

Programming 3 Assignment 4

Assignment Overview

- The specifications associated with this assignment will continue building a CRUD application along with a relational database. The purpose of this application is to allow School Administrators to maintain database data.
- Work associated with this assignment will reinforce the student's learning in the following areas:
 - ASP.NET MVC
 - Object Relational Model
 - The Singleton Pattern
 - Data Migrations
 - Data Import
 - Stored Procedures

Evaluation

- The evaluation of this assignment will be in the form of a Code Review and Run Time Demonstration.
- During the Code Review evaluation will be based on:
 - Comments
 - Documentation
 - Adherence to Standards – and Pattern Best Practices
 - Coding choices (i.e. refactoring, no repeated code, readability etc...)
 - Implementation of Requirements
 - Verbal discussion of code
- During the Run-Time Demonstration evaluation will be based on:
 - A variety of run-time tests

Specifications

MVC Project

- Continue with the ASP.NET MVC project called **BITCollege_FL** based on the following specifications:

Class Diagram

- The class diagram used in earlier Assignments has been modified. New classes and methods have been added.
- Modify your SchoolModels.cs file based on the class diagram below and the specific instructions that follow.

Programming 3 Assignment 4

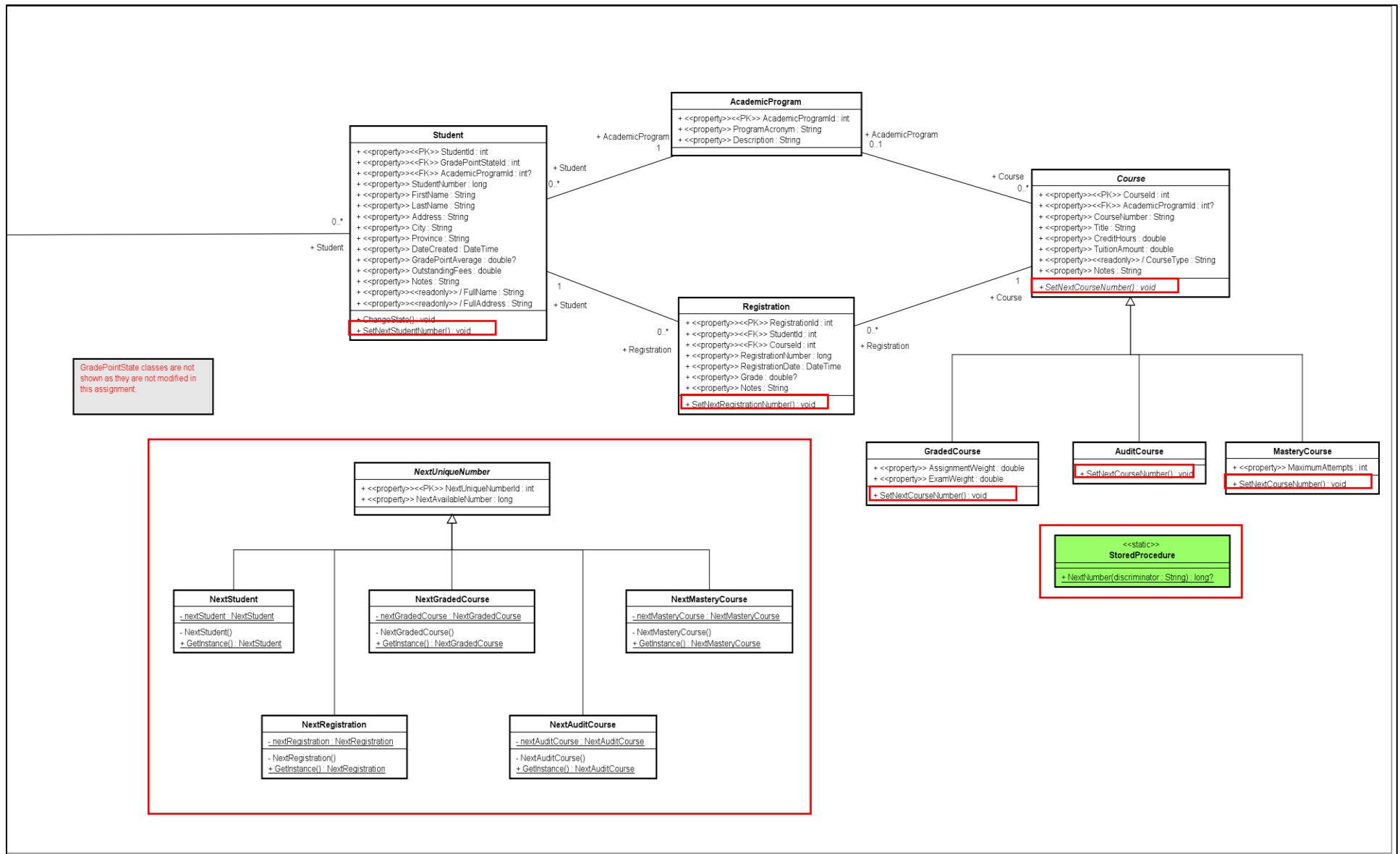


Figure 1 (electronic version in the Assignment 4 Files folder)

Programming 3 Assignment 4

New Models

- The following changes will result in the need for a migration.

NextUniqueNumber

- Define this model based on the class diagram above as well as the following specification:
 - The NextUniqueNumberId value must be automatically generated as an identity field.
 - NextUniqueNumberId must have a key annotation.
 - The NextAvailableNumber value is required.

NextUniqueNumber sub classes

- The models that are defined as sub classes to the NextUniqueNumber abstract base class implement the Singleton Pattern. Specific requirements for these models appear below. For now, define the models and code empty methods to satisfy the compiler.

StoredProcedure

- The class depicted using a green background is a class used to execute SQL Server stored procedures. Specific instructions for this class appear below. For now, define the class and code an empty method to satisfy the compiler.

Programming 3 Assignment 4

Existing Models

- The following changes may result in the need for a migration.

Student

- Code an empty method for `SetNextStudentNumber`. Specific instructions appear below.

Registration

- Code an empty method for `SetNextRegistrationNumber`. Specific instructions appear below.

Course

- Define the `SetNextCourseNumber` abstract method.

GradedCourse

- Override the inherited `SetNextCourseNumber` method. Specific instructions appear below.

MasteryCourse

- Override the inherited `SetNextCourseNumber` method. Specific instructions appear below.

AuditCourse

- Override the inherited `SetNextCourseNumber` method. Specific instructions appear below.

Programming 3 Assignment 4

Controllers & Views

- Define Controllers and Views for the following models using the Scaffolding Option which will generate *MVC 5 Controller with views, using Entity Framework*:
 - NextUniqueNumber (do not generate views for this model)
 - NextAuditCourse
 - NextGradedCourse
 - NextMasteryCourse
 - NextRegistration
 - NextStudent
- **Note**: StoredProcedure is not a model class and as such does not require controllers and views.

Programming 3 Assignment 4

Singleton Pattern

NextUniqueNumber

- In the NextUniqueNumber base class, instantiate a protected static variable of your data context object (BITCollege_FLContext) so that the following can be achieved in each of the derived classes.
 - **Note:** This data context object is defined as:
 - Protected because it is intended to be used in the sub classes.
 - Static as it will be used in the static GetInstance method implementations below.

NextUniqueNumber Sub Classes

- Implement the Singleton Pattern for **EACH** of the following models:

Model	Private Constructor requirement
NextStudent	Set NextAvailableNumber to 20000000
NextGradedCourse	Set NextAvailableNumber to 200000
NextAuditCourse	Set NextAvailableNumber to 2000
NextMasteryCourse	Set NextAvailableNumber to 20000
NextRegistration	Set NextAvailableNumber to 700

- Code the GetInstance method for **EACH** of the NextUniqueNumber sub classes:
 - Review Assignment 3 as necessary when implementing the Singleton Patterns for the above models.
- **Reminder:** Modify the Index ActionResult method in each of the NextUniqueNumber sub class Controllers to incorporate the use of the GetInstance method.
- **Reminder:** Delete the Create and Index Views. Create a copy of the Details view and rename that copy to Index.cshtml for each of the NextUniqueNumber sub classes.

Test

- Test these Singleton implementation for **EACH** of the NextUniqueNumber sub classes using techniques learned in Assignment 3.
- **Note:** If you have not yet done a migration, one will be necessary now.

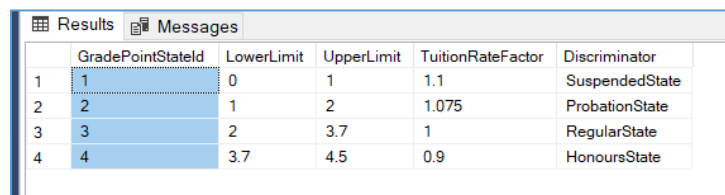
Programming 3 Assignment 4

SQL Sever Management Studio

- Open SQL Server Management Studio and connect to the **BITCollege_FLContext** database.

Key Seed Values

- Ensure the Id field values for the GradePointStates table have consecutive key values for GradePointStateId starting at 1 as shown below:



The screenshot shows the 'Results' pane in SQL Server Management Studio. It displays a table with the following data:

	GradePointStateId	LowerLimit	UpperLimit	TuitionRateFactor	Discriminator
1	1	0	1	1.1	SuspendedState
2	2	1	2	1.075	ProbationState
3	3	2	3.7	1	RegularState
4	4	3.7	4.5	0.9	HonoursState

Figure 2

- If not, delete all records from this table and run the following query. Replace DATABASENAME with your actual database name:

```
--sets next available identity field to 1
USE [DATABASENAME]
GO
DBCC CHECKIDENT (GradePointStates, RESEED, 0)
```

Code 1

- Once the above script completes, re-enter the data as shown above. This can be done in SQL Server Management Studio, or by running the appropriate views in your MVC application.

Programming 3 Assignment 4

- Ensure the next available id fields correspond to the scripts below by executing the following scripts in SQL Server Management Studio. Replace DATABASENAME with your actual database name:

```
--sets next available Students identity field to 101
USE [DATABASENAME]
GO
DBCC CHECKIDENT (Students, RESEED, 100)
GO

--sets next available Courses identity field to 101
USE [DATABASENAME]
GO
DBCC CHECKIDENT (Courses, RESEED, 100)
GO

--sets next available Registrations identity field to 101
USE [DATABASENAME]
GO
DBCC CHECKIDENT (Registrations, RESEED, 100)
GO
```

Code 2

Data Entry Scripts

- Open **EACH** of the SQL Server scripts accompanying this assignment found in the Assignment 04 Files\SQL Server Scripts.
 - In **EACH** of the scripts, replace [DATABASENAME] at the top of the INSERT statement with the actual name of your database.
- Run these scripts **in the order** specified below:
 - studentinsert.sql
 - courseinsert.sql
 - registrationinsert.sql

Programming 3 Assignment 4

Visual Studio

- Return to Visual Studio.

Stored Procedure

- Create a folder within the Migrations folder of your project and name it SQL
- Add the following SQL files found in the Assignment 04 Files\Stored Procedures folder to the SQL folder created above:
 - create_next_number.sql
 - drop_next_number.sql
- **Note:** Review the code in the above sql scripts so that you understand the functionality of the stored procedure.
- If a Project Resource File (Resources.resx) does not exist in the Project Properties folder, Create a Project Resource file in that location
 - Add New Item -> General -> Resources File
 - If the newly created Project Resource file is not within Project Properties folder, move it to the BITCollege_FL Properties folder

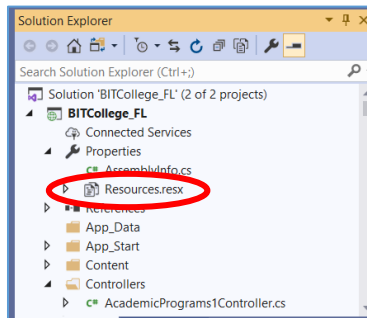


Figure 3

- Open the Resources.resx file, and add the two files currently in the SQL Migrations folder to the Project Resources file as File resources (drag and drop the two files into the open resx file).
- Create a new migration for the project called StoredProcedures using the Package Manager Console.
- In the migration code that is generated code the up and down methods as follows:

Programming 3 Assignment 4

```
public override void Up()
{
    //call script to create the stored procedure
    this.Sql(Properties.Resources.create_next_number);
}

public override void Down()
{
    //call script to drop the stored procedure
    this.Sql(Properties.Resources.drop_next_number);
}
```

Code 3

- **Note:** In the above, replace the word “Resources” with the name of your Project Resource File
- Update the database based on the StoredProcedure migration requirements using the Package Manager Console
- The next_number stored procedure should now reside in database within the Programmability\Store Procedures folder (a refresh may be required)

SQL Server Management Studio

- Return to SQL Server Management Studio.
- Test the stored procedure within SQL Server Management Studio by executing the stored procedure (Right click on the stored procedure and select Execute Stored Procedure...) using NextRegistration as the discriminator argument
- **Note:** The stored procedure contains two parameters. The first is an input parameter representing the discriminator of the row whose value needs to be incremented, the second argument is an output parameter which will provide the value of the next available number. When testing, only provide the input parameter.

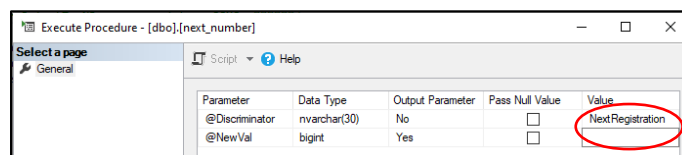


Figure 4

- Do not proceed if the value of the NextRegistration row in the NextUniqueNumbers table is not set as the @NewVal output parameter value.

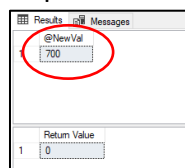


Figure 5

Programming 3 Assignment 4

Visual Studio

- Return to Visual Studio.

SchoolModels

StoredProcedure class

- Code the NextNumber method in the StoredProcedures class as follows:
- Ensure appropriate class/method documentation is added.
- Include exception handling such that if an exception occurs a value of null is returned from the static method
 - Wrap the try/catch block around the entire contents of the method.
- Ensure the data source and the name of the database below (in bold font) matches the name of your database and server connection.
- It may be necessary to incorporate using statements based on some of the classes used in the code below.

```
public static class StoredProcedures
{
    public static long? NextNumber(string discriminator)
    {
        long? returnValue = 0;

        SqlConnection connection = new SqlConnection("Data Source=DBMS2_SERVER; " +
            "Initial Catalog=BITCollege_FLContext;Integrated Security=True");
        SqlCommand storedProcedure = new SqlCommand("next_number", connection);
        storedProcedure.CommandType = CommandType.StoredProcedure;
        storedProcedure.Parameters.AddWithValue("@Discriminator", discriminator);
        SqlParameter outputParameter = new SqlParameter("@NewVal", SqlDbType.BigInt)
        {
            Direction = ParameterDirection.Output
        };
        storedProcedure.Parameters.Add(outputParameter);
        connection.Open();
        storedProcedure.ExecuteNonQuery();
        connection.Close();
        returnValue = (long?)outputParameter.Value;
        return returnValue;
    }
}
```

Code 4

Using the Stored Procedure

- When using the NextNumber method, remember that it returns a long?. Null will be returned when an exception occurs. If an exception does not occur, the long? will need to be cast to a long (in some cases) to satisfy the compiler.

Programming 3 Assignment 4

Course Sub Classes

- In **EACH** of the Course sub classes, code the SetNextCourseNumber methods to set the CourseNumber property to the appropriate value as follows:
 - For GradedCourses, set the CourseNumber to the value of “G-” followed by the value returned from the NextNumber static method defined above.
 - E.g. “G-200000”
 - For MasteryCourses, set the CourseNumber to the value of “M-” followed by the value returned from the NextNumber static method defined above.
 - E.g. “M-20000”
 - For AuditCourses, set the CourseNumber to the value of “A-” followed by the value returned from the NextNumber static method defined above.
 - E.g. “A-2000”

Registration

- In the Registration model, code the SetNextRegistrationNumber method to set the RegistrationNumber property to the appropriate value returned from the NextNumber static method defined above.

Student

- In the Student model, code the SetNextStudentNumber method to set the StudentNumber property to the appropriate value returned from the NextNumber static method defined above.

Programming 3 Assignment 4

Controllers

- Modify **EACH** of the following controllers to make use of the SetNext...() methods

AuditCoursesController

- Modify the AuditCoursesController [HttpPost] Create method by calling the SetNextCourseNumber method.
 - Ensure this is the first line of code within the “if” statement.

GradedCoursesController

- Modify the GradedCoursesController [HttpPost] Create method by calling the SetNextCourseNumber method.
 - Ensure this is the first line of code within the “if” statement.

MasteryCoursesController

- Modify the MasteryCoursesController [HttpPost] Create method by calling the SetNextCourseNumber method.
 - Ensure this is the first line of code within the “if” statement.

RegistrationsController

- Modify the RegistrationsController [HttpPost] Create method by calling the SetNextRegistrationNumber method.
 - Ensure this is the first line of code within the “if” statement.

StudentsController

- Modify the StudentsController [HttpPost] Create method by calling the SetNextStudentNumber method.
 - Ensure this is the first line of code within the “if” statement.

Programming 3 Assignment 4

SchoolModels

- The process of setting the StudentNumber, RegistrationNumber and CourseNumber properties is now automated. These fields no longer require validation as the stored procedure is causing only valid data to be generated for these fields.
- The following changes may result in the need for a migration.

Student

- Remove any [Range] and [Required] data annotations for the StudentNumber field

Registration

- Remove any [Range] and [Required] data annotations for the RegistrationNumber field

Course

- Remove any [Range] and [Required] data annotations for the CourseNumber field.

Views

- For the following **Create** views **remove** the `<div class="form-group">` block for the fields whose values will now be generated by the stored procedure:
 - AuditCourse (remove `<div class="form-group">` for CourseNumber)
 - GradedCourse (remove `<div class="form-group">` for CourseNumber)
 - MasteryCourse (remove `<div class="form-group">` for CourseNumber)
 - Student (remove `<div class="form-group">` for StudentNumber)
 - Registration (remove `<div class="form-group">` for RegistrationNumber)
- For example, remove the following block in the AuditCourse Create view (do the same for the remaining views listed above):

```
<div class="form-group">
  @Html.LabelFor(model => model.CourseNumber, htmlAttributes: new { })
  <div class="col-md-10">
    @Html.EditorFor(model => model.CourseNumber, new { htmlAttributes = new { } })
    @Html.ValidationMessageFor(model => model.CourseNumber, "", new { })
  </div>
</div>
```

Figure 6

- For **EACH** of the following **Edit** views the fields cannot be completely deleted as their values will default to 0 upon update. Instead, modify the razor syntax such that the `@Html.EditorFor(...` is replaced with `@Html.HiddenFor(...`. In these cases, still remove the remaining `<div class="form-group">` block)

Programming 3 Assignment 4

- See example below:
 - AuditCourse (make the changes shown below for CourseNumber)
 - GradedCourse (make the changes shown below for CourseNumber)
 - MasteryCourse (make the changes shown below for CourseNumber)
 - Student (make the changes shown below for StudentNumber)
 - Registration (make the changes shown below for RegistrationNumber)

Before:

```
<div class="form-group">
  @Html.LabelFor(model => model.CourseNumber, htmlAttributes: new { @class = "control"
  <div class="col-md-10">
    @Html.EditorFor(model => model.CourseNumber, new { htmlAttributes = new { @class = "form-control"
    @Html.ValidationMessageFor(model => model.CourseNumber, "", new { @class = "text-danger"
  </div>
</div>
```

After:

```
@Html.HiddenFor(model => model.CourseNumber, new { htmlAttributes = new { @class = "form-control"
@Html.ValidationMessageFor(model => model.CourseNumber, "", new { @class = "text-danger"
@Html.EditorFor(model => model.CourseNumber, new { htmlAttributes = new { @class = "form-control"
@Html.ValidationMessageFor(model => model.CourseNumber, "", new { @class = "text-danger"
```

Test

- Ensure the Singleton pattern has been tested for each of the models in which it is implemented. This can be confirmed by retrieving the records in the NextUniqueNumbers table. There should be five records.
- Test the use of the Stored Procedure by running the application and using your views to insert new records for the following:
 - Students
 - Registrations
 - GradedCourses
 - MasteryCourses
 - AuditCourses
- In each case, the new record should persist and the record should be given the next available number through successful execution of the stored procedure.

Backup

- Backup your Solution.
- Backup your database and its corresponding log file.
- The database and log file can be found in the following directory:
- C:\Program Files\Microsoft SQL Server\MSSQL15.MSSQLSERVER\MSSQL\DATA

Programming 3 Assignment 4

- **Note:** it may be necessary to stop the SQL Server Service to perform this backup.

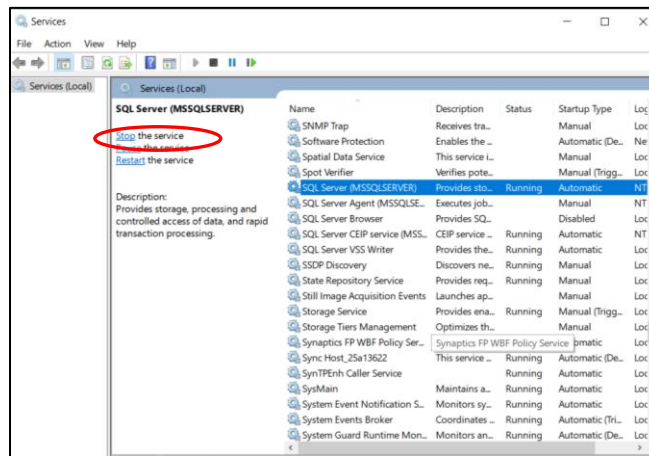


Figure 7

Evaluation: In Person/Hybrid Delivery

- This assignment must be evaluated on or before the due date/time specified in Learn.
 - The assignment is due at the beginning of class on the due date specified. Your instructor will evaluate the assignment on the students' machine in a random order. Remote evaluations will be based on a first come, first served basis.