THE UNIVERSITY OF BUEA

P.o box 63

Buea, South West Region

Cameroon

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER ENGINEERING

REPUBLIC OF CAMEROON

PEACE - WORK - FATHERLAND

CEF440:INTERNET PROGRAMMING (J2EE) AND MOBILE PROGRAMMING

Design and Implementation of a **Centralized Database Design for Archiving Images and Metadata**(Mobile-Based Archival and Retrieval of Missing Objects Application using Image

Group 28: members: TASK 6

Names	Matricule
TCHUIDJAN JORDAN BRYANT	FE21A320
FUKA NEVILLE TENYI	FE21A199
NFOR WILLY LINWE	FE21A254
FRU BELTMOND CHINJE	FE21A198
EYONG OSCAR ENOWNYUO	FE21A187

COURSE INSTRUCTOR:
Dr. NKEMENI VALERY

ACADEMIC YEAR 2023-2024

Table of contents

Catalog

A . Introduction 3
B. Requirements 4
C. Database Technology 5
Dynamic Schema 5
GridFS 5
High Performance 5
Text Indexes 5
D. Database Schema Design 6
1. Entity-Relationship Diagram (ERD) 6
2. Data Flow Diagram (DFD) 6
3. Schema Diagram 6
1. Entity-Relationship Diagram (ERD) 8
1. Image 8
2. User 8
3. Search 8
Explanation of the Entity-Relationship Diagram (ERD) 8
2. Data Flow Diagram (DFD) 10
1. Entities 10
2. Processes 10
3. Data Stores 10
Explanation of DFD diagram 11
3.Schema Diagram 11
1. User Table 11
2. Search Table 11
3. Image Table 11
E. SOME TABLES CREATED ON MongoDB 12
Column: 12
Diagram 12
Columns: 13
Sql code to create the metadata table 13
Columns : 13
Columns : 15
Diagram 15
Columns : 15
Sql code 16

A. Introduction

This report outlines the design and implementation of a centralized database system for storing images and their associated metadata. The database is intended to support efficient indexing and retrieval, enabling users to access relevant data quickly for various search processes. The stored metadata includes descriptions, timestamps, and locations of the archived objects.

B. Requirements

2.1 Functional Requirements

1. **Storage of Images**: The system must store images either as binary data or as paths to image files.

2. **Metadata Storage**: The system must store metadata for each image, including:
   - Description: A textual description of the image.
   - Timestamp: The date and time when the image was archived.
   - Location: The geographical location associated with the image.
   - Keywords: Additional keywords for tagging and searching.

3. **Efficient Retrieval**: The system should support efficient retrieval of images and metadata based on various search criteria.

2.2 Non-Functional Requirements

1. **Performance**: The system should handle a high volume of data and maintain quick search and retrieval times.

2. **Scalability**: The system should be scalable to accommodate growing data volumes.

3. **Security**: The system should ensure data integrity and protect against unauthorized access.

C. Database Technology

Based on the requirements, a relational database was chosen for its structured nature and support for complex queries. PostgreSQL is selected due to its robustness and support for advanced indexing and full-text search.Here we will use MongoDB,MongoDB, a popular NoSQL database, is often considered a strong choice for projects involving storage and retrieval of images and their associated metadata.

Here are several reasons why MongoDB might be the best choice for this project:

1. **Schema Flexibility**

Dynamic Schema

- **Flexible Documents**: MongoDB uses a flexible, schema-less design, allowing you to store complex data structures in a single document. This is particularly useful for storing image metadata, which can vary in structure.
- **Adaptable**: You can easily add or remove fields from documents without needing to modify the schema, making it easier to handle evolving data requirements.

2. Storage of Binary Data

GridFS

- **Efficient Storage**: MongoDB's GridFS is designed for storing large binary data, such as images. It splits large files into smaller chunks and stores them in separate documents, ensuring efficient storage and retrieval.
- **Scalability:** GridFS can handle files larger than the BSON-document size limit (16 MB), making it ideal for high-resolution images.

3. Performance and Scalability

High Performance

- **Indexing**: MongoDB supports various index types, including compound indexes, geospatial indexes, and text indexes, which can significantly improve query performance for metadata searches.
- **Horizontal Scaling**: MongoDB is designed to scale horizontally using sharding, allowing you to distribute data across multiple servers and ensure high availability and performance.

4. Geospatial Queries

Location-Based Metadata

- **Geospatial Indexes:** MongoDB supports geospatial indexing, which is ideal for querying images based on location metadata. This makes it easier to perform location-based searches and analyses.
- **Rich Query Capabilities**: You can perform complex geospatial queries, such as finding images within a certain radius or bounding box.

5. Full-Text Search

Text Indexes

- **Built-In Full-Text Search:** MongoDB supports full-text search indexes, enabling efficient searching of textual metadata fields like descriptions and keywords.
- **Advanced Search Features:** You can perform rich text search queries, including stemming, tokenization, and relevance scoring.

Conclusion

MongoDB provides a flexible, high-performance, and scalable solution for storing and retrieving images and their metadata. Its schema-less design, efficient handling of binary data with GridFS, built-in support for geospatial queries, and make it an excellent choice for this project. By leveraging MongoDB, you can ensure efficient storage, quick retrieval, and easy management of your archived images and their associated metadata.

D. Database Schema Design

Database design involves a variety of diagrams to visually represent different aspects of the database structure and its interactions. Here are the main types of diagrams commonly used in this database design:

1. Entity-Relationship Diagram (ERD)

An ERD is used to visually represent the data objects (entities), the relationships between them, and the attributes of the entities.

- **Entities**: Represented by rectangles.
- **Relationships**: Represented by diamonds or lines connecting entities.
- **Attributes**: Represented by ovals or listed within entities.
- **Primary Keys**: Underlined attributes.
- **Foreign Keys**: Attributes that link entities.

Example:

```
Customer---------
CustomerID (PK)
Name
Email

Order
---------
OrderID (PK)
OrderDate
CustomerID (FK)

Product
---------
ProductID (PK)
Name
Price

Relationships:
-------------
Customer "places" Order
Order "includes" Product
```

2. Data Flow Diagram (DFD)

A DFD illustrates the flow of data within a system. It shows how data enters a system, how it is processed, and how it is stored.

- **Processes**: Represented by circles or rounded rectangles.
- **Data Stores**: Represented by open-ended rectangles.
- **Data Flows**: Represented by arrows.
- **External Entities**: Represented by squares.

Example:

```
[Customer] --> (Place Order) --> [Order System](Order System) --> [Order Database]
```

3. Schema Diagram

A schema diagram provides a detailed view of the database schema, including tables, columns, data types, and the relationships between tables.

- **Tables**: Represented by rectangles or boxes.
- **Columns**: Listed within the tables.
- **Primary Keys**: Often highlighted or underlined.
- **Foreign Keys**: Indicated by connections between tables.

Example:

```
Table: Customer----------------
CustomerID (PK)
Name
Email

Table: Order
-------------
OrderID (PK)
OrderDate
CustomerID (FK)

Table: Product
---------------
ProductID (PK)
Name
Price
```

Now Let's create the Entity-Relationship Diagram (ERD), Data Flow Diagram (DFD), and Schema Diagram for a centralized database system for storing images and their associated metadata.

1. Entity-Relationship Diagram (ERD)

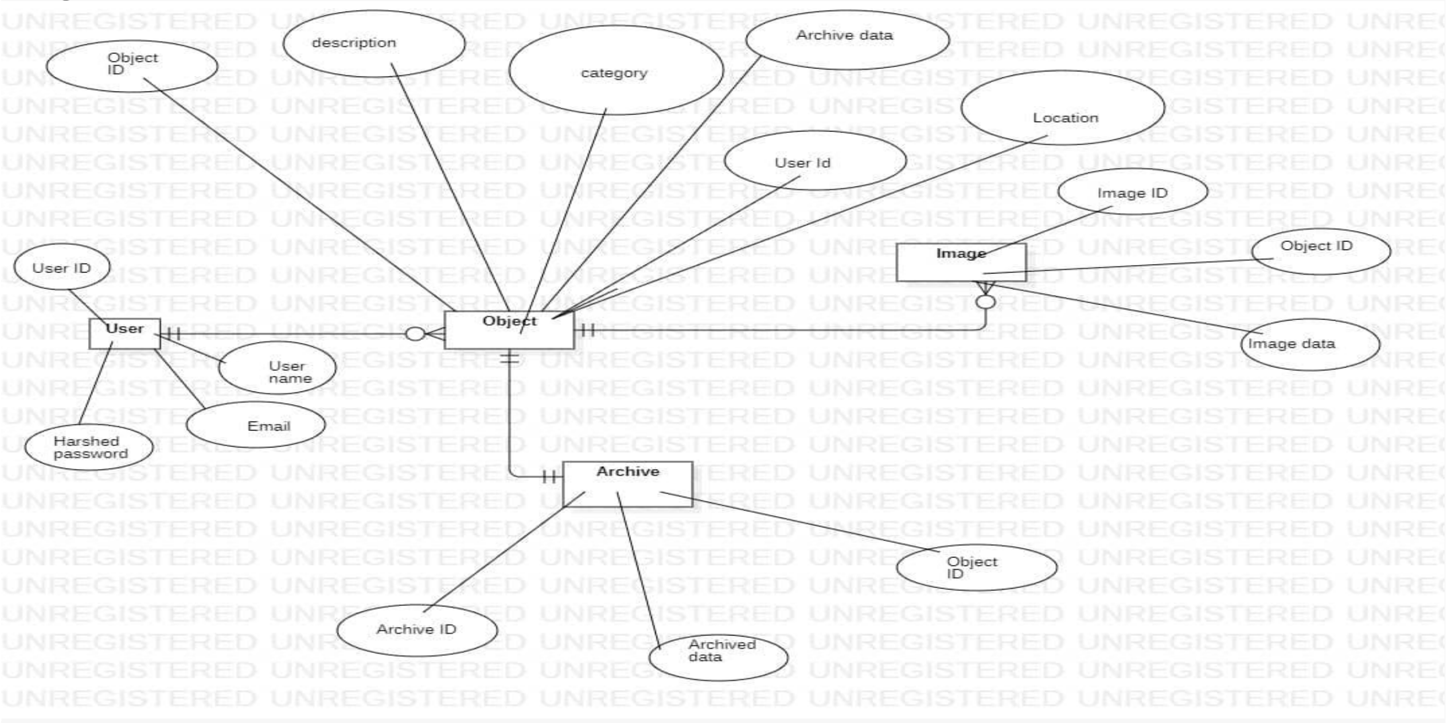
Entities and Attributes

1. **Image**
    - ImageID (PK)
    - FilePath
    - Description
    - Timestamp
    - Location
2. **User**
    - UserID (PK)
    - Username
    - Email
3. **Search**
    - SearchID (PK)
    - UserID (FK)
    - SearchQuery
    - SearchTimestamp

Relationships

- A `User` can perform many `Searches`.
- Each `Search` may retrieve multiple `Images`.

Diagram



Explanation of the Entity-Relationship Diagram (ERD)

Entities and Their Attributes

User

1.
- **UserID (PK)**: A unique identifier for each user.
- **Username**: The name chosen by the user.
- **Email**: The email address of the user.

2.
Search

3.
- **SearchID (PK)**: A unique identifier for each search query.
- **UserID (FK)**: A foreign key that links each search to a specific user.
- **SearchQuery**: The search terms or parameters entered by the user.
- **SearchTimestamp**: The date and time when the search was performed.

4.
Image

5.
- **ImageID (PK)**: A unique identifier for each image.
- **FilePath**: The location of the image file in storage.
- **Description**: Textual information describing the image.
- **Timestamp**: The date and time when the image was stored or created.
- **Location**: The geographical location associated with the image.

Relationships

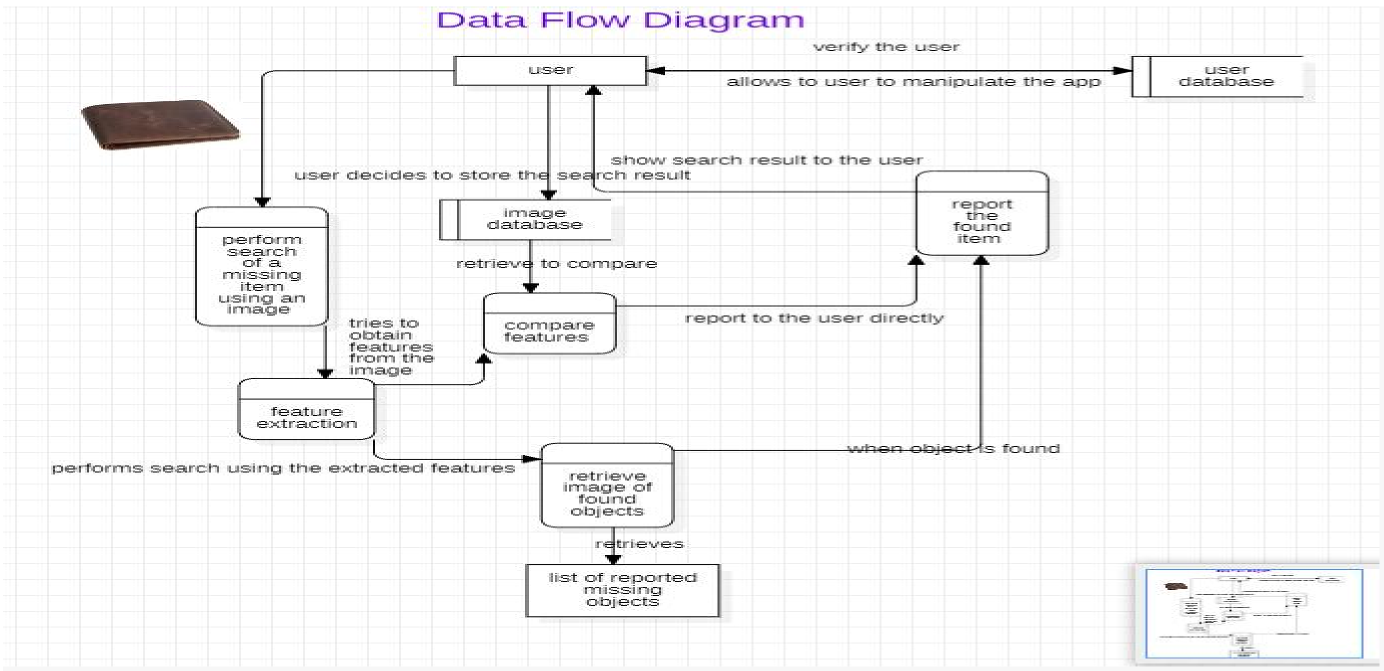
- A **User** can perform multiple **Searches**, which is a one-to-many relationship.
- Each **Search** can retrieve multiple **Images**, which suggests a many-to-many relationship between searches and images. However, for simplicity, we treat it as multiple one-to-many relationship

2. Data Flow Diagram (DFD)

Processes and Data Flows

1. **Entities**
    - User
2. **Processes**
    - Perform Search
    - Retrieve Images
3. **Data Stores**
    1.
    - User Database
    - Image Database

Diagram


Data Flow Diagram
user
verify the user
allows to user to manipulate the app
user database
show search result to the user
user decides to store the search result
image database
report the found item
perform search of a missing item using an image
retrieve to compare
compare features
report to the user directly
tries to obtain features from the image
feature extraction
when object is found
performs search using the extracted features
retrieve image of found objects
retrieves
list of reported missing objects

Explanation of DFD diagram

In this DFD:

- The `User` initiates a search by entering a search query.
- The **Perform Search** process handles the query and interacts with the **User Database** to fetch necessary user-related information.
- The **Retrieve Images** process uses the search criteria to query the **Image Database** and retrieve relevant images.
- The results are then delivered back to the user.

3.Schema Diagram

Tables and Columns

1. **User Table**

```
Table: User------------
UserID (PK)
Username
Email
```

2. **Search Table**

```
Table: Search--------------
SearchID (PK)
UserID (FK)
SearchQuery
SearchTimestamp
```

3. **Image Table**

```
Table: Image-------------
ImageID (PK)
FilePath
Description
Timestamp
Location
```

Relationships

- `UserID` in the `Search` table is a foreign key referencing the `User` table.
- The `Search` table connects to the `Image` table through search results.

Diagram

```
+------------------------+    +------------------------+|        User          |     |
Search          |
|------------------------|    |------------------------|
| UserID (PK)            |<--->| SearchID (PK)         |
| Username               |    | UserID (FK)            |
| Email                  |    | SearchQuery            |
+------------------------+    | SearchTimestamp        |
                              +------------------------+

                              +------------------------+
                              |        Image           |
                              |------------------------|
                              | ImageID (PK)           |
                              | FilePath               |
                              | Description            |
                              | Timestamp              |
```

These diagrams provides us with a comprehensive visual representation of the database system for storing images and their associated metadata.

E. SOME TABLES CREATED ON MY SQL (Implementation)

4.1.1 Images Table

Purpose: Store information about the images, including a reference to their metadata.

Column:

- `**id**` (Primary Key): Unique identifier for each image.
- `image_data`: Binary data of the image or a URL/path to the image file.
- `metadata_id` (Foreign Key): References the `metadata` table to link this image to its metadata.

Sql codes to create the image table

```
CREATE TABLE images (
   id SERIAL PRIMARY KEY,
   image_data BYTEA, -- or VARCHAR for URL/path to image
   metadata_id INTEGER REFERENCES metadata(id)
);
```

Diagram

4.1.2 Object Table

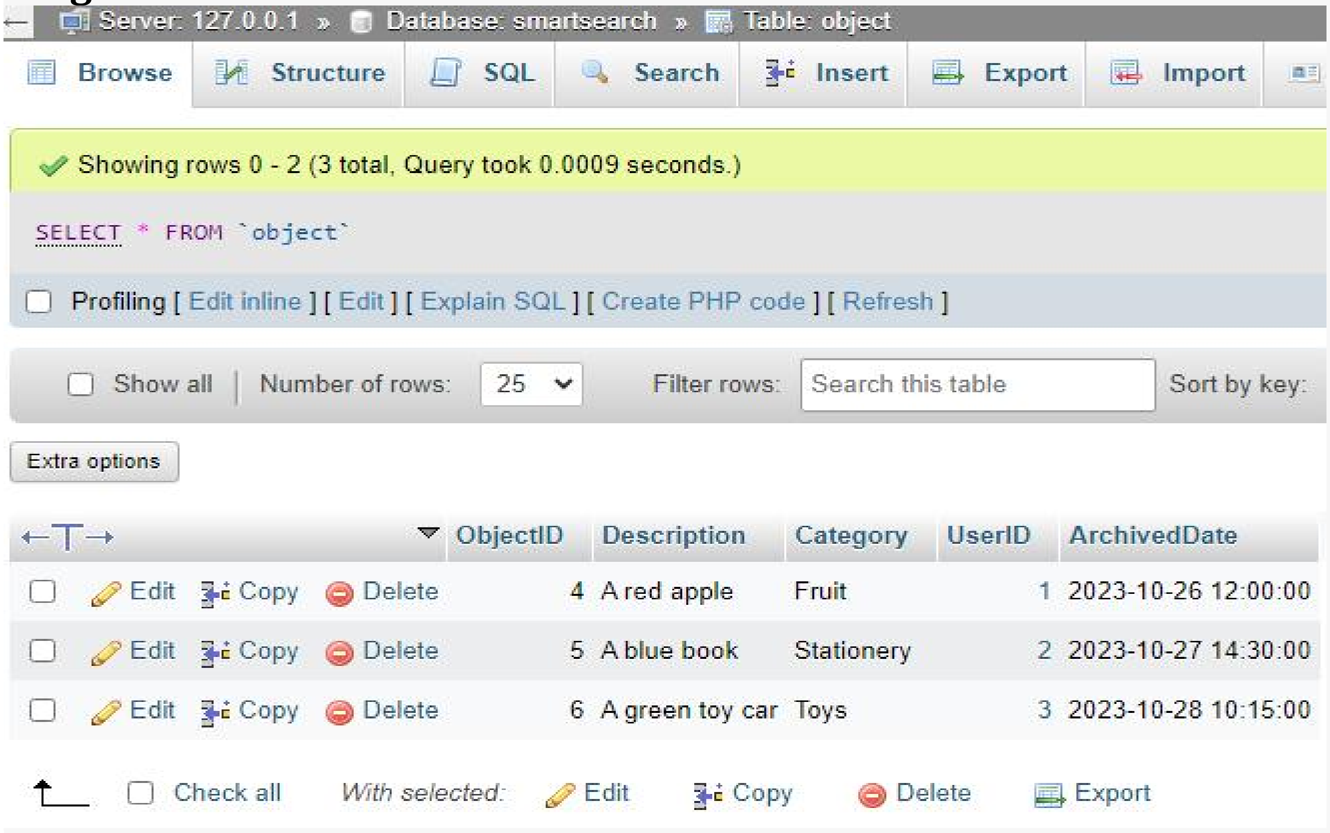
Purpose: Store metadata related to each image, such as descriptions, timestamps, locations, and keywords.

Columns:
- `**id**` (Primary Key): Unique identifier for each metadata entry.
- `description`: Textual description of the image.
- `timestamp`: Date and time when the image was archived.
- `location`: Geographical location associated with the image.
- `keywords`: Additional keywords for tagging and searching.

Sql code to create the metadata table

```
CREATE TABLE metadata (
   id SERIAL PRIMARY KEY,
   description TEXT,
   timestamp TIMESTAMP,
   location VARCHAR(255),
   keywords TEXT
);
```

Diagram



Server: 127.0.0.1 » Database: smartsearch » Table: object
Browse
Structure
SQL
Search
Insert
Export
Import
Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
Show all
Number of rows: 25
Filter rows: Search this table
Sort by key:
Extra options
Check all
With selected:
Edit
Copy
Delete
Export

Showing rows 0 - 2 (3 total, Query took 0.0009 seconds.)

```
SELECT * FROM `object`
```

	ObjectID	Description	Category	UserID	ArchivedDate
Edit Copy Delete	4	A red apple	Fruit	1	2023-10-26 12:00:00
Edit Copy Delete	5	A blue book	Stationery	2	2023-10-27 14:30:00
Edit Copy Delete	6	A green toy car	Toys	3	2023-10-28 10:15:00

User Table

Purpose: Stores user information, enabling us to track which user performed which searches and maintain user-specific data.

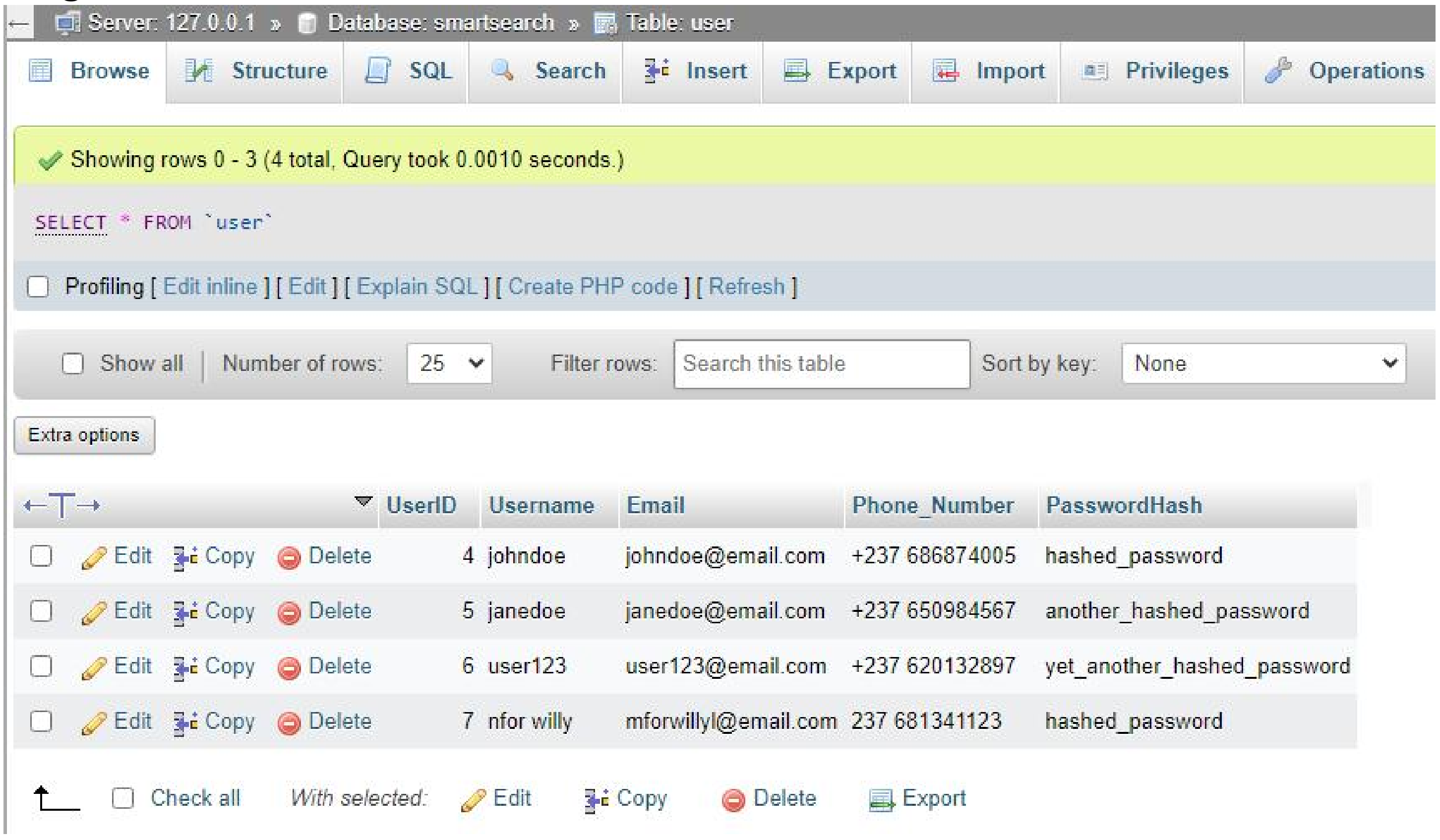
Columns :

User Id (PK)
Username,
Email
Password hash

Sql code :

```
CREATE TABLE User (
  UserID INT PRIMARY KEY AUTO_INCREMENT,
  Username VARCHAR(50) NOT NULL,
  Email VARCHAR(100) NOT NULL,
  PasswordHash VARCHAR(255) NOT NULL
);
```

Diagram


Server: 127.0.0.1 » Database: smartsearch » Table: user
Browse
Structure
SQL
Search
Insert
Export
Import
Privileges
Operations

Showing rows 0 - 3 (4 total, Query took 0.0010 seconds.)

```
SELECT * FROM `user`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
Show all
Extra options
Check all
With selected:
Edit
Copy
Delete
Export

Number of rows: 25 | Filter rows: Search this table | Sort by key: None

UserID	Username	Email	Phone_Number	PasswordHash
4	johndoe	johndoe@email.com	+237 686874005	hashed_password
5	janedoe	janedoe@email.com	+237 650984567	another_hashed_password
6	user123	user123@email.com	+237 620132897	yet_another_hashed_password
7	nfor willy	mforwillyl@email.com	237 681341123	hashed_password

Search Table

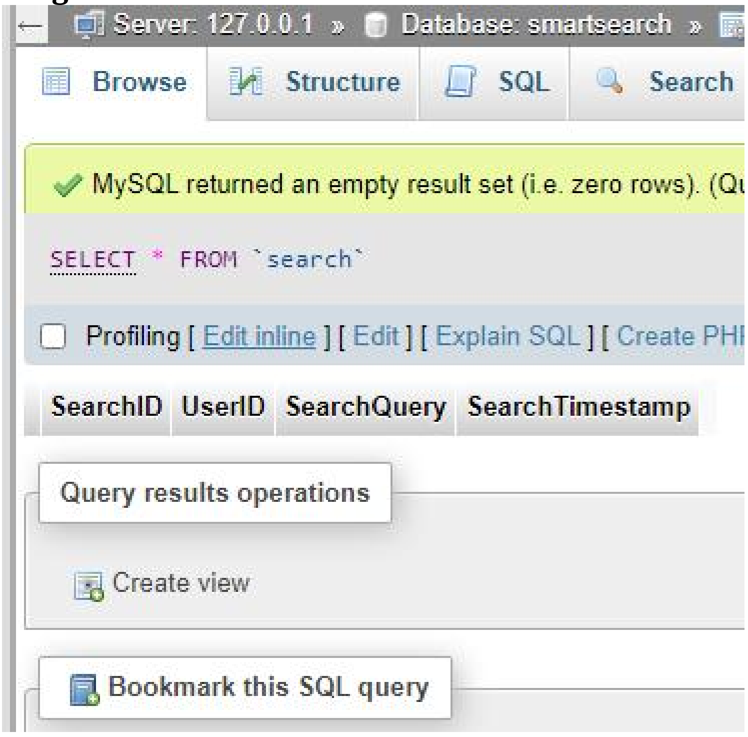
Purpose: Records each search query along with the user who performed it and the timestamp. This helps in tracking search history and analyzing user behavior.

Columns :
SearchID (PK)
UserID (FK)
SearchQuery
SearchTimestamp

Sql code :

```
CREATE TABLE Search (
   SearchID INT PRIMARY KEY AUTO_INCREMENT,
   UserID INT,
   SearchQuery TEXT NOT NULL,
   SearchTimestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY (UserID) REFERENCES User(UserID)
);
```

Diagram


Server: 127.0.0.1 » Database: smartsearch »
Browse
Structure
SQL
Search

✓ MySQL returned an empty result set (i.e. zero rows). (Qu

```
SELECT * FROM `search`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PH

SearchID	UserID	SearchQuery	SearchTimestamp

Query results operations
Create view
Bookmark this SQL query

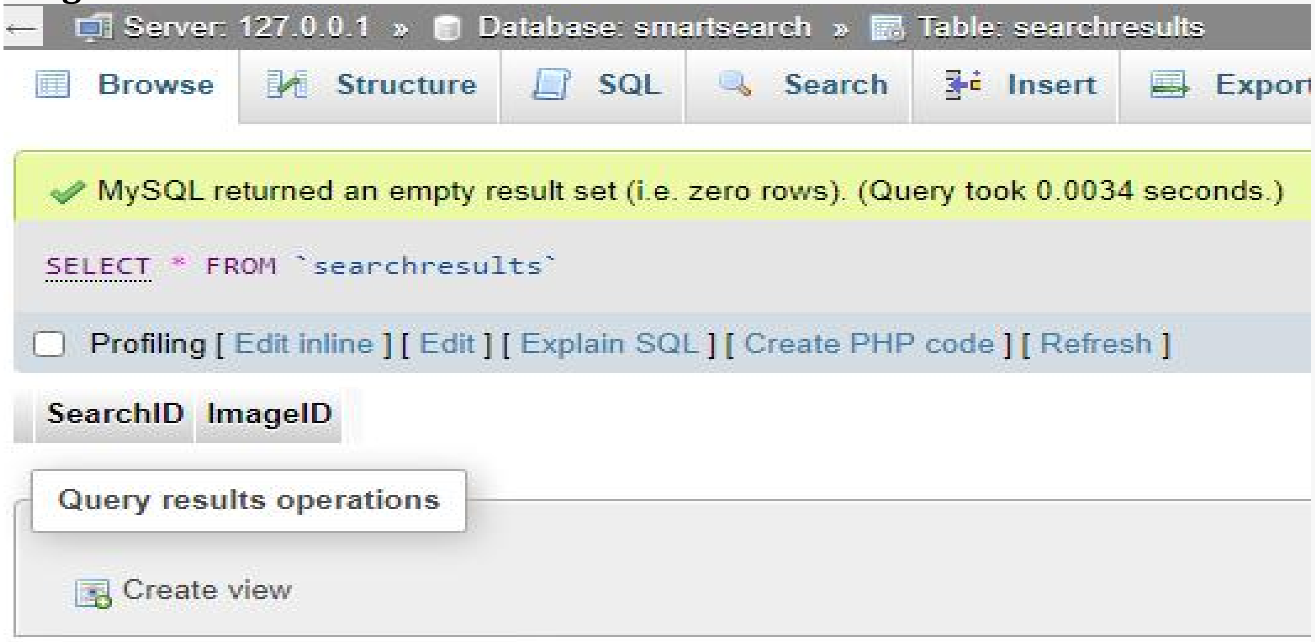
SearchResults Table (Optional)

Purpose: Facilitates the many-to-many relationship between searches and images, storing the results of each search query.
Columns :
SearchID (FK)
ImageID (FK)

Sql code :

```
CREATE TABLE SearchResults (
   SearchID INT,
   ImageID INT,
   PRIMARY KEY (SearchID, ImageID),
   FOREIGN KEY (SearchID) REFERENCES Search(SearchID),
   FOREIGN KEY (ImageID) REFERENCES Image(ImageID)
);
```

Diagram


Server: 127.0.0.1 » Database: smartsearch » Table: searchresults
Browse
Structure
SQL
Search
Insert
Expor

MySQL returned an empty result set (i.e. zero rows). (Query took 0.0034 seconds.)

```
SELECT * FROM `searchresults`
```

Profiling [Edit inline] [Edit] [Explain SQL] [Create PHP code] [Refresh]
SearchID
ImageID
Query results operations
Create view

4.2 Indexes for Efficient Searching

To ensure efficient searching and retrieval of data, indexes are created on relevant columns.

1. **Full-Text Search Indexes**: For `description` and `keywords` columns.
2. **Timestamp Index:** For the `timestamp` column.
3. **Location Index:** For the `location` column.

Sql code

```
-- Full-text search indexes
CREATE INDEX idx_object_description ON metadata USING gin(to_tsvector('english', description));
CREATE INDEX idx_metadata_keywords ON metadata USING gin(to_tsvector('english', keywords));

-- Timestamp index
CREATE INDEX idx_metadata_timestamp ON metadata (timestamp);

-- Location index
CREATE INDEX idx_metadata_location ON metadata (location);
```