Relational Database Management System

Contents

1. Introduction

1.1 Database Management System

A database management system (DBMS) is system software for creating and managing databases. The DBMS provides users and programmers with a systematic way to create, retrieve, update and manage data. A DBMS makes it possible for end users to create, read, update and delete data in a database.

1.2 Relational Database Management System

A relation database refers to a database that stores data in a structured format, using rows and columns. This makes it easy to locate and access specific values within the database. It is "relational" because the values within each table are related to each other. Tables may also be related to other tables. The relational structure makes it possible to run queries across multiple tables at once.

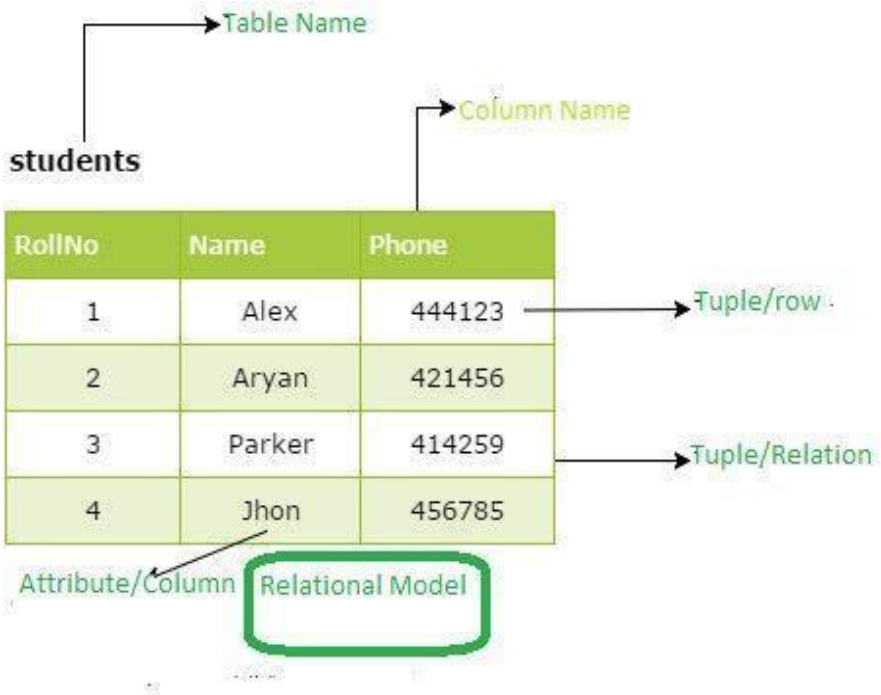


Figure. Example of Relational Database Management System

1.3 Introduction to SQL Server Management Studio (SSMS)

SQL Server Management Studio is a GUI tool included with SQL Server for configuring, managing, and administering all components within Microsoft SQL Server. The tool includes both script editors and graphical tools that work with objects and features of the server. It can be installed on the server or on the client (or both).

1.3.1 Different Edition of SQL Server

SQL Server is available in various editions. The following will show multiple editions with its features.

(1) **Enterprise**: This is the top-end edition with a full feature set.

(2) **Standard:** This has less features than Enterprise, when there is no requirement of advanced features.

(3) **Workgroup**: This is suitable for remote offices of a larger company.

(4) **Web**: This is designed for web applications.

(5) **Developer**: This is similar to Enterprise, but licensed to only one user for development, testing and demo. It can be easily upgraded to Enterprise without reinstallation.

(6) **Express**: This is free entry level database. It can utilize only 1 CPU and 1 GB memory, the maximum size of the database is 10 GB.

(7) **Compact**: This is free embedded database for mobile application development. The maximum size of the database is 4 GB.

(8) **Datacenter**: The major change in new SQL Server 2008 R2 is Datacenter Edition. The Datacenter edition has no memory limitation and offers support for more than 25 instances.

(9) **Business Intelligence:** Business Intelligence Edition is a new introduction in SQL Server 2012. This edition includes all the features in the Standard edition and support for advanced BI features such as Power View and PowerPivot, but it

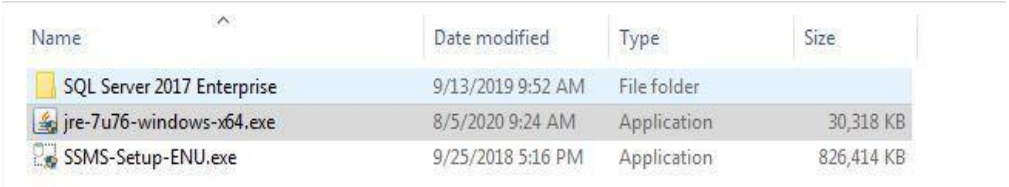
lacks support for advanced availability features like Always on Availability Groups and other online operations.

10) **Enterprise Evaluation:** The SQL Server Evaluation Edition is a great way to get a fully functional and free instance of SQL Server for learning and developing solutions. This edition has a built-in expiry of 6 months from the time that you install it.

2 Microsoft SQL Server Installation

Step 1: Install Jre-7y76-windows-x64

1. Before Install SQL SERVER , should be install jre-7u76-windows-x64.

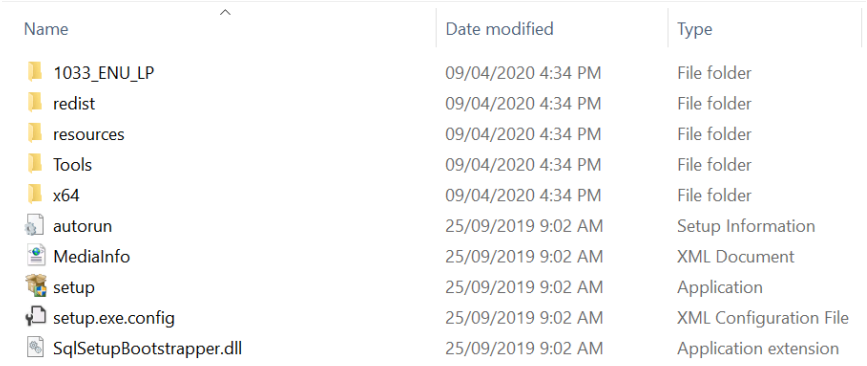


2. Java Setup Installation appear and click install.

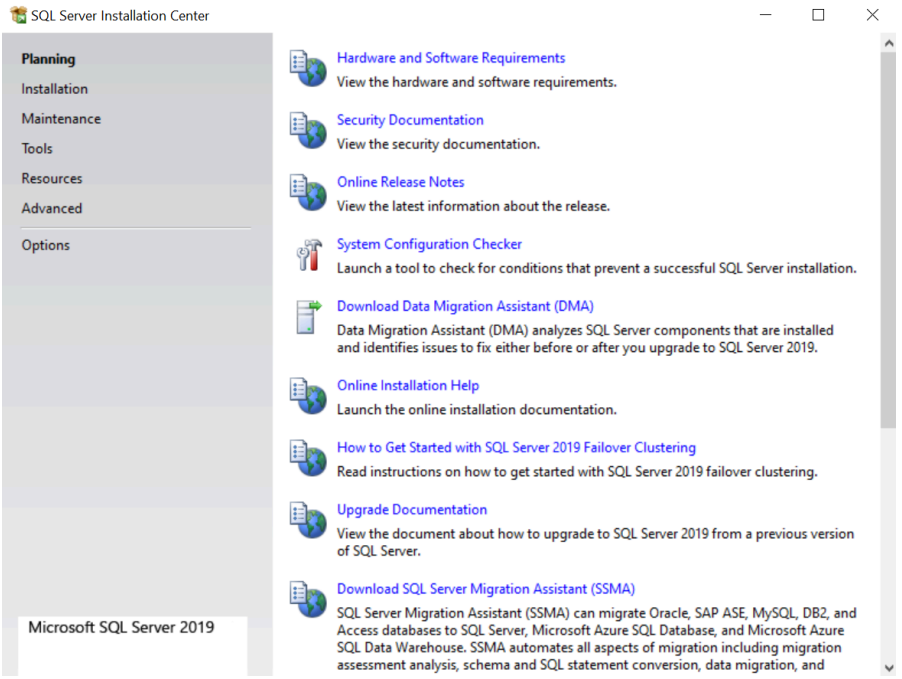
3. Installation completed.

Step2: Install SQL Server 2019

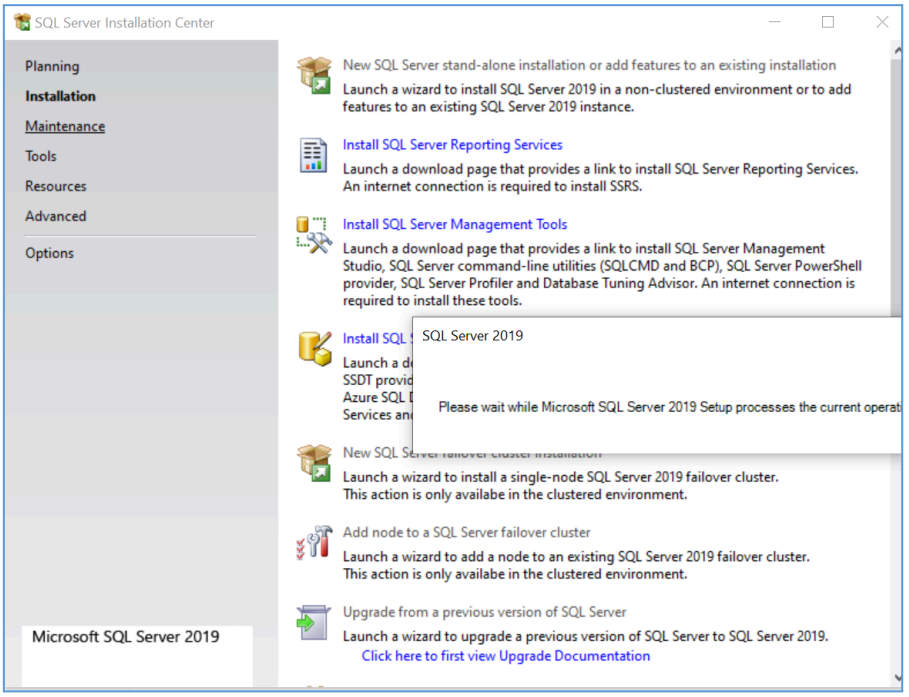
1. Open SQL SERVER 2019 Installer Folder and run setup file.



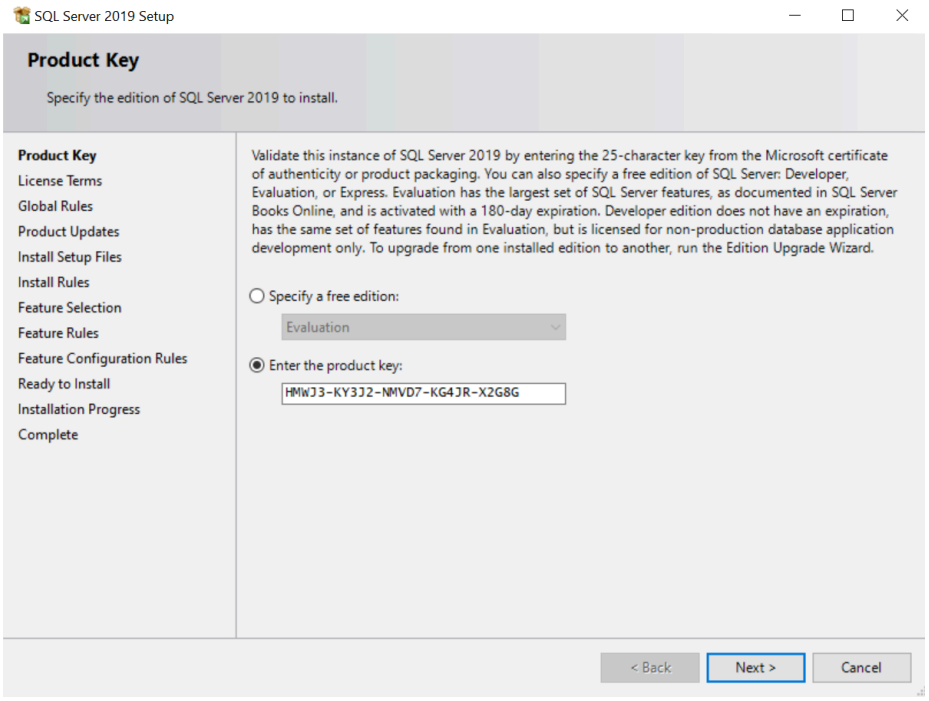
2. SQL Server Installation Center appears.



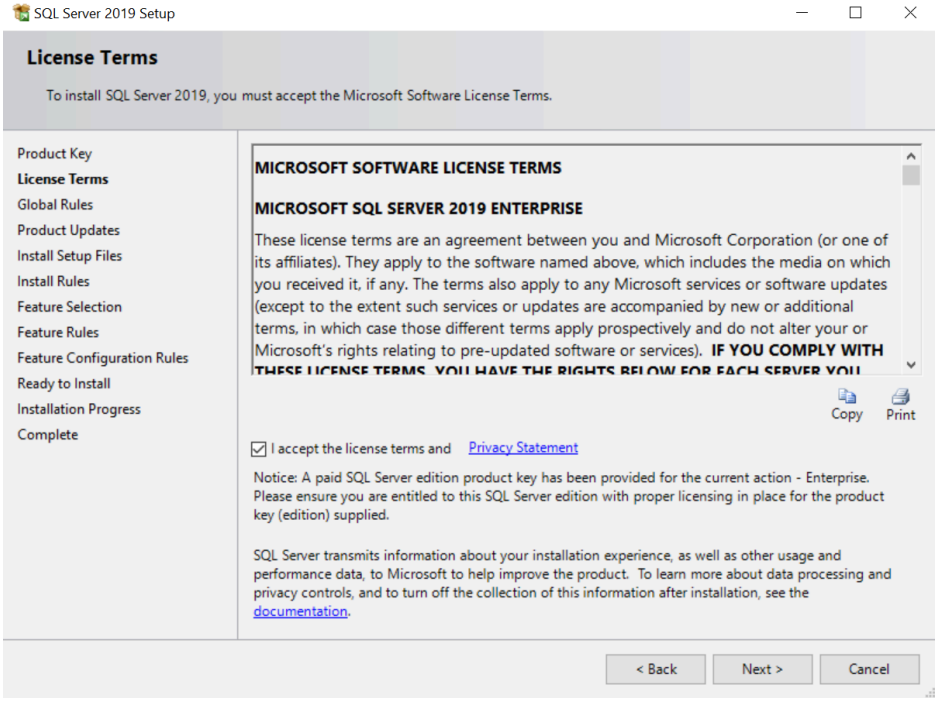
3. Go to Installation and click on New SQL Server stand-alone installation or add features to an existing installation.



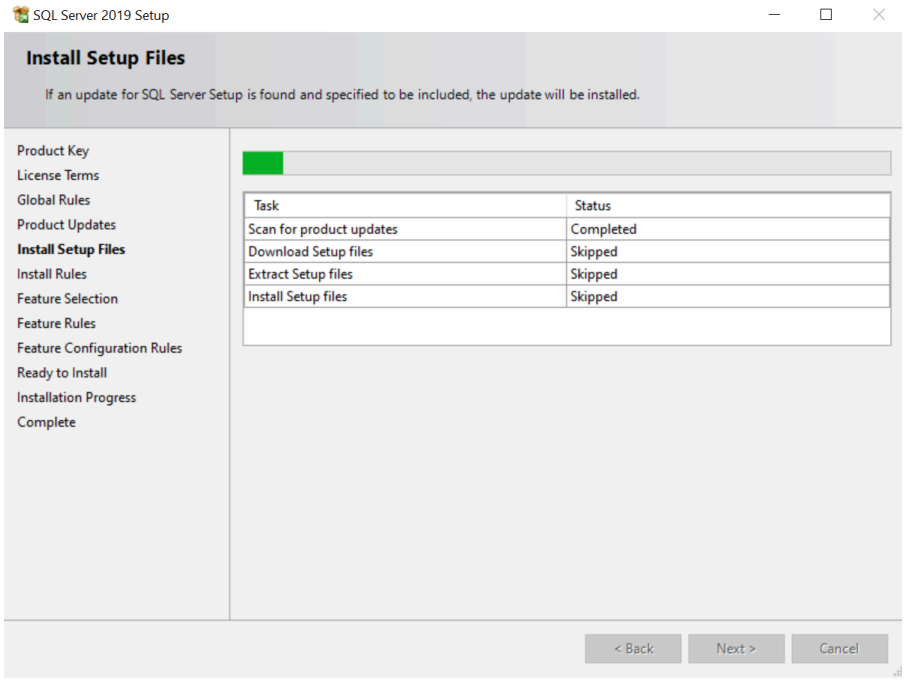
4. Enter The Product Key



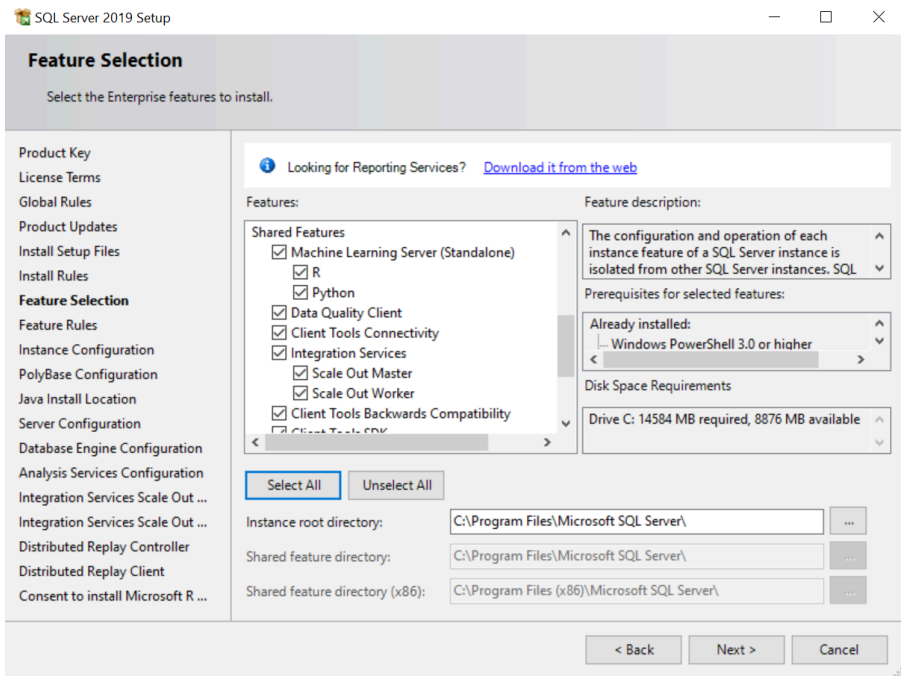
5. Accept the license terms and privacy statement



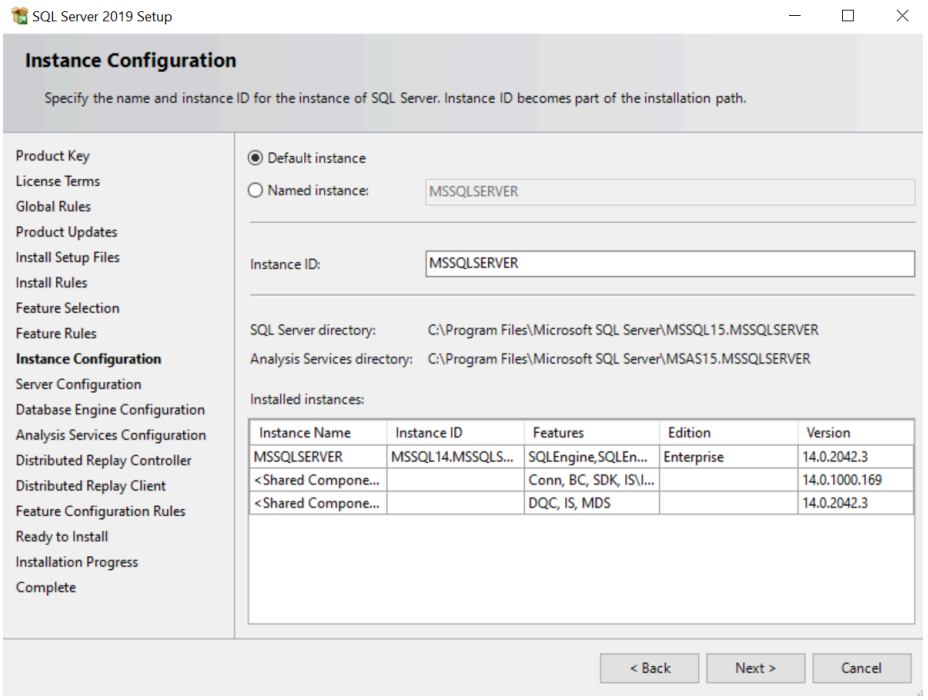
Installation Step



6. The Feature Selection screen appears. Click Select All and click Next.



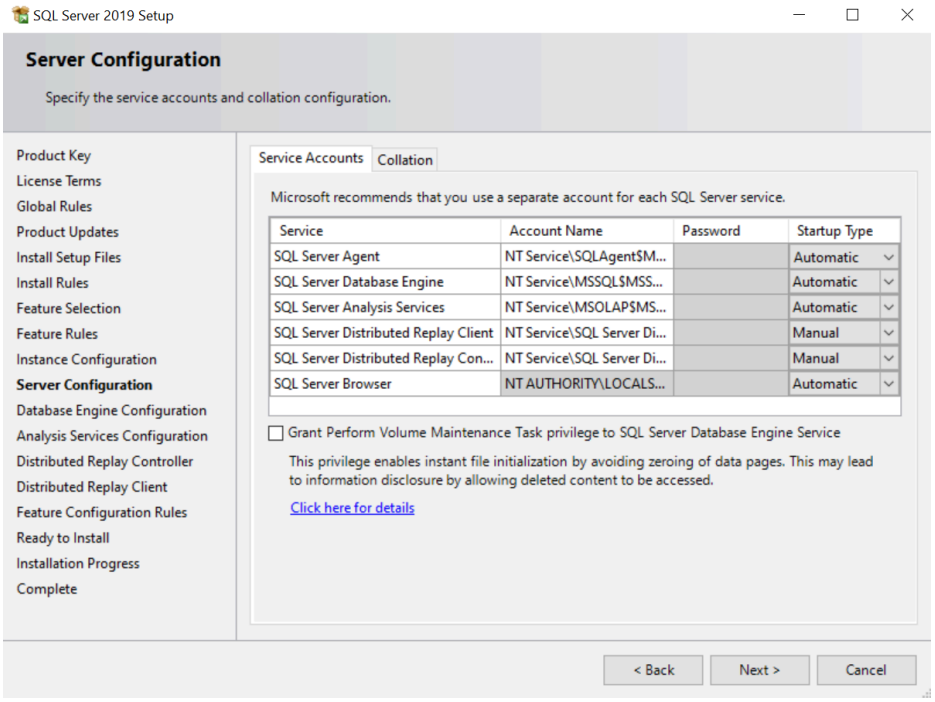
7. Choose Default instance and click on Next.



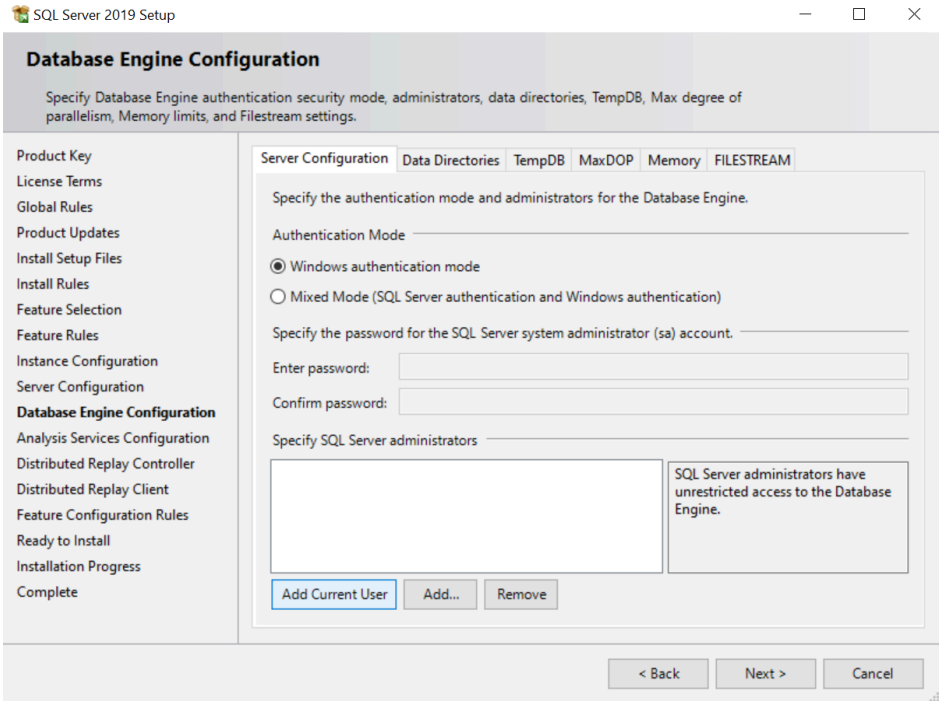
8. The Server Configuration screen appears. Select Account Names as follows and click Next:

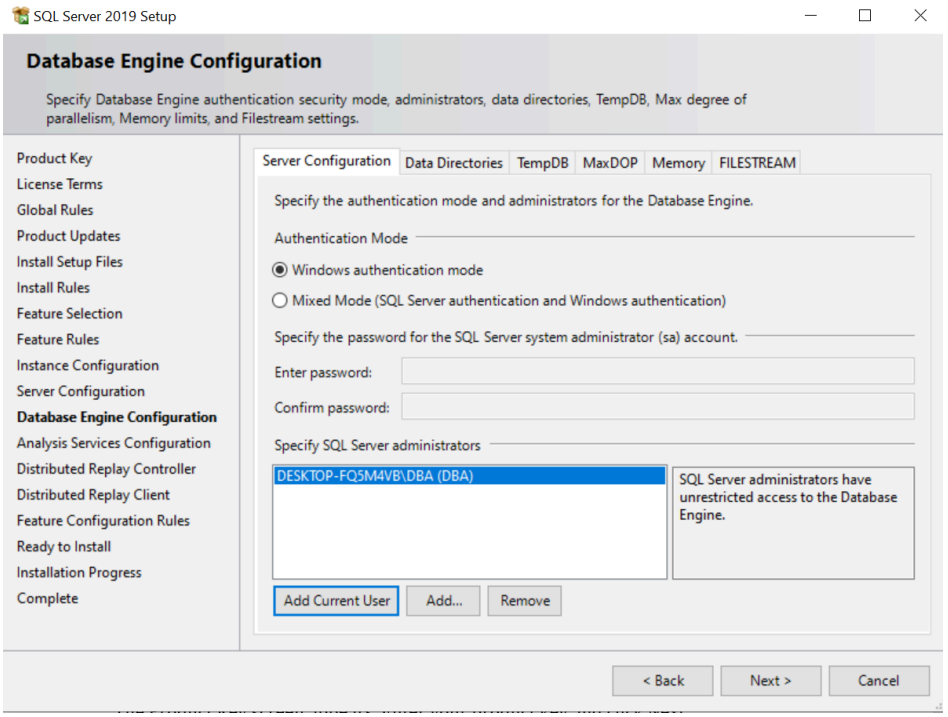
   SQL Server Agent: NT Service\SQLAgent$SQLSERVER2019
   SQL Server Database Engine: NT Service\MSSQL$SQLSERVER2019SQL Server Analysis

   SQL Server Analysis Services: NT Service\MSOLAP$SQLSERVER2019

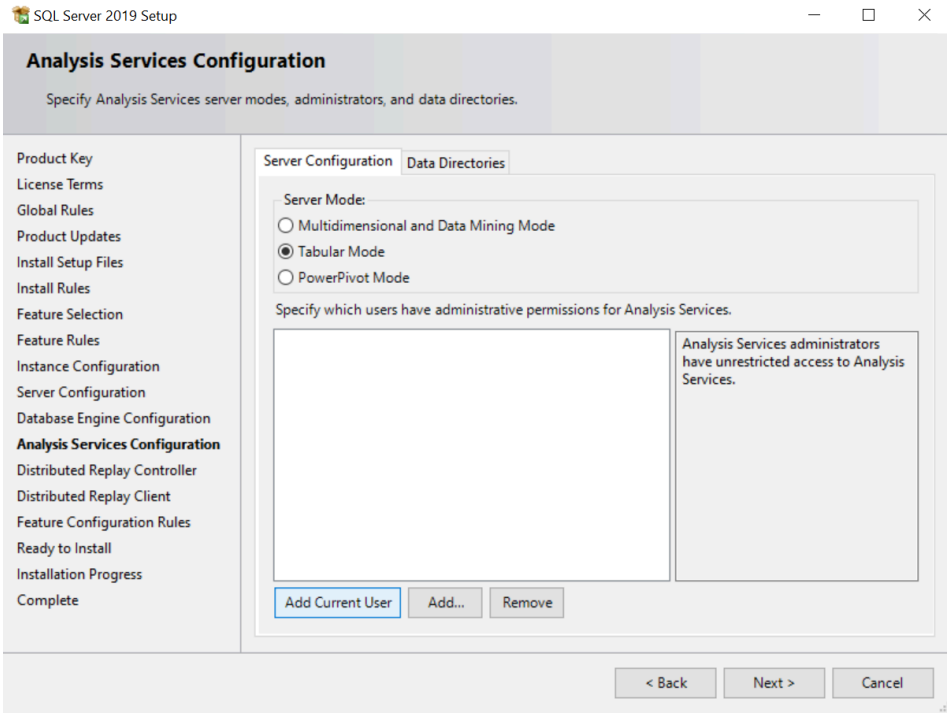


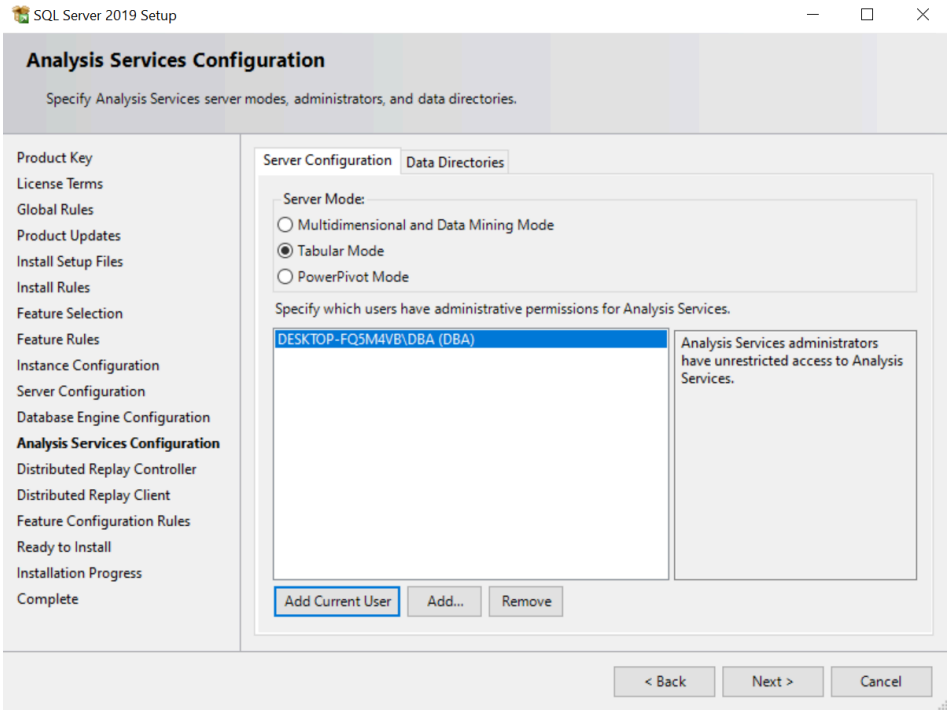
9. Choose Windows authentication mode and click on Add Current User and then click Next.



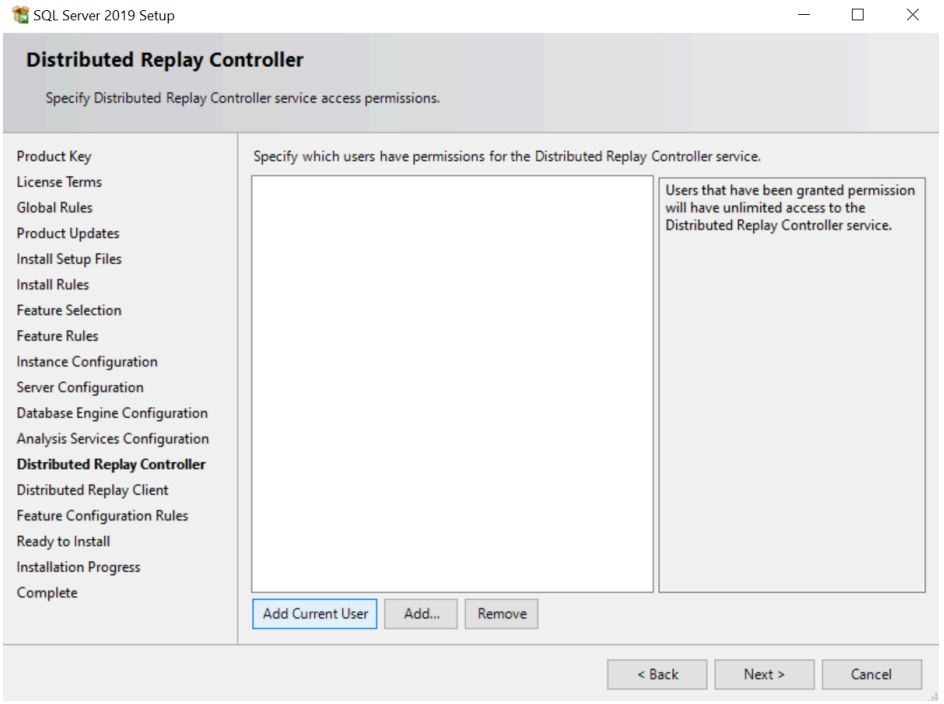


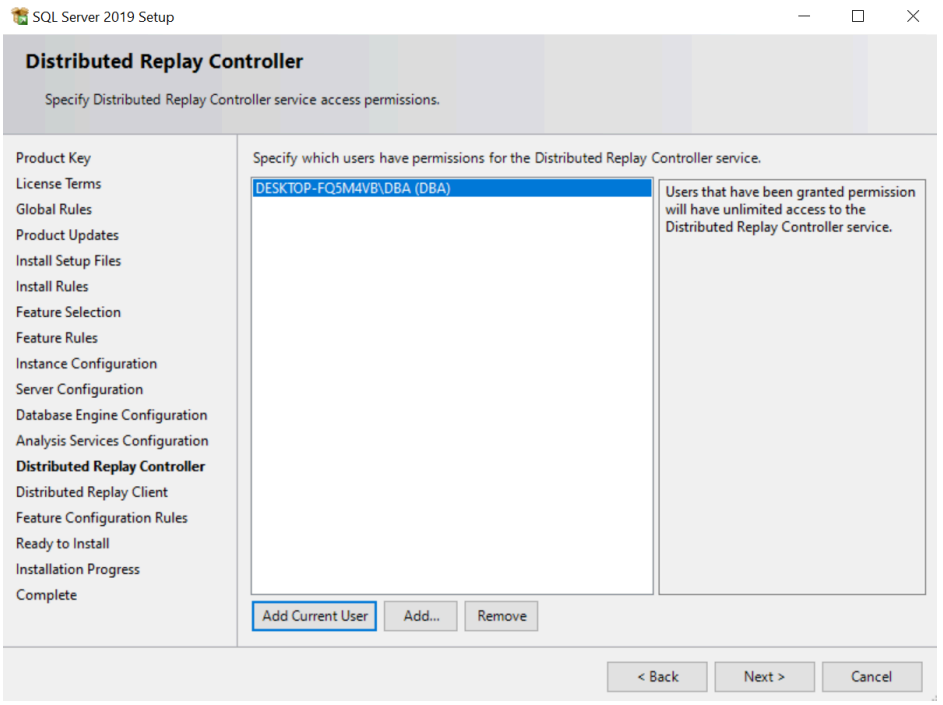
10. Choose Tabular Mode and click on Add Current User and then click Next.



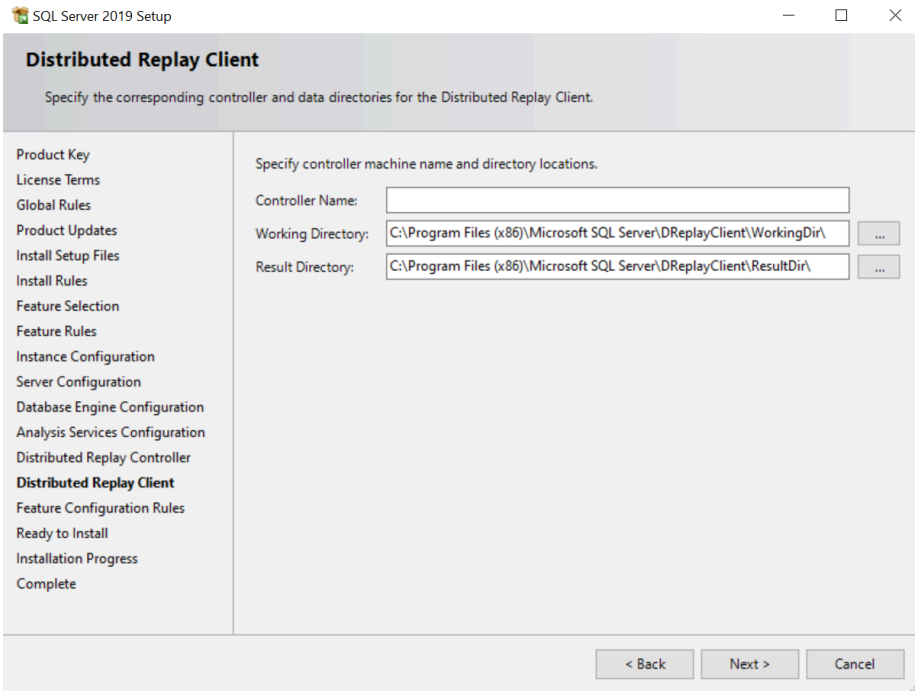


11. Add Current user and click on Next.

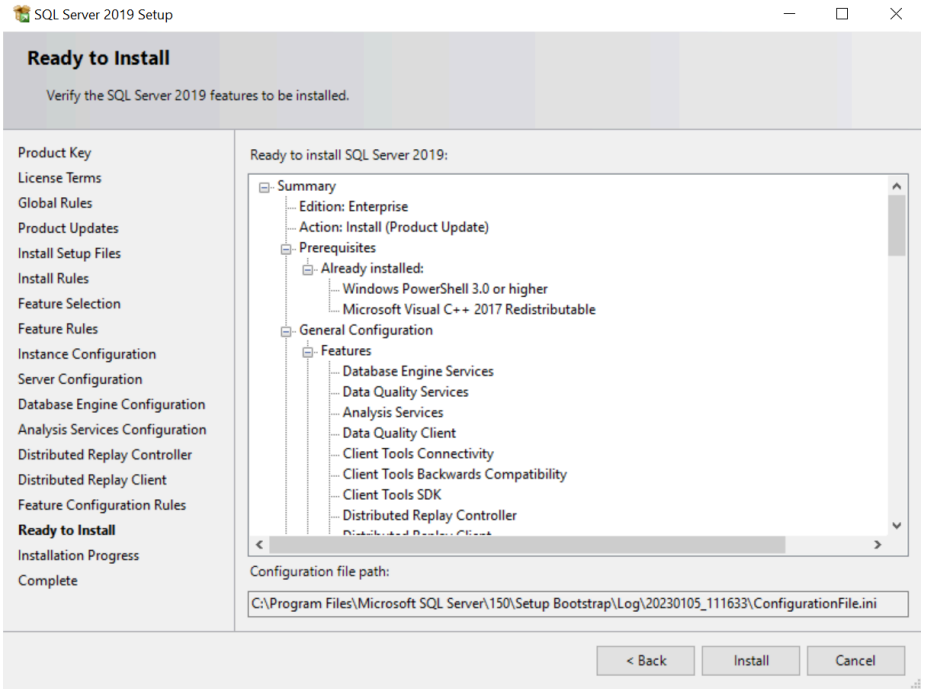




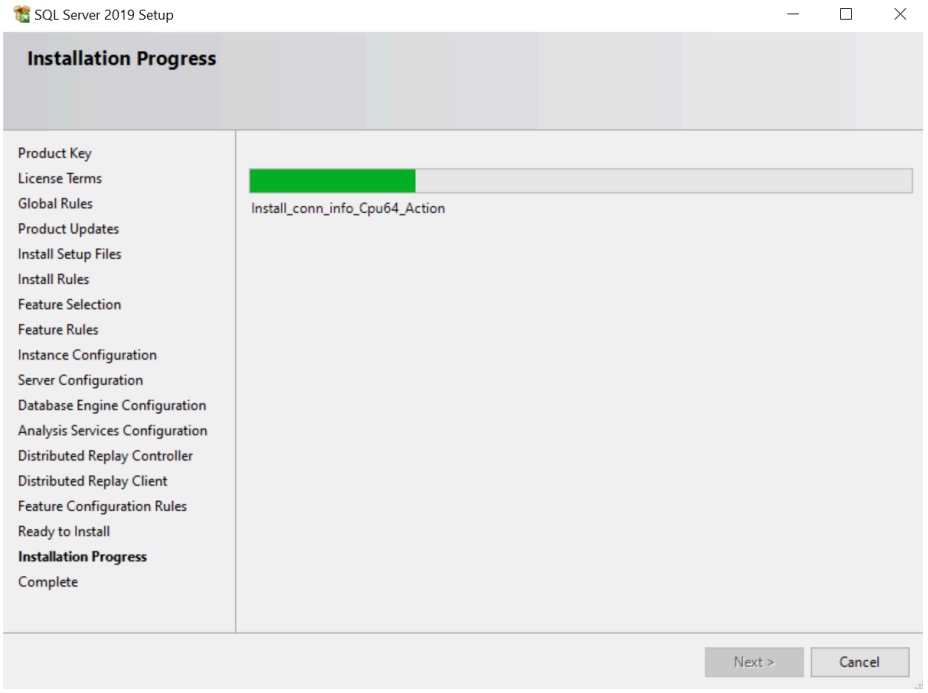
12. Click Next.



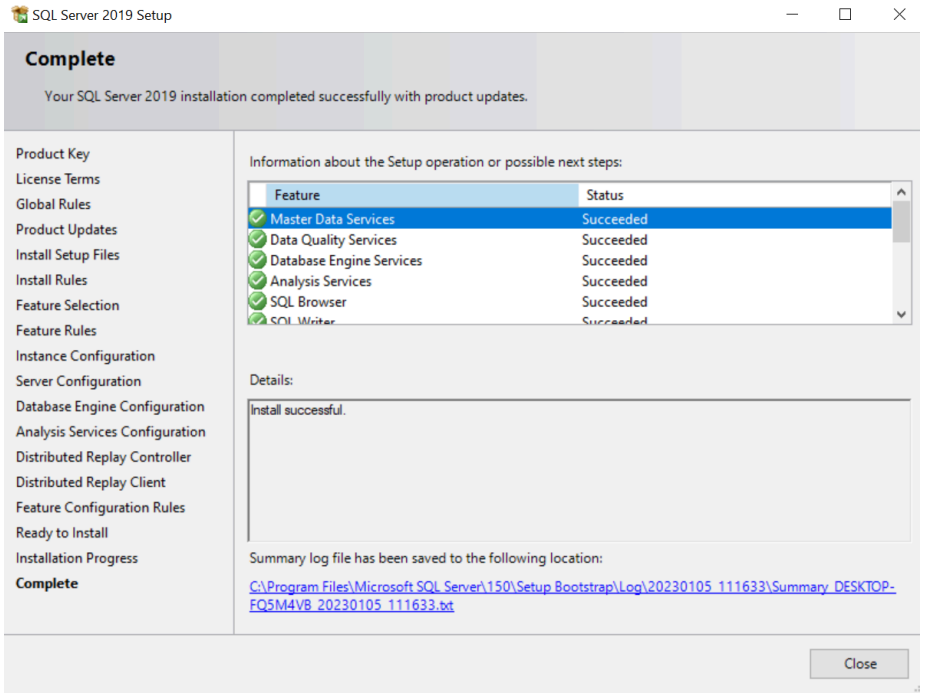
13. Click Install



14. Installation Progress screen appears.



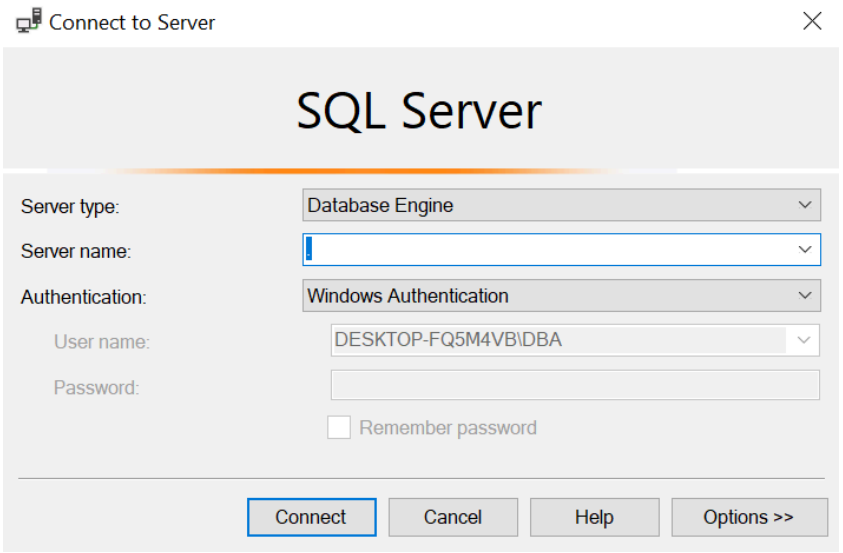
15. Installation Completed.



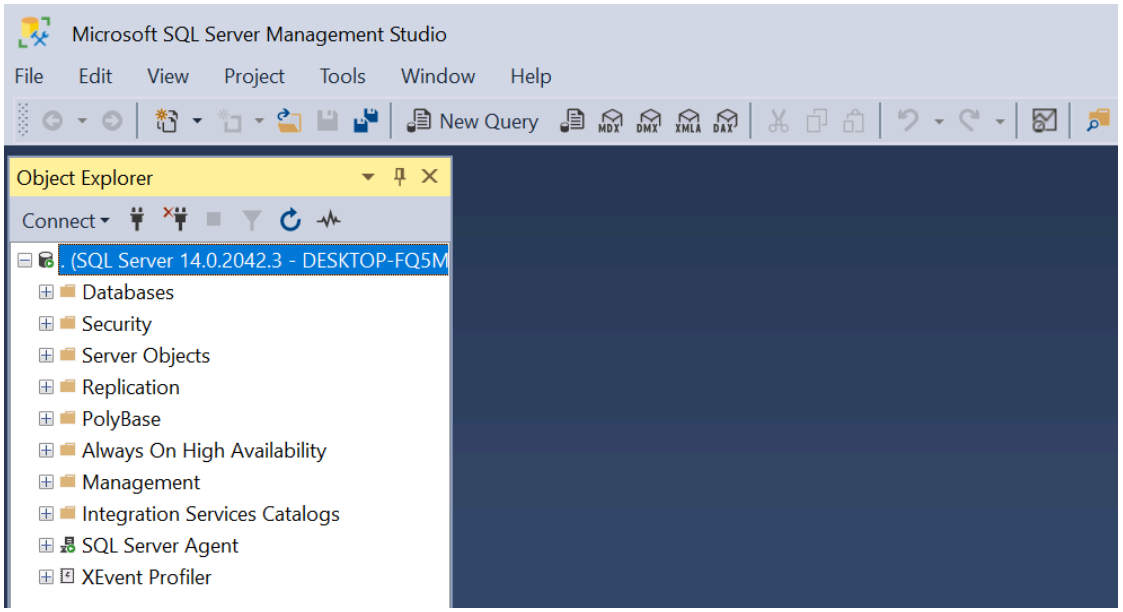
2.1 Connecting to SQL Server Management Studio

Step 1: Open SQL Server Management Studio

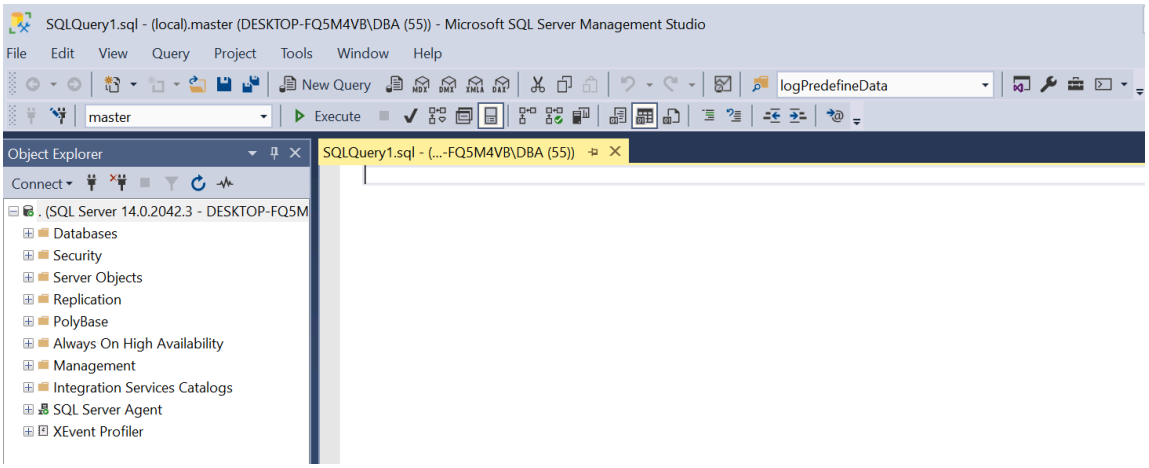
Step 2: Connect to SQL server with Windows Authentication .



Login View.



Step 3: Open SQL Query Window



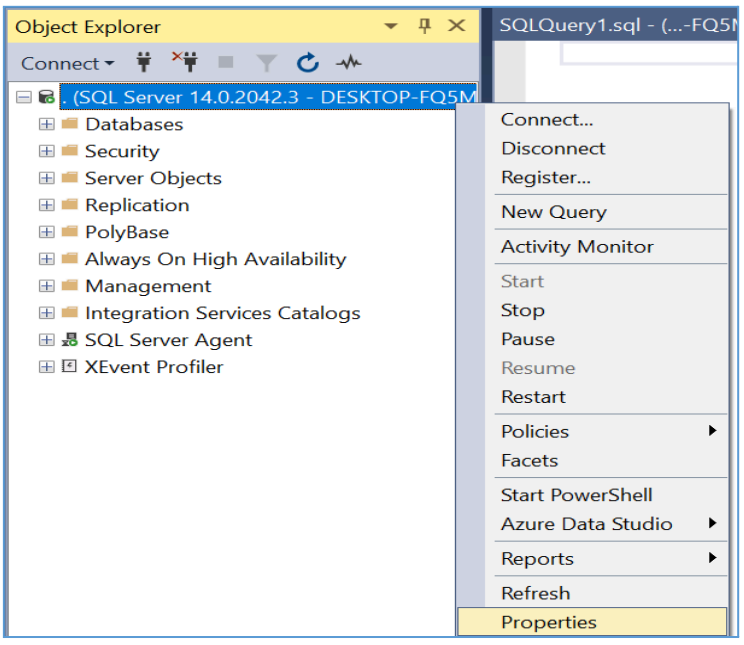
2.2 Checking SQL Server Version

Using Transact-SQL

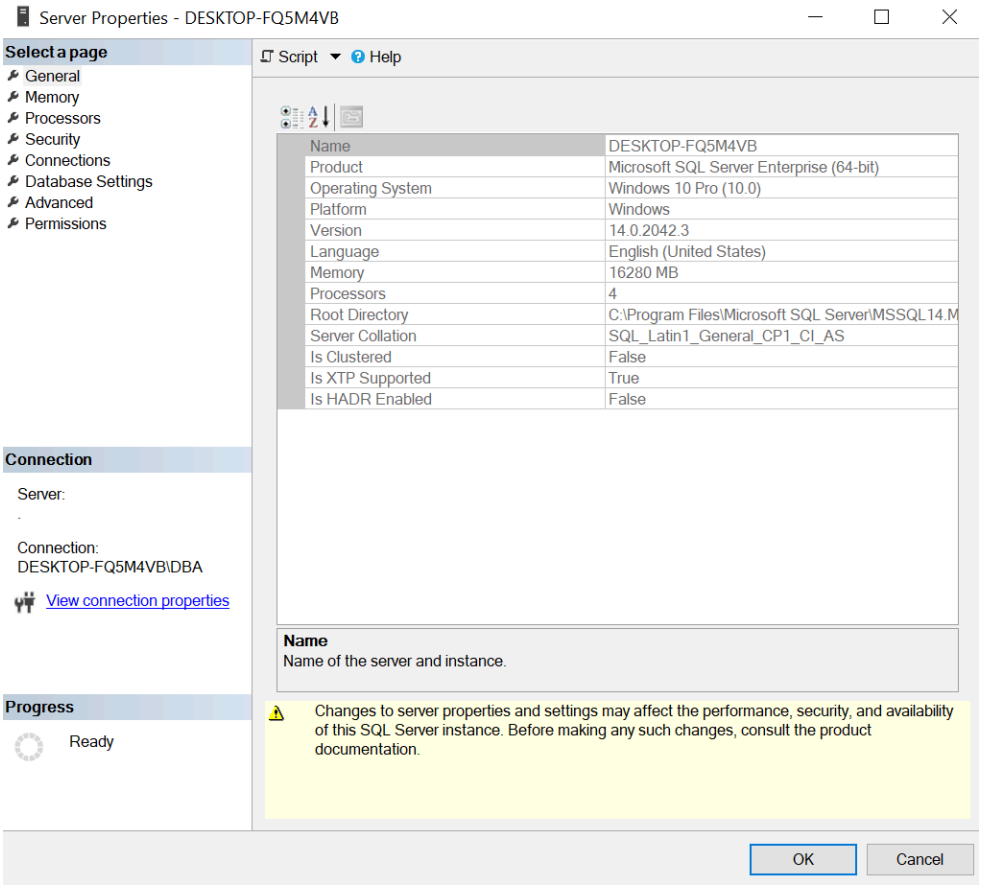
```
SELECT @@version
```

Using SQL Server Management Studio

Step1: Right-Click SQL Server and then Click Properties



Step2: Check Server Name, Editions and Collation



2.3 SQL Database Collation

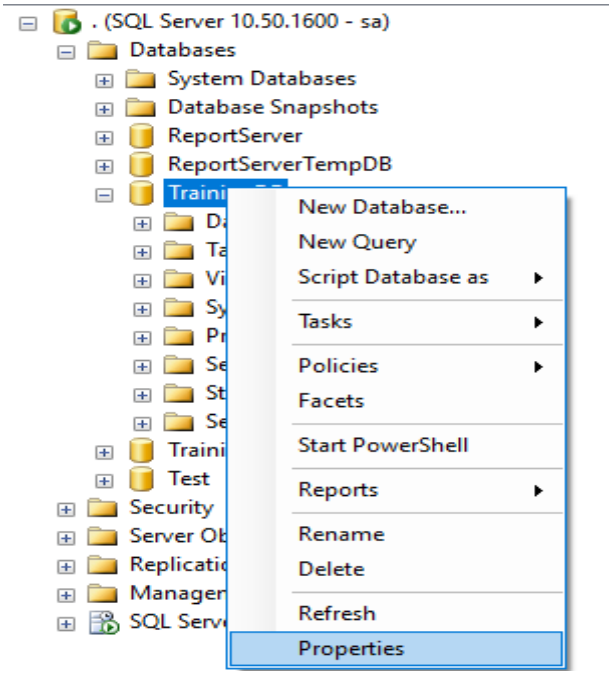
A collation specifies the bit patterns that represent each character in a dataset. Collations also determine the rules that sort and compare data. SQL Server supports storing objects that have different collations in a single database.

Using Transact-SQL

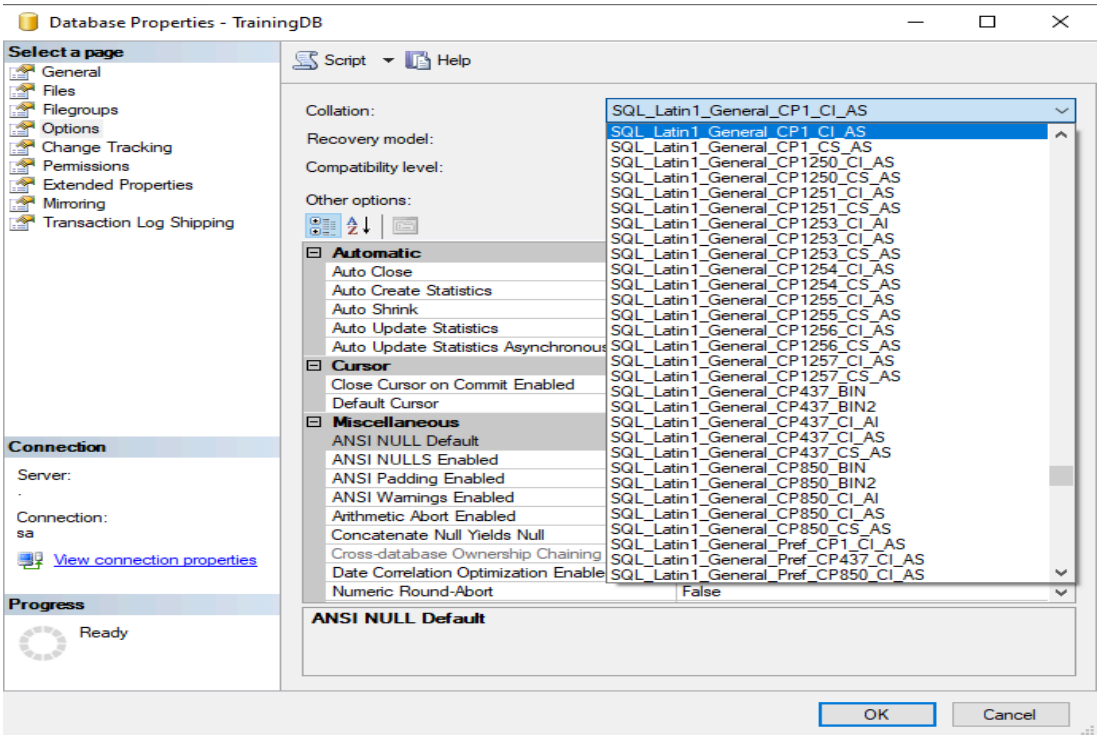
```
ALTER DATABASE TrainingDB COLLATE SQL_Latin1_General_CP1_CI_AS;
```

Using SQL Server Management Studio

Step1: Right-Click Database which want to use and then Click Properties.



Step2: Click Options and Select 'SQL_Latin1_General_CP1_CI_AS' from the Collation drop-down list and Click OK.



2.4 SQL Data Type

Data type	Description
CHAR(n)	Character string. Fixed-length n. Non-Unicode
VARCHAR (n) or VARCHAR (max)	Character string. Fixed-length n. Maximum length n. Maximum number of characters is 2GB. Non-Unicode.
TEXT	Variable-length character string. Non-Unicode.
NCHAR (n)	Character string. Fixed-length n. Unicode.
NVARCHAR(n) or NVARCHAR (max)	Character string. Variable length. Maximum length n. Maximum number of characters is 2GB. Unicode
NTEXT	Variable-length character string. Unicode
BINARY(n)	Binary string. Fixed-length n
VARBINARY(n) or VARBINARY (max)	Binary string. Variable length. Maximum length n. Maximum number of characters is 2GB. Non-Binary.
IMAGE	Variable length. Binary data.
BIT	Stores Integer that can be TRUE (1) or FALSE (0) values or Null.
TINYINT	Stores whole numbers up to 255.
SMALLINT	Stores whole numbers from lowest (-32768) to highest (32767)
INT	Stores whole numbers from lowest (-2,147,483,648) to highest (2,147,483,647).
BIGINT	Stores large integer from lowest (-9,223,372,036,854,775,808) to highest (9,223,372,036,854,775,807)
BOOLEAN	Stores TRUE or FALSE values
VARBINARY(n)	Binary string. Variable length. Maximum length n
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)
REAL	Approximate numerical, mantissa precision 7
FLOAT	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

XML	Stores XML data

Remark: Precision is the number of digits in a number. Scale is the number of digits to the right of the decimal point in a number

2.5 SQL Constraints

SQL Not Null

A NOT NULL constraint in SQL is used to prevent inserting NULL values into the specified column.

Example

```
CREATE TABLE StudentInfo
(
        StudentID INT Not Null,
        First_Name VARCHAR(50) NULL
)
```

Unique

The UNIQUE constraint in SQL is used to ensure that no duplicate values will be inserted into a specific column or combination of columns that are participating in the UNIQUE constraint and not part of the PRIMARY KEY.

In other words, the index that is automatically created when you define a UNIQUE constraint will guarantee that no two rows in that table can have the same value for the columns participating in that index, with the ability to insert only one unique NULL value to these columns, if the column allows NULL.

Example

```
CREATE TABLE Customer2
(
        ID INT Unique,
        Name VARCHAR(50) NULL
)
```

Primary Key

The PRIMARY KEY constraint consists of one column or multiple columns with values that uniquely identify each row in the table. PRIMARY KEY does not allow NULL values.

Example

```
CREATE TABLE Customer3
```

```
(
        ID INT PRIMARY KEY,
        Name VARCHAR(50) NULL
)
```

Foreign Key

A FOREIGN KEY is a key used to link two tables together. A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table.

Check Constraint

The CHECK constraint is used to limit the value range that can be placed in a column. If you define a CHECK constraint on a single column it allows only certain values for this column.

Default

A DEFAULT constraint is used to assign default values to the columns.

Aggregate Function

a. **MIN Function**
   The MIN () function returns the smallest value of the selected column.
   ```
   SELECT MIN (StudentID) FROM StudentInfo;
   ```
b. **MAX Function**
   The MAX () function returns the largest value of the selected column.
   ```
   SELECT MAX (StudentID) FROM StudentInfo;
   ```
c. **COUNT Function**
   The COUNT () function returns the number of rows that matches a specified criterion.
   ```
   SELECT COUNT (StudentID) FROM StudentInfo;
   ```
d. **AVG Function**
   The AVG () function returns the average value of a numeric column.
   ```
   SELECT AVG (List_Price) FROM StudentInfo;
   ```
e. **SUM Function**
   The SUM () function returns the total sum of a numeric column.
   ```
   SELECT SUM (Quantity) FROM Orders;
   ```

SQL Operators

a. Relational Operator

Operator	Description
=	Equal To
<>	Not Equal To
>	Greater Than
<	Less Than
>=	Greater Than or Equal To
<=	Less Than or Equal To
BETWEEN	Between an inclusive range
LIKE	Search for a Pattern
IN	specify multiple values

b. Arithmetic Operator

Operator	Meaning	Functionality
+	Addition	Adds two numeric values
-	Subtraction	Subtracts one numeric value from another
*	Multiplication	multiply tow numeric expressions
/	Division	Divides one numeric value by another
%	Modulation	Returns the remainder of one numeric value divided by another

Like Operator

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

```
SELECT * FROM StudentInfo WHERE Name like '%Kyaw%'
```

IN Operator

The IN operator allows you to specify multiple values in a WHERE clause.

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

(OR)

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

Example

```
SELECT * FROM StudentInfo WHERE Addr IN (USA, Yangon);
```

Between Operator

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.

The BETWEEN operator is inclusive: begin and end values are included.

```
SELECT column_name (s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

Example

```
SELECT * FROM StudentInfo
WHERE StudentID BETWEEN '0001' AND '0003';
```

Logical Operator

AND

The AND operator displays a record if all the conditions separated by AND are TRUE.

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

Example

```
SELECT * FROM StudentInfo
WHERE Addr=Yangon AND City=Bago;
```

OR

The OR operator displays a record if any of the conditions separated by OR is TRUE.

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

Example

```
SELECT * FROM StudentInfo
WHERE Addr='Yangon' OR City=USA;
```

NOT

The NOT operator displays a record if the condition(s) is NOT TRUE.

```
SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

Example

```
SELECT * FROM StudentInfo
WHERE StudentID NOT BETWEEN '0001' AND '0003';
```

Order By

The ORDER BY keyword is used to sort the result-set in ascending or descending order.

```
SELECT * FROM StudentInfo ORDER BY StudentID;
```

Group By

The GROUP BY statement group rows that have the same values into summary rows, like "find the number of customers in each City".

The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

Example

```
SELECT COUNT (StudentID), Addr
FROM StudentInfo
GROUP BY Addr;
```

Having Clause

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

Example

```
SELECT COUNT (StudentID), Addr
FROM StudentInfo
GROUP BY Addr
HAVING COUNT (StudentID) > 1
ORDER BY StudentID DESC;
```

Name Alias

i. Column Alias

```
SELECT column_name AS alias_name
FROM table_name;
```

Example

```
SELECT StudentID AS ID, First_Name AS Name
FROM StudentInfo;
```

ii. Table Alias

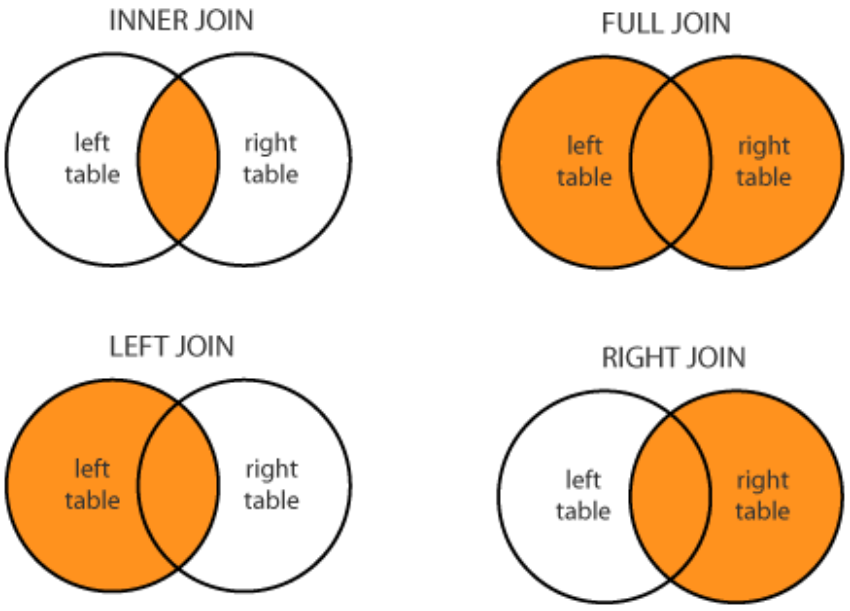
```
SELECT column_name(s)
FROM table_name AS alias_name;
```

Example

```
SELECT s.First_Name
FROM StudentInfo AS s
WHERE s.ID='0002';
```

Different SQL Joins

A **JOIN** clause is used to combine rows from two or more tables, based on a related column between them.



i. Inner Join

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

ii. Left Join

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

iii. Right Join

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

iv. Full Join

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```

v. Self Join

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

2.6 Index

Indexes are special lookup tables that the database search engine can use to speed up data retrieval. Simply put, an index is a pointer to data in a table. An index in a database is very similar to an index at the end of a book.

Using Transact-SQL

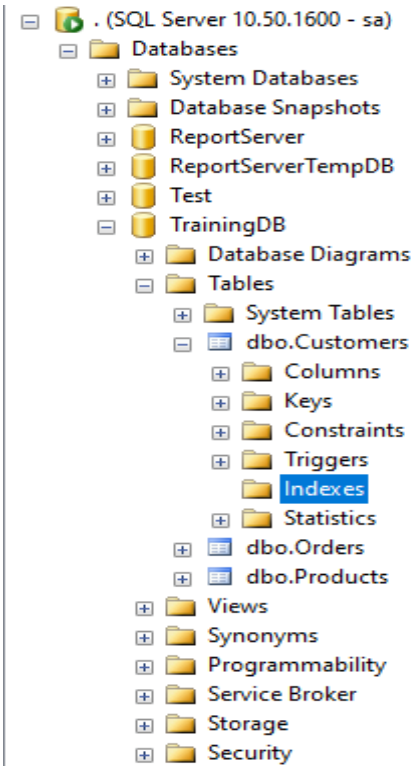
```
CREATE INDEX index_name
ON table_name (column_name)
```

Example

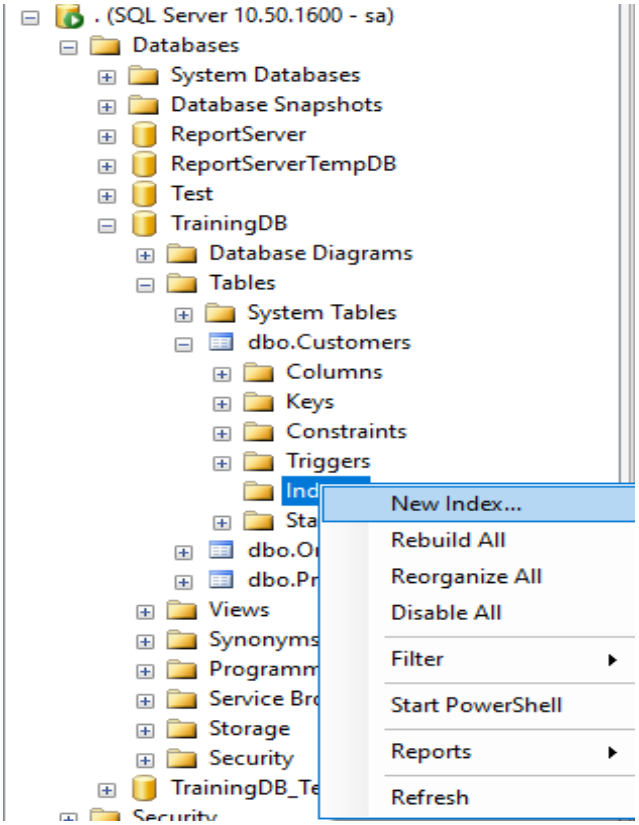
```
CREATE INDEX Cust_Index
ON Customers (Id,CustomerName)
```

Using SQL Server Management Studio

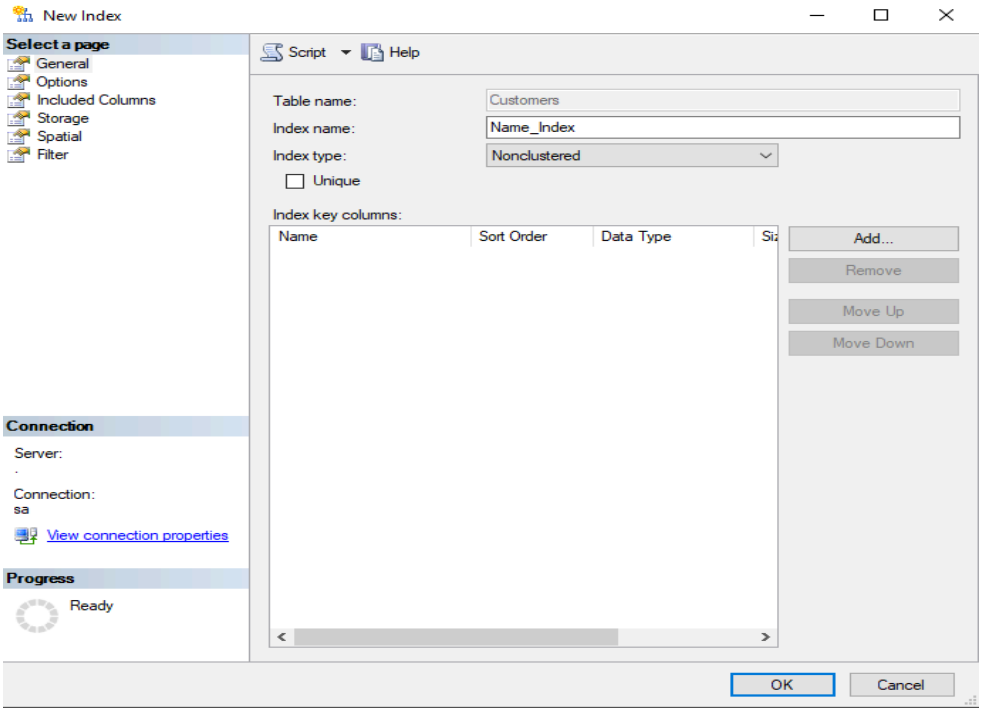
Step1: Right-Click Index Folder of a table



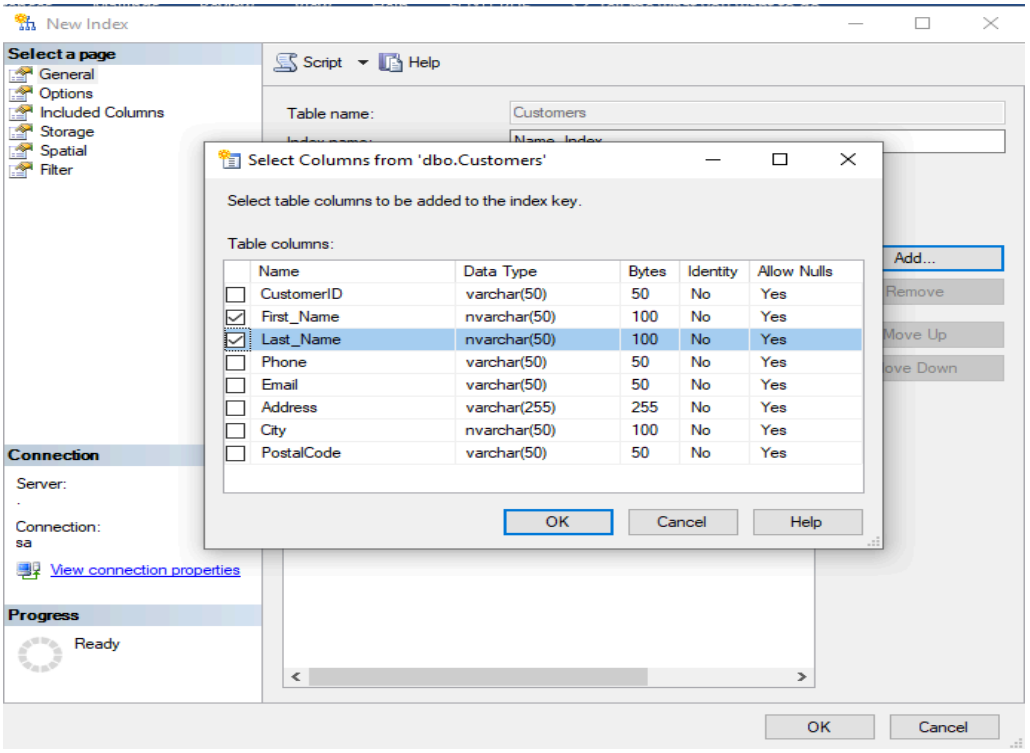
Step2: Click New Index…



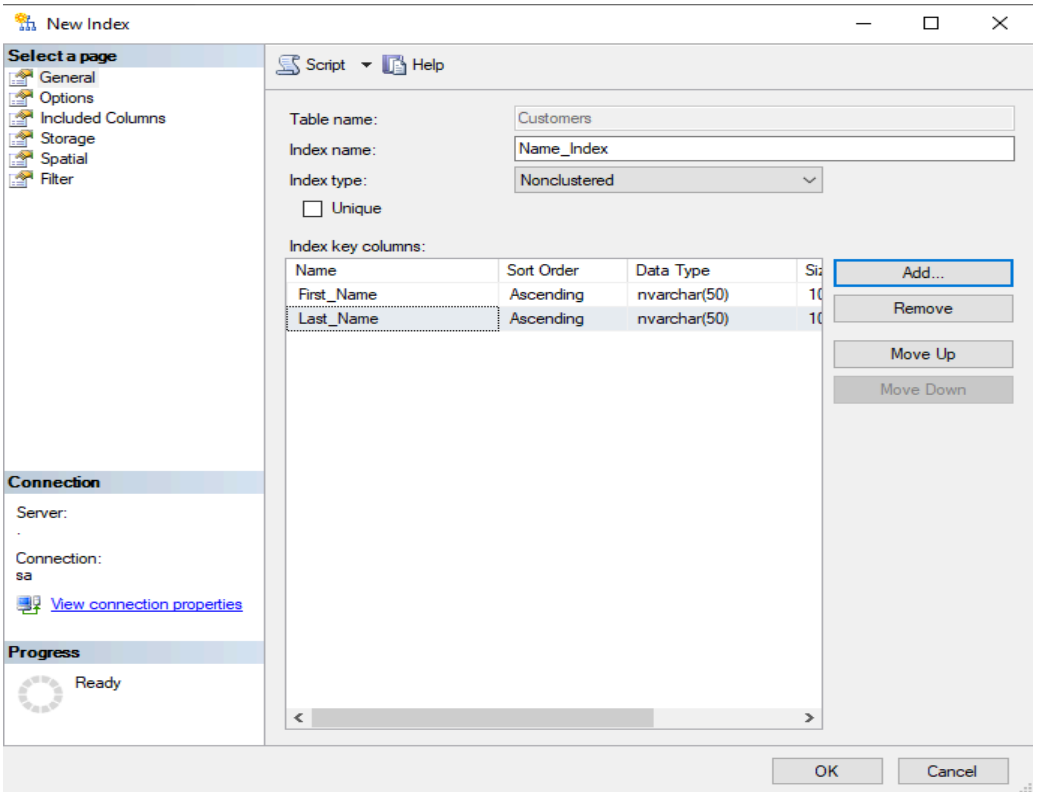
Step3: Give Index Name and Click Add…

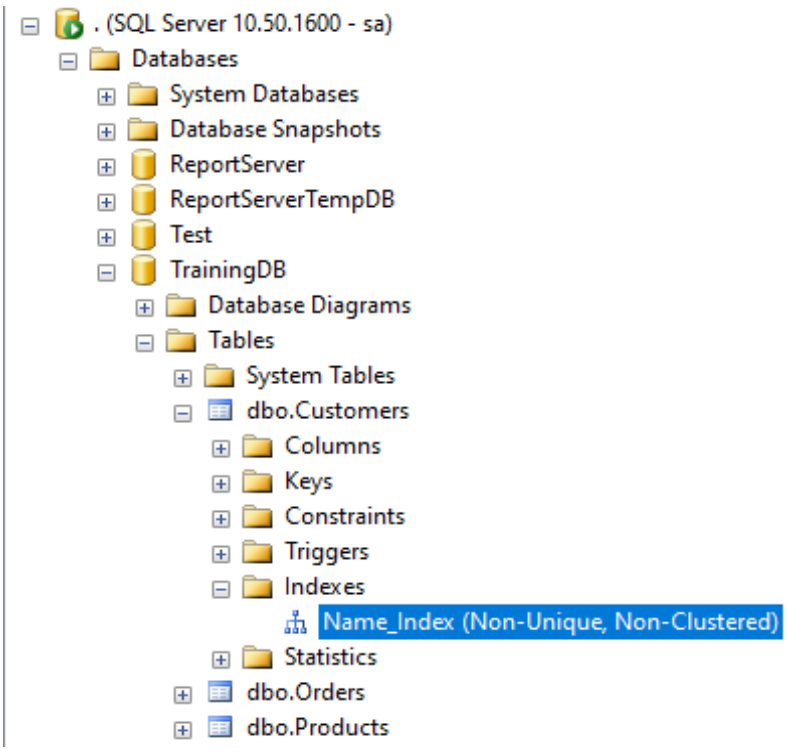


Step4: Choose Column to create index and Click OK.



Step5: Click OK





3. Structure Query Language (SQL)

Structured Query Language (SQL) is a standard computer language for relational database management and data manipulation. SQL is used to query, insert, update and modify data. Most of the relational database has SQL.

What Can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

3.1 Data Definition Language (DDL)

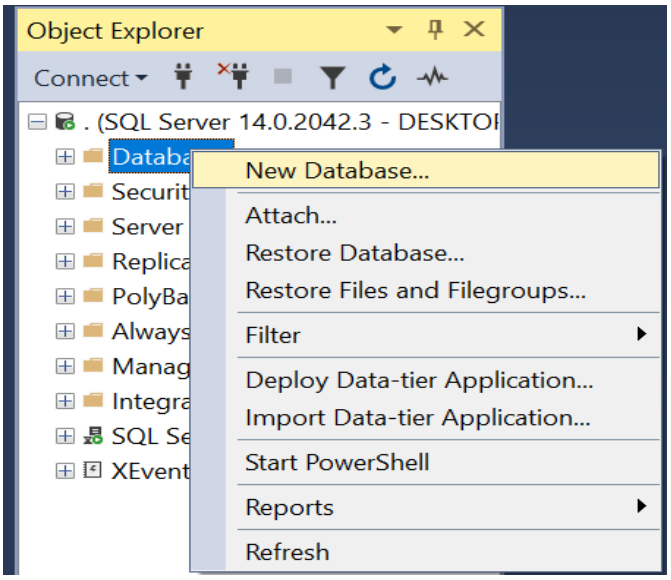
Data Definition Language is used to specify the database schema and database structures such as table and index structure. The most basic items of DDL are the CREATE, ALTER, RENAME and DROP statements:

1. CREATE creates an object (a table, for example) in the database.
2. DROP deletes an object in the database, usually irretrievably.
3. ALTER modifies the structure an existing object in various ways—for example, adding a column to an existing table.

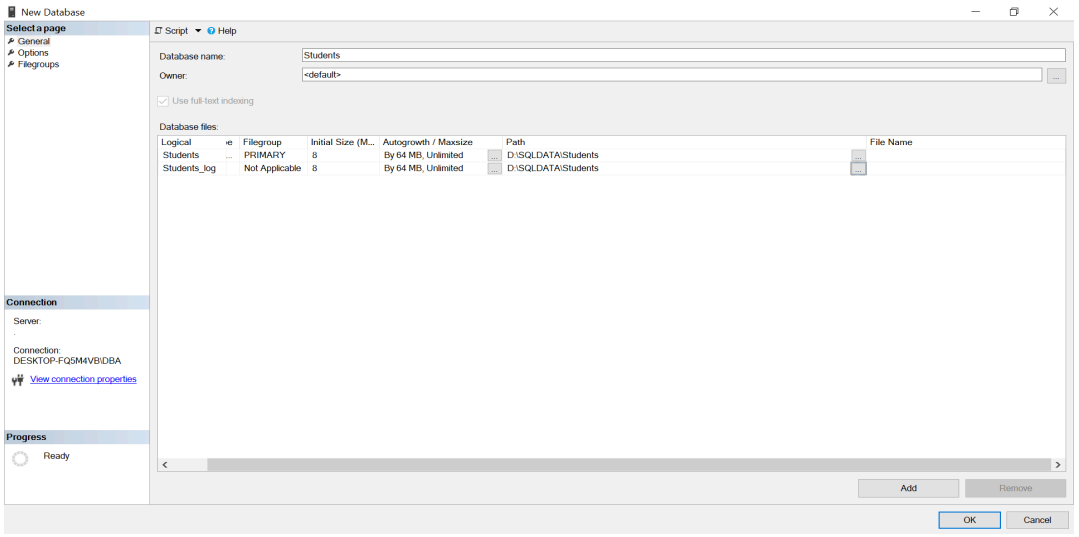
3.1.1 CREATE Database Object

(1) Create New Database using SSMS

Step 1. R-Click Database------> New Database



Step 2: Name the Database and Choose File Path that exists for mdf and ldf file.



(2) Create Database Using T-SQL

CREATE DATABASE <your database name>

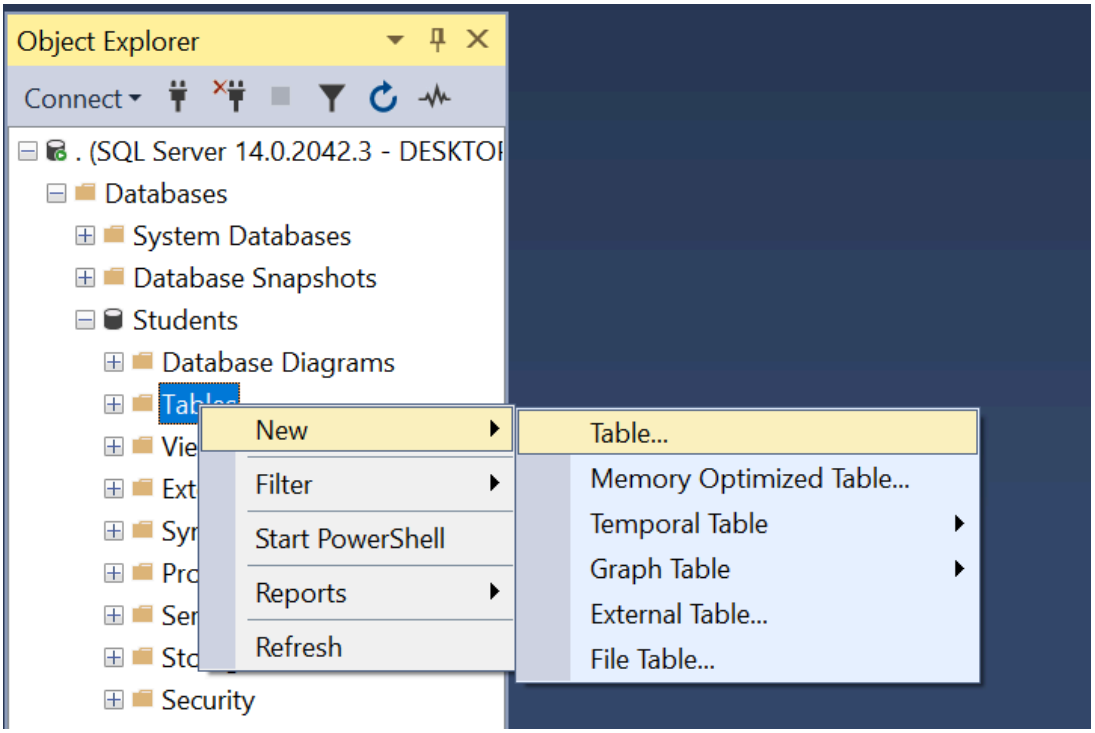
Example

CREATE DATABASE Students;

3.1.2 Create Table Object

Tables in SSMS

Step 1: R-Click Tables----> Select New Table



Step 2: Identify the column name and data type

Column Name	Data Type	Allow Nulls
StudentID	varchar(50)	☑
First_Name	varchar(50)	☑
Last_Name	varchar(50)	☑
Phone	varchar(50)	☑
Address	nvarchar(255)	☑

Create Table in T-SQL

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype
);
```

Example

```
CREATE TABLE studentInfo (
        StudentID varchar (50),
        First_Name varchar (50),
        Last_Name varchar (50),
        Phone varchar (50),
        Address nvarchar (255),
);
```

Altering the Table

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

Alter Table-Add Column

To add a column in a table, use the following syntax:

```
ALTER TABLE table_name ADD column_name datatype;
```

Example

```
ALTER TABLE studentInfo ADD DateOfBirth Date;
```

DESKTOP-FQ5M4VB....- dbo.studentInfo*

Column Name	Data Type	Allow Nulls
StudentID	varchar(50)	☑
First_Name	varchar(50)	☑
Last_Name	varchar(50)	☑
Phone	varchar(50)	☑
Address	nvarchar(255)	☑
DateOfBirth	date	☑

Alter Table-Drop Column

To change the data type of a column in a table, use the following syntax:

ALTER TABLE *table_name* ALTER COLUMN *column_name datatype*;

Example

ALTER TABLE studentInfo ALTER COLUMN Address nvarchar(500);

DESKTOP-FQ5M4VB....- dbo.studentInfo*

Column Name	Data Type	Allow Nulls
StudentID	varchar(50)	☑
First_Name	varchar(50)	☑
Last_Name	varchar(50)	☑
Phone	varchar(50)	☑
Address	nvarchar(500)	☑
DateOfBirth	date	☑

Rename Column in Table

To rename a column in an existing table, use the following syntax:

sp_rename 'table_name.old_column_name', 'new_column_name', 'COLUMN';

Example

sp_rename 'studentInfo.Address', 'Addr', 'COLUMN';

DESKTOP-FQ5M4VB....- dbo.studentInfo

Column Name	Data Type	Allow Nulls
StudentID	varchar(50)	☑
First_Name	varchar(50)	☑
Last_Name	varchar(50)	☑
Phone	varchar(50)	☑
Addr	nvarchar(500)	☑
DateOfBirth	date	☑

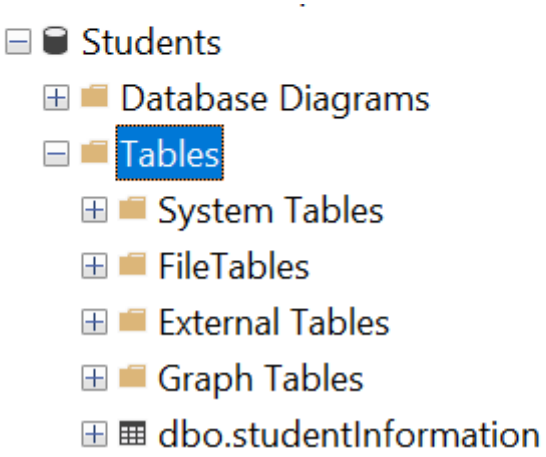
Rename Table

To rename a column in an existing table, use the following syntax:

```
sp_rename 'old_table_name', 'new_table_name';
```

Example

```
sp_rename 'studentInfo', 'studentInformation';
```



Dropping the Table

The DROP TABLE statement is used to drop an existing table in a database.

Syntax

```
DROP TABLE table_name;
```

Example

```
DROP TABLE studentInformation;
```

Truncate Table

The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

Syntax

TRUNCATE TABLE *table_name*;

Example

TRUNCATE TABLE *studentInformation*;

Dropping the Database

Syntax

DROP DATABASE <your database name>

Example

DROP DATABASE Students

3.2 Data Manipulation Language (DML)

Data Manipulation Language (DML) is used to access, modify or retrieve the data from the database. The acronym CRUD refers to all of the major functions that need to be implemented in a relational database application to consider it complete.

Operation	SQL	Description
CREATE	INSERT	Inserts new data into a database
READ (RETRIEVE)	SELECT	Extracts data from a database
UPDATE	UPDATE	Updates data in a database
DELETE	DELETE	Deletes data from a database

Insert Statement

The INSERT INTO statement is used to insert new records in a table.

```
INSERT INTO studentInfo
(StudentID, First_Name, Last_Name,Phone,Addr,DateOfBirth)
VALUES
( '0001', 'Aung', 'Kyaw','09788888881','Yangon','1991-01-01');

INSERT INTO studentInfo
(StudentID, First_Name, Last_Name,Phone,Addr,DateOfBirth)
VALUES
( '0002', 'Su', 'Nadi','09788888851','Bago','1994-01-01');
```

Update Statement

The UPDATE statement is used to modify the existing records in a table.

```
UPDATE studentInfo SET First_Name='Kyaw' WHERE StudentID='0001';

UPDATE studentInfo SET Addr='Toungoo' WHERE StudentID='0002';
```

Select Statement

a. Select all Data

The SELECT statement is used to select data from a database.

```
SELECT * FROM StudentInfo;
```

Results | Messages

	StudentID	First_Name	Last_Name	Phone	Addr	DateOfBirth
1	0001	Kyaw	Kyaw	09788888881	Yangon	1991-01-01
2	0002	Su	Nadi	09788888851	Toungoo	1994-01-01
3	0003	Aye	Chan	09798888881	Yangon	1992-01-01
4	0004	John	Cena	09780888851	USA	1981-01-01

b. Select Distinct

The SELECT DISTINCT statement is used to return only distinct (different) values.

```
SELECT DISTINCT(Addr)  FROM StudentInfo;
```

c. Select Top

The SELECT TOP clause is used to specify the number of records to return.

```
SELECT TOP(1) FROM StudentInfo;
```

Where Clause

The WHERE clause is used to filter records.

```
SELECT * FROM StudentInfo WHERE StudentInfo='0001';

SELECT First_Name,Last_Name FROM StudentInfo WHERE StudentInfo='0001';

SELECT Phone FROM StudentInfo WHERE First_Name='John';

SELECT * FROM StudentInfo WHERE StudentID='0001';
```

Delete Statement

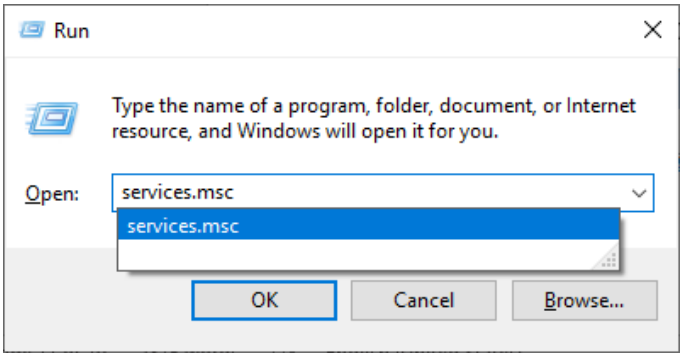
The DELETE statement is used to delete existing records in a table.

```
DELETE FROM studentInfo WHERE StudentID='0001';

DELETE FROM studentInfo WHERE StudentID='0002' and Addr='Toungoo';
```

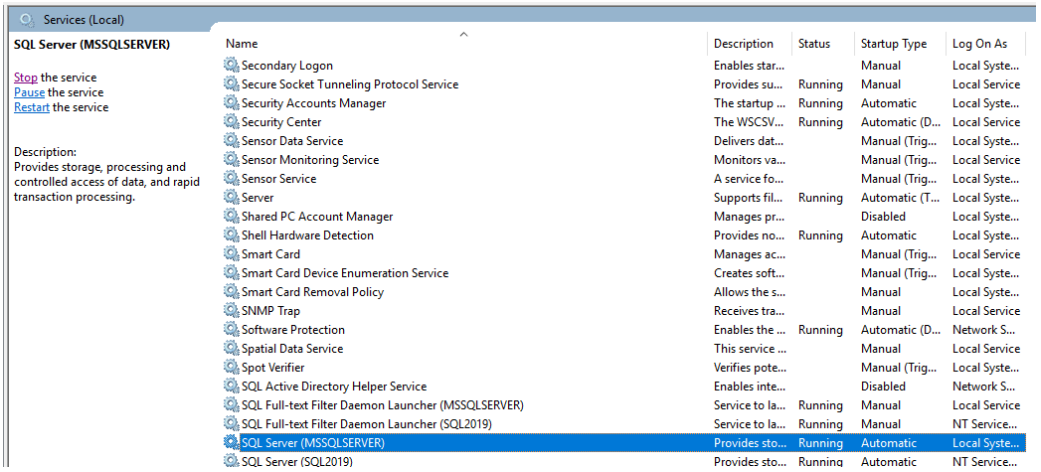
4. Manage MSSQL Service

4.1 To Start/Stop/ Restart SQL service Using Service Manager

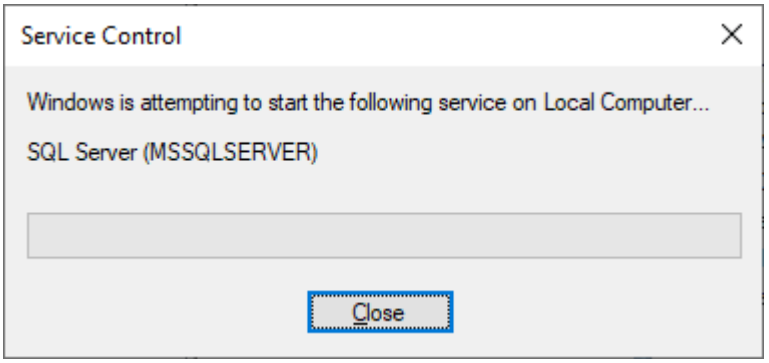
Step 1: Open Run box and type Services.msc and then click OK



Step 2: Choose SQL Server (MSSQLSERVER)



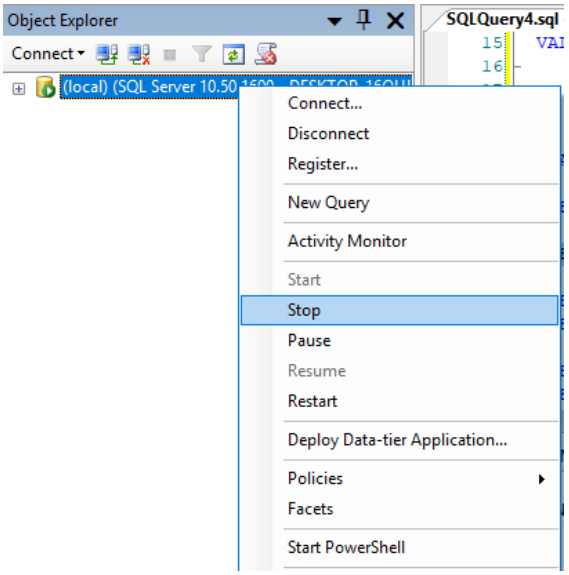
Step 3: Click required service (start/ stop, Pause or Restart)



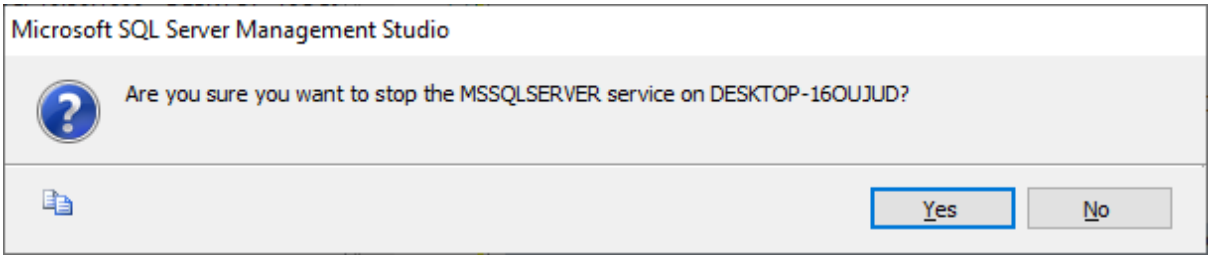
4.2 To Start/Stop/ Restart SQL service in SQL Server Explore

To Stop SQL Service

Step 1: Right Click on the Database Server and Click Stop

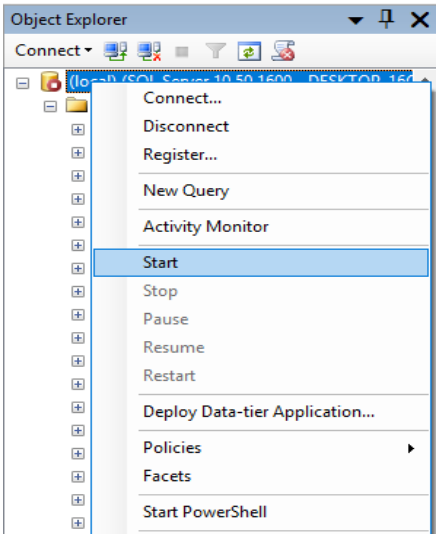


Step 2: Click Yes to stop Database service



4.3 To Start SQL Service

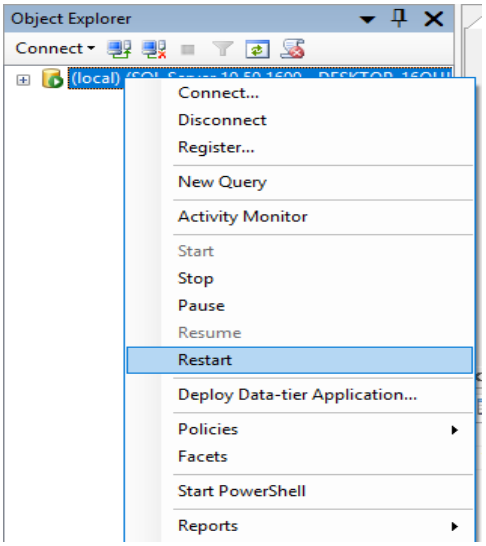
Step 1: Right Click on the Database Server and Click Start



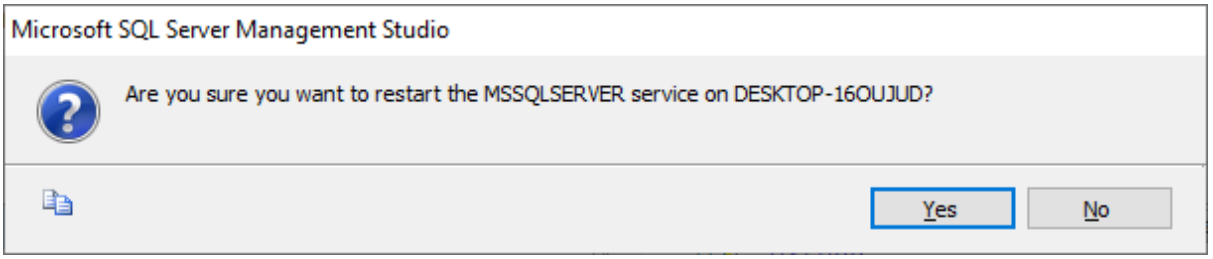
Step 2: Click Yes to start Database service

4.4 To Restart SQL Service

Step 1: Right Click on the Database Server and Click Restart



Step 2: Click Yes to Restart Database service



5. Normalization

Database normalization is the process of structuring a relational database in accordance with a series of normal forms in order to reduce data redundancy and improve data integrity. It is a multi-step process that puts data into tabular form, removing duplicated data from the relational tables.

The main purpose of using Normalization:

- Eliminating redundant (useless) data.
- Ensuring data dependencies make sense i.e., data is logically stored.
- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies.
- Normalization divides the larger table into the smaller table and links them using relationship.

5.1 Database Normal Form

Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
4NF	A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
5NF	A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.

Customer Table

Customer_First_Name	Customer_Last_Name	Item
Joe	Bloggs	Cake, Bread, Coffee
John	Smith	Tea, Cake

1NF

Customer_First_Name	Customer_Last_Name	Item 1	Item 2	Item 3
Joe	Bloggs	Cake	Bread	Coffee
John	Smith	Tea	Cake	

2NF

Customer_First_Name	Customer_Last_Name	Item
Joe	Bloggs	Cake
Joe	Bloggs	Bread
Joe	Bloggs	Coffee
John	Smith	Tea
John	Smith	Cake

3NF

Customer

Customer_ID	Customer_First_Name	Customer_Last_Name
C001	Joe	Bloggs
C002	John	Smith

Item

Item_ID	Item
I001	Cake
I002	Bread
I003	Coffee
I004	Tea

Customer_Order

Customer_ID	Item_ID
C001	I001
C001	I002
C001	I003
C002	I004
C002	I003

1NF (First Normal Form) Rules

- Each table cell should contain a single value.
- Each record needs to be unique.

1NF Example

FULL NAMES	PHYSICAL ADDRESS	MOVIES RENTED	SALUTATION
Janet Jones	First Street Plot No 4	Pirates of the Caribbean	Ms.
Janet Jones	First Street Plot No 4	Clash of the Titans	Ms.
Robert Phil	3rd Street 34	Forgetting Sarah Marshal	Mr.
Robert Phil	3rd Street 34	Daddy's Little Girls	Mr.
Robert Phil	5th Avenue	Clash of the Titans	Mr.

2NF (Second Normal Form) Rules

- Rule 1- Be in 1NF
- Rule 2- Single Column Primary Key

Member

MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION
1	Janet Jones	First Street Plot No 4	Ms.
2	Robert Phil	3rd Street 34	Mr.
3	Robert Phil	5th Avenue	Mr.

Member_Rented

MEMBERSHIP ID	MOVIES RENTED
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans

Database Key

A KEY is a value used to identify a record in a table uniquely. A KEY could be a single column or combination of multiple columns.

Note: Columns in a table that are NOT used to identify a record uniquely are called non-key columns.

Primary Key

A primary is a single column value used to identify a database record uniquely.

It has following attributes

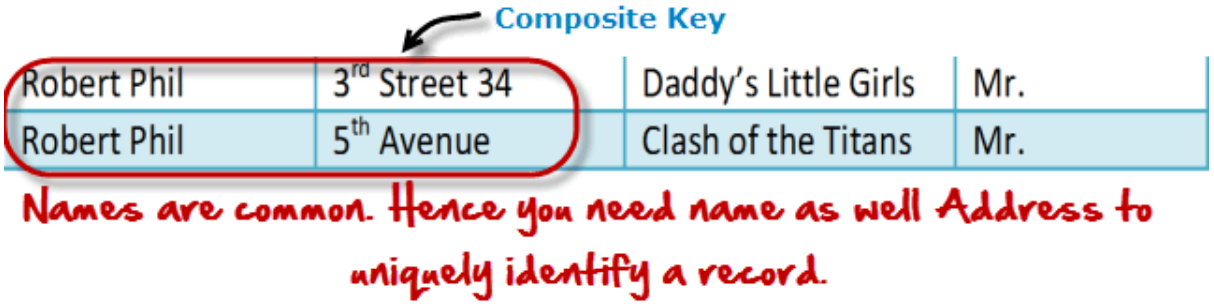
- A primary key cannot be NULL

- A primary key value must be unique
- The primary key values should rarely be changed
- The primary key must be given a value when a new record is inserted.

Composite Key

A composite key is a primary key composed of multiple columns used to identify a record uniquely.

In our database, we have two people with the same name Robert Phil, but they live in different places.



Hence, we require both Full Name and Address to identify a record uniquely. That is a composite key.

Database - Foreign Key

In the following table, Membership_ID is the Foreign Key that is reference from Primary Key of Member Table.

MEMBERSHIP ID	MOVIES RENTED
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans

Foreign Key references the primary key of another Table, it helps connect to tables.

- A foreign key can have a different name from its primary key
- It ensures rows in one table have corresponding rows in another
- Unlike the Primary key, they do not have to be unique. Most often they aren't
- Foreign keys can be null even though primary keys can not

Functional Dependency

The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.

Transitive functional dependencies

A transitive functional dependency is when changing a non-key column, might cause any of the other non-key columns to change. The following example show changing the non-key column Full Name may change Salutation.

MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION
1	Janet Jones	First Street Plot No 4	Ms.
2	Robert Phil	3rd Street 34	Mr.
3	Robert Phil	5th Avenue	Mr.

3NF (Third Normal Form) Rules

- Rule 1- Be in 2NF
- Rule 2- Has no transitive functional dependencies

To become 2NF table into 3NF, need to divide member table to create a new table which stores Salutations.

Now, there are no transitive functional dependencies, between Full Name and Salutation.

MEMBERSHIP ID	FULL NAMES	PHYSICAL ADDRESS	SALUTATION ID
1	Janet Jones	First Street Plot No 4	2
2	Robert Phil	3rd Street 34	1
3	Robert Phil	5th Avenue	1

MEMBERSHIP ID	MOVIES RENTED
1	Pirates of the Caribbean
1	Clash of the Titans
2	Forgetting Sarah Marshal
2	Daddy's Little Girls
3	Clash of the Titans

SALUTATION ID	SALUTATION
1	Mr.
2	Ms.
3	Mrs.
4	Dr.

Boyce-Codd Normal Form (BCNF)

Even when a database is in 3rd Normal Form, still there would be anomalies resulted if it has more than one **Candidate** Key. Sometimes is BCNF is also referred as **3.5 Normal Form.**

4NF (Fourth Normal Form) Rules

If no database table instance contains two or more, independent and multivalued data describing the relevant entity, then it is in 4th Normal Form.

5NF (Fifth Normal Form) Rules

A table is in 5th Normal Form only if it is in 4NF and it cannot be decomposed into any number of smaller tables without loss of data.

5.2 Advantages and Disadvantages of Normalization

Advantages

- Data Consistency
- Data is always real and it is not ambiguous.
- Data becomes non-redundant
- Only copy original copy of data is available for each user and for every time. There are no multiple copies of the same data for different persons, when data is changed in one file and stay in one file. Then the data is consistent and non-redundant.
- Reduce Insertion, deletion and updating anomalies
- Database table compaction
- Better Performance
- Fast Queries

Disadvantages

- Required experienced database designer
- Difficult and expansive
- Requires detailed database design
- Requires more joins to get the coveted effect. A crudely composed question can cut the database down
- Maintenance overhead. The higher the level of normalization, the more stupendous the amount of tables in the database.

6. Database Programming

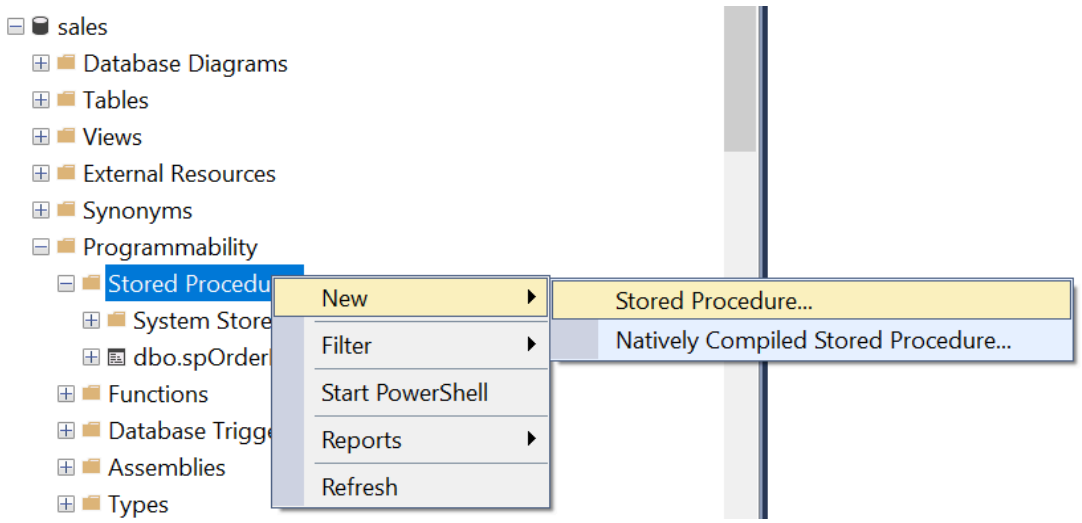
SQL (structured query language) is a programming language that enables programmers to work with that data. While an application might be programmed in a language like Python, PHP or Ruby, databases are not configured to understand these. To users, the information in a database can be accessed by using SQL Procedures, Views and Functions.

6.1 SQL Stored Procedures

A stored procedure is the prepared SQL code that you can save so that the code can be reused over and over again. A stored procedure is a subroutine available to applications that access a relational database management system.

Category	Featured syntax elements
Basic Syntax	CREATE PROCEDURE
Passing parameters	@parameter
	• = default
	• OUTPUT
	• table-valued parameter type
	• CURSOR VARYING
Modifying data by using a stored procedure	UPDATE
Error Handling	TRY...CATCH

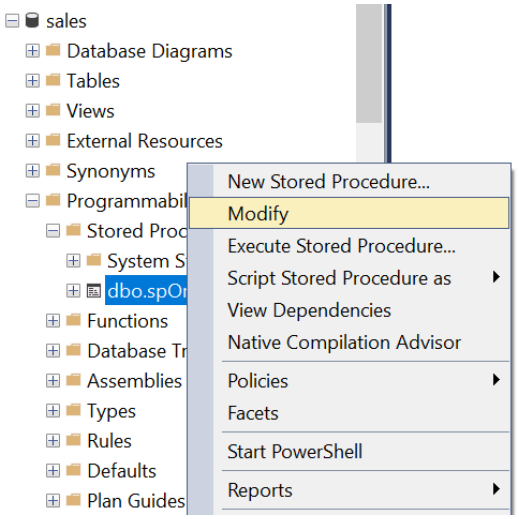
Create Procedure (database_name>>Programmability>>Stored Procedure>>New>>Stored Procedure)



Syntax

```
-- ============================================
-- Author:      <Author,,Name>
-- Create date: <Create Date,,>
-- Description: <Description,,>
-- ============================================
CREATE PROCEDURE <Procedure_Name, sysname, ProcedureName>
    -- Add the parameters for the stored procedure here
    <@Param1, sysname, @p1> <Datatype_For_Param1, , int> = <Default_Value_For_Param1, , 0>,
    <@Param2, sysname, @p2> <Datatype_For_Param2, , int> = <Default_Value_For_Param2, , 0>
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 <@Param1, sysname, @p1>, <@Param2, sysname, @p2>
END
GO



  ---------------------------------------------
CREATE PROCEDURE spOrderList
    @fromDate date,
    @toDate date
AS
BEGIN

    SET NOCOUNT ON;

    SELECT c.first_name, c.last_name, d.product_name, d.model_year, a.order_date,
        d.list_price, b.quantity, d.list_price*b.quantity as amount FROM orders a
        INNER JOIN order_items b on a.order_id=b.order_id
        INNER JOIN customers c on c.customer_id=a.customer_id
        INNER JOIN products d on d.product_id=b.product_id
    WHERE CONVERT(date, a.order_date, 101) between @fromDate and @toDate

END
GO
```

Modify Procedure (database_name>>Programmability>>Stored Procedure>>Store Procedure Name>>Modify)




```
ALTER PROCEDURE [dbo].[spOrderList]
    @fromDate date,
    @toDate date
AS
BEGIN

    SET NOCOUNT ON;

    SELECT c.first_name, c.last_name, d.product_name, d.model_year, a.order_date,
        d.list_price, b.quantity, d.list_price*b.quantity as amount FROM orders a
        INNER JOIN order_items b on a.order_id=b.order_id
        INNER JOIN customers c on c.customer_id=a.customer_id
        INNER JOIN products d on d.product_id=b.product_id
    WHERE CONVERT(date, a.order_date, 101) between @fromDate and @toDate
    ORDER BY a.order_date

END
```

Drop Procedure

Syntax

DROP PROCEDURE procedure_name;

Execute Procedure

Exec storedProcedureName, para1, para2,---

exec spOrderList '2016-01-01','2016-01-30'

6.1.1 Advantages & Disadvantages of Stored Procedure

Advantages

Stored procedures are so popular and have become so widely used in Relational Database Management Systems (RDBMS). The list below are advantages of using stored procedure.

1) **Maintainability**

Because scripts are in one location, updates and tracking of dependencies based on schema changes becomes easier.

2) **Testing**

Can be tested independent of the application.

3) **Isolation of Business Rules**

Having Stored Procedures in one location means that there's no confusion of having business rules spread over potentially disparate code files in the application.

4) **Speed / Optimization**

Stored procedures are cached on the server.

Execution plans for the process are easily reviewable without having to run the application.

5) **Utilization of Set-based Processing**

The power of SQL is its ability to quickly and efficiently perform set-based processing on large amounts of data; the coding equivalent is usually iterative looping, which is generally much slower.

6) **Security**

Limit direct access to tables via defined roles in the database.

Provide an "interface" to the underlying data structure so that all implementation and even the data itself is shielded.

Securing just the data and the code that accesses it is easier than applying that security within the application code itself.

Disadvantages

There are certainly drawbacks to Stored Procedures that preclude them from being the one-stop shop solution to application database access. The list below contains some reasons why Stored Procedures might not be right for your application solution.

1) **Limited Coding Functionality**

Stored procedure code is not as robust as app code, particularly in the area of looping (not to mention that iterative constructs, like cursors, are slow and processor intensive)

2) **Portability**

Complex Stored Procedures that utilize complex, core functionality of the RDBMS used for their creation will not always port to upgraded versions of the same database. This is especially true if moving from one database type (Oracle) to another (MS SQL Server).

3) **Testing**

Any data errors in handling Stored Procedures are not generated until runtime.

4) Location of Business Rules

Since SP's are not as easily grouped/encapsulated together in single files, this also means that business rules are spread throughout different Stored Procedures. App code architecture helps to ensure that business rules are encapsulated in single objects. There is a general opinion that business rules / logic should not be housed in the data tier.

5) Utilization of Set-based Processing

Too much overhead is incurred from maintaining Stored Procedures that are not *complex* enough. As a result, the general consensus is that simple SELECT statements should *not* be bound to Stored Procedures and instead implemented as inline SQL.

6) Cost

Depending on your corporate structure and separation of concern for development, there is the potential that Stored Procedure development could potentially require a dedicated database developer. Some businesses will not allow developers access to the database at all, requiring instead a separate DBA. This will automatically incur added cost.

Some companies believe (and sometimes it's true, but not always) that a DBA is more of a SQL expert than an application developer, and therefore will write better Stored Procedures. In that case, an extra developer in the form of a DBA is required.

6.2 SQL Views

In **SQL**, a **view** is a virtual table based on the result-set of an **SQL** statement. A **view** contains rows and columns, just like a real table that can be added **SQL** functions,

WHERE, and JOIN statements to a **view** and present the data as if the data were coming from one single table.

Views are used for security purpose in databases, views restrict the user from viewing certain column and rows means by using view we can apply the restriction on accessing the particular rows and columns for specific user. Views display only those data which are mentioned in the query, so it shows only data which is returned by the query that is defined at the time of creation of the View.

6.2.1 Create / Alter View Using SQL Query

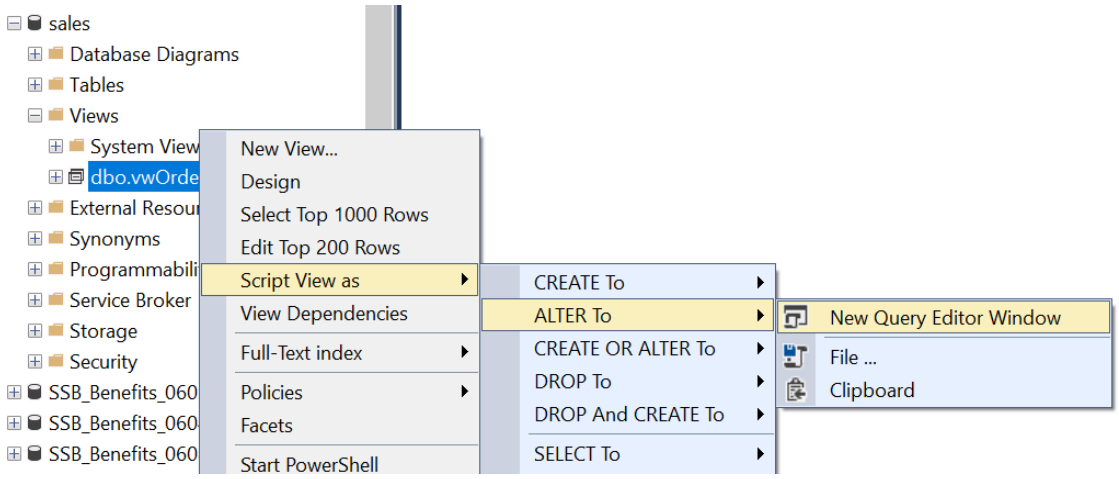
Create View

```
CREATE VIEW vwOrderList
AS
    SELECT c.first_name, c.last_name, d.product_name, d.model_year, a.order_date,
        d.list_price, b.quantity, d.list_price*b.quantity as amount FROM orders a
        INNER JOIN order_items b on a.order_id=b.order_id
        INNER JOIN customers c on c.customer_id=a.customer_id
        INNER JOIN products d on d.product_id=b.product_id
```

Retrieve Data from View

```
SELECT * FROM vwOrderList
```

Modify View



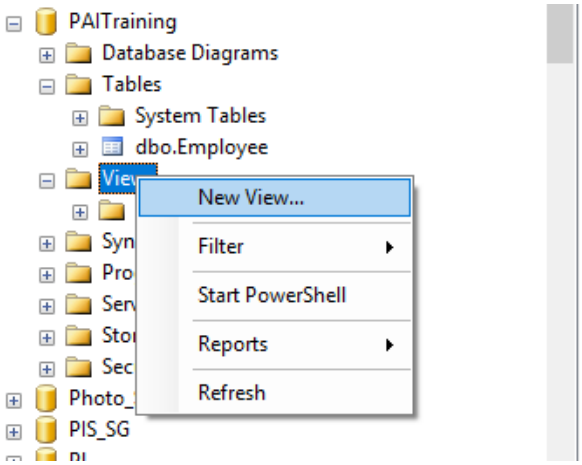

```
ALTER VIEW [dbo].[vwOrderList]
AS
    SELECT c.first_name, c.last_name, d.product_name, d.model_year, a.order_date,
        d.list_price, b.quantity, d.list_price*b.quantity as amount FROM orders a
        INNER JOIN order_items b on a.order_id=b.order_id
        INNER JOIN customers c on c.customer_id=a.customer_id
        INNER JOIN products d on d.product_id=b.product_id
GO
```

Drop View

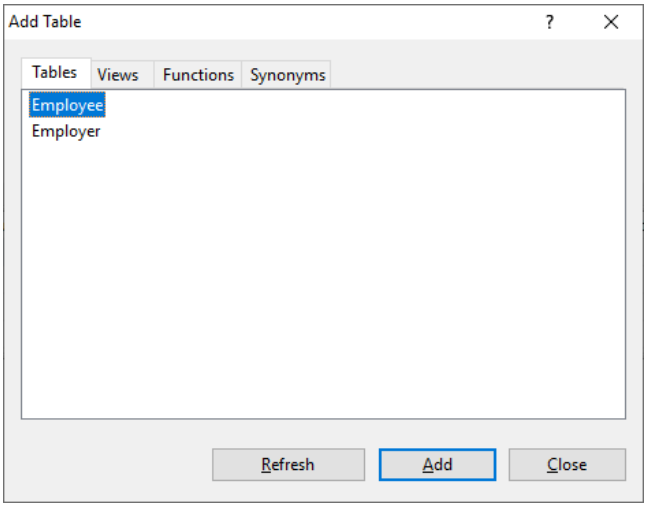
drop view vwOrderList

6.2.2 Create / Alter View Using Design

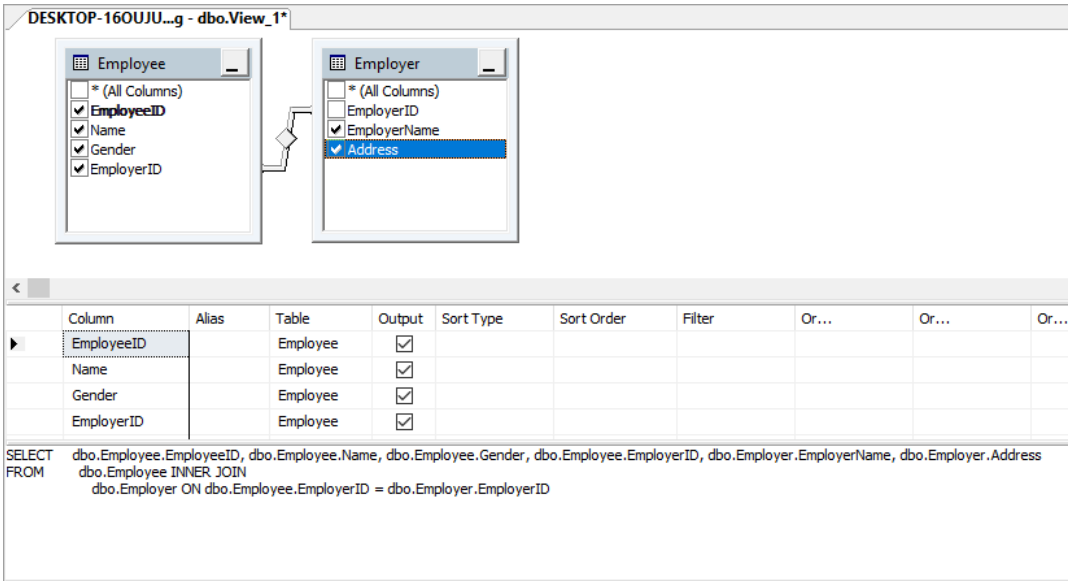
Step 1: In the database, Right Click On the View and click New View.



Step 2: Add table from the list.



Step 3: Join Table choose the required column to Select.



6.2.3 Advantages & Disadvantages of Views

Advantages

1) **Security**

Each user can be given permission to access the database only through a small set of views that contain the specific data the user is authorized to see, thus restricting the user's access to stored data

2) **Query Simplicity**

A view can draw data from several different tables and present it as a single table, turning multi-table queries into single-table queries against the view.

3) **Structural simplicity**

Views can give a user a "personalized" view of the database structure, presenting the database as a set of virtual tables that make sense for that user.

4) **Consistency**

A view can present a consistent, unchanged image of the structure of the database, even if the underlying source tables are split, restructured, or renamed.

5) **Data Integrity**

If data is accessed and entered through a view, the DBMS can automatically check the data to ensure that it meets the specified integrity constraints.

6) **Logical data independence.**

View can make the application and database tables to a certain extent independent. If there is no view, the application must be based on a table. With the view, the program can be established in view of above, to view the program with a database table to be separated.

Disadvantages of views

1) Performance

Views create the appearance of a table, but the DBMS must still translate queries against the view into queries against the underlying source tables. If the view is defined by a complex, multi-table query then simple queries on the views may take considerable time.

2) Update restrictions

When a user tries to update rows of a view, the DBMS must translate the request into an update on rows of the underlying source tables. This is possible for simple views, but more complex views are often restricted to read-only.

6.3 SQL Functions

Functions can be used anywhere in **SQL**, like AVG, COUNT, SUM, MIN, DATE and so on with select statements. **Functions** compile every time. **Functions** must return a value or result. **Functions** only work with input parameters.

6.3.1 Types of Built-In Functions

SQL functions	Description
SQL Aggregate Function	This function can produce a single value for an entire group or table. They operate on sets of rows and return results based on groups of rows.
	Some Aggregate functions are - SQL Count function SQL Sum function SQL Avg function SQL Max function SQL Min function
SQL Arithmetic Function	A mathematical function executes a mathematical operation usually based on input values that are provided as arguments, and return a numeric value as the result of the operation. Mathematical functions operate on numeric data such as decimal, integer, float, real, smallint, and tinyint.
	Some Arithmetic functions are - abs() ceil() floor() exp() ln()

	mod() power() sqrt()
SQL Character Function	A character or string function is a function which takes one or more characters or numbers as parameters and returns a character value. Basic string functions offer a number of capabilities and return a string value as a result set.
	Some Character functions are - lower() upper() trim() translate()

6.3.2 Types of User Defined Functions

6.3.2.1 Scalar Function

The user-defined scalar function also returns a single value as a result of actions performed by the function. We return any datatype value from a function.

```
Create function fnGetEmpFullName
(
        @FirstName varchar(50),
        @LastName varchar(50)
)
returns varchar(101)
As
Begin
        return @FirstName + ' '+ @LastName
end
```

Executing Scalar Value Function

```
SELECT dbo. fnGetEmpFullName('The', 'Swe Zin')
```

6.3.2.2 Table-Valued Function

The user-defined inline table-valued function returns a table variable as a result of actions performed by the function. The value of the table variable should be derived from a single SELECT statement.

Create function fnGetMulEmployee()

```
returns @Emp Table
(
   EmpID int,
   FirstName varchar(50),
   Salary int
)
As
begin
    Insert into @Emp Select e.EmpID,e.FirstName,e.Salary from Employee e;
   --Now update salary of first employee
    update @Emp set Salary=25000 where EmpID=1;
   --It will update only in @Emp table not in Original Employee table
return
end
```

Executing Table Value Function

```
SELECT * FROM dbo.fnGetMulEmployee (1, 'Swe Zin', 10000)
```

6.4 SQL Triggers

SQL Server **triggers** are special stored procedures that are executed automatically in response to the database object, database, and server events. Data manipulation language (DML) **triggers** which are invoked automatically in response to INSERT, UPDATE , and DELETE events against tables.

SQL Server triggers are special stored procedures that are executed automatically in response to the database object, database, and server events.

```
CREATE TRIGGER [schema_name.]trigger_name
ON table_name
AFTER {[INSERT],[UPDATE],[DELETE]}
[NOT FOR REPLICATION]
AS
{sql_statements}
```

Example

Step1. Create Student table

```
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| tid   | int(4)      | NO   | PRI | NULL    | auto_increment |
| name  | varchar(30) | YES  |     | NULL    |                |
| subj1 | int(2)      | YES  |     | NULL    |                |
| subj2 | int(2)      | YES  |     | NULL    |                |
| subj3 | int(2)      | YES  |     | NULL    |                |
| total | int(3)      | YES  |     | NULL    |                |
| per   | int(3)      | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
```

Step 2. Create Trigger

```
CREATE TRIGGER stud_marks
before INSERT
on
Student
for each row
set Student.total = Student.subj1 + Student.subj2 + Student.subj3, Student.per =
Student.total * 60 / 100;
```

Step 3. Insert data into Student Table

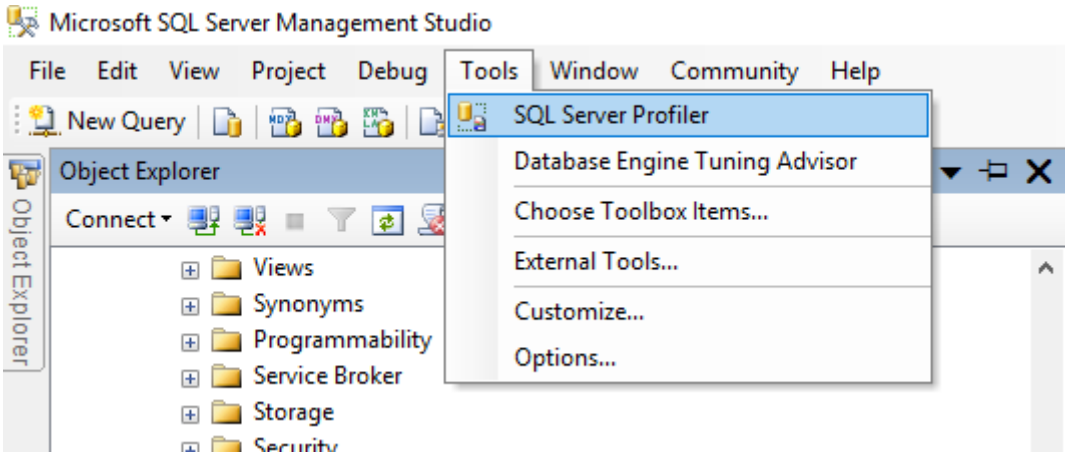
```
INSERT INTO Student VALUES (0, "ABCDE", 20, 20, 20, 0, 0);
```

7. Database Administration Tools

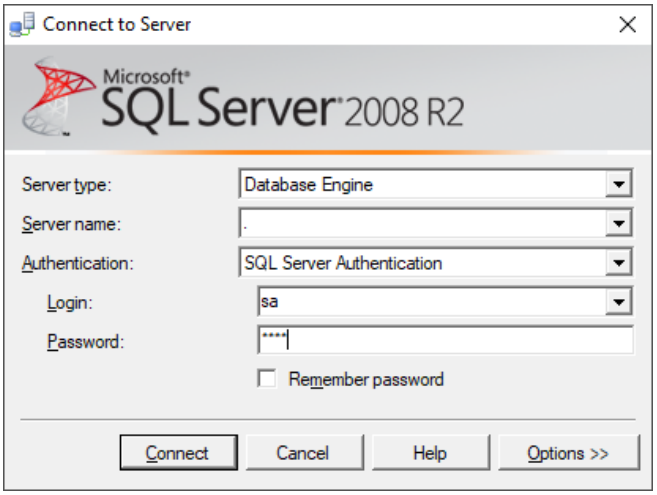
7.1 SQL Server Profiler

Microsoft **SQL Server Profiler** is a graphical user interface to **SQL** Trace for monitoring an instance of the Database Engine or Analysis Services. It allows to capture and save data about each event to a file or table to analyze later.

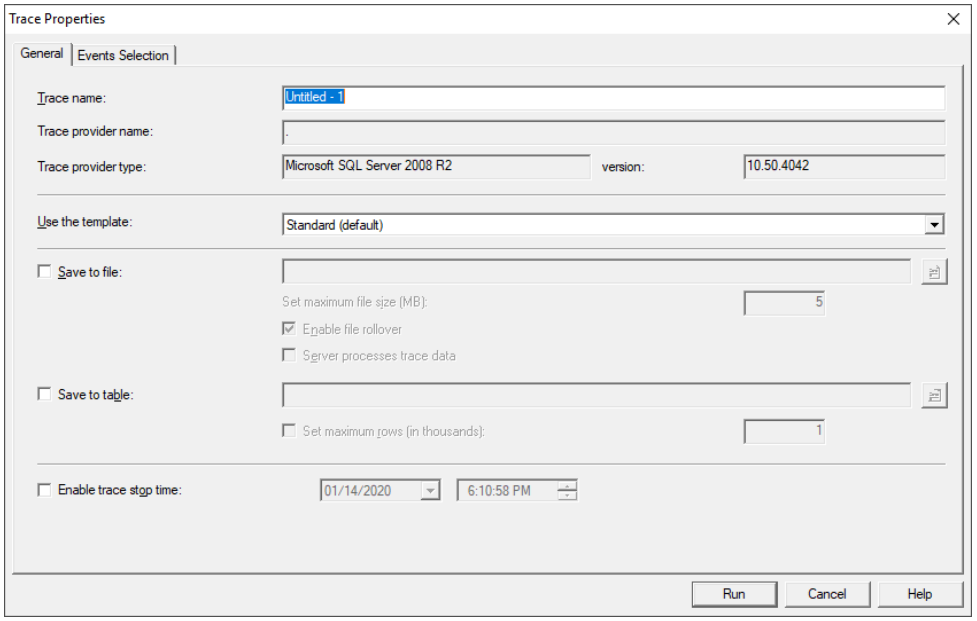
Step 1: In SQL Server Management Studio, Click **Tools** and choose **SQL Server Profiler**



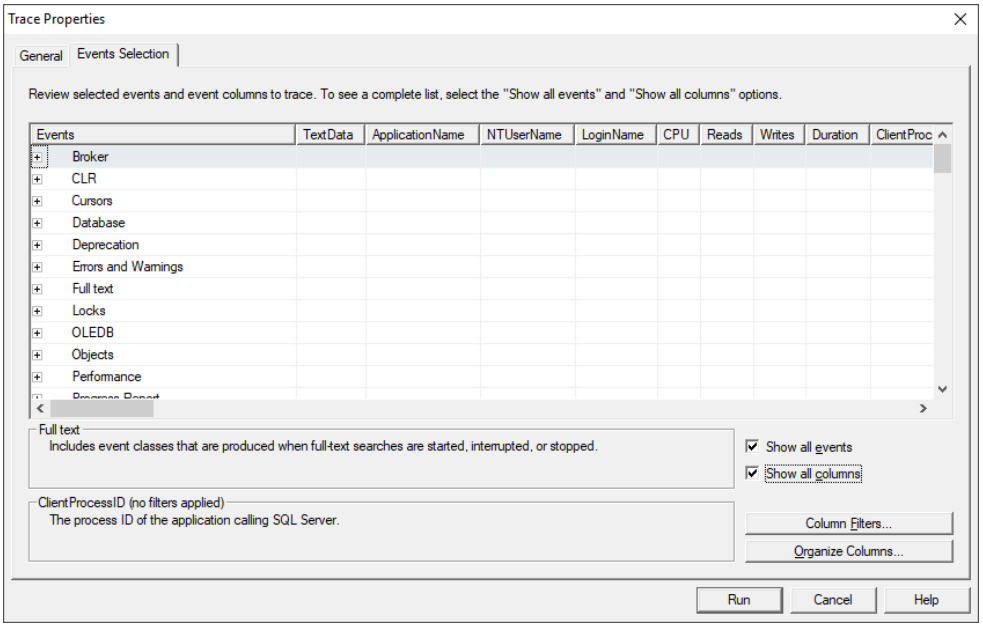
Step 2: Connect to SQL Server Profiler with credential



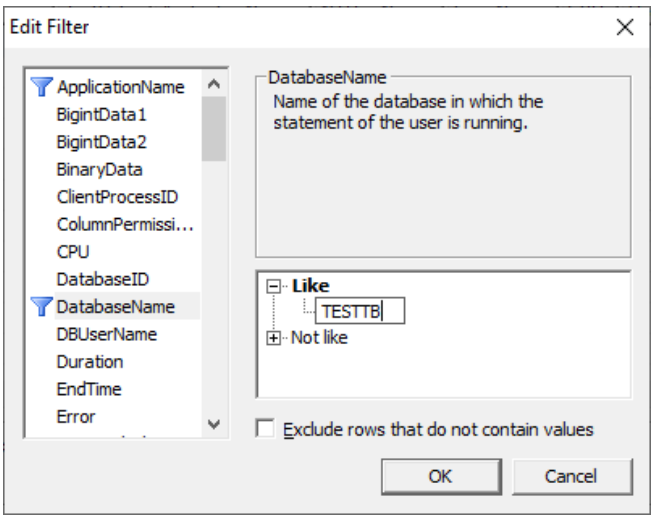
Step 3: Choose Events Selection



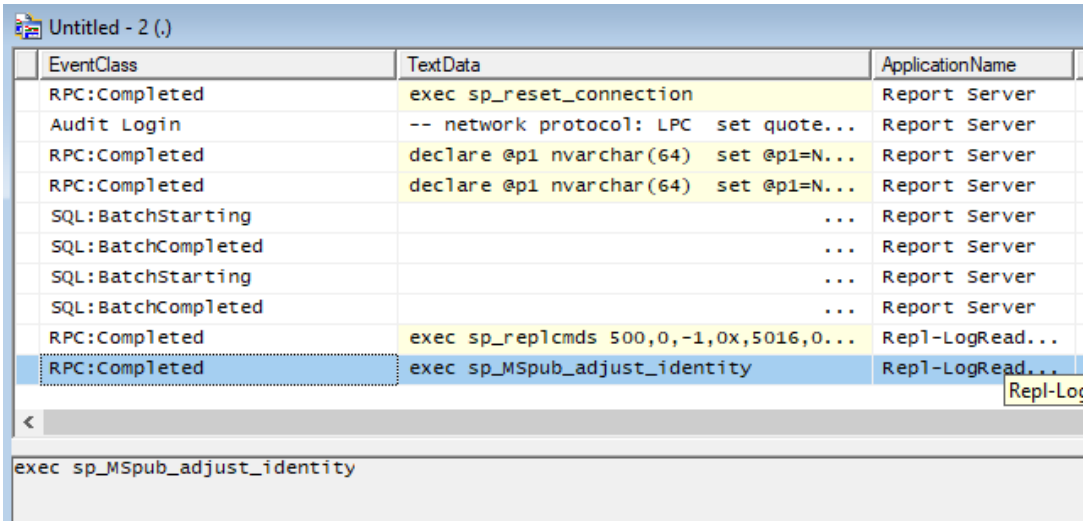
Step 4: Click Show all events and Show all columns and Click Filters



Step 5: Choose Database Name and Type the Database Name

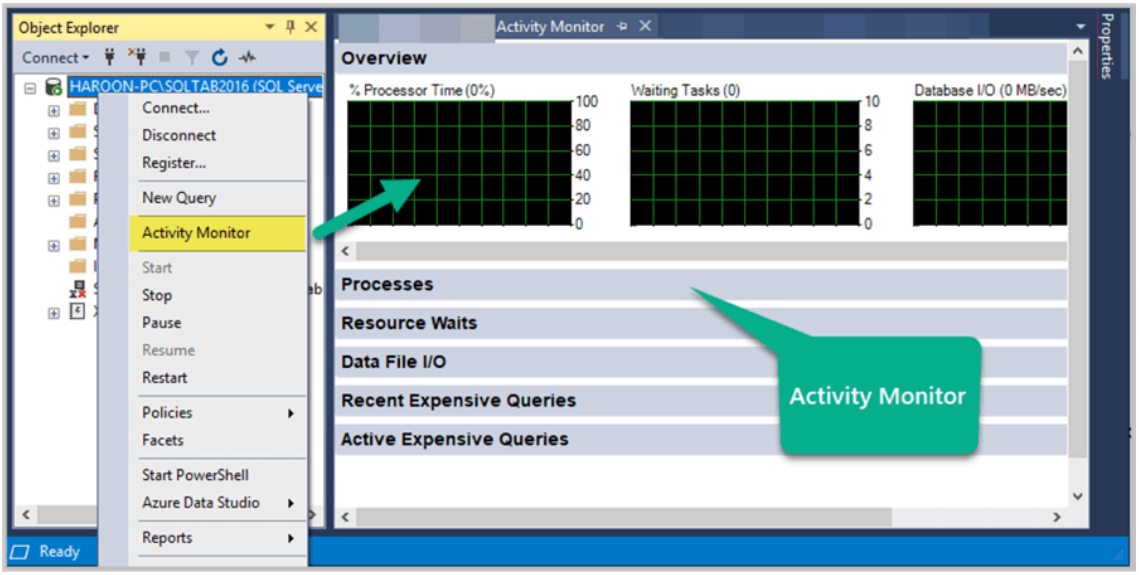


Step 6: Tracing the Query



7.2 Activity Monitor

Activity Monitor displays information about current SQL Server processes and how these processes affect the current instance of SQL Server. Activity Monitor is mainly for monitoring of SQL Server with respect to databases and their queries and other measures to help understand what is impacting SQL Server at the current time. This can help you troubleshoot issues as well as see how SQL Server is using server resources.



Activity monitor panel will display the followings:

1) Percent Processor Time

The percentage of elapsed time that the processor spends to execute non-idle threads for the instance across all CPUs.

2) Waiting Tasks

The number of tasks that are waiting for processor, I/O, or memory resources.

3) Database I/O

The transfer rate, in megabytes per second, of data from memory to disk, disk to memory, or disk to disk.

4) Batch Requests/sec

The number of SQL Server batches that are received by the instance.

7.2.1 Processes

The next section down is Processes. This pane shows a list of all the active users who are connected to the SQL Server instance. It has the following columns:

Session ID - A unique number that is assigned to each user connection when the connection is made.

User Process - Displays 0 for a system process and 1 for a user process. By default, the filter setting for this column is 1. This displays only user processes.

Login - The SQL Server login name under which the session is currently executing.

Database - The name of the database that is included in the connection properties of processes that are currently running.

Task State - The state of the task. For tasks in a runnable or sleeping state, the task state is blank. Otherwise, this can be one of the following values: Background, Running, or Suspended.

Command - The kind of command that is being processed under the task.

Application - The name of the application program that created the connection.

Wait Time (ms) - The time, in milliseconds, in which this task is waiting for a resource. When the task is not waiting, the wait time is 0.

Wait Type - The name of the last or current wait type.

Wait Resource - The name of the resource that is needed.

Blocked By - If there are blocking sessions, the ID of the session that is blocking the task.

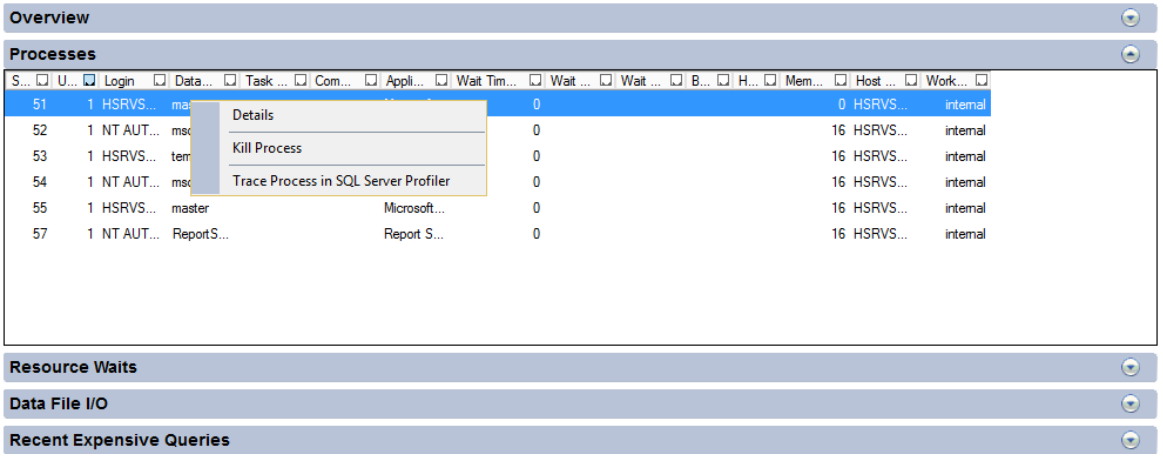
Head Blocker - If there are blocking sessions, identifies the session that causes the first blocking condition. A value of 1 represents a head blocker for other sessions.

Memory Use (KB) - The amount of memory, in kilobytes, that is being used by the task.

Host Name - The name of the computer that made the connection to the instance of SQL Server.

Workload Group - The name of the Resource Governor workload group for the session.

You can right click any of the Session IDs and run a SQL Server Profiler Trace to capture all its activities or you can see the Session Details. You can even KILL a process. We will be going over these items in a future blog post.



7.2.2 Resource Waits

The Resource Waits pane provides valuable information with respect to SQL Server wait times and counts for key resources related to the performance of the instance. This helps identify potential bottlenecks with respect to Memory, CPU, Network I/O, etc. The columns in this pane are:

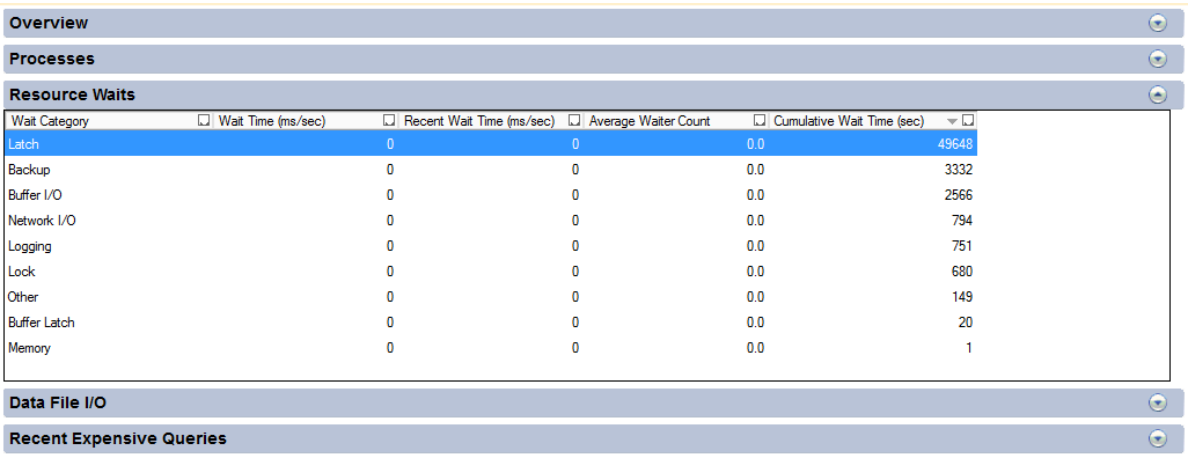
Wait Category - The categories that accumulate wait type statistics. The individual wait types are shown in the Active User Tasks pane.

Wait Time (ms/sec) **- The wait time in milliseconds per second for all tasks that are waiting for one or more resources in the wait category since the last update interval.**

Recent Wait Time (ms/sec) - The weighted average wait time in milliseconds per second for all tasks that are waiting for one or more resources in the wait category since the last update interval.

Average Waiter Count **- The number of tasks that are waiting for one or more resources in the wait category at a typical moment during the last sample interval.**

Cumulative Wait Time (sec) - The total amount of time in seconds that tasks have waited for one or more resources in the wait category since SQL Server was last started on the instance.



7.2.3 Data File I/O

The forth pane down is Data File I/O. This provides disk level I/O (input/output) information related to all the data and log files of user and system databases. This information can be used to quickly identify databases which are performing badly due to disk bottlenecks. The columns in the Data File I/O pane are:

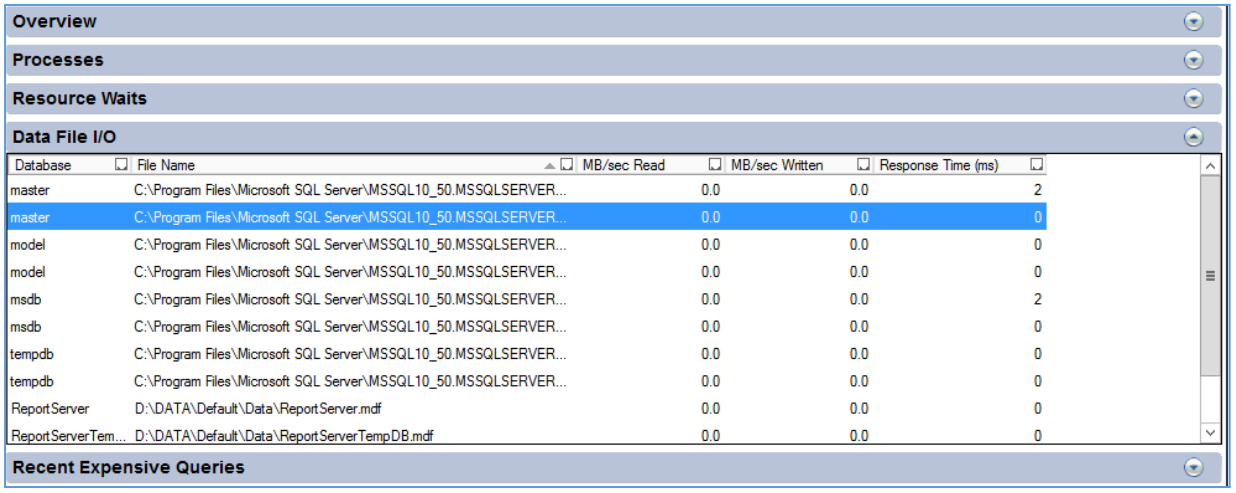
Database - The name of the database.

File Name **- The name of the files that belong to the database.**

MB/sec Read - Recent read activity, in megabytes per second, for the database file.

MB/sec Written **- Recent write activity, in megabytes per second, for the database file.**

Response Time (ms) - Average response time, in milliseconds, of recent read-and-write activity to the database file.



7.2.4 Recent Expensive Queries

At the bottom is the Recent Expensive Queries pane. This allows you to identify poorly performing queries in an instance. You can also right click any of the queries and choose the "Edit Query Text" option to edit the query. Right clicking also lets you see the execution plan of a query by selecting the option "Show Execution Plan". The columns for this pane are:

Query **- The query statement that is being monitored.**

Executions/min – The number of executions per minute for the query.

CPU (ms/sec) **- The rate of CPU use by the query.**

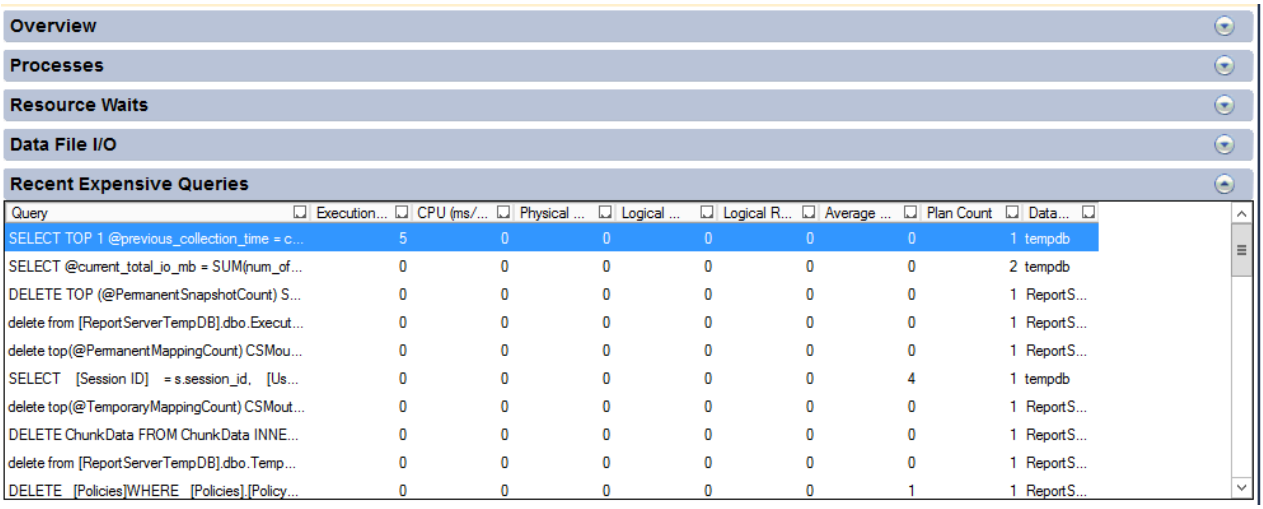
Physical Reads/sec - The rate per second of physical reads by the query.

Logical Writes/sec - The rate per second of logical writes by the query.

Logical Reads/sec **- The rate per second of logical reads by the query.**

Average Duration (ms) – The average duration in milliseconds of running this query.

Plan Count **- The number of cached query plans for this query.**



8. Advance Database Administration

8.1 User Management and Permission

A securable is a specific SQL server resource who access is controlled by the database engine through use of permissions. SQL server includes securable at three different scopes: server, database and schema. Server-scoped securable include such resources as logins, server roles, availability groups, endpoints, and database as a whole. Most SQL Server databases have a number of users viewing and accessing data, which makes security a major concern for the administrator. SQL server security roles, which grant and deny permissions to group of users will reduce the security workload.

This document is intended to provide how to create logins ID to prevent accessing data and schema from unauthorized user.

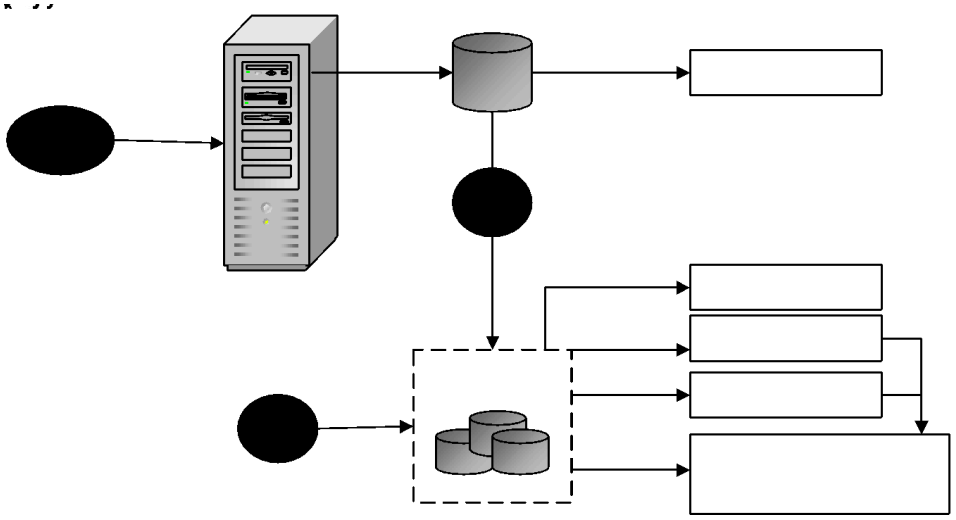


Fig 3.1 MSSQL Server Securable Model

8.1.1 Server and Database Roles in SQL Server

All versions of SQL Server use role-based security, which allows you to assign permissions to a role, or group of users, instead of to individual users.

Database Roles and Users

Logins must be mapped to database user accounts in order to work with database objects. Database users can then be added to database roles, inheriting any permission sets associated with those roles. All permissions can be granted.

The public Role

The public role is contained in every database. It cannot be dropped and you cannot add or remove users from it.

The dbo User Account

The dbo, or database owner, is a user account that has implied permissions to perform all activities in the database. Members of the sysadmin fixed server role are automatically mapped to dbo.

The guest User Account

The guest account is a built-in account in all versions of SQL Server. By default, it is disabled in new databases. If it is enabled, you can disable it by revoking its CONNECT permission by executing the Transact-SQL REVOKE CONNECT FROM GUEST statement.

Fixed Server Roles

Fixed server roles have a fixed set of permissions and server-wide scope. They are intended for use in administering SQL Server and the permissions assigned to them cannot be changed. Logins can be assigned to fixed server roles without having a user

account in a database. The following table shows the fixed server-level roles and their capabilities.

Fixed server-level role	Description
sysadmin	Members of the **sysadmin** fixed server role can perform any activity in the server.
serveradmin	Members of the **serveradmin** fixed server role can change server-wide configuration options and shut down the server.
securityadmin	Members of the **securityadmin** fixed server role 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 of the **processadmin** fixed server role can end processes that are running in an instance of SQL Server.
setupadmin	Members of the **setupadmin** fixed server role can add and remove linked servers by using Transact-SQL statements. (**sysadmin** membership is needed when using Management Studio.)
bulkadmin	Members of the **bulkadmin** fixed server role can run the BULK INSERT statement.
diskadmin	The **diskadmin** fixed server role is used for managing disk files.
dbcreator	Members of the **dbcreator** fixed server role 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

Fixed server-level role	Description
	permissions can be granted, denied, or revoked from the public fixed server roles.

8.1.2 Fixed Database Roles

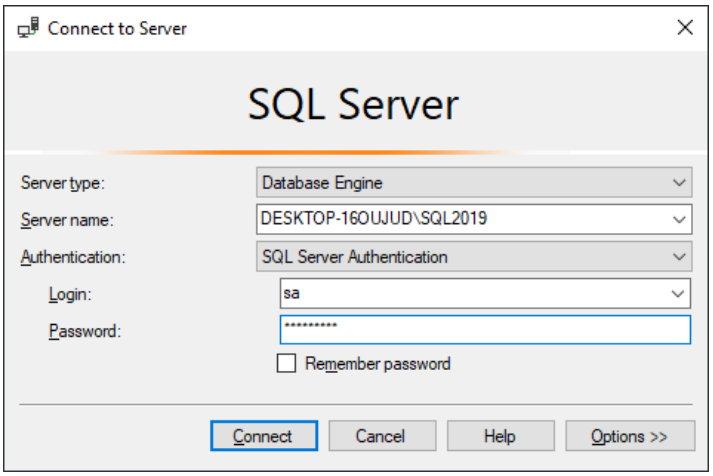
Fixed database roles have a pre-defined set of permissions that are designed to allow you to easily manage groups of permissions. Members of the db_owner role can perform all configuration and maintenance activities on the database. The following table shows the fixed-database roles and their capabilities. These roles exist in all databases. Except for the **public** database role, the permissions assigned to the fixed-database roles cannot be changed.

Fixed-Database role name	Description
db_owner	Members of the **db_owner** fixed database role can perform all configuration and maintenance activities on the database, and can also drop the database in SQL Server. (In SQL Database and SQL Data Warehouse, some maintenance activities require server-level permissions and cannot be performed by **db_owners**.)
db_securityadmin	Members of the **db_securityadmin** fixed database role can modify role membership for custom roles only and manage permissions. Members of this role can potentially elevate their privileges and their actions should be monitored.
db_accessadmin	Members of the **db_accessadmin** fixed database role can add or remove access to the database for Windows logins, Windows groups, and SQL Server logins.
db_backupoperator	Members of the **db_backupoperator** fixed database role can back up the database.
db_ddladmin	Members of the **db_ddladmin** fixed database role can run any Data Definition Language (DDL) command in a database.
db_datawriter	Members of the **db_datawriter** fixed database role can add, delete, or change data in all user tables.
db_datareader	Members of the **db_datareader** fixed database role can read all data from all user tables.

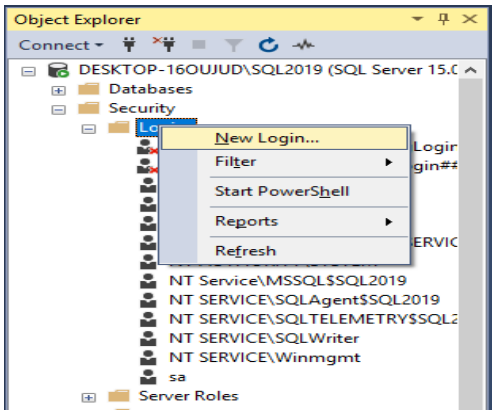
Fixed-Database role name	Description
db_denydatawriter	Members of the **db_denydatawriter** fixed database role cannot add, modify, or delete any data in the user tables within a database.
db_denydatareader	Members of the **db_denydatareader** fixed database role cannot read any data in the user tables within a database.

8.1.3 Create Datab ase Users and Permission

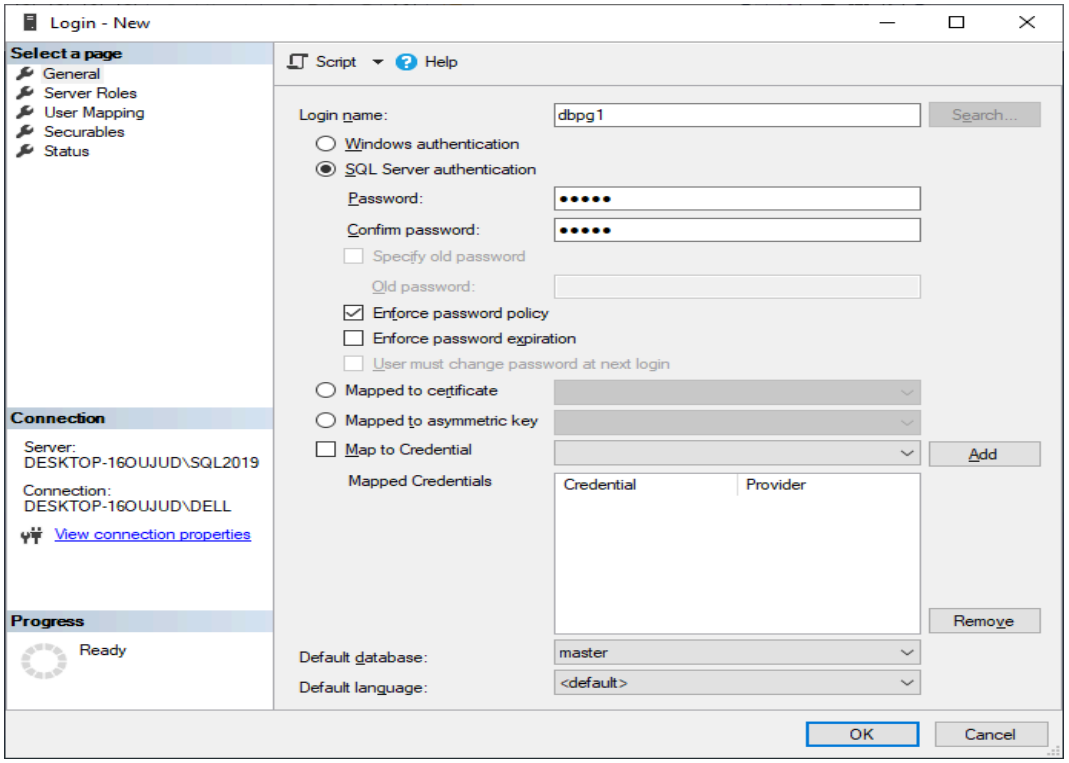
Step 1. Connect to Database with admin account (SA or sysadmin role).

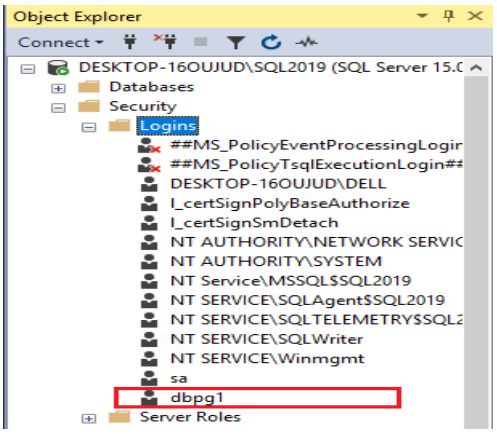


Step 2. Right click Login to create a new login



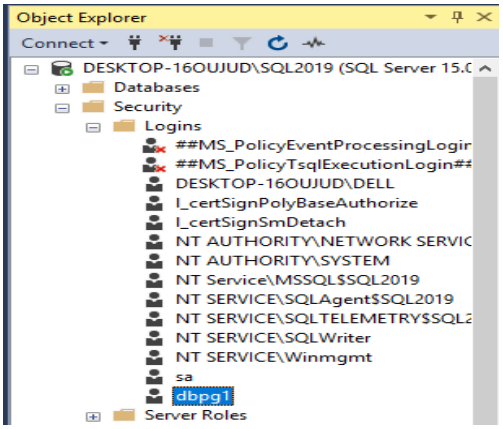
Step 3. Give a name for new login user and configure SQL Server authentication



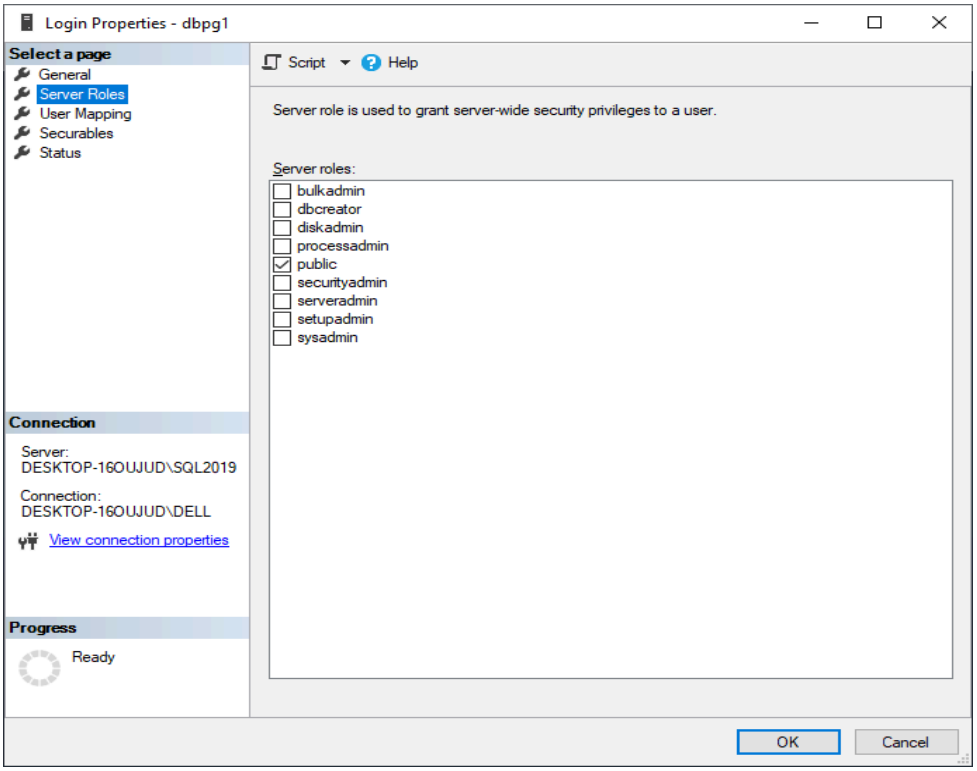


To Create Server role as Public

Step 1. Double clicking on new user name

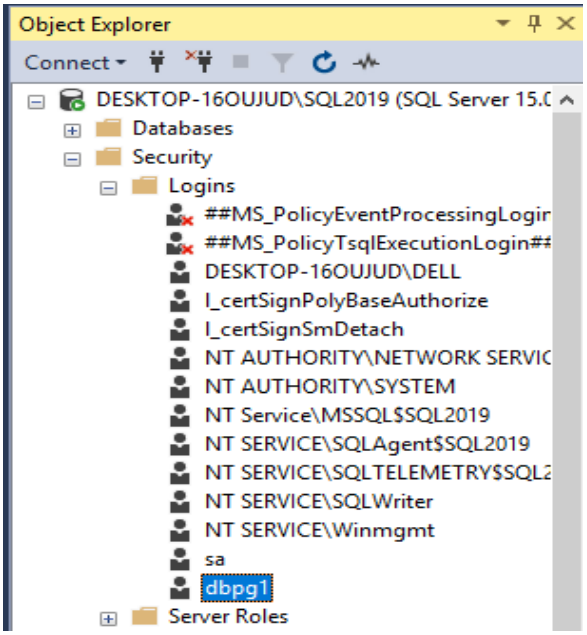


Step 2. Select Server Roles page and check server roles. Server roles must be public.



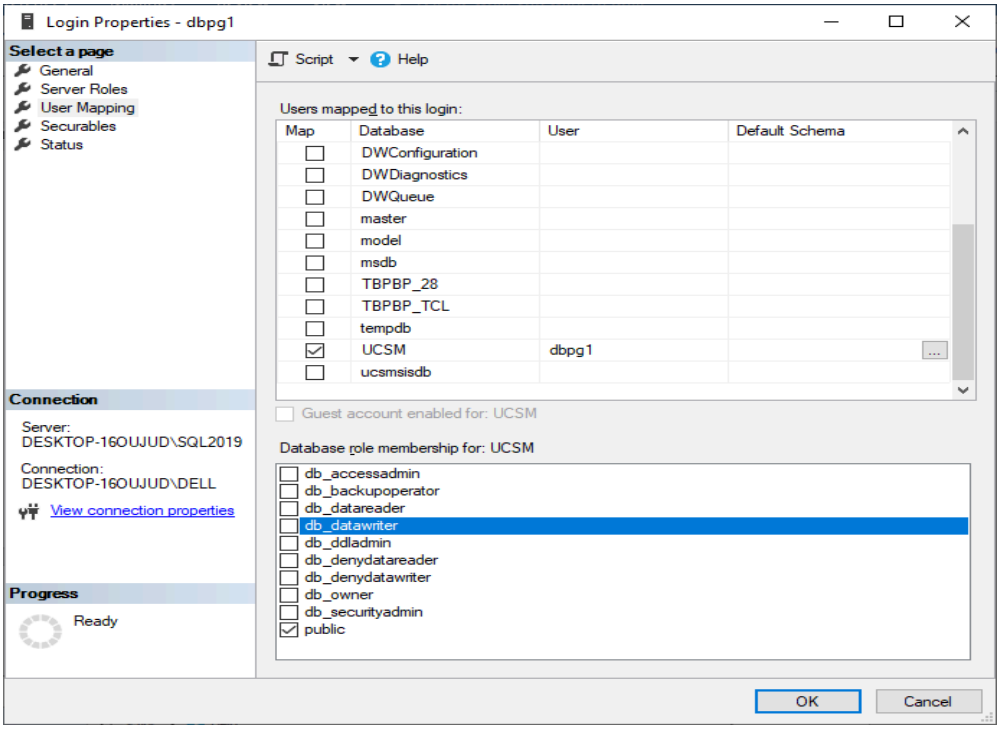
Define User Mapping

Step 1. Double clicking on new user name

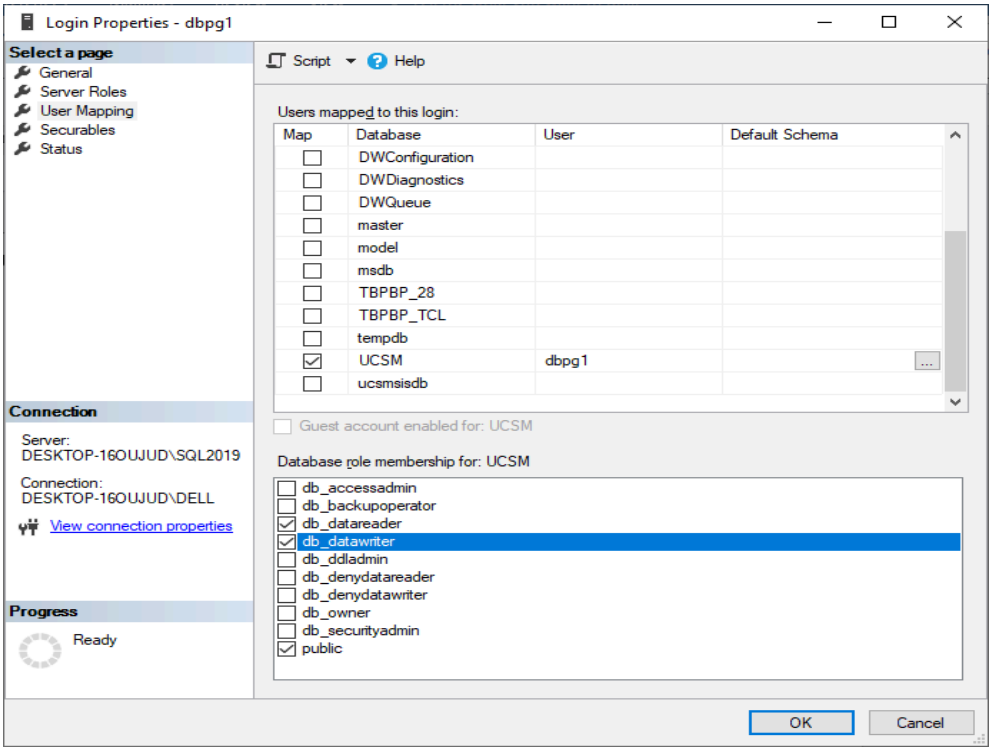


Step 2. Select User Mapping page and choose database name to map with this user.

(user will not be allowed to access other database)

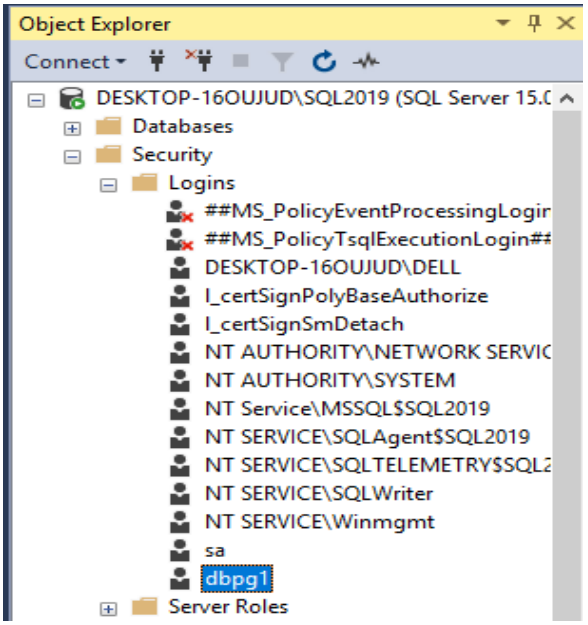


Step 3. Select Database role as db_datareader and db_datawriter, and then click OK button.

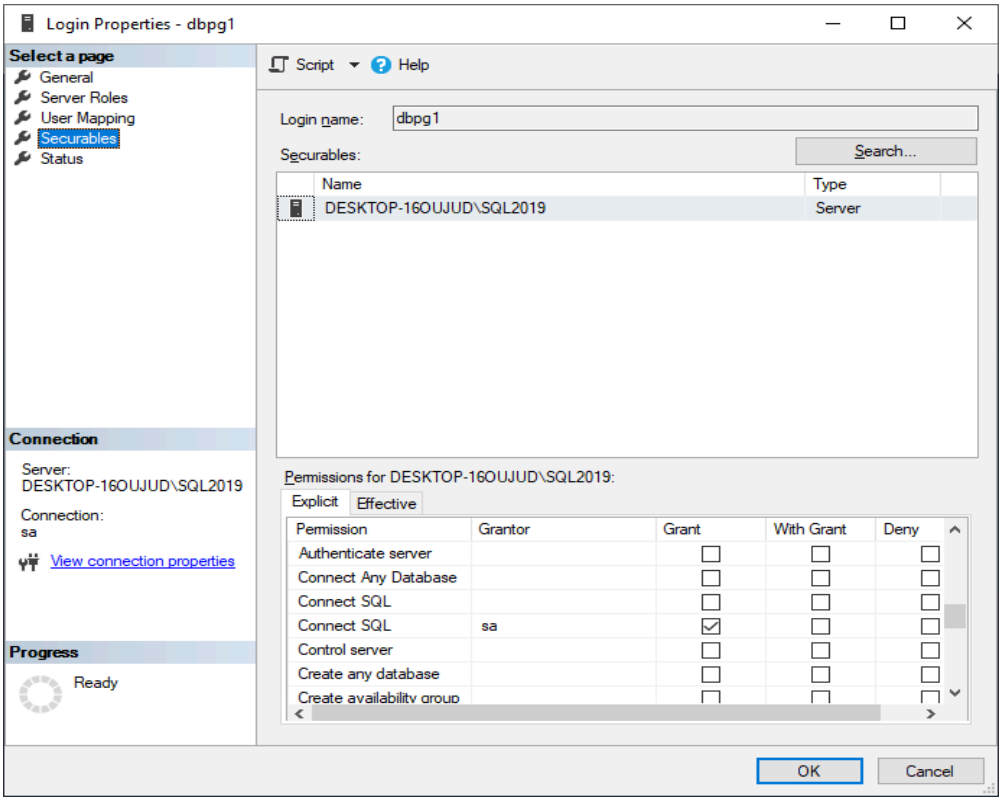


Check Securable

Step 1. Double clicking on new user name

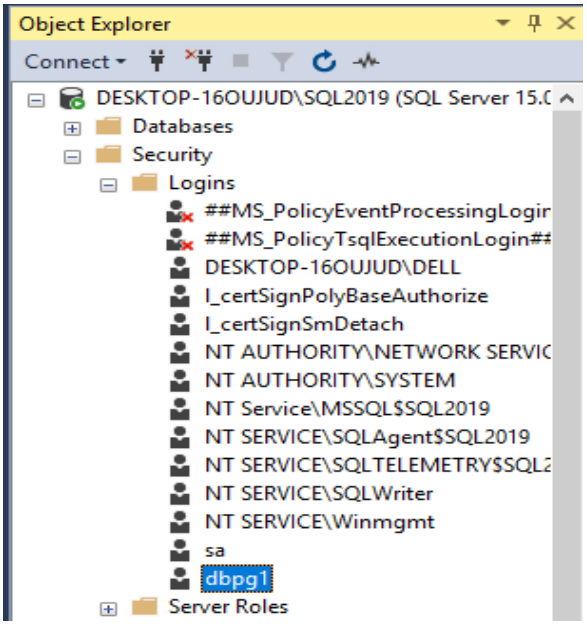


Step 2. Select Securable page and check the permission.

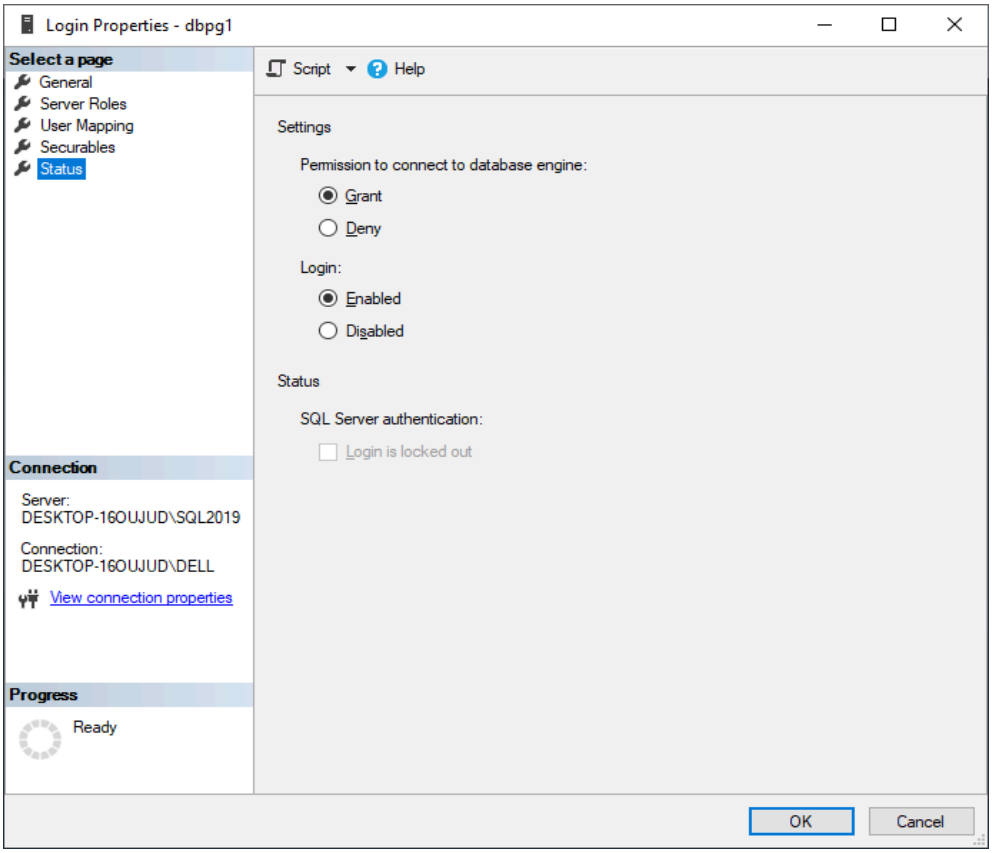


Check User Status

Step 1. Double clicking on new user name

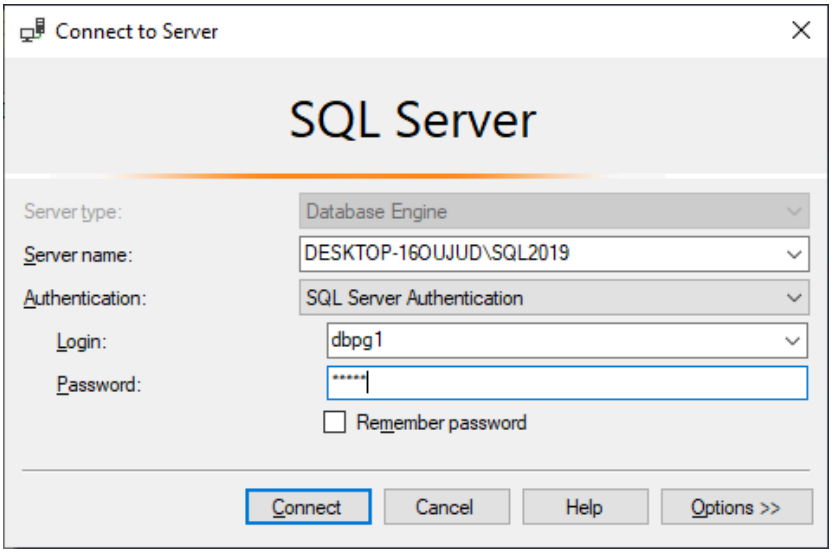


Step 2. Select Status page and check the setting. Setting must be Grant and Enable as shown below figure.

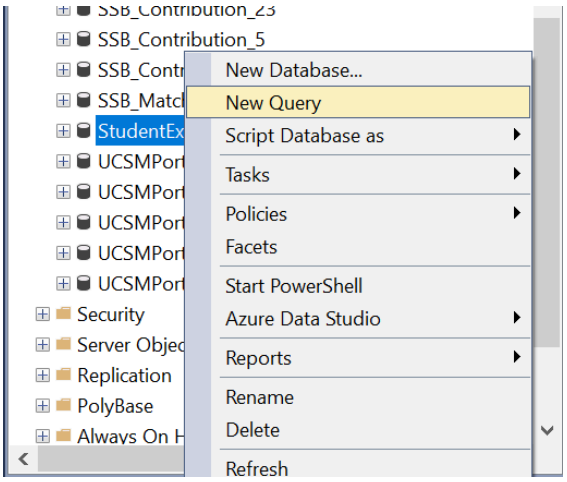


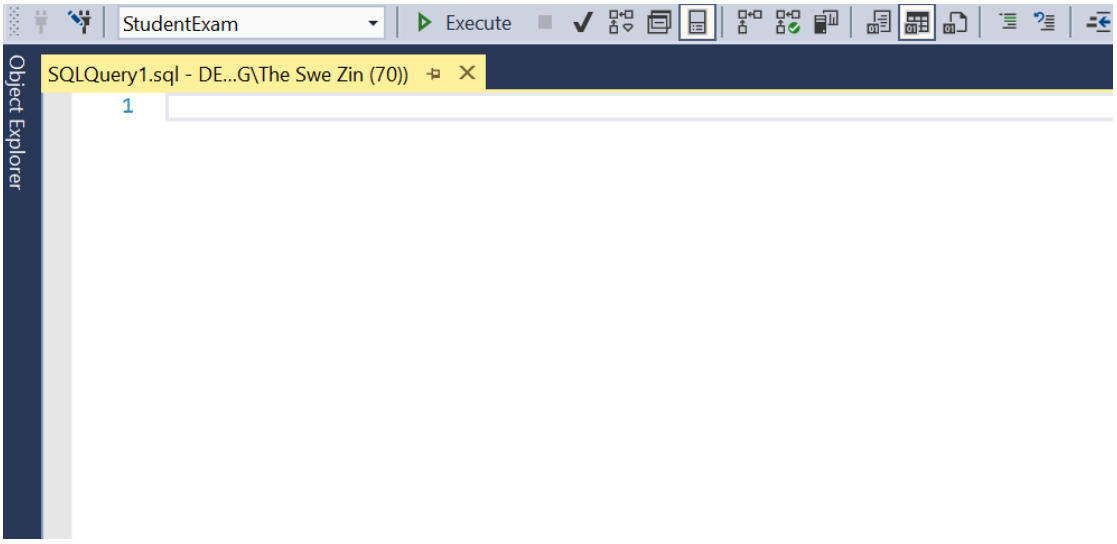
Connect to SQL Server with new User

Step 1: Connect to MSSQL



Step 2: Run Query Editor Window with Accessible Database





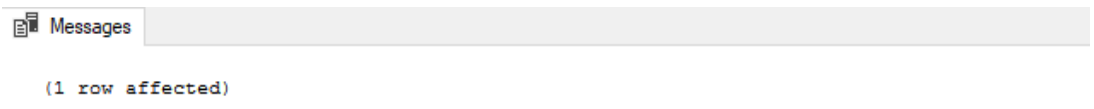
Step 3: To retrieve data from user table (student)

Query: select * from student

	studentID	gender	Name	nationality	placeofBirth	dateofBirth	registerNo
1	1	F	Aye Thandar	MMR	Yangon	2000-10-01	MKPT-0001
2	2	M	Aung Myou	MMR	Yangon	1998-11-01	MKPT-0002

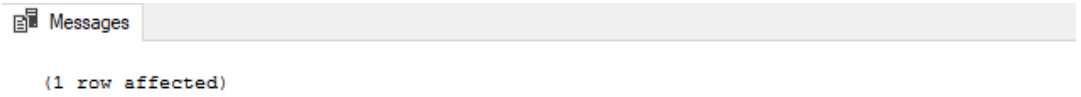
Step 4: Insert data to user table (student)

Insert into student ----------



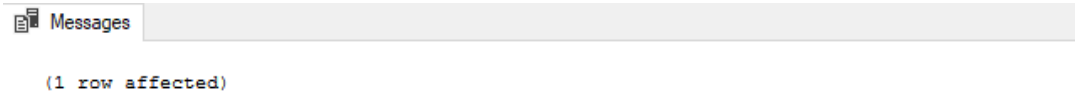
Step 5: Update data to user table (Student)

Update student set name='Maung Myint' where registerNo='3'



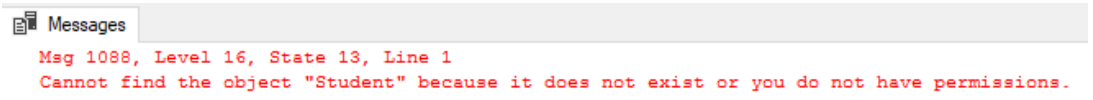
Step 6: Delete data From user table (Student)

```
delete from student where registerNo ='2'
```



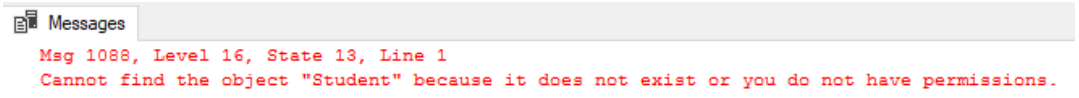
Step 7: To Add column to user table (student)

```
Use StudentExam

Alter Table student Add studentAge int null

GO
```



Step 8: To Drop user table (Student)

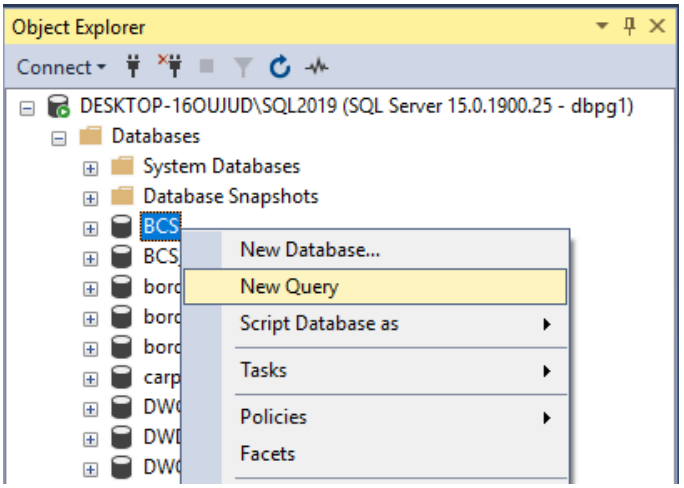
```
Use StudentExam

Drop Table student

GO
```



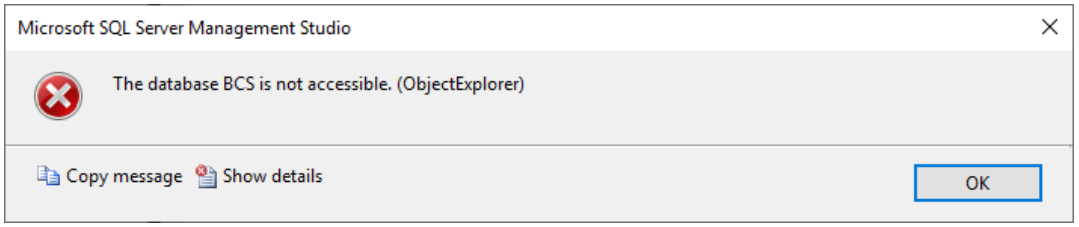
Step 9: Add new Table

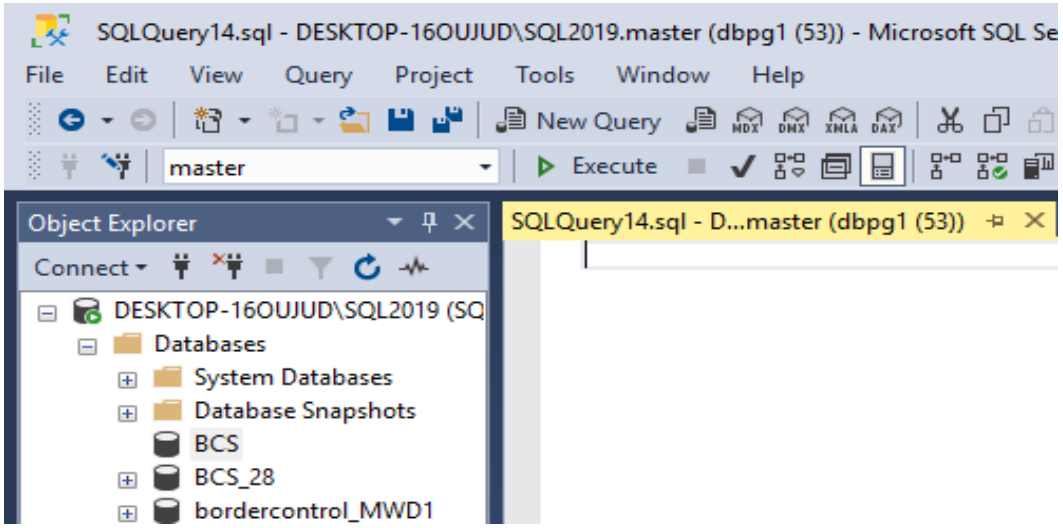
```
Use StudentExam
CREATE TABLE REGISTER
(
        registerID varchar(10) not null,
        registerYear varchar(50) not null,
        createOn datetime not null,
        createdBy varchar(50) not null,
        modyfiedOn datetime null,
        modyfiedBy varchar(50)
)
GO
```

Step 10: Run Query Editor Window for Not-Accessible Databases



This type of user can't access to other database because system admin doesn't allow to connect other database.





9. Database Backup and Recovery

9.1.1 Backup Database

Backup is a copy of data/database, etc. Backing up MS SQL Server database is essential for protecting data. MS SQL Server backups are mainly three types - Full or database backup, Differential or Incremental, and Transactional Log or Log.

9.1.2 Database Backup Types

(1) Full **backups**

The most common types of SQL Server backups are complete or full backups, also known as database backups. These backups create a complete backup of database as well as part of the transaction log; all of the contents are contained in one backup.

(2) Differential **backups**.

A "Differential" backup is a backup of any extent that has changed since the last "Full" backup was created.

(3) Incremental backups

An incremental backup covers all files that have been changed since the last backup was made, regardless of backup type. If your business runs a full backup on Friday and an incremental backup on Monday, the incremental backup would copy all files changed between Friday and Monday. If you run a differential backup on Tuesday and an incremental backup on Thursday, the incremental backup would affect all files modified between Tuesday and Thursday.

(4) Transaction log **backups**

If your database is set to the "Full" or "Bulk-logged" recovery model then you will be able to issue "Transaction Log" backups. By having transaction log backups along with full backups you have the ability to do a point in time restore, so if someone accidently deletes all data in a database you can recover the database to the point in time right before the delete occurred.

Pros and Cons of Backups Type

Backup Type	Pros	Cons
Full	• Potential for fast, total recovery of data assets. • All back-ups are contained in a single version. • Minimal time needed to restore business operations.	• Requires the most storage space. • Relatively time-consuming to complete the backup process.
Incremental	• Minimal time to complete backup. • Requires the least storage space.	• Recovery time may be slower. • Requires a full backup in addition to incremental backups for complete recovery and requires the piecing together of data from multiple backup sets. • Small potential for incomplete data recovery if one or more backup sets has failed.
Differential	• Requires less storage space than full backups. • Only two backups (last full and most recent incremental) are required for recovery.	• Slower than incremental. • Requires an initial full backup for complete recovery. • IT will need to piece together two backup sets. • Potential for failed recovery if one or more backups is incomplete.

9.1.3 Database Recovery

Database recovery is the process of restoring the **database** to a correct (consistent) state in the event of a failure. In other words, it is the process of restoring the **database** to the most recent consistent state that existed shortly before the time of system failure.

9.1.4 Database Recovery Models

Every database require a recovery model which signifies that what sort of backup is required or can be perform by the user to restore the data which could be lost due to any hardware failure or other issue.

There are generally three types of recovery models of database, these are explained as following below.

1) **Simple Recovery :**

In this model, the transaction logs get automatically removed without causing and change to the files size, because of this it is difficult to make log backups. Simple Recovery does not support backup of transaction log. It supports both full and bulk_logged backup operations. Some operations that aren't supported by this model are : Log shipping, AlwaysOn or Mirroring and Point-in-time restore. In this case the database is use only for testing and development. The data in this operation is static. It does not have the provision for point-to-time recovery.

2) **Full Recovery :**

Unlike simple recovery, it supports backups of transaction log. There will no loss of work due to damaged or lost files as this model keeps tract of every operation performed on database. It supports point-in-time for recovery for database, because of which it can recover upto an arbitrary point. When this model is use by database, the transaction logs will grow in huge number (infinitely) which will cause a problem like system crash. So to prevent it we must backup transaction log on regular basis.

3) **Bulk logged :**

This model has similarity with Full Recovery Model as in both transaction logs are backup. It has high performance for bulk operations. It helps in importing bulk data quicker than other model and this keeps the transaction file size low. It did not support point-in-time recovery. If you perform the transactions under this model which require transaction log restoration, then there could be data loss.

**********END**********