-5678	George H. Delarosa
503	Cape Girardeau	MO	456	345-6789	Maria J. Talndo
504	Dalton	GA	901	456-7890	Peter F. McAvee

Table name: TYPE
Primary key: TYPE_CODE
Foreign key: none

TYPE_CODE	TYPE_DESCRIPTION
1	Single box, double-axle
2	Single box, single-axle
3	Tandem trailer, single-axle

1. Create a script to create the Trucking database and the following tables. Use the appropriate data types and assign the primary keys and foreign keys.
2. Create a script to insert the given values using the Trucking database.
3. Create and execute a function that returns the total number of truck per base. Use BASE_CODE as input parameter
4. Create and execute a function to display the truck number, base city, base manager and type description.

5. Create and execute a function that display the total number of truck purchased per month. Display the month and the total number of truck. Arrange the list from highest to lowest number of truck.
8. Conclusion
9. Assessment (Rubric for Laboratory Performance):

Activity No. 13	
Trigger	
Course Code: CPE011	**Program:**
Course Title: Database Management System	**Date Performed:**
Section:	**Date Submitted:**
Name:	**Instructor:**

1. Objective(s):

This activity aims to create and implement trigger in databases

2. Intended Learning Outcomes (ILOs):

The students should be able to:
2.1 Create triggers in database
2.2Implement and execute triggers.

3. Discussion :

A trigger is a special kind of stored procedure that automatically executes when an event occurs in the database server. These events can be categorized as

1. Data Manipulation Language (DML) and
2. Data Definition Language (DDL) events.

DML triggers execute when a user tries to modify data through a data manipulation language (DML) event. DML events are INSERT, UPDATE, or DELETE statements on a table or view. These triggers fire when any valid event is fired, regardless of whether or not any table rows are affected.DML triggers use the deleted and inserted logical (conceptual) tables. They are structurally similar to the table on which the trigger is defined, that is, the table on which the user action is tried. The deleted and inserted tables hold the old values or new values of the rows that may be changed by the user action.

DDL triggers execute in response to a variety of data definition language (DDL) events. These events primarily correspond to Transact-SQL CREATE, ALTER, and DROP statements, and certain system stored procedures that perform DDL-like operations.

The trigger actions specified in the Transact-SQL statements go into effect when the operation is tried. Triggers can include any number and kind of Transact-SQL statements, with exceptions. A trigger is designed to check or change data based on a data modification or definition statement; it should not return data to the user. The Transact-SQL statements in a trigger frequently include control-of-flow language.

Trigger Limitations

- A trigger is created only in the current database; however, a trigger can reference objects outside the current database.
- If the trigger schema name is specified to qualify the trigger, qualify the table name in the same way.
- The same trigger action can be defined for more than one user action (for example, INSERT and UPDATE) in the same CREATE TRIGGER statement.

- INSTEAD OF DELETE/UPDATE triggers cannot be defined on a table that has a foreign key with a cascade on DELETE/UPDATE action defined.

- Any SET statement can be specified inside a trigger. The SET option selected remains in effect during the execution of the trigger and then reverts to its former setting.

- When a trigger fires, results are returned to the calling application, just like with stored procedures. To prevent having results returned to an application because of a trigger firing, do not include either SELECT statements that return results or statements that perform variable assignment in a trigger. A trigger that includes either SELECT statements that return results to the user or statements that perform variable assignment requires special handling; these returned results would have to be written into every application in which modifications to the trigger table are allowed. If variable assignment must occur in a trigger, use a SET NOCOUNT statement at the start of the trigger to prevent the return of any result sets.

The following Transact-SQL statements are not allowed in a DML trigger:

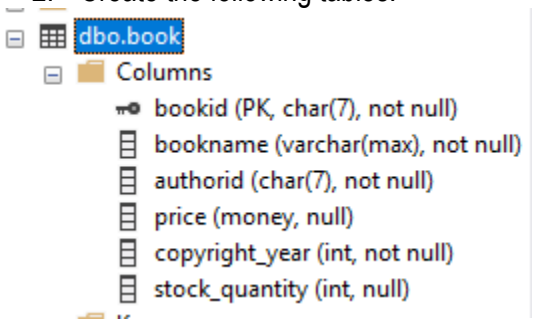
ALTER DATABASE	CREATE DATABASE	DROP DATABASE
RESTORE DATABASE	RESTORE LOG	RECONFIGURE

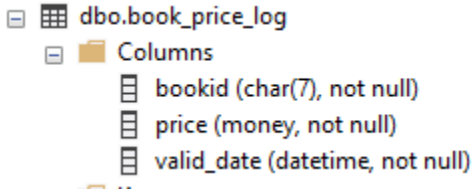
4. Resources:

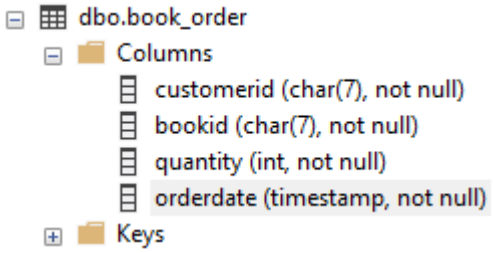
Personal Computer with installed SQL Server
ABCLibrary Database

5. Procedure:

1. Create ABCLibrary Database
2. Create the following tables:







3. Insert the following data in the book table.

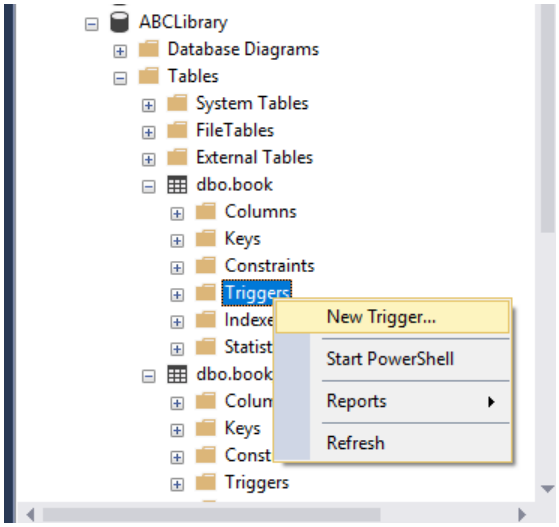
bookid	bookname	authorid	price	copyright_year	stock_quantity
BK-0001	Software Engineering	1	500.2500	2015	60
BK-0002	System Analysis and Design	2	300.0000	2016	20
BK-0003	Connectiing Networks	3	750.2500	2017	70
BK-0004	Embedded System	4	1000.7500	2016	80
BK-0005	Robotics	5	789.9500	2015	90
BK-0006	Image Processing	6	800.9500	2014	100
BK-0007	Computer Architecture	7	1500.7500	2014	30
BK-0008	Routing and Switching	8	2000.7500	2016	78
BK-0009	Artificial Intelligence	9	5400.7500	2015	65
BK-0010	Internet of Things	10	1005.2500	2015	77

Create Data Manipulation Language (DML) Trigger

Example No. 1

Step 1: In **Object Explorer**, connect to an instance of Database Engine and then expand that instance.

Step 2: Expand **Databases**, expand the **ABCLibrary** database, and then expand **Tables**. Expand **book** table. Right-click **Trigger** and choose **New Trigger**.



Step 3: Modify the trigger using the given screenshot. Type your name as author and the creation date.

```
GO
-- ============================================
-- Author:      <Author,,Name>
-- Create date: <Create Date,,>
-- Description: <Update Book Price log table>
-- ============================================
CREATE TRIGGER UpdateBookPrice
   ON book
   AFTER INSERT, UPDATE
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;

    -- Insert statements for trigger here
    INSERT book_price_log
    (bookid,price,valid_date)
    SELECT bookid, price, getdate() from inserted
    SELECT * from inserted

END
```

Step 4: Click execute or press F5 to save the trigger.

Step 5: Test the UpdateBookPriceLog trigger. Create a new query and type the given statement. Click Execute.

A. Inserting Book Information

```
use ABCLibrary
insert into book
(bookid,bookname,authorid,price,copyright_year,stock_quantity)
values
('BK-0011', 'Database Management Systems 2', 11,2500.75,2017,35)
select * from book
select * from book_price_log
```

121 %

Results | Messages

	bookid	bookname	authorid	price	copyright_year	stock_quantity
1	BK-0011	Database Management Systems 2	11	2500.75	2017	35

	bookid	bookname	authorid	price	copyright_year	stock_quantity
1	BK-0001	Software Engineering	1	500.25	2015	60
2	BK-0002	System Analysis and Design	2	300.00	2016	20
3	BK-0003	Connectiing Networks	3	750.25	2017	70
4	BK-0004	Embedded System	4	1000.75	2016	80
5	BK-0005	Robotics	5	789.95	2015	90
6	BK-0006	Image Processing	6	800.95	2014	100
7	BK-0007	Computer Architecture	7	1500.75	2014	30
8	BK-0008	Routing and Switching	8	2000.75	2016	78
9	BK-0009	Artificial Intelligence	9	5400.75	2015	65
10	BK-0010	Internet of Things	10	1005.25	2015	77

	bookid	price	valid_date
1	BK-0011	2500.75	2018-02-08 14:32:48.280

121 %

Results | Messages

	bookid	bookname	authorid	price	copyright_year	stock_quantity
1	BK-0011	Database Management Systems 2	11	500.75	2015	45

	bookid	bookname	authorid	price	copyright_year	stock_quantity
1	BK-0001	Software Engineering	1	NULL	2015	NULL
2	BK-0002	System Analysis and Design	2	NULL	2016	NULL
3	BK-0003	Connectiing Networks	3	NULL	2017	NULL
4	BK-0004	Embedded System	4	NULL	2016	NULL
5	BK-0005	Robotics	5	NULL	2015	NULL
6	BK-0006	Image Processing	6	NULL	2014	NULL
7	BK-0007	Computer Architecture	7	NULL	2014	NULL
8	BK-0008	Routing and Switching	8	NULL	2016	NULL
9	BK-0009	Artificial Intelligence	9	NULL	2015	NULL
10	BK-0010	Internet of Things	10	NULL	2015	NULL
11	BK-0011	Database Management S...	11	500...	2015	45

	bookid	price	valid_date
1	BK-0011	500.75	2018-02-07 13:31:02.523

B. Updating Book Information

```
use ABCLibrary
update book
set price=500.25
where bookid = 'BK-0001'
SELECT * from book
SELECT * from book_price_log
```

121 %

Results | Messages

	bookid	bookname	authorid	price	copyright_year	stock_quantity
1	BK-0001	Software Engineering	1	500.25	2015	60

	bookid	bookname	authorid	price	copyright_year	stock_quantity
1	BK-0001	Software Engineering	1	500.25	2015	60
2	BK-0002	System Analysis and Design	2	300.00	2016	20
3	BK-0003	Connectiing Networks	3	750.25	2017	70
4	BK-0004	Embedded System	4	1000.75	2016	80
5	BK-0005	Robotics	5	789.95	2015	90
6	BK-0006	Image Processing	6	800.95	2014	100
7	BK-0007	Computer Architecture	7	1500.75	2014	30
8	BK-0008	Routing and Switching	8	2000.75	2016	78
9	BK-0009	Artificial Intelligence	9	5400.75	2015	65

	bookid	price	valid_date
1	BK-0011	2500.75	2018-02-08 14:32:48.280
2	BK-0001	500.25	2018-02-08 14:34:14.220

Example No. 2

Step 1: In **Object Explorer**, connect to an instance of Database Engine and then expand that instance.

Step 2: Expand **Databases**, expand the **ABCLibrary** database, and then expand **Tables**. Expand **book_order** table. Right-click **Trigger** and choose **New Trigger**.

```
-- =============================================
-- Author:      <Author,,Name>
-- Create date: <Create Date,,>
-- Description: <Update the quantity of stock in book table>
-- =============================================
CREATE TRIGGER UpdateBookStockQuantity
   ON  book_order
   AFTER INSERT
AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;
     -- Insert statements for trigger here
     Update book set stock_quantity =
     stock_quantity - (select quantity from inserted)
     where bookid = (select bookid from inserted)

END
```

Step 4: Click execute or press F5 to save the trigger.

Step 5: Test the UpdateBookStock trigger. Create a new query and type the given statement. Click Execute.

```
use ABCLibrary
go
insert into book_order
(customerid,bookid,quantity,orderdate)
VALUES
('CUS-001','BK-0001',10,getdate())
select * from book_order
select * from book
```

121 %

Results | Messages

	bookid	bookname	authorid	price	copyright_year	stock_quantity
1	BK-0001	Software Engineering	1	500.25	2015	50

	customerid	bookid	quantity	orderdate
1	CUS-001	BK-0001	10	2018-02-08 15:20:37.990

	bookid	bookname	authorid	price	copyright_year	stock_quantity
1	BK-0001	Software Engineering	1	500.25	2015	50
2	BK-0002	System Analysis and Design	2	300.00	2016	20
3	BK-0003	Connectiing Networks	3	750.25	2017	70
4	BK-0004	Embedded System	4	1000.75	2016	80
5	BK-0005	Robotics	5	789.95	2015	90
6	BK-0006	Image Processing	6	800.95	2014	100
7	BK-0007	Computer Architecture	7	1500.75	2014	30
8	BK-0008	Routing and Switching	8	2000.75	2016	78
9	BK-0009	Artificial Intelligence	9	5400.75	2015	65

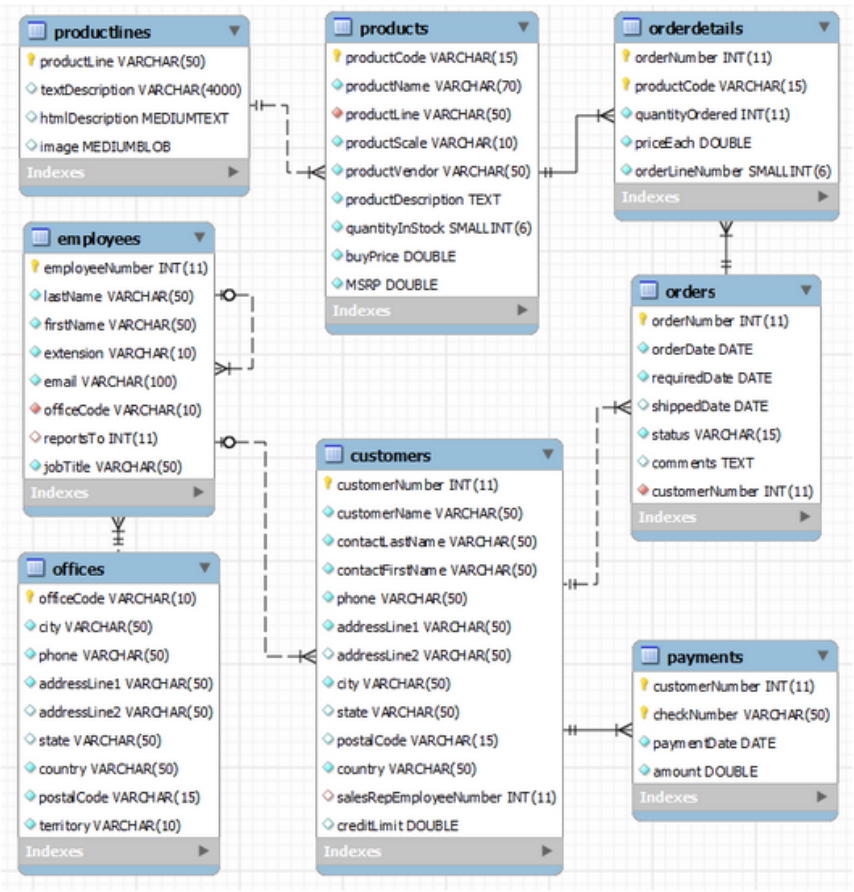
6. Database Output

Copy screenshot(s) of your output after completing the procedures provided in Part 5.

a. Example No. 1
b. Example No. 2

7. Supplementary Activity

Do the following tasks and copy screenshot(s) of your output.



a. Use the given ERD to create the ClassicModels database and tables. Assign the appropriate data type.
b. Insert five(5) Office records where the Office code starts with OFC-001.
c. Insert five(5) employee records (5) with managerial position. Use Manager as Job Title.

Employee Number	Lastname	Firstname	Office Code
1	Sy	Henry	OFC-001
2	Cojuangco	Edward	OFC-002
3	Reyes	Gino	OFC-003
4	Lacson	Ping	OFC-004
5	Magno	Michael	OFC-005

d. Create a trigger that automatically updates the reportsTo field of the employee depending on the OFFICE CODE. Insert 2 records for each office code.

Example: Since Employee Number 6 resides at Office Code OFC-001,the reportsTo column will be automatically equals to 1 since the manager of OFC-001 is Employee Number 1.

Employee Number	Lastname	Firstname	reportsTo	Office Code
6	Villanueva	Alonica	1	OFC-001

5. Create a trigger that automatically updates the Sales Rep Employee and Credit Limit field of the customer table

depending on the country. Insert 2 records for each customer per country. Update the last 2 records to change the country to Spain and France respectively.

Country	Employee Number	Credit Limit
Spain	6	100000
France	7	200000
USA	8	300000
Paris	9	400000
Africa	10	500000

6. Create a table PRODUCT_PRICE_LOG.

   productCode varchar(15)
   price money
   valid_date datetime

7. Create a trigger that updates the table PRODUCT_PRICE_LOG after inserting and updating the products record.
8. Create a trigger that automatically updates the priceEach of the orderdetails record depending on the price on the Product table.
9. Create a trigger that automatically updates the status depending on the following condition:
   - If the shipped date is less than or equal on the required date , the status is OK FOR DELIVERY
   - If the shipped is greater than the required date , the status is PENDING.

8. Conclusion

9. Assessment (Rubric for Laboratory Performance):

Activity No. 14	
Transactions	
Course Code: CPE011	**Program:**
Course Title: Database Management System	**Date Performed:**
Section:	**Date Submitted:**
Name:	**Instructor:**

1. Objective(s):

This activity aims to create and manage transactions in databases

2. Intended Learning Outcomes (ILOs):

The students should be able to:
2.1 Create and apply transactions in databases
2.2Troubleshoot error(s) encountered in developing transactions.

3. Discussion :

A transaction is a single unit of work. If a transaction is successful, all of the data modifications made during the transaction are committed and become a permanent part of the database. If a transaction encounters errors and must be canceled or rolled back, then all of the data modifications are erased.

SQL Server operates in the following transaction modes:

1. Autocommit transactions - Each individual statement is a transaction.
2. Explicit transactions - Each transaction is explicitly started with the BEGIN TRANSACTION statement and explicitly ended with a COMMIT or ROLLBACK statement.
3. Implicit transactions - A new transaction is implicitly started when the prior transaction completes, but each transaction is explicitly completed with a COMMIT or ROLLBACK statement.
4. Batch-scoped transactions - Applicable only to multiple active result sets (MARS), a Transact-SQL explicit or implicit transaction that starts under a MARS session becomes a batch-scoped transaction. A batch-scoped transaction that is not committed or rolled back when a batch completes is automatically rolled back by SQL Server.

BEGIN TRANSACTION

Marks the starting point of an explicit, local transaction. Explicit transactions start with the BEGIN TRANSACTION statement and end with the COMMIT or ROLLBACK statement.

COMMIT TRANSACTION

Marks the end of a successful implicit or explicit transaction. If @@TRANCOUNT is 1, COMMIT TRANSACTION makes all data modifications performed since the start of the transaction a permanent part of the database, frees the resources held by the transaction, and decrements @@TRANCOUNT to 0. If @@TRANCOUNT is greater than 1, COMMIT TRANSACTION decrements @@TRANCOUNT only by 1 and the transaction stays active.

ROLLBACK TRANSACTION

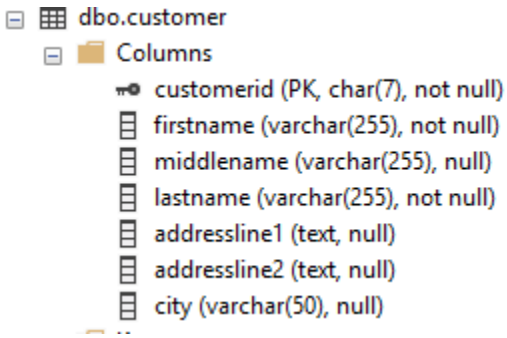
Rolls back an explicit or implicit transaction to the beginning of the transaction, or to a savepoint inside the transaction. You can use ROLLBACK TRANSACTION to erase all data modifications made from the start of the transaction or to a savepoint. It also frees resources held by the transaction.

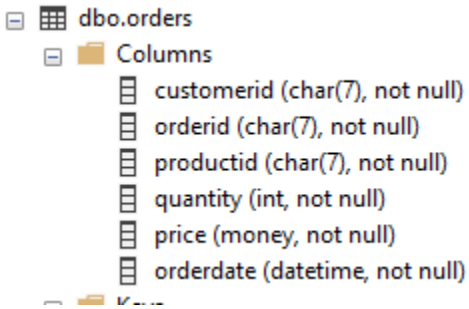
4. Resources:

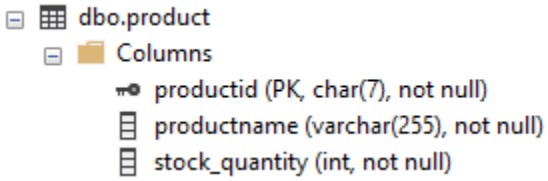
Personal Computer with installed SQL Server

5. Procedure:

1. Create the ABCFlowerShop database
2. Create the following tables in ABCFlowerShop database







3. Insert the following data into product table.

	productid	productname	stock_quantity
	RS-0001	Roses	100
	SM-0002	Sampaguita	200
	CL-0003	Calla Lily	145
	GM-0004	Gumamela	240

COMMIT TRANSACTION

A. Using T-SQL

Step 1: In **Object Explorer**, connect to an instance of Database Engine and then expand that instance.

Step 2: Create a new query. Type the given T-SQL. Execute the given query.

```
use ABCFlowerShop
go
begin tran InsertCustomer
      insert into customer
      values ('CUS-001', 'Joshua',
      'Santos', 'Reyes', '888 Mahal Kita St.',
       'Paraiso ', 'Manila')
      commit tran
go
select * from customer
```

121 %

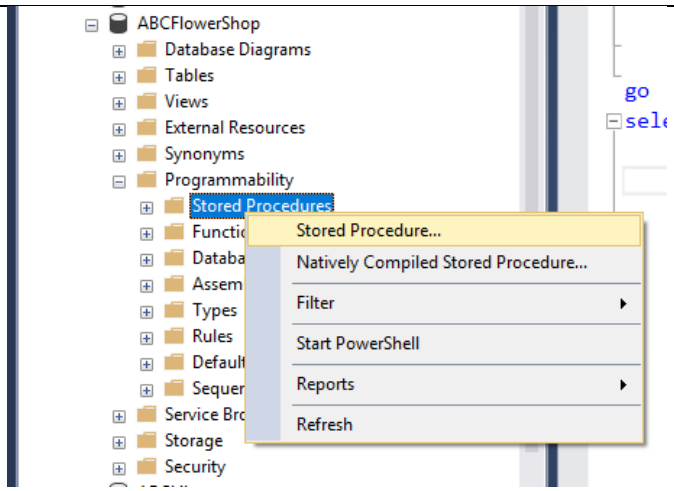
Results | Messages

	customerid	firstname	middlename	lastname	addressline1	addressline2	city
1	CUS-001	Joshua	Santos	Reyes	888 Mahal Kita St.	Paraiso	Manila

B. Using Stored Procedure

Step 1: In **Object Explorer**, connect to an instance of Database Engine and then expand that instance.

Step 2: Expand **Databases**, expand the **ABCFlowerShop** database, and then expand **Programmability**. Right-click **Stored Procedures**. Choose **Stored Procedure**



Step 3: Create the **uspInsertOrder** stored procedure. Type your name as author and the current date in Create data field. Click **Execute** or Press **F5** to save the stored procedure.

```
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:      <Author,,Name>
-- Create date: <Create Date,,>
-- Description: <Insert Order Transaction>
-- =============================================
CREATE PROCEDURE uspInsertOrder
    -- Add the parameters for the stored procedure here
    @customerid char(7), @orderid char(7) ,@productid char(7),
    @quantity int, @price money, @orderdate datetime
AS
BEGIN TRAN InserOrder
    BEGIN TRY
        INSERT  orders (customerid,orderid,productid,
        quantity,price,orderdate)
        VALUES (@customerid,@orderid,
        @productid,@quantity,@price,@orderdate)
        COMMIT TRAN InsertOrder
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN InsertOrder
    END CATCH
```

Step 4: Create a new query to test the **uspInsertOrder** stored procedure. Click **Execute** or Press **F5**

```
use ABCFlowerShop
exec uspInsertOrder @customerid='CUS-001',
     @orderid='OR-0001',
     @productid='RS-0001',
     @quantity=5,
     @price=500.75,
     @orderdate='12-08-2015'
go
select * from orders
```

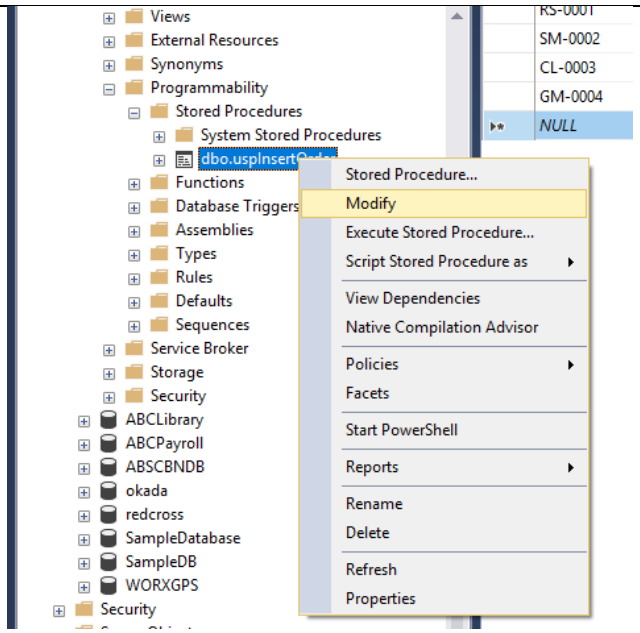
Results | Messages

customerid	orderid	productid	quantity	price	orderdate
CUS-001	OR-0001	RS-0001	5	500.75	2015-12-08 00:00:00.000

ROLLBACK TRANSACTION

Step 1: In **Object Explorer**, connect to an instance of Database Engine and then expand that instance.

Step 2: Expand **Databases**, expand the **ABCFlowerShop** database, and then expand **Programmability**. Expand **Stored Procedures** and Right-click **uspInsertOrder**. Choose **Modify**.

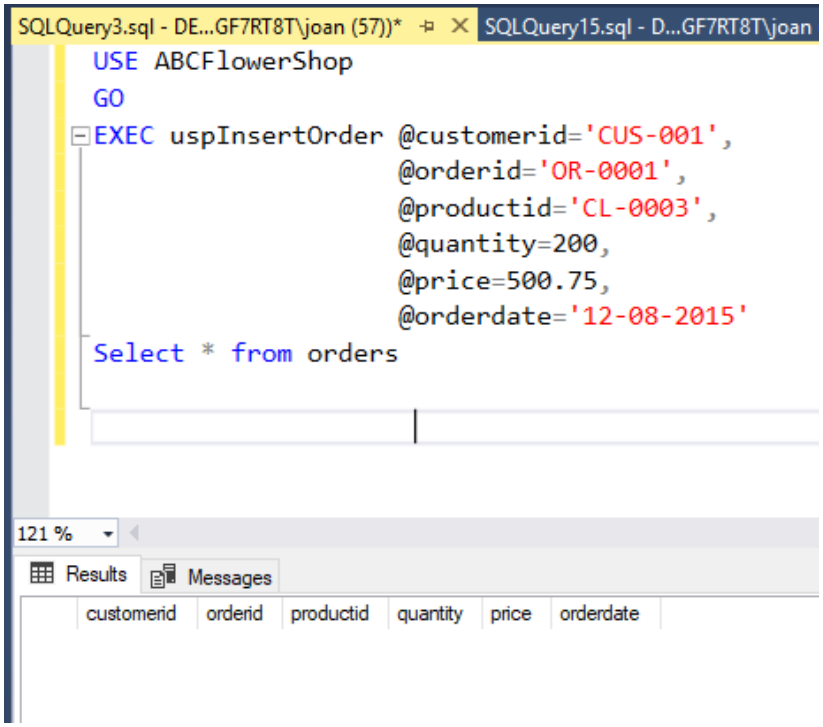


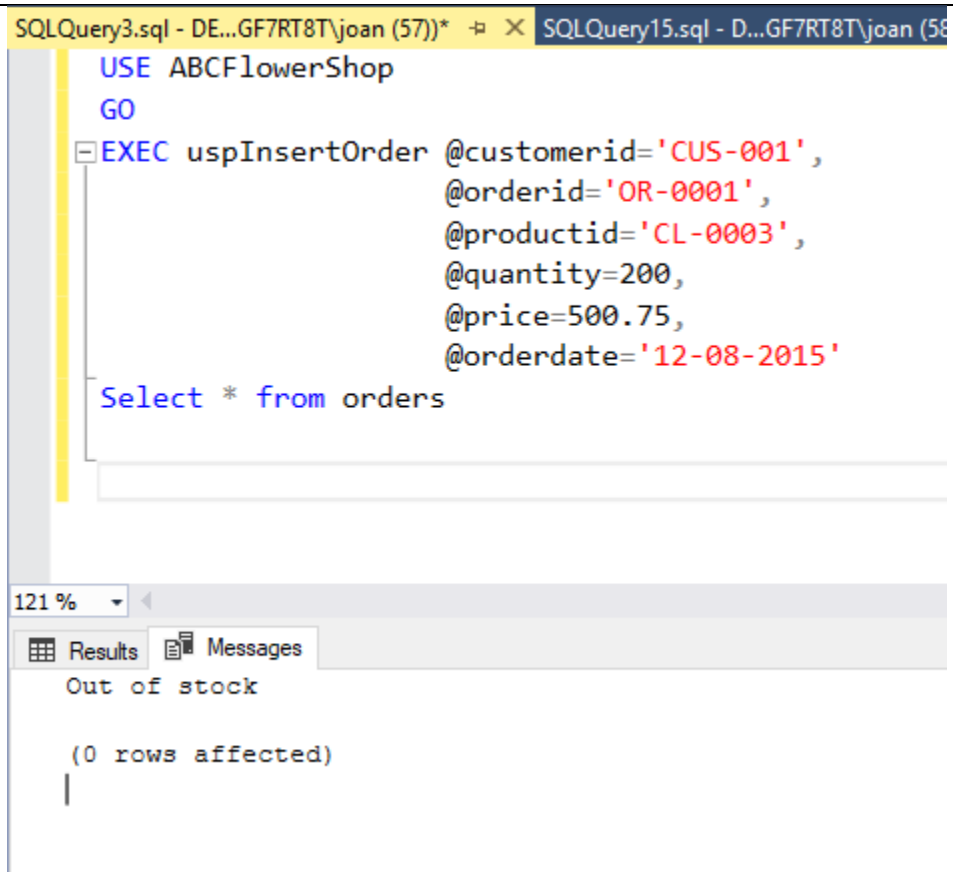
Step 3:Modify the **uspInsertOrder** stored procedure using the given figure to rollback the transaction if the ordered product is out of stock.

```
-- ============================================
-- Author:      <Author,,Name>
-- Create date: <Create Date,,>
-- Description: <Insert Order Transaction>
-- ============================================
ALTER PROCEDURE [dbo].[uspInsertOrder]
    -- Add the parameters for the stored procedure here
    @customerid char(7), @orderid char(7) ,@productid char(7),
    @quantity int, @price money, @orderdate datetime

AS
DECLARE @stock_quantity int
BEGIN
SET NOCOUNT ON
SET XACT_ABORT ON
    BEGIN TRY
        BEGIN TRAN InsertOrder
        SELECT @stock_quantity = stock_quantity from product
        where productid = @productid
        INSERT  orders (customerid,orderid,productid,
        quantity,price,orderdate)
        VALUES (@customerid,@orderid,
        @productid,@quantity,@price,@orderdate)
        if @quantity > @stock_quantity
```

```
            BEGIN
            ROLLBACK TRAN InsertOrder
            PRINT 'Out of stock'
            END
        else

            COMMIT TRAN InsertOrder

    END TRY
    BEGIN CATCH
         SELECT ERROR_MESSAGE()
         IF @@TRANCOUNT>0
            ROLLBACK TRAN InsertOrder

    END CATCH

END
```

Step 4: Create a new query to test the rollback transaction of **uspInsertOrder** stored procedure. Click **Execute** or Press **F5**





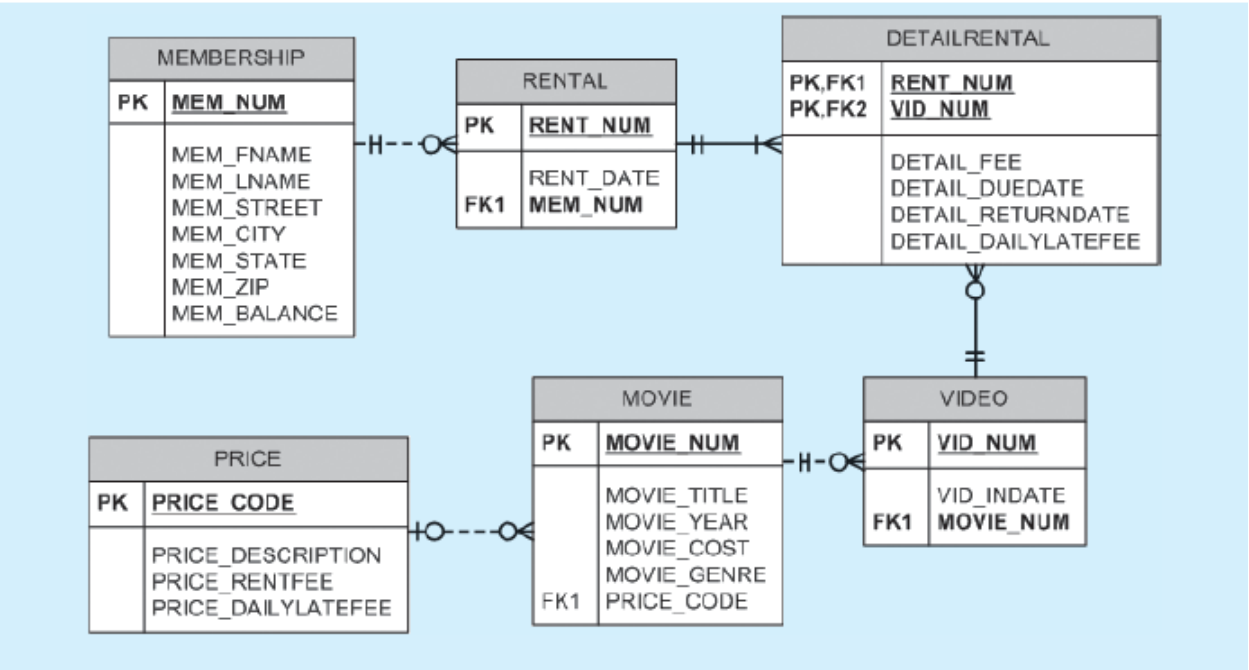
6. Database Output

Copy screenshot(s) of your output after completing the procedures provided in Part 5.

7. Supplementary Activity

Do the following tasks and copy screenshot(s) of your output.

1. Create a database TinyVideo.
2. Using the given ERD, create the following tables and relationship. Assign the appropriate data types.



3. Create a transaction to insert a new movie and display an appropriate error message if the user inserts a duplicate movie.

4. Create a transaction to insert a rental transaction and perform the following conditions:
    a. Verify that the membership number exists in the MEMBERSHIP table. If it does not exist, then a message should be displayed stating that the membership does not exist and no data should be written to the database.
    b. If the membership does exist, then retrieve the membership balance and display a message stating the balance amount as the previous balance. If the membership balance is greater than 500.00, allow the rental transaction. Otherwise, no data should be written to the database.
5. Create a transaction to insert a video transaction and perform the following conditions:
    a. Verify that the video number exists in the VIDEO table. If it does not exist, then display a message that the video does not exist, and do not write any data to the database.
    b. If the video number does exist, then verify that the VID_STATUS for that video is "IN".
        i. If the status is not "IN", then display a message that the return of the video must be entered before it can be rented again, and do not write any data to the database.
        ii. If the status is "IN", then retrieve the values of PRICE_RENTFEE, PRICE_DAILYLATEFEE, and PRICE_RENTDAYS associated with the video from the PRICE table. Calculate the due date for the video rental by adding the number of days found in PRICE_RENTDAYS above to 11:59:59PM (hours:minutes:seconds) on the current system date.

8. Conclusion

9. Assessment (Rubric for Laboratory Performance):

Activity No. 15	
Database Implementation	
Course Code: CPE011	**Program:**
Course Title: Database Management System	**Date Performed:**
Section:	**Date Submitted:**
Name:	**Instructor:**

1. Objective(s):

This activity aims to implement a software application that is connected to a SQL Database.

2. Intended Learning Outcomes (ILOs):

The students should be able to:
2.1 Setup the necessary configurations for connection to a MySQL database and SQL Server database respectively.
2.2 Create a software application that can connect to a MySQL database and a SQL Server database using the PyMySQL and the pyodbc module respectively.
2.3 Execute SQL Statements and Stored Procedures through the software application's database connection.

3. Discussion:

An application which can either be a desktop, web, or mobile application needs a database and a database management system in order to manage large amounts of records of their users. To connect an application to a database management system such as MySQL or MS SQL, these are the following steps to be followed:

1. Install the correct drivers and modules/packages that will allow your application's connection to the application.
2. Create a connection object which can be imported from the module/package then use your configuration of the database management system you wish to connect to.
   Common parameters include:

   host=URL/Address of the SQL Server (ex. "localhost"),
   port=port number of the SQL Server (ex. 3306),
   db=database name (ex. "mydb"),
   user=username (ex. "root"),
   password=userpassword, (optional)

3. Create a cursor object which can be imported from the module/package that will serve as your SQL Controller by providing a method that will allow you to execute SQL Commands and Stored Procedures as well as handle retrieval of values from the database.
4. Close the connections after performing the SQL Commands or Stored Procedures.

The activity will be following this pattern throughout the procedure.

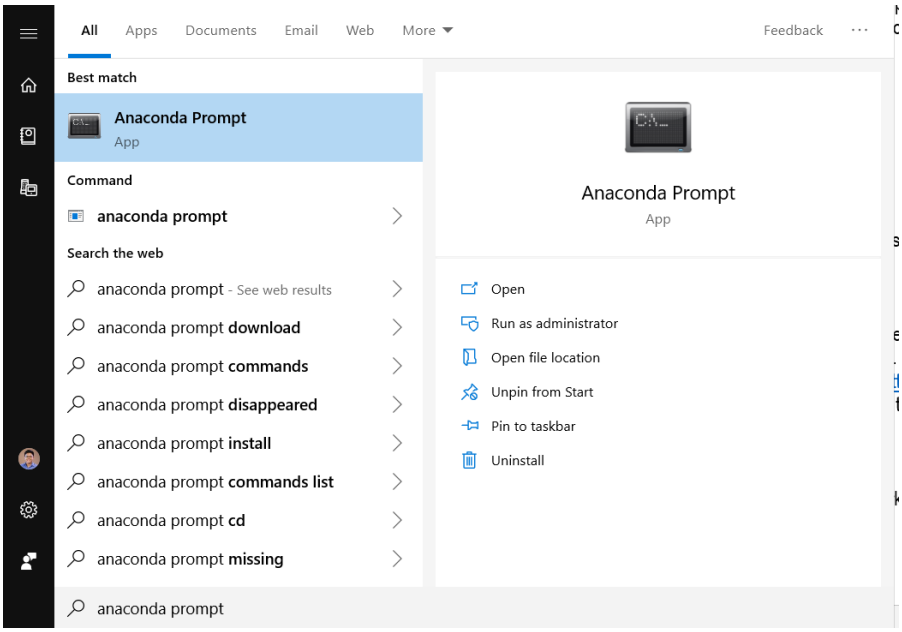
The next set of discussions will focus more on the installation of required drivers and software for a successful Python and SQL Database connection.

In the Python programming language, there are several modules/packages available to install in order to establish connection to a SQL Database such as MySQL or MS SQL. In MySQL, one of the widely used modules is PyMySQL which is a pure-MySQL client (more information can be found here: https://github.com/PyMySQL/PyMySQL). To install this library, simply open your computer or workstation's command prompt or terminal that has a Python interpreter installed and run the command:

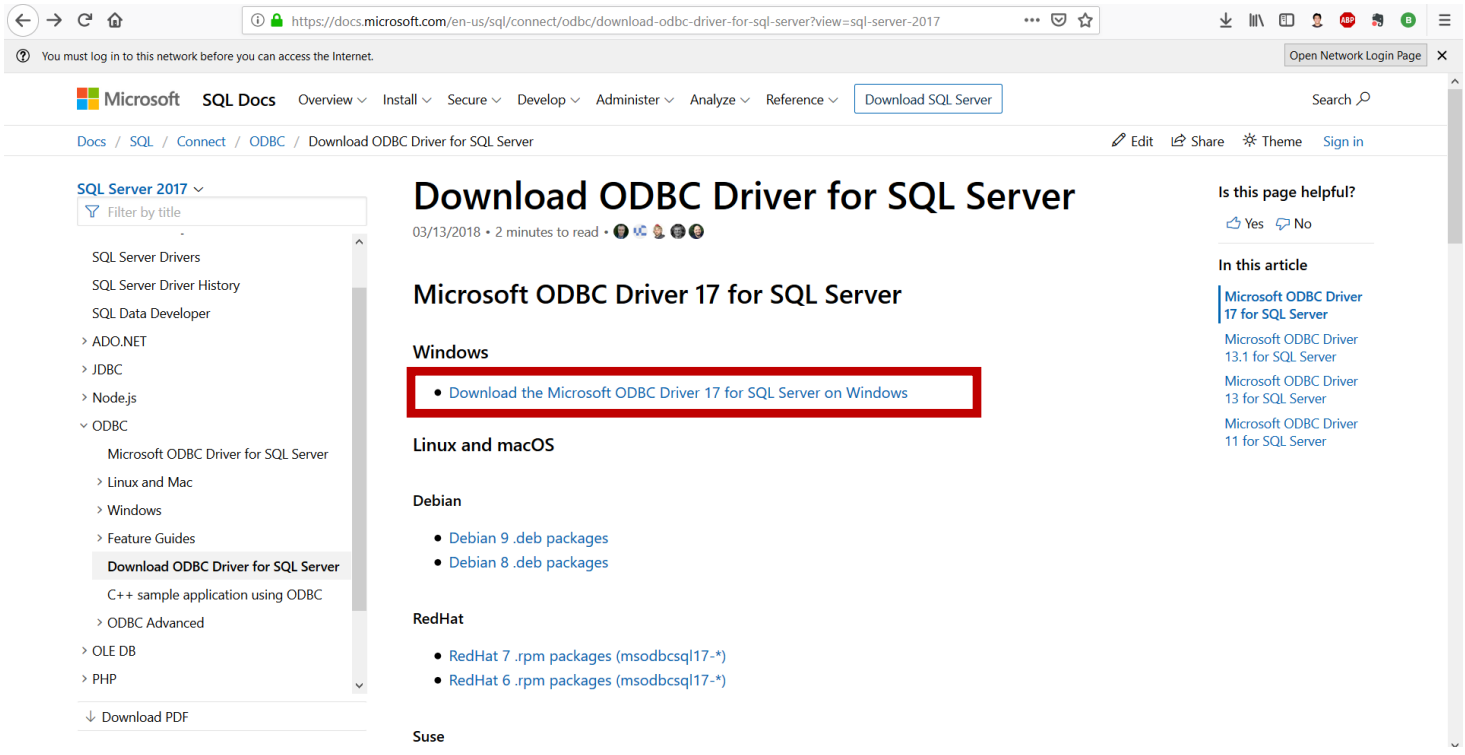
pip install pymysql

The module wheel file (.whl) can also be downloaded here: https://pypi.org/project/PyMySQL/#files

For Anaconda distributions, if this does not work on your command prompt or terminal, open your Anaconda Prompt and run the pip install command there.



In MS SQL or SQL Server, we will use the ODBC driver or the Open Database Connectivity (ODBC) which is an interface provided by Microsoft for allowing applications to access data from database management systems using SQL. We first need to install this free ODBC driver from the Microsoft website using this link: https://docs.microsoft.com/en-us/sql/connect/odbc/download-odbc-driver-for-sql-server?view=sql-server-2017



After successful installation, the Python module for connecting to a SQL Server is the pyodbc module. Which can be installed using the steps previously performed in installing the PyMySQL module:

pip install pyodbc

The wheel file (.whl) is also available for download through this link: https://pypi.org/project/pyodbc/#files

For Anaconda distributions, if this does not work on your command prompt or terminal, open your Anaconda Prompt and run the pip install command there.

More information about the methods and parameters inside the pydobc module can be found here: https://github.com/mkleehammer/pyodbc/wiki

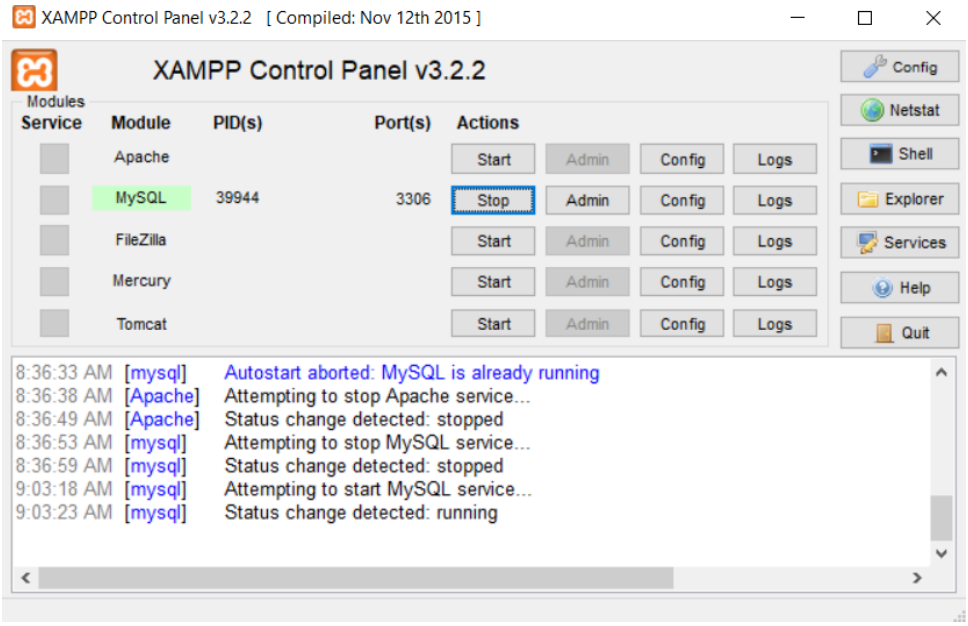
4. Resources:

Personal Computer with installed Python 3, SQL Server and XAMPP
PyMySQL Module
Pyodbc module and the ODBC driver

5. Procedure:

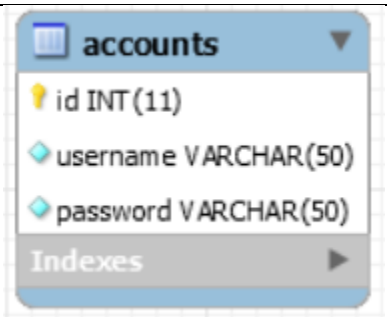
Database Connection
MySQL Database Connection
I. Initial Setup

1. Ensure that the MySQL service is running through XAMPP.



Note: The XAMPP should indicated that the service is running before continuing the activity

2. Create a new database mydb_<lastname> with the table Accounts using SQL Commands.
   Accounts table should contain: id, username, password. **Note: id and username should be unique.**



3. Insert at least 2 records into the Accounts table.
4. Open your Python editor and copy the following code to your program and then run/execute the program shown below:

```
import pymysql
# Creation of PyMySQL Connection Object
try:
    conn = pymysql.connect(host="localhost",port=3306,db="mydb",user="root",password="")
except pymysql.MySQLError as e:
    print(e)
    sys.exit()
finally:
    print("Connection established sucessfully")
    conn.close() # You always have to close the connection
```

5. The connection should be established successfully.

II. Executing SQL and Stored Procedures

A. Creating the SQLDBConnector

1. Create a SQLDBConnector.py file in your workspace folder.
2. Type the following source codes to the SQLDBConnector.py.

```
import pymysql
import sys

class SQLDBConnector():
    def __init__(self, db="",username="",password="",host="",port=3306):
        self.db = db
        self.username = username
        self.password = password
        self.host = host
        self.port = port
```

```
    def executequery(self,sql="",parameters=()):
        try:
            self.conn = pymysql.connect(db=self.db,user=self.username
                                        ,password=self.password,host=self.host
                                        ,port=self.port)
            self.cursor = self.conn.cursor()
            self.cursor.execute(sql,parameters)
            result = self.cursor.fetchall()
            self.conn.close()
            return result
        except pymysql.MySQLError as e:
            print(e)
            sys.exit()


    def callprocedure(self,procname="",parameters=()):
        try:
            self.conn = pymysql.connect(db=self.db,user=self.username
                                        ,password=self.password,host=self.host
                                        ,port=self.port)
            self.cursor = self.conn.cursor()
            self.cursor.callproc(procname,parameters)
            result = self.cursor.fetchall()
            self.conn.close()
            return result
        except pymysql.MySQLError as e:
            print(e)
            sys.exit()
```

3. Save the changes to the file. You may need to modify this code for INSERT, UPDATE, DELETE statements to work by using connection.commit() method.

B. Creating the Login Window

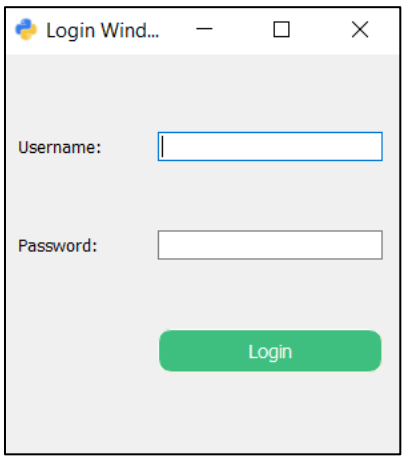
1. Create a login.py file in the same workspace folder.
2. Type the following source codes to the SQLDBConnector.py.

```
import sys
from PyQt5.QtWidgets import QMainWindow, QApplication, QPushButton,
QLineEdit, QLabel, QGridLayout, QWidget, QMessageBox
from PyQt5.QtCore import pyqtSlot
from PyQt5.QtGui import QIcon

from SQLDBConnector import SQLDBConnector

class App(QWidget):

    def __init__(self):
        super().__init__()
        self.title= "Login Window"
        # Set the window position, dimension, and icon variables
        self.x=200
        self.y=200
        self.width=300
        self.height=300
        self.icon='pythonico.ico'
        # Initialize GUI components
        self.initUI()
```

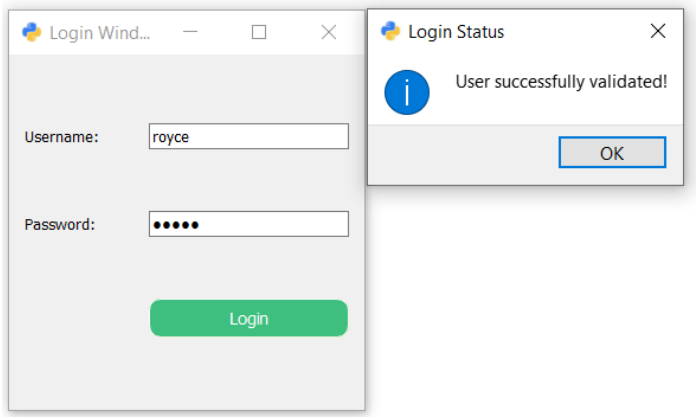
```
    def initUI(self):
        self.setWindowTitle(self.title)
        self.setGeometry(self.x,self.y,self.width,self.height)
        self.setWindowIcon(QIcon(self.icon))
        # Create the grid layout and its components
        self.createGridLayout()
        self.setLayout(self.layout)
        # Display the window
        self.show()

    def createGridLayout(self):
        self.layout = QGridLayout()

        self.layout.setColumnStretch(1,2)

        self.usernamelbl = QLabel("Username: ", self)
        self.usernamefield = QLineEdit(self)
        self.passwordlbl = QLabel("Password: ", self)
        self.passwordfield = QLineEdit(self)
        self.passwordfield.setEchoMode(QLineEdit.Password)
        self.loginbutton = QPushButton('Login', self)
        self.loginbutton.setStyleSheet("""QPushButton {background-color: rgba(63, 191, 127, 1);
                                        color: white;
                                        border-style: outset;
                                        border-width: 0.7px;
                                        border-radius: 10px;
                                        border-color: beige;
                                        min-width: 9em;
                                        max-width: 5em;
                                        max-height: 0.99em;
                                        min-height: 0em;
                                        padding: 7px;
                                        font-size: 14px;
                                        }
                                        QPushButton:hover { background-color:rgba(189,191,63,0.8); }
                                        QPushButton:pressed{background-color: rgba(63, 127, 191, 0.4);
                                        border-style: inset;}
                                        """)
        self.loginbutton.clicked.connect(self.authenticateUser)
        self.layout.addWidget(self.usernamelbl,0,1)
        self.layout.addWidget(self.usernamefield, 0,2)
        self.layout.addWidget(self.passwordlbl, 1, 1)
        self.layout.addWidget(self.passwordfield, 1, 2)
        self.layout.addWidget(self.loginbutton, 2, 2)

    def authenticateUser(self):
        if len(self.usernamefield.text())==0 or len(self.passwordfield.text())==0:
            QMessageBox.warning(self, "Login Status", "Both username and password should have values!",
            QMessageBox.Ok, QMessageBox.Ok)
        else:
            self.conn1 = SQLDBConnector(db="mydb",username="root",password="",host="localhost",port=3306)
            self.result = self.conn1.executequery("SELECT * FROM Accounts WHERE username=%s and password=%s",
            parameters=(self.usernamefield.text(),self.passwordfield.text()))
            if len(self.result) > 0:
                QMessageBox.information(self, "Login Status", "User successfully validated!", QMessageBox.Ok,
                QMessageBox.Ok)
            else:
                QMessageBox.warning(self, "Login Status", "User validation failed!", QMessageBox.Ok,
                QMessageBox.Ok)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = App()
    sys.exit(app.exec_())
```

Note: You will need to replace the SQLDBConnector object parameters for a successful connection.

3. Save the changes to the file and run using the Python editor or the command prompt.
4. The output should appear similar to the image below.

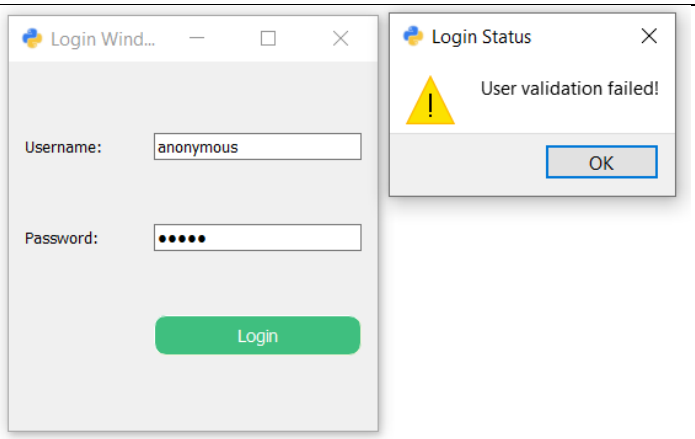


5. Enter a username and password that matches records in the created Accounts table. The output should look similar to the one below.



You may use one of the two accounts instructed to be created earlier in the activity. Both of them should be successfully validated.

6. Enter a username and password that does not match a record in the created Accounts table. The output should look similar to the one below.



C. Executing Stored Function / Stored Procedures

1. Using the MySQL Workbench query editor or the Command Terminal, copy the following code below.

Stored function | SQL File 3*

Limit to 1000 rows

```
USE `mydb`;
-- SHOW TABLES;

DELIMITER $$
USE `mydb`$$
CREATE FUNCTION `authenticateUser` (username VARCHAR(50), password VARCHAR(50))
RETURNS INTEGER
BEGIN
IF EXISTS(SELECT * FROM Accounts WHERE username=username AND password=password) THEN
    RETURN 1;
ELSE
    RETURN 0;
END IF;
END$$

DELIMITER ;
```

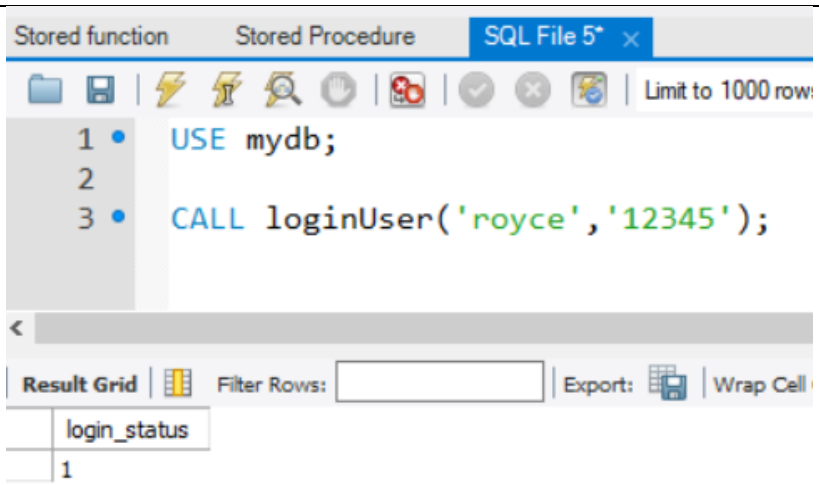
2. Execute the SQL code.
3. Using the MySQL Workbench query editor or the Command Terminal, copy the following code below.

Stored function | Stored Procedure | SQL File 5*

Limit to 1000 rows

```
USE mydb;

DELIMITER //
CREATE PROCEDURE loginUser(IN username VARCHAR(50), IN password VARCHAR(50))
BEGIN
  SELECT authenticateUser(username,password) as 'login_status';
END //
DELIMITER ;
```

4. Execute the SQL code.
5. Test the Stored Procedure and Function by copying the following code and executing it.



6. Inserting an existing Account record will result in 1 else it will be 0.

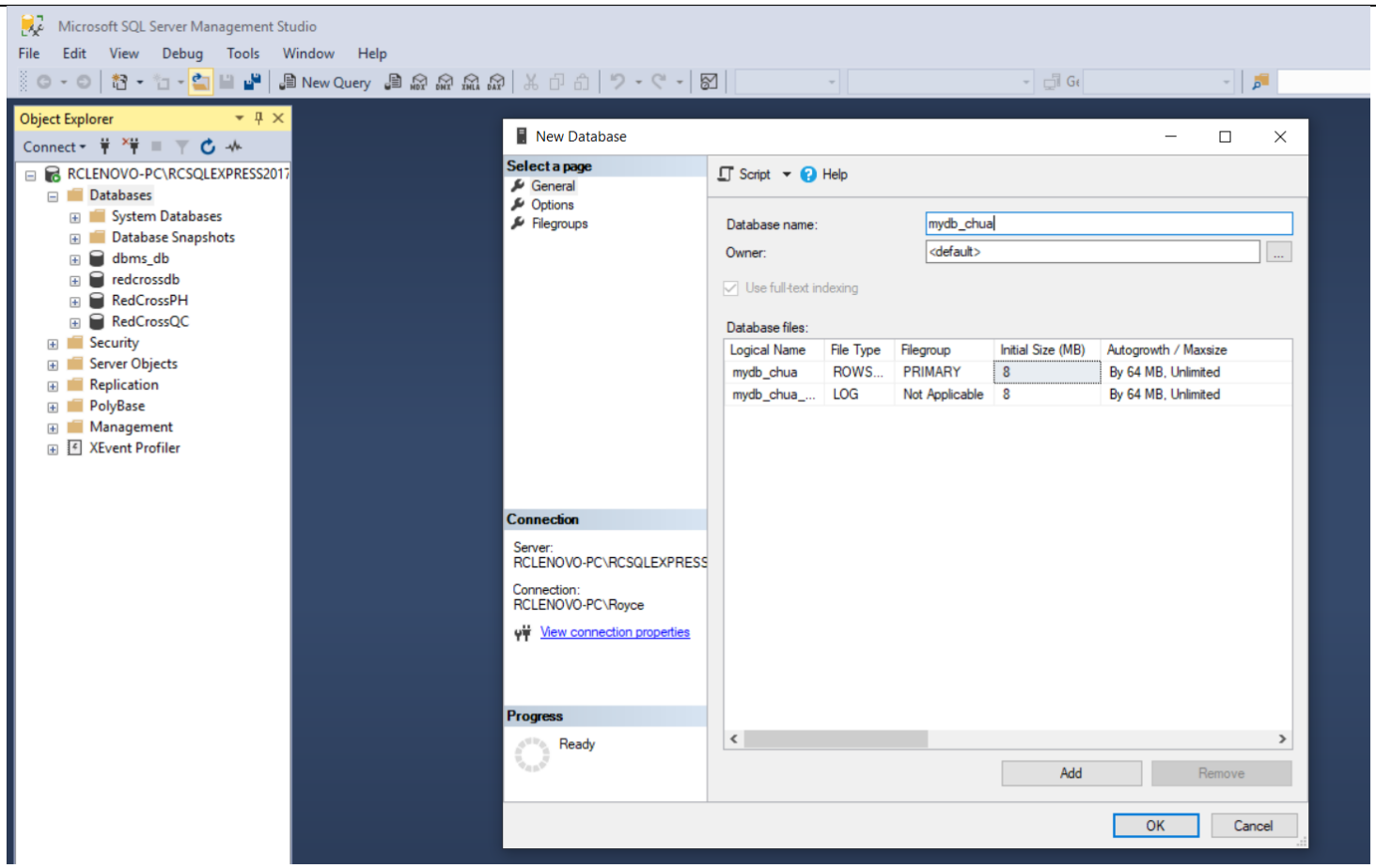
7. Modify the login.py code by replacing the codes in the Python function authenticateUser() with the code below. The ones emphasized with red lines are the codes that were changed.

```
def authenticateUser(self):
    if len(self.usernamefield.text())==0 or len(self.passwordfield.text())==0:
        QMessageBox.warning(self, "Login Status", "Both username and password should have values!",
        QMessageBox.Ok, QMessageBox.Ok)
    else:
        self.conn1 = SQLDBConnector(db="mydb",username="root",password="",host="localhost",port=3306)
        self.result = self.conn1.callprocedure('authenticate_user',parameters=(self.usernamefield.text(),
        self.passwordfield.text()))
        if self.result == 1:
            QMessageBox.information(self, "Login Status", "User successfully validated!", QMessageBox.Ok,
            QMessageBox.Ok)
        else:
            QMessageBox.warning(self, "Login Status", "User validation failed!", QMessageBox.Ok,
            QMessageBox.Ok)
```

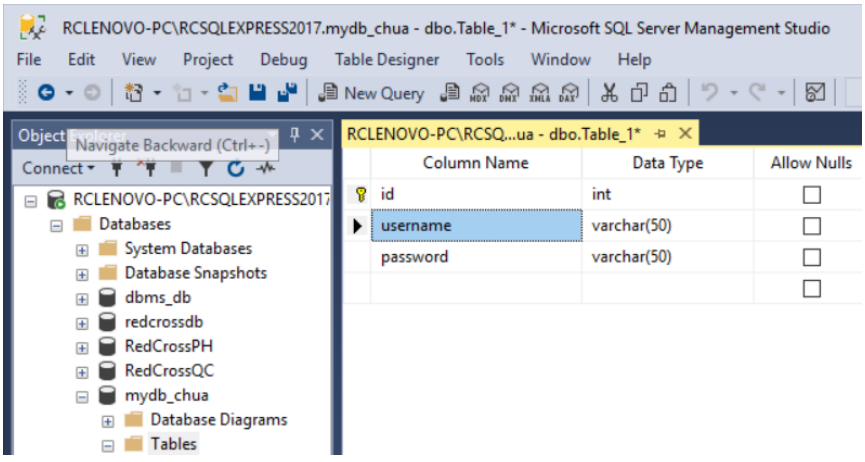
8. Run login.py and validate the result again. The output should be the same with procedures 5 and 6.

MS SQL Database Connection

1. Login to the SQL Server Management Studio using your credentials.
2. Create the database mydb_<lastname>.



3. Create the Accounts table and insert at least two values to the table.



4. Insert at least two values to the table.

```
use mydb_chua;

SELECT * FROM Accounts;
```

146 %

Results | Messages

	id	username	password
1	1	royce	12345
2	2	john	54321

5. Create the scalar-valued stored function named **authenticateUser**.

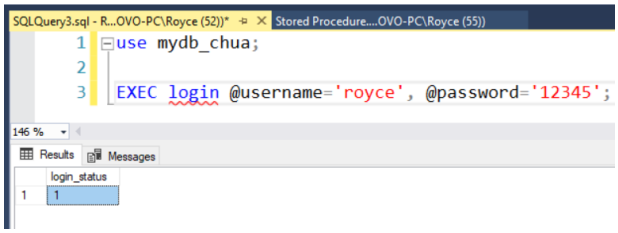
```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:           Royce Chua
-- Create date: September 26, 2019
-- Description:      Authenticates a user based on username and password
-- =============================================
CREATE FUNCTION authenticateUser
(
       -- Add the parameters for the function here
       @username VARCHAR(50), @password VARCHAR(50)
)
RETURNS INT
AS
BEGIN
       -- Declare the return variable here
       DECLARE @login_status INT=0;

       -- Add the T-SQL statements to compute the return value here
       IF EXISTS(SELECT * FROM Accounts WHERE username=@username AND password=@password)
              SET @login_status = 1;

       RETURN @login_status;
END
GO
```

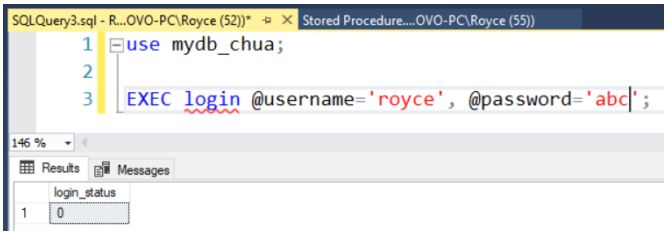
6. Create the stored procedure login.

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:           Royce Chua
-- Create date: September 26, 2019
-- Description:      This procedure executes
-- =============================================
CREATE PROCEDURE login
       -- Add the parameters for the stored procedure here
       @username VARCHAR(50) = '',
       @password VARCHAR(50) = ''
AS
BEGIN
       -- SET NOCOUNT ON added to prevent extra result sets from
       -- interfering with SELECT statements.
       SET NOCOUNT ON;

    -- Insert statements for procedure here
```

```
        SELECT authenticateUser(@username, @password) as 'login_status';
    END
    GO
```

7. Test the stored procedure and function.





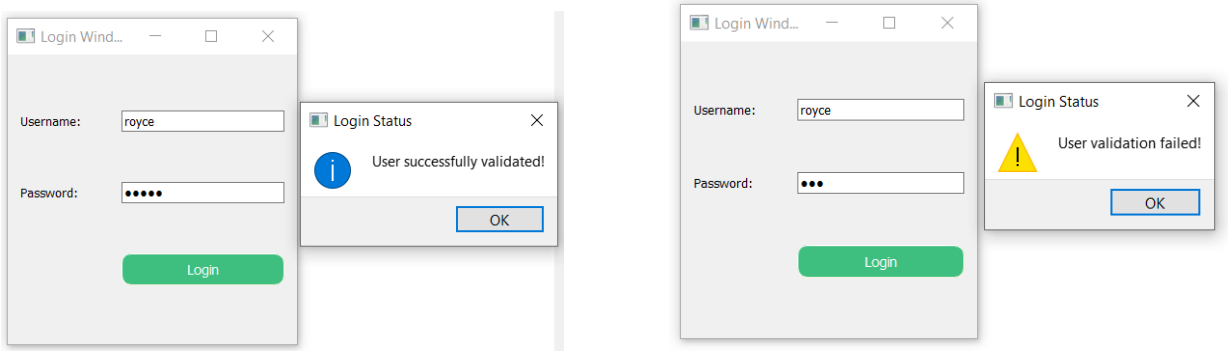
8. The result of the login_status should be similar earlier on the MySQL.
9. Modify the SQLDBConnector.py with the code below

```
import pyodbc
import sys

class SQLDBConnector():
    def __init__(self, db="",uid="",password="",server=""):
        self.db = db
        self.uid = uid
        self.password = password
        self.server = server

    def executequery(self,sql="",parameters=()):
        try:
            self.conn = pyodbc.connect("DRIVER={SQL Server};SERVER=%s;DATABASE=%s;UID=%s;PWD=%s"% (
                self.server, self.db, self.uid, self.password
            ))
            self.cursor = self.conn.cursor()
            self.cursor.execute(sql,parameters)
            result = self.cursor.fetchall()
            self.conn.close()
            return result
        except pyodbc.DatabaseError as e:
            print(e)
            sys.exit()

    def callprocedure(self,procname="",parameters=()):
        try:
            self.conn = pyodbc.connect("DRIVER={SQL Server};SERVER=%s;DATABASE=%s;UID=%s;PWD=%s"% (
                self.server, self.db, self.uid, self.password
            ))
            self.cursor = self.conn.cursor()
            if len(parameters)>0:
                param_no = "?,"*len(parameters)
                self.cursor.execute("{CALL %s (%s)}" % (procname,param_no[:len(param_no)-1]),parameters)
            else:
                self.cursor.execute("{CALL %s}" % procname)
            result = self.cursor.fetchall()
            self.conn.close()
            return result
        except pyodbc.DatabaseError as e:
            print(e)
            sys.exit()
```

10. Modify the authenticate_user() function in the login.py by using the code below.

```
def authenticateUser(self):
        if len(self.usernamefield.text())==0 or len(self.passwordfield.text())==0:
            QMessageBox.warning(self, "Login Status", "Both username and password should have
values!", QMessageBox.Ok, QMessageBox.Ok)
        else:
            self.conn1 =
SQLDBConnector(db="mydb_chua",uid="dbms_user",password="123456",server="RCLENOVO-PC\RCSQLEXPRESS2017")
            self.result =
self.conn1.callprocedure('login',parameters=(self.usernamefield.text(),self.passwordfield.text()))
            if self.result[0][0] == 1:
                QMessageBox.information(self, "Login Status", "User successfully validated!",
QMessageBox.Ok, QMessageBox.Ok)
            else:
                QMessageBox.warning(self, "Login Status", "User validation failed!", QMessageBox.Ok,
QMessageBox.Ok)
```

11. Run login.py through your Python editor or command prompt. The output should look similar to the results of procedure 7 but this time it is displayed through the GUI.



6. Database Output

Copy screenshot(s) of your output after completing the procedures provided in Part 5 and write your observations in each screenshot.

7. Supplementary Activity

Do the following tasks and copy screenshot(s) of your output using Microsoft SQL Server Management Studio.

1. The database that will be used is the ABCLibrary database. You may use an existing ABCLibary database or create a new one.
2. Create a GUI application that can insert, delete and modify the book information in ABCLibrary Database.
3. Create stored procedures that can insert, delete and modify the book information using the book table in ABCLibrary database.
4. Call the stored procedures from your book information.

8. Conclusion

9. Assessment (Rubric for Laboratory Performance):