Comparing Relational and NoSQL Databases for the Book Recommendation Dataset

Group Information

Group Number: 22
Students:

- Enzo Chatalov - fc54414, Contribution: 25%
- Agnieszka Radomska - fc64357, Contribution: 25%
- Duarte Alexandre Pedro Gonçalves - fc64465, Contribution: 25%
- Tommaso Tragno - fc64699, Contribution: 25%

Dataset Description

Source

The dataset selected is the **Book Recommendation Dataset** from Kaggle, available here. This dataset is designed for analyzing and building recommendation systems, consisting of three CSV files:

1. **Books.csv:** Contains metadata about books such as ISBN, titles, authors, publication year, and more.
2. **Ratings.csv:** Contains user ratings for books, including user IDs, book IDs, and rating scores.
3. **Users.csv:** Contains user demographic information like location and age.

Characteristic

- **Common column:** All three files share the `bookID` or `userID` fields as relational keys.
- **Volume:** Moderate-sized dataset (107.51 MB) with potential for complex analysis in both relational and NoSQL structures.
    - Books: 271361 records;
    - Ratings: 1149781 records
    - Users: 278860 records

Data Validation

A robust data validation and preprocessing has been performed to ensure the integrity of the dataset before use. Here's a detailed breakdown of the validation steps:

1. Load the three CSV files (`Books.csv`, `Ratings.csv`, `Users.csv`) into separate pandas DataFrames.
2. Initial check for missing values: this step checks if any missing values (`Na`) exist in the datasets before validation, for so, it was used `isna().any().any()` to identify if any column contains missing values.
3. Handling of missing values: for each dataframe, we dropped the rows with missing `'User-ID'` or `'ISBN'`, because they will be used as primary key and foreign key for the relational DB and missing values for these columns are considered invalid and are removed. A message prints the count of dropped rows after each operation, however, no row contains any missing value for these columns.
   Then, we filled the remaining missing cells with predefined strings (such as 'not available' or '0'), using the pandas method `fillna()`.
4. Removing duplicates: We eventually dropped duplicate rows from all DataFrames to ensure unique records, which is a vital step for maintaining data consistency and avoiding redundancy in further processing. A message prints the count of dropped rows after each operation, but no duplicated rows were found
5. Data type conversion: Certain columns are converted to numeric types to ensure consistent data types.
6. Final check for missing values: check the correct execution of the pipeline using `isna().any().any()` to identify if any column contains missing values, where it should return `False` for all the datasets.

Key Outcomes

- **Clean and Consistent Data:** All missing and inconsistent values are addressed, ensuring the datasets are ready for analysis or database insertion.
- **Data Integrity:** Invalid rows (missing `User-ID` or `ISBN`) are removed to maintain relational integrity.
- **Standardized Formats:** Columns like `Age`, `Year-Of-Publication`, and `Book-Rating` are standardized to numeric formats, improving usability for calculations or queries.

This robust validation pipeline ensures the dataset is well-prepared for further processing, analysis, or storage in relational and NoSQL databases.

Additional Issue Identified: Error During MySQL Insertion for `Books`

After executing the described data validation pipeline, an error was encountered during the insertion of the `Books` records into the MySQL database. In order to find the records that give this error, we try `except` statements to handle the error and print the affected rows.

```
for _, row in df_books.iterrows():
    try:
        cursor.execute(
            "INSERT IGNORE INTO books (ISBN, Book_Title, Book_Author, Year_Of_Publication, Publisher, Image_URL_S,
            (row['ISBN'], row['Book-Title'], row['Book-Author'], row['Year-Of-Publication'], row['Publisher'], row
        )
    except:
        print(row)
```

Luckily, the error was caused by only 3 records.

```
ISBN                                                 078946697X
Book-Title            DK Readers: Creating the X-Men, How It All Beg...
Book-Author                                                  2000
Year-Of-Publication                                          <NA>
Publisher             http://images.amazon.com/images/P/078946697X.0...
Image-URL-S           http://images.amazon.com/images/P/078946697X.0...
Image-URL-M           http://images.amazon.com/images/P/078946697X.0...
Image-URL-L                                          not available
Name: 209538, dtype: object
ISBN                                                 2070426769
Book-Title            Peuple du ciel, suivi de 'Les Bergers\";Jean-M...
Book-Author                                                  2003
Year-Of-Publication                                          <NA>
Publisher             http://images.amazon.com/images/P/2070426769.0...
Image-URL-S           http://images.amazon.com/images/P/2070426769.0...
Image-URL-M           http://images.amazon.com/images/P/2070426769.0...
Image-URL-L                                          not available
Name: 220731, dtype: object
ISBN                                                 0789466953
Book-Title            DK Readers: Creating the X-Men, How Comic Book...
Book-Author                                                  2000
Year-Of-Publication                                          <NA>
Publisher             http://images.amazon.com/images/P/0789466953.0...
Image-URL-S           http://images.amazon.com/images/P/0789466953.0...
Image-URL-M           http://images.amazon.com/images/P/0789466953.0...
Image-URL-L                                          not available
Name: 221678, dtype: object
Data inserted successfully.
```

The error stemmed from issues with the separation of values in the `Books.csv` file.

```
221679  0441142648,Demons (Magic Tales Anthology Series),Jack Dann,1987,Ace Books,http://images.amazon.com/images/P/0441142648.01.THUMBZZZ.jpg,http://image
221680  0789466953,"DK Readers: Creating the X-Men, How Comic Books Come to Life (Level 4: Proficient Readers)\"";James Buckley""", ,2000,DK Publishing Inc
221681  0345314603,Five Years to Freedom,James Rowe,1991,Ballantine Books,http://images.amazon.com/images/P/0345314603.01.THUMBZZZ.jpg,http://images.amazon
```

```
220732  0758200064,Soulmates Dissipate,Mary B. Morrison,2002,Dafina,http://images.amazon.com/images/P/0758200064.01.THUMBZZZ.jpg,http://images.amazon.com/i
220733  2070426769,"Peuple du ciel, suivi de 'Les Bergers\"";Jean-Marie Gustave Le ClÃ?Ã©zio""", ,2003,Gallimard,http://images.amazon.com/images/P/20704267
220734  0679808450,"Good Night, Sleep Tight! Shhh... (Chunky Shape Book)",Gyo Fujikawa,1990,Random House Children's Books,http://images.amazon.com/images/P
```

```
209539  1572972173,The Ultimate X-Men,Stan Lee,1996,Berkley Publishing Group,http://images.amazon.com/images/P/1572972173.01.THUMBZZZ.jpg,http://images.ama
209540  078946697X,"DK Readers: Creating the X-Men, How It All Began (Level 4: Proficient Readers)\"";Michael Teitelbaum""", ,2000,DK Publishing Inc,http:/
209541  0425170829,X-Men Legends (X-Men),Stan Lee,2000,Berkley Publishing Group,http://images.amazon.com/images/P/0425170829.01.THUMBZZZ.jpg,http://images.
```

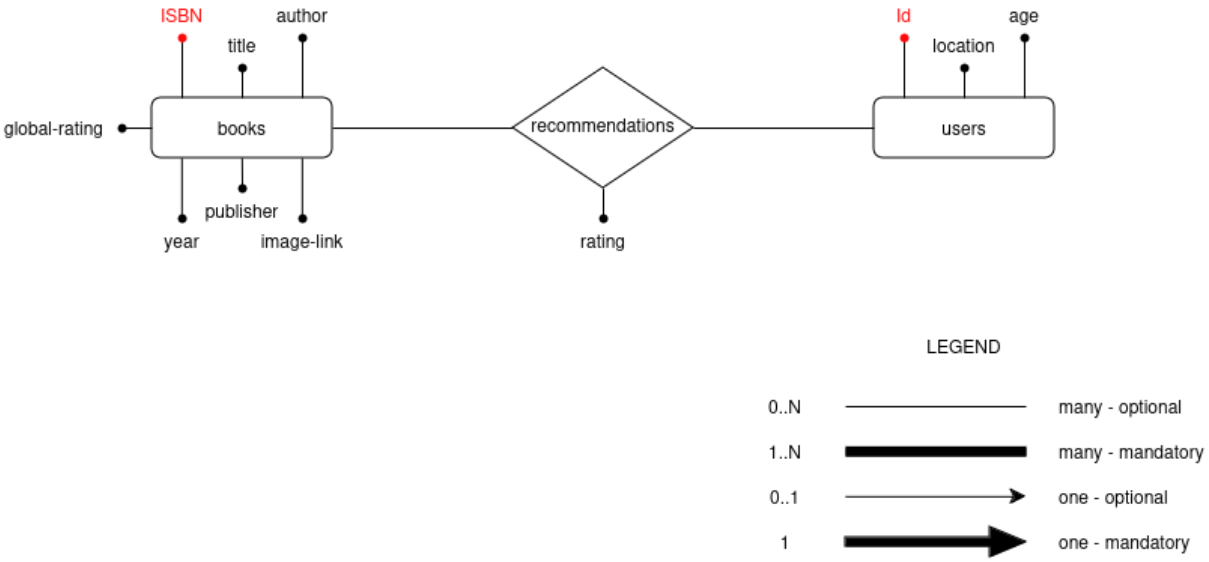
The root cause was identified as follows:

- **Incorrect Delimiters in the CSV File:** Some rows in the `Books.csv` file contained values that included characters like commas or double quotes, which were not properly escaped.
- **Mismatched Column Alignment:** This caused misalignment of columns during the `pandas` DataFrame parsing or while attempting to construct SQL queries for record insertion.

After the manual correction of the mismatch, the data were further validated by the pipeline and inserted effectively into mySQL without any error.

Entity-Relational Diagram (ERD)

The **ER Diagram** is a high-level abstract diagram, helping conceptualize the relationships and the general structure of the database without implementation-specific details.



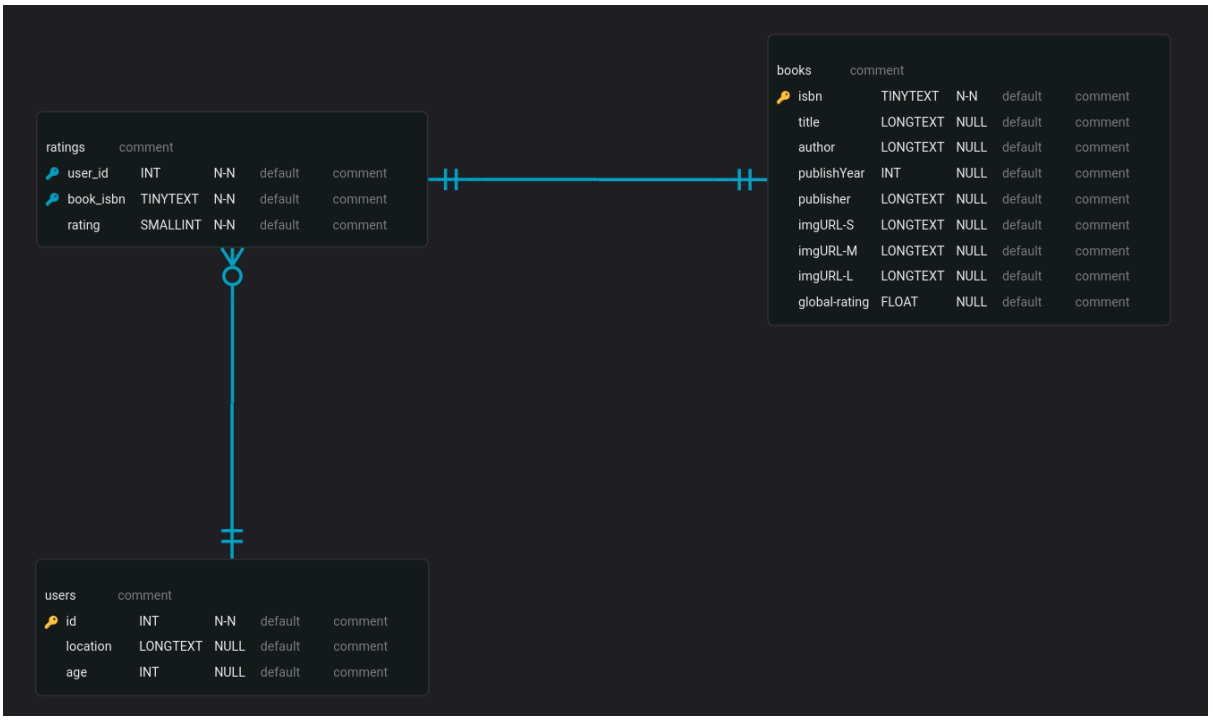
The **Entity-Relational Diagram** showcases the logical relationships between entities in a database system. Here's a breakdown of its components:

1. **Entities**:
    - Represented as rectangles (`books`, `users`).
    - These are core objects in the database.
2. **Attributes**:
    - Shown inside the entities or associated with them.
    - Example: The `books` entity has attributes such as `ISBN`, `author`, `publisher`, `title`, etc.
3. **Relationships**:
    - Represented by lines connecting entities and rhombus (`recommendation`).
    - A book can have many recommendations, each associated with a specific user and a rating. Cardinality: `0..N` (One book may be rated by many users).
    - A user can provide many ratings for different books. Cardinality: `0..N` (One user may rate multiple books).
4. **Key Features**:
    - Primary keys (unique identifiers) are highlighted in red each entity.
    - Relationships between entities include `users` rating `books`, which connects both entities with the `recommendation` table, which contains the attribute `rating`.

Relational Diagram

The **Relational Diagram** (or schema diagram) is more focused on implementation, detailing the database structure and how data will be stored in tables.. It is the physical representation

of the database structure, and shows how the tables are implemented in a database management system.



The many-to-many relations between books and users should be splitted in a new table, which contains the foreign key of both books and users, and the attribute rating of the relation.

```
books(ISBN,title,author,year,publisher,image-link,global-rating)
users(ID,location,age)
recommendations(users.ID,books.ISBN,rating)
```

Here's a summary of the Relational Diagram features:

1. **Tables**:
   - Represented as boxes (`books`, `users`, `ratings`).
   - Each box contains the column names, data types, and constraints (e.g., `INT`, `VARCHAR`, `PRIMARY KEY`).
2. **Relationships**:
   - Lines between tables represent foreign key constraints.
   - The `ratings` table acts as a junction between `users` and `books`, connecting them with foreign keys.
3. **Data Types**:
   - Defined for each column within the table.
   - Example: Columns like `user_id` or `book_id` are of type `INT`, while `title` or `author` may be `TEXT` or `VARCHAR`.
4. **Keys and Constraints**:

- Primary keys (`id` in `users`, `ISBN` in `books`) ensure unique identification of records.
- Foreign keys link related tables and enforce referential integrity.
- The `Ratings` table serves as a bridge table (junction table) linking `Users` and `Books` via foreign keys. This represents a many-to-many relationship between `Users` and `Books`.

Observation: the **Global-Rating** attribute for the `books` entity has been added in order to evaluate the mean rating for each book from the `recommendation` entity.

MongoDB Collection Structure Based on the ERD for MySQL

The MongoDB schema is designed to represent the relationships and attributes from the ERD while leveraging MongoDB's document-oriented, flexible schema capabilities.
The collection represents the structure of the dataset itself, as it is already as defined from the Relational Diagram.
Each `.csv` file has been loaded as a separate collection, maintaining a normalized structure similar to MySQL. Ideal for large datasets with complex relationships.

Project Infrastructure

The project infrastructure is based on a local environment where both MySQL and MongoDB are installed for data management. A `config.json` file is used to centralize connection details and credentials, ensuring easy database access while keeping sensitive data separate from the application logic.
Before executing the code, you need to create the `config.json` file in the same folder of the Jupyter Notebook, and add the following informations:

```
{
      "mongo": {
      "username": "your_mongo_username",
      "password": "your_mongo_password",
      "host": "your_mongo_host",
      "port": "your_mongo_port"
      },
      "mysql": {
      "username": "your_mysql_username",
      "password": "your_mysql_password",
      "host": "your_mysql_host",
      "port": "your_mysql_port"
      }
}
```

Queries

We designed the following queries, to be executed on both MongoDB and mySQL for a time comparison evaluation.

- Simple Query 1: find all the books published in the year 2000;
- Simple Query 2: find all the users that are older than 30 years old;
- Complex Query 1: Update all the rating from a specific user to a specific value;
- Complex Query 2: Add a new column in the Books table with the mean ratings of every book

Time Comparison

For each query execution, and also for the database population, we measured the elapsed time, in seconds.

Query	MongoDB	MySQL
Insertion	~283.88	~417.47
Simple 1	0.00069	~1.02
Simple 2	0.000118	~0.24
Complex 1	~1.30	~0.03
Complex 2	~17777.60	~48.53

Conclusions

As shown from the previous table, we get better elapsed times on complex queries from MySQL, due to ACID compliance and query optimization (ensures reliable transactions in complex queries and efficiently handles multi-table joins), while the MongoDB got better elapsed on the insertion and simple ones.

- **Simple Queries:** MongoDB drastically outperformed MySQL in this queries. MongoDB's document model and indexing on fields allow faster lookups for simple retrieval queries, while MySQL's slower performance can be attributed to schema constraints and the cost of scanning and joining tables.
- **Complex Queries:** MySQL significantly outperformed MongoDB for updates, because it is optimized for structured data and transactional consistency, which enables efficient updates. MongoDB's slower performance results from its lack of ACID compliance.

For real world data management with complex queries handling, the MySQL should be the choice for database, otherwise the MongoDB should suffice for handling simple queries.