Assignment
Create a database and access it.

Data access with SQL Client

Write SQL scripts and build a console application in C#. Follow the guidelines given below, feel free to expand on the functionality. It must meet the minimum requirements prescribed.

NOTE: This assignment is to be completed in groups of **2**. In addition to this, you should complete this assignment using Pair Programming.

1) Set up the development environment.

Make sure you have installed at least the following tools:

- Visual Studio 2022 with .NET 6
- SQL Server Management Studio

2) Optional: Database diagrams

You can use SSMS to draw relationship diagrams of your database, you can include these in your repository.

3) Use plain C# to create a console application, and SQL scripts to create a database with the following minimum requirements (See Appendix A-B for details):

a) SQL scripts to create a database according to the specifications in Appendix A.
b) A *repository* implementation in a C# console application to access data from the provided *Chinook* database (see Appendix B).
c) *Summary (///) tags* for each method you write, explaining what the method does, any exceptions it can throw, and what data it returns (if applicable). You do not need to write summary tags for overloaded methods.

4) Submit

a) Create a GitLab repository containing all your code.
b) You can include the generated class diagram in this repository if you have made one.
c) The repository must be either public, or I am added as a Maintainer (@NicholasLennox).
d) Submit only the link to your GitLab repository (not the "clone with SSH").
e) Only one person from each group needs to submit but add the names of both group members in the submission.

Appendix A: SQL scripts to create database.

1) Introduction and overview

You are to create several scripts which can be run to create a database, setup some tables in the database, add relationships to the tables, and then populate the tables with data.

The database and its theme are surrounding superheroes. The database can be called **SuperheroesDb**.

REQUIREMENT: Create a script called *01_dbCreate.sql* that contains a statement to create the database.

NOTE: This script and all following scripts can be written in SSMS through the query window. You can test the scripts and save them from there.

NOTE: If you are unfamiliar with SQL, the W3 Schools set of tutorials are a fantastic resource.

2) Tables

There are three main tables you need to create, this is **Superhero**, **Assistant**, and **Power**.

Superhero has: Autoincremented integer Id, Name, Alias, Origin.

Assistant has: Autoincremented integer Id, Name.

Power has: Autoincremented integer Id, Name, Description.

REQUIREMENT: Create a script called *02_tableCreate.sql* that contains statements to create each of these tables and setup their primary keys.

NOTE: At this point in time, no relationships are to be configured aside from primary keys. Datatypes should be up to you, and what you think is best.

3) Table relationships

Here we are going to include some relationships between the tables. The **ALTER** SQL keyword will be used to change the existing tables to add the keys. A description of the relationships can be seen below:

- One Superhero can have multiple assistants, one assistant has one superhero they assist.
- One Superhero can have many powers, and one power can be present on many Superheroes.

Firstly, to setup the Superhero-assistant relationship, we need to add a foreign key. This represents the superhero this assistant is linked to (SuperheroId in Assistant), and ALTER any tables needed to cater for this foreign key, and to setup the constraint.

REQUIREMENT: Create a script called *03_relationshipSuperheroAssistant.sql* that contains statements to ALTER any tables needed to add the foreign key and setup the constraint to configure the described relationship between Superhero and assistant.

Finally, to setup the Superhero-power relationship, we need to add a linking table. The name of this table is up to you but should be based on convention. This table is there purely for linking, meaning it needs to contain only two fields, which are both **foreign keys** and a **composite primary key**.

REQUIREMENT: Create a script called *04_relationshipSuperheroPower.sql* that contains statements to create the linking table. This script should contain any ALTER statements needed to set up the foreign key constraints between the linking tables and the Superhero and Power tables.

4) Inserting data

Now it is time to populate the database using INSERT statements. Firstly, we should add superheroes.

REQUIREMENT: Create a script called *05_insertSuperheroes.sql* that inserts three new superheroes into the database.

Then we need to add assistant data.

REQUIREMENT: Create a script called *06_insertAssistants.sql* that inserts three assistants, decide on which superheroes these can assist.

Finally, we need to add powers. This requires you to add the powers first, then associate them with Superheroes.

REQUIREMENT: Create a script called *07_powers.sql* that inserts four powers. Then the script needs to give the superheroes powers. Try have one superhero with multiple powers and one power that is used by multiple superheroes, to demonstrate the many-to-many.

5) Updating data

Pick one superhero to update their data.

REQUIREMENT: Create a script called *08_updateSuperhero.sql* where you can update a superheroes name. Pick any superhero to do this with.

6) Deleting data

Referential integrity will stop us from deleting any records which have got relationships used. Meaning, a superhero cannot be deleted if it is used in the assistant table.

REQUIREMENT: Create a script called *09_deleteAssistant.sql* where you can delete any assistant. You can delete that assistant by name (his name must be unique to avoid deleting multiple assistants), this is done to ease working with autoincremented numbers – my PC may skip a number or two.

NOTE: The scripts should be able to be run in order with no errors to create a database with the requirements stated. That is how I will initially check for functionality, then I will investigate each statement for specifics.

Appendix B: Reading data with SQL Client

1) Introduction and overview

Some hotshot media mogul has heard of your newly acquired skills in C#. They have contracted you and a friend to stride on the edge of copyright glory and start re-making iTunes, but under a different name. They have spoken to lawyers and are certain a working prototype should not cause any problems and ensured that you will be safe. The lawyer they use is the same that Epic has been using, so they are familiar with Apple.

The second part of the assignment deals with manipulating SQL Server data in Visual Studio using a library called SQL Client. For this part of the assignment, you are given a database to work with. It is called *Chinook*.

Chinook models the iTunes database of customers purchasing songs. You are to create a C# console application, install the SQL Client library, and create a repository to interact with the database.

NOTE: These requirements are separate from Appendix A.

2) Customer requirements

For customers in the database, the following functionality should be catered for:

1. Read all the customers in the database, this should display their: Id, first name, last name, country, postal code, phone number and email.
2. Read a specific customer from the database (by Id), should display everything listed in the above point.
3. Read a specific customer by name. HINT: LIKE keyword can help for partial matches.
4. Return a page of customers from the database. This should take in *limit* and *offset* as parameters and make use of the SQL limit and offset keywords to get a subset of the customer data. The customer model from above should be reused.
5. Add a new customer to the database. You also need to add only the fields listed above (our customer object)
6. Update an existing customer.
7. Return the number of customers in each country, ordered descending (high to low). i.e. USA: 13, …
8. Customers who are the highest spenders (total in invoice table is the largest), ordered descending.
9. For a given customer, their most popular genre (in the case of a tie, display both). Most popular in this context means the genre that corresponds to the most tracks from invoices associated to that customer.

NOTE: You should create a class (model) for each data structure you intend to use. Do not return a formatted string. This is a minimum of: Customer, CustomerCountry, CustomerSpender, CustomerGenre. Place these classes in a Models folder.

3) Repositories

You are expected to implement the repository pattern in this assignment. The version of the pattern to implement is up to you.

NOTE: Consult the provided material for examples of the Repository pattern.