Movie Database Project

RASHID OMENI WILL SUBMIT THIS DOCUMENT AGAIN WITH THE VIDEO LINK ATTACHED AS THE FINAL VERSION.

Overview

This project integrates SQL and NoSQL databases to manage a movie database system. It leverages MySQL for relational data storage and MongoDB for document-oriented storage, providing a robust and scalable environment for handling complex data interactions.

Prerequisites

- Python 3.10.12
- Make

Installation

Step 1: Clone the Repository

```
git clone https://github.com/jpinn97/movie_database.git
cd movie_database
```

Step 4: Full Installation and Setup

```
make full-setup
```

This will install the necessary packages and dependencies for the application. The setup includes the following steps:

- Install PHP, XAMPP, Composer, and MongoDB.
- Copy PHP files to the XAMPP server directory.

Step 5: Start the Application

```
make start
```

This will start the Apache server, MySQL and MongoDB services, it will then populate the databases with the movie data.

Usage

After starting the servers, access the application by navigating to http://localhost in your web browser. Utilize the PHP interface to interact with both MySQL and MongoDB databases by executing predefined queries.

The default credentials are required for MySQL: username is `root` and password is empty.

MySQL and MongoDB with PHP Frontend.

MySQL

The provided dataset consisted of 5 csv files containing data for: `Artist`, `Country`, `Movie`, `Role`, and `Internet_user`. The given schema for the database:

```
Movie (movieId, title, year, genre, summary, #producerId, #countryCode)
Country (code, name, language)
Artist (artistId, surname, name, DOB)
Role (#movieId, #actorId, roleName)
Internet_user (email, surname, name, region)
Score_movie (#email, #movieId, score)
```

A Python script performs the following tasks:

- Connects to MySQL database.
- Runs a creation_script.sql server that reads a creation_script.sql file to create the database and tables. It splits each query by the delimiter `;` and executes them. Example:

```sql
CREATE TABLE Country (
    code CHAR(2) PRIMARY KEY,
    name VARCHAR(255),
    language VARCHAR(100)
);
```

- Performs path manipulation to insert the csv files absolute location dynamically in the read SQL query.
- Inserts the data from the csv files into the tables using the `LOAD DATA INFILE` command.
- Executes a sanitization script to standardize the data.

Frontend

The frontend consists of a simple HTML that uses HTMX to make AJAX requests to the PHP backend. It passes credentials if necessary to the predefined query that is selected by the user, which connects and retrieves the data, before displaying it in a table.

The HTMX allows for a more dynamic frontend by swapping elements, without the need for a full page reload.

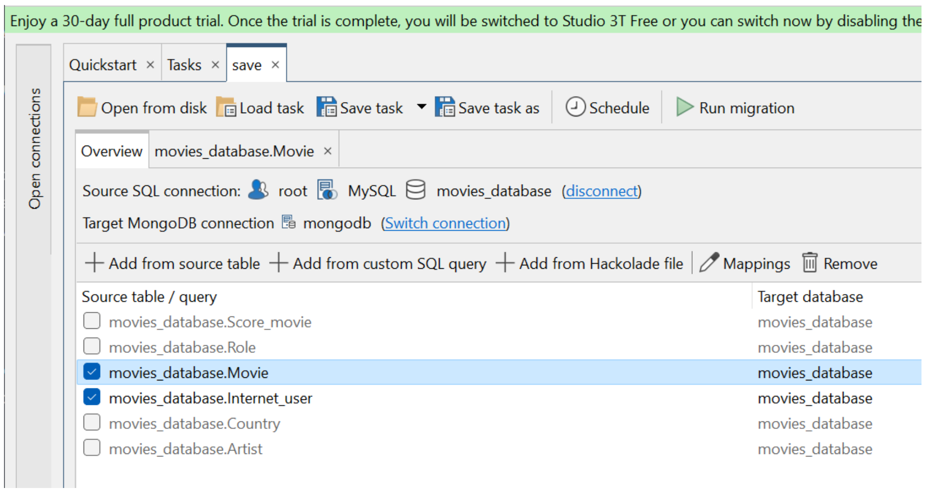
```html
<button
  hx-post="/php/query_1.php"
  hx-trigger="click"
  hx-target="#result"
  hx-vals='{"username": document.getElementById("username").value, "password": document.getE
>
  Query 1
</button>
```

The query_1 button is pressed, which sends the username and password from the interface to the PHP query via the `hx-vals` attribute, it then swaps the content of the `#result` div with the result of the query via the `hx-target` attribute.

MongoDB

Studio3t is a GUI for MongoDB that can be used to migrate data from MySQL to MongoDB. It can be downloaded from here.

A connection is made to our current MySQL implementation, as well as our target MongoDB instance. The authentication information, hostname and ports are specified. We can then use the import wizard to migrate data from MySQL to MongoDB.


Enjoy a 30-day full product trial. Once the trial is complete, you will be switched to Studio 3T Free or you can switch now by disabling the

Source SQL connection: root MySQL movies_database (disconnect)

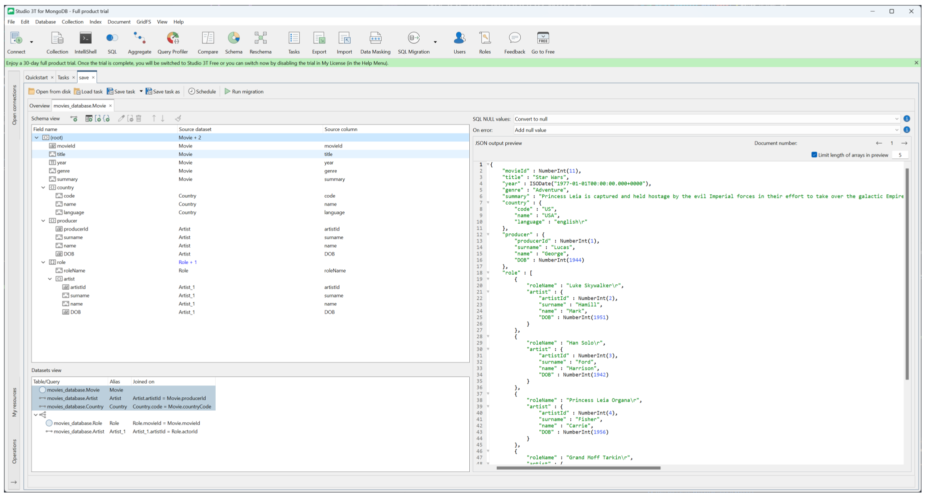
Target MongoDB connection mongodb (Switch connection)

	Source table / query	Target database
☐	movies_database.Score_movie	movies_database
☐	movies_database.Role	movies_database
✓	movies_database.Movie	movies_database
✓	movies_database.Internet_user	movies_database
☐	movies_database.Country	movies_database
☐	movies_database.Artist	movies_database

Figure 1: Studio3t

We select the Movie and Internet_user tables from MySQL and click `Mappings`, where we can transform the tables into documents.

Essentially, a minimal no-code approach is taken to transform the tables into documents. Moving from a relational database to a document-orientated database requires denormalization. Information that was previously stored across multiple tables is now stored in as few objects as possible, this is because to get all the information we need, we would have to perform multiple joins in a relational database.



Schema view

Field name	Source dataset	Source column
(root)	Movie + 2	
movieId	Movie	movieId
title	Movie	title
year	Movie	year
genre	Movie	genre
summary	Movie	summary
country		
code	Country	code
name	Country	name
language	Country	language
producer		
producerId	Artist	artistId
surname	Artist	surname
name	Artist	name
DOB	Artist	DOB
role	Role + 1	
roleName	Role	roleName
artist		
artistId	Artist_1	artistId
surname	Artist_1	surname
name	Artist_1	name
DOB	Artist_1	DOB

Datasets view

Table/Query	Alias	Joined on
movies_database.Movie	Movie	
movies_database.Artist	Artist	Artist.artistId = Movie.producerId
movies_database.Country	Country	Country.code = Movie.countryCode
movies_database.Role	Role	Role.movieId = Movie.movieId
movies_database.Artist	Artist_1	Artist_1.artistId = Role.actorId

SQL NULL values: Convert to null

On error: Add null value

JSON output preview

Document number: ← 1 →

☑ Limit length of arrays in preview 5

```
{
    "movieId" : NumberInt(11),
    "title" : "Star Wars",
    "year" : ISODate("1977-01-01T00:00:00.000+0000"),
    "genre" : "Adventure",
    "summary" : "Princess Leia is captured and held hostage by the evil Imperial forces in their effort to take over the galactic Empire
    "country" : {
        "code" : "US",
        "name" : "USA",
        "language" : "english\r"
    },
    "producer" : {
        "producerId" : NumberInt(1),
        "surname" : "Lucas",
        "name" : "George",
        "DOB" : NumberInt(1944)
    },
    "role" : [
        {
            "roleName" : "Luke Skywalker\r",
            "artist" : {
                "artistId" : NumberInt(2),
                "surname" : "Hamill",
                "name" : "Mark",
                "DOB" : NumberInt(1951)
            }
        },
        {
            "roleName" : "Han Solo\r",
            "artist" : {
                "artistId" : NumberInt(3),
                "surname" : "Ford",
                "name" : "Harrison",
                "DOB" : NumberInt(1942)
            }
        },
        {
            "roleName" : "Princess Leia Organa\r",
            "artist" : {
                "artistId" : NumberInt(4),
                "surname" : "Fisher",
                "name" : "Carrie",
                "DOB" : NumberInt(1956)
            }
        },
        {
            "roleName" : "Grand Moff Tarkin\r",
            "artist" : {
```

Figure 2: Studio3t

The complexity of a query is increased by the number of joins that are required, which can slow down the query on large datasets, as well as make the query harder to read and maintain.

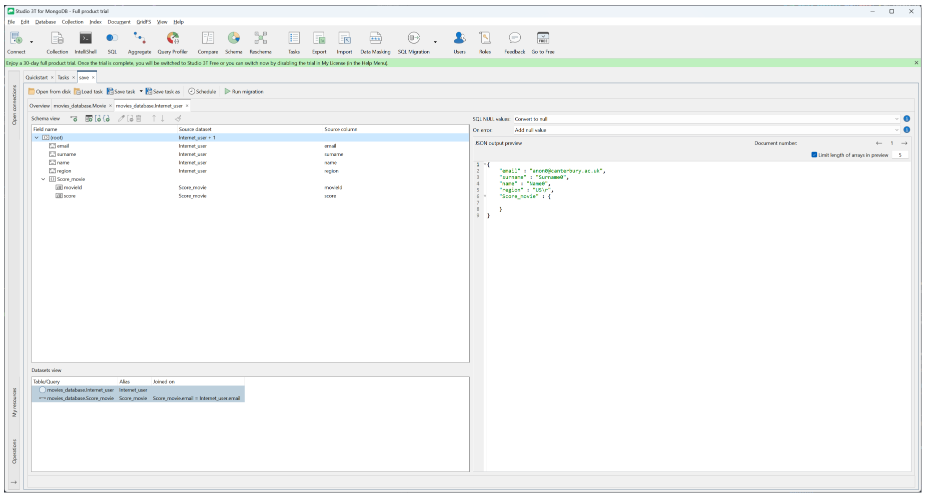
Given our dataset, the data is tightly coupled, we can use an object-orientated approach to store related information together to avoid expensive joins.

There is no need to define a complex schema with a document, the structure is flexible, and can be changed more easily than in relational databases. This is useful when we are working with data that is constantly changing, as we can add new fields to a document without affecting the existing documents.

For instance, after visually inspecting the dataset, we look at the database schema, to understand the relationships between the tables. The `Movie` table becomes the main document of the collection, with the `country` and `role` becoming sub-documents. We create a new sub-document, the `producer`, which maps the artists who aren't referenced by the `role` table. The `artist` table is embedded in the `role` sub-document. This makes the data more readable, easier to query and natural.

A movie has a country, a producer and a list of roles (characters) played by artists. By doing so, we've encapsulated the data that into a single document, which needn't change often, this decreases the number of write operations, as well as the number of queries required to get the data we need.

The `Internet_user` and `Score_movie` table are moved into their own collection. Given the nature of the data, the `movie` documents are unlikely to change often. Taking into account that amount of internet user ratings are variable, and the document size limit of MongoDB, there are methods to handle this, but we


Studio 3T for MongoDB - Full product trial
File Edit Database Collection Index Document GridFS View Help
Connect Collection IntelliShell SQL Aggregate Query Profiler Compare Schema Reschema Tasks Export Import Data Masking SQL Migration Users Roles Feedback Go to Free
Quickstart Tasks save
Open from disk Load task Save task Save task as Schedule Run migration
Overview movies_database.Movie movies_database.Internet_user
Schema view
Open connections
My resources
Operations

Enjoy a 30-day full product trial. Once the trial is complete, you will be switched to Studio 3T Free or you can switch now by disabling the trial in My License (in the Help Menu).

Field name	Source dataset	Source column
(root)	Internet_user + 1	
email	Internet_user	email
surname	Internet_user	surname
name	Internet_user	name
region	Internet_user	region
Score_movie		
movieId	Score_movie	movieId
score	Score_movie	score

SQL NULL values: Convert to null
On error: Add null value
JSON output preview
Document number: 1
Limit length of arrays in preview: 5

```
{
    "email" : "anon0@canterbury.ac.uk",
    "surname" : "Surname0",
    "name" : "Name0",
    "region" : "US\r",
    "Score_movie" : {

    }
}
```

Datasets view

Table/Query	Alias	Joined on
movies_database.Internet_user	Internet_user	
movies_database.Score_movie	Score_movie	Score_movie.email = Internet_user.email

Figure 3: Studio3t

needn't write endless queries to a `movie` document to update the ratings. We can simply update the `Internet_user` and `Score_movie` documents, and query the `movieId` to get the ratings. This allows us to maintain maximum read operations to the `movie` collection.

The `Score_movie` table is embedded into the `Internet_user` document, and denormalized to remove redundant data. We now have a collection of `movie` documents and a collection of `Internet_user` documents.

Exporting, Preparing, and Importing into MongoDB

The migration can go directly into our MongoDB, but we want to ensure the document is cleansed of anything that has occurred from the MySQL database, such as carriage returns, we can export the data to a JSON file.

The exported JSON has data types from MongoDB that aren't valid JSON. We parse a few commands to replace the invalid data types with MongoDB Extended JSON v2

This must be done, because to import the file we must stream it using JSON, and datatypes like "ObjectID()" aren't valid JSON.

```
sed -i 's/ISODate(\(.*\))/{"$date": \1}/g' json/Movie.json
sed -i 's/NumberInt(\(.*\))/{"$numberInt": "\1"}/g' json/Movie.json
sed -i 's/ObjectId(\(.*\))/{"$oid": \1}/g' json/Movie.json
```

Conclusion

The decision between NoSQL and SQL databases hinges on the specific requirements of each application, such as data structure, scalability, performance, and transactional reliability. By supporting ACID (Atomicity, Consistency, Isolation, Durability), SQL databases excel in environments where structured data integrity and complex querying capabilities are paramount, making them suitable for applications that involve complex transactions and require precise data consistency.

On the other hand, NoSQL databases offer flexibility, scalability, and performance efficiency, especially beneficial for handling large volumes of diverse data types and for applications that need rapid scaling. Ultimately, the choice is not strictly one or the other; many modern applications benefit from a hybrid approach that utilizes the strengths of both NoSQL and SQL technologies to meet their comprehensive data management needs.

Tasks

Jamie(a), Rashid(b):

- index.php [a]
- query_1.php, query_2.php, query_3.php, [b]
- m_query_1.php, m_query_2.php, m_query_3.php [a]
- makefile [a]
- creation_script.sql [b]
- import_script.sql [a]
- sanitize_script.sql [a]
- denormalize mysql db to mongodb [a]
- xampp/mongo py script [a]

Version: V001

Authors

Jamie Pinnington Abdul Rashid Omeni

GitHub

https://github.com/jpinn97/movie_database

Video

PLACEHOLDER

License

Not under any license.

References:

- https://www.apachefriends.org/