Problem Set 1

ACA - Advanced C# 06/03/2024

Deadline

The deadline is **Wednesday, 13 March 2024 22:00:00**

Galactic Archive

Background

In the vast universe, the Galactic Archive Initiative (GAI) aims to gather and organize knowledge from across galaxies. It's essential to keep this information well-organized and easy to find. Exploration ships collect data from many places, and now there's a need for a smart system to sort and keep track of all this data. This system will put the data into a central database, using a unique path for each file and folder to make sure everything stays in order and is easy to access.

Known as the best developer in the universe, you have been chosen for this important task. You'll build a system that works with the data collection tools already on the ships. Your system will organize folders and files in a database, marking each with its unique path. It will also update the database whenever files and folders are added or changed. This way, the archive of galactic knowledge will always be up to date and easy for everyone to use.

Setup

To kickstart your involvement in the Galactic Archive project, you've been granted access to the core data collection framework. This existing codebase is your foundation upon which the new indexing system will be built.
You can find all necessary files and submission instructions in the following repository: Problem Set 2 Repo

Note: You should not modify any of the existing code.

Database Schema and Considerations

The output of the indexing service is an SQL Server Database. The following is the minimum requirements that the table should follow, but you are free to add a Primary Key, Constraints, Indexes, Columns, etc...

Directories Table Schema

Column	Type and Constraints
Path	VARCHAR(256) NOT NULL
Name	VARCHAR(256) NOT NULL
DirectoryPath	VARCHAR(256) NOT NULL
LastModified	DATETIME NOT NULL

Files Table Schema

Column	Type and Constraints
Path	VARCHAR(256) NOT NULL
Name	VARCHAR(256) NOT NULL
DirectoryPath	VARCHAR(256) NOT NULL
Size	BIGINT
LastModified	DATETIME NOT NULL

Considerations

1. **Why is there no foreign key reference for parent directories?**

   - In software development, decisions often involve weighing trade-offs. In this case, we're choosing between the data integrity and query performance benefits of a foreign key relationship and the ability to optimize data processing efficiency. Without enforcing a foreign key constraint, we gain flexibility in the sequence of data processing—allowing us to handle child elements before their parents. This flexibility opens up opportunities for parallel processing, significantly enhancing the performance of our data handling code.

2. **Path length is 256?**

   - Yes, the Windows file path max length is 256 characters, and for our purposes let's assume that the Cosmos has similar laws.

Problem 1 - Star System Setup

Your mission is to lay the foundational stone of the Galactic Archive by setting up the initial database schema. You'll modify the **initDbSchema.sql** file to include creation scripts for the defined database schema. Your script will bring the universe of our data into existence, ensuring it's structured to hold the vast knowledge collected across galaxies.

0.1 Steps

1. **Create a database:**
   - Create a database with a name of your choosing
2. **Schema Creation:**
   - Apply the SQL statements on your database so that it contains the intended schema.
3. **Save the SQL statements:**
   - When finished, put all of the SQL statements inside the **initDbSchema.sql** file located in the **GalacticArchive.IndexingEngine** project. Make sure that the SQL statements are as clear and readable as possible. This file will be used to assess your solution.

0.2 Requirements

1. **Schema Definition:**
   - Write clear and concise SQL statements to create the minimal required tables as per the schema provided earlier.
2. **Primary Key:**
   - Add a Primary Key on the tables, you will definitely need it for the next problem.
3. **(Optional) Indexing for Performance:**
   - You are encouraged to strategically add indexes and constraints to the tables. Think about which columns are most likely to be queried against and create indexes on those columns to enhance query performance.
   - Consider the balance between read and write operations while deciding on the indexes. Too many indexes can slow down insert and update operations, so choose wisely.

Problem 2 - Galactic Indexing Engine

Your objective is to develop the "Galactic Indexing Engine", a system designed to dynamically map the structure of files and directories into our database. This engine will update the database to reflect changes in the filesystem, including new additions and modifications. Your focus is on creating efficient algorithms that ensure the database accurately mirrors the current state of the filesystem.

Before diving into the development of the "Galactic Indexing Engine," it's crucial to configure your project environment properly. This setup involves specifying key variables in the ‘appsettings.Development.json‘ file, which the application will use to determine its operational context. Here are the steps to configure these essential settings:

0.3 Setup:

Open the ‘appsettings.Development.json‘ file located in the root directory of the **GalacticArchive** project. Ensure the file includes the following key variables:

1. **"RootPath":** This variable defines the root directory from which the data collection and indexing process will commence. Set this variable to the path of the directory you wish to use as the starting point for indexing. For example:

   ```
   "RootPath": "C:\\Users\\YourUsername\\Documents\\DataCollectionRoot"
   ```

2. **"ConnectionStrings:DefaultConnection":** This variable specifies the SQL Server connection string used by the application to connect to your database. Replace the placeholder values with your actual database server details. For example, for most of you running a local instance of SQL Server, it will look like this:

   ```
   "ConnectionStrings": {
       "DefaultConnection": "Server=.;Database=YOUR_DATABASE_NAME;
   Trusted_Connection=True;"
   }
   ```

0.4 Requirements

1. **Database Population:** Implement the "GalacticSqlServerIndexer" class to handle ‘FileSystemInfo‘ objects, distinguishing between ‘DirectoryInfo‘ and ‘FileInfo‘. Accurately insert or update their details in the database, including name, path, size (for files), and last modified date.

2. **Idempotent Operations:** Develop a mechanism within your indexer to effectively handle changes (additions, modifications) since the last database update. Ensure the operations are idempotent, meaning the operations can be performed multiple times without duplicating or losing data.

   - For **additions**, insert new files or directories.
   - For **modifications**, update existing records with new information.
   - For **unchanged** items, don’t update the row.
   - For **deletions**, the Galactic Archive has decided that once a file or directory is indexed, it should always be remembered. So don’t do anything here.

3. **(Optional) Parallelism and Bulk Operations:** Since performance is critical when it comes to cosmos-level scale, try to add parallelism and bulk operations to squeeze as much performance as possible.