Final Report: Social Media Analysis Database

Harley Gribble, Jason Zeng, Matthew Virginia

Abstract

This project presents the design and implementation of a database-backed web application for the structured analysis of social media content across multiple platforms and research projects. Developed using MySQL and Flask, the system enables users to create and manage posts, users, institutes, and projects, while defining project-specific analytic fields and recording manual annotations. The data model incorporates normalized tables, composite and foreign keys, and constraint-driven relationships, including support for ternary associations. A browser-based interface allows for form-driven data entry, post-project assignment, and field-level annotation, with built-in validation for logical consistency. The application also includes tools to query posts and monitor analysis coverage by field. The resulting system fulfills all core project requirements and provides a flexible, extensible framework for research teams conducting qualitative analysis of online content. Full Project is available in GitHub repository: https://github.com/b1akp1astik/social-media-analysis-db.

Table of contents

1. Introduction

The purpose of this project was to design and implement a research-oriented database system for managing and analyzing social media posts across multiple platforms. The system allows users to store original and reposted content, associate posts with research projects, and annotate them with manually entered analysis results. Rather than performing automated analysis, the system provides a structured interface for researchers to input and retrieve field-level annotations tied to specific projects and posts.

This functionality is critical in academic and industry research settings where large volumes of social media data need to be evaluated and tagged based on dynamic, project-specific criteria. To support this, the system enables users to define custom analytical fields for each project and record partial or complete analysis results per post. Researchers can also query posts by metadata such as media platform, author, or timestamp, and view analysis coverage across defined fields.

The application is built using a Python Flask backend connected to a MySQL relational database. HTML and Jinja2 templates are used to create a browser-based interface for data entry and exploration. Pytest was used to implement and run unit tests to ensure the correctness of the backend logic. The system was developed and tested on a local environment and supports basic CRUD functionality, complex querying, and validation to maintain data integrity.

---

2. Project Requirements

This project was developed in response to a detailed set of functional requirements focused on building a structured and queryable database system for social media post analysis. The following requirements, drawn from the project instructions and reflected in our completed implementation, define the scope of the system:

- **Social Media Post Storage**
  The system must store both original and reposted social media posts. Each post record includes metadata such as the author (a user associated with a specific media platform), timestamp, post content, and indicators such as whether the post contains multimedia content. Additional optional attributes include the number of likes/dislikes and location details (city, state, country). Reposts are stored as independent entities that reference the original post and capture the time and user responsible for the reposting.

- **User Demographics and Platform Association**
  Each user in the system is uniquely identified by their username and associated media platform. The system must support demographic attributes including first and last name, country of birth and residence, age, gender, and verified status. This ensures support for diverse user populations while enforcing uniqueness within platforms.

- **Projects and Institutes**
  Users can define research projects that serve as containers for social media post analysis. Each project is linked to a managing institute, includes a project manager's name, and has a defined start and end date. Projects can be created independently and later associated with posts and analytic fields.

- **Field Definitions and Analysis Entry**
  Projects can define a flexible set of **fields** to be used in post-level analysis (e.g., "Sentiment", "Topic", "Bias"). Posts assigned to a project can then be annotated with string-valued results for each field. The system supports **partial results**—not all fields are required to be filled in at once, allowing for progressive analysis over time.

- **Post-Project Assignment**
  The system allows researchers to assign any stored post to one or more projects. This enables reuse of posts across different analytic efforts without duplication, and ensures that each analysis result is properly contextualized within its associated project and fields.

- **Query and Search Capabilities**
  Users can perform searches over stored posts using filters such as media platform, posting date, username, or post author's name. In addition, project views allow researchers to list all posts associated with a project and view analysis progress, including field-level coverage statistics that indicate the percentage of posts annotated per field.

These requirements guided the schema design, application logic, and user interface implementation, ensuring a system that is both flexible and aligned with real-world research workflows.

3. Database Schema

3.1 Relational Schema Overview

The logical relational schema was implemented in MySQL. Each entity and relationship maps to one or more tables in the final database. The design emphasizes normalization, constraint enforcement, and flexibility to support dynamic analysis use cases.

Entity-to-Table Mapping

- **User** → `User`
  Stores users uniquely identified by their platform (`MediaName`) and username. Includes optional demographic and verification attributes.
- **SocialMedia** → `SocialMedia`
  A simple table listing supported platforms (e.g., Twitter, Instagram).
- **Post** → `Post`
  Stores original posts authored by users, including content, timestamp, optional location metadata, and like/dislike counts.
- **Repost** → `Repost`
  Records when a user reposts another post. The repost has its own timestamp and is connected to the original post and reposting user via foreign keys.
- **Institute** → `Institute`
  A simple table listing the name of each research institute managing projects.
- **Project** → `Project`
  Stores metadata for each analysis project, including its manager, dates, and associated institute.
- **Field** → `Field`
  Stores the labels of analytical fields defined within a specific project context. Fields are only meaningful within the scope of the project that defines them.

Relationship Tables

- **ProjectPost** (M:N join between `Post` and `Project`)
  Enables assigning multiple posts to the same project, and vice versa. This allows posts to be reused across different research contexts without duplication.

- **PostAnalysis** (Ternary relationship between `Post`, `Project`, and `Field`)
  Captures the actual analysis result as a string value for a specific post, project, and field combination. This design supports partial results and flexible field definitions, and avoids redundancy by uniquely identifying each post-project-field triple.

Design Considerations

- **Composite Primary Keys**
  Used for entities such as `User`, `Post`, `Field`, and `PostAnalysis` to ensure uniqueness across platform or project contexts.
- **Foreign Keys and Referential Integrity**
  Enforced throughout the schema to prevent orphaned records. For example:
  - `User` → `SocialMedia`
  - `Post` → `User`
  - `Repost` → `Post` and `User`
  - `Field` → `Project`
  - `Project` → `Institute`
- **Normalization**
  The schema adheres to at least Third Normal Form (3NF), avoiding redundant data and ensuring that all attributes are functionally dependent on the primary key of their respective tables.
- **Nullability and Partial Data**
  Many fields are nullable to accommodate partial demographic entry and field-level analysis. This is aligned with the project requirement to allow incomplete records and gradually entered data.

Relational Schema Diagram

The figure below shows the implemented schema as visualized in DBeaver. It illustrates the primary and foreign key relationships and table-level structure.

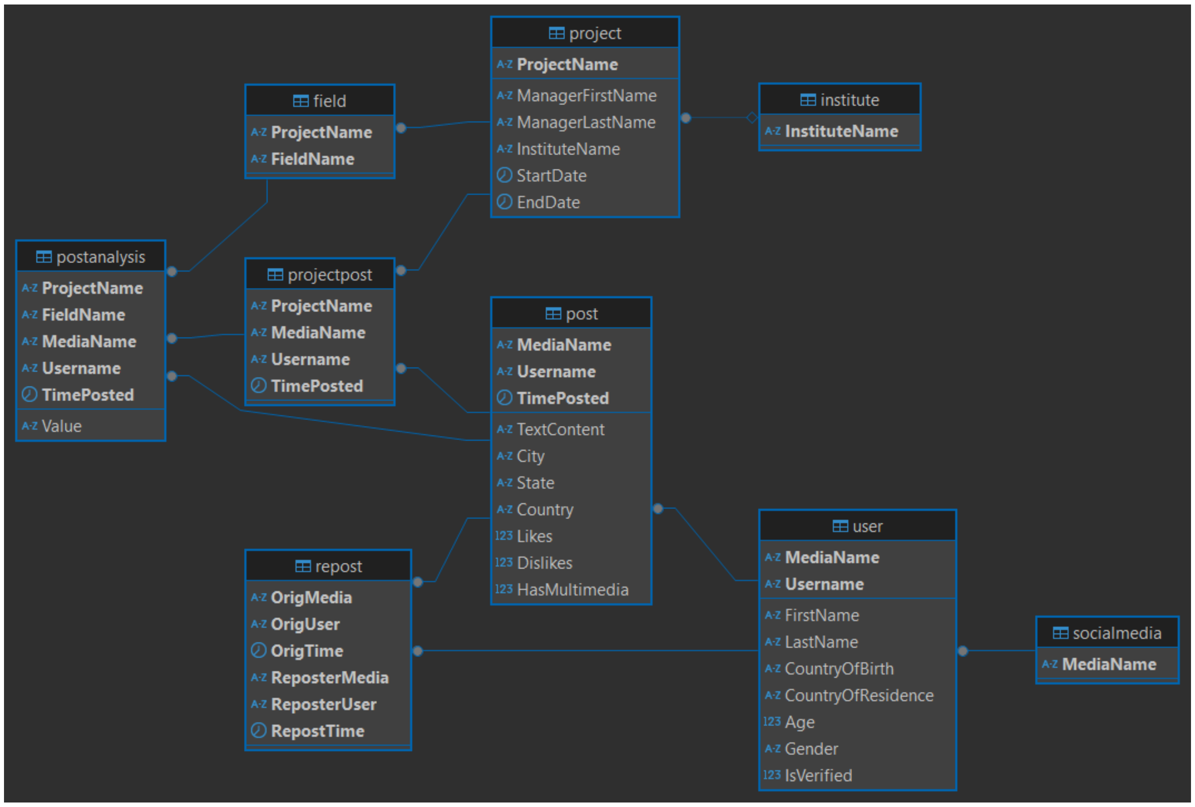


Figure 1: Relational Schema Diagram

3.2 SQL Table Definitions

The following tables were implemented in MySQL to reflect the conceptual and logical design of the system. Each table definition is accompanied by a brief explanation of its purpose, keys, and relevant constraints.

Table: `SocialMedia`

Stores supported media platforms (e.g., Twitter, Instagram).
Each user must belong to one platform, enforced through a foreign key in the `User` table.

```
CREATE TABLE SocialMedia (
  MediaName VARCHAR(50) PRIMARY KEY
);
```

Table: `User`

Stores user profiles and demographic information.
Composite primary key (`MediaName`, `Username`) ensures uniqueness per platform.
Includes optional fields such as country, gender (as ENUM), and verification status.

```
CREATE TABLE User (
  MediaName        VARCHAR(50),
  Username         VARCHAR(40),
  FirstName        VARCHAR(50),
  LastName         VARCHAR(50),
  CountryOfBirth   VARCHAR(50),
  CountryOfResidence VARCHAR(50),
  Age              INT,
  Gender           ENUM('Male','Female','Other'),
  IsVerified       BOOLEAN,
  PRIMARY KEY (MediaName, Username),
  FOREIGN KEY (MediaName) REFERENCES SocialMedia(MediaName)
);
```

Table: `Post`

Stores original posts authored by users.
Composite primary key (`MediaName`, `Username`, `TimePosted`) ensures one post per user per time.
Optional metadata includes location, like/dislike counts, and multimedia flag.
`Likes` and `Dislikes` are constrained to be non-negative.

```
CREATE TABLE Post (
  MediaName    VARCHAR(50),
  Username     VARCHAR(40),
  TimePosted   DATETIME,
  TextContent  TEXT     NOT NULL,
  City         VARCHAR(50),
  State        VARCHAR(50),
  Country      VARCHAR(50),
  Likes        INT DEFAULT 0    CHECK (Likes    >= 0),
  Dislikes     INT DEFAULT 0    CHECK (Dislikes >= 0),
  HasMultimedia BOOLEAN,
  PRIMARY KEY (MediaName, Username, TimePosted),
  FOREIGN KEY (MediaName, Username)
    REFERENCES User(MediaName, Username)
);
```

Table: **`Repost`**

Represents reposting of a post by another user.
Includes repost timestamp and constraints ensuring reposts occur **after** the original post.
Composite PK spans both users and timestamps for uniqueness.

```
CREATE TABLE Repost (
  OrigMedia      VARCHAR(50),
  OrigUser       VARCHAR(40),
  OrigTime       DATETIME,
  ReposterMedia  VARCHAR(50),
  ReposterUser   VARCHAR(40),
  RepostTime     DATETIME,
  PRIMARY KEY (
    OrigMedia, OrigUser, OrigTime,
    ReposterMedia, ReposterUser, RepostTime
  ),
  FOREIGN KEY (OrigMedia, OrigUser, OrigTime)
    REFERENCES Post(MediaName, Username, TimePosted),
  FOREIGN KEY (ReposterMedia, ReposterUser)
    REFERENCES User(MediaName, Username),
  CONSTRAINT chk_repost_after_orig
    CHECK (RepostTime >= OrigTime)
);
```

Table: **`Institute`**

Stores the name of each institution managing projects.
Simple table with a single primary key.

```sql
CREATE TABLE Institute (
  InstituteName VARCHAR(100) PRIMARY KEY
);
```

Table: `Project`

Represents a research analysis project.
Includes manager information and project timeline.
Each project is affiliated with a single institute.

```sql
CREATE TABLE Project (
  ProjectName      VARCHAR(100) PRIMARY KEY,
  ManagerFirstName VARCHAR(50),
  ManagerLastName  VARCHAR(50),
  InstituteName    VARCHAR(100),
  StartDate        DATE,
  EndDate          DATE,
  FOREIGN KEY (InstituteName) REFERENCES Institute(InstituteName),
  CHECK (EndDate >= StartDate)
);
```

Table: `Field`

Defines named analytic fields (e.g., sentiment) used within a specific project.
Composite PK ensures uniqueness of field names per project.

```sql
CREATE TABLE Field (
  ProjectName VARCHAR(100),
  FieldName   VARCHAR(50),
  PRIMARY KEY (ProjectName, FieldName),
  FOREIGN KEY (ProjectName) REFERENCES Project(ProjectName)
);
```

Table: `ProjectPost`

Join table that links posts to projects for analysis.
Supports many-to-many mapping.
Composite PK uniquely identifies each assignment.

```
CREATE TABLE ProjectPost (
  ProjectName VARCHAR(100),
  MediaName   VARCHAR(50),
  Username    VARCHAR(40),
  TimePosted  DATETIME,
  PRIMARY KEY (ProjectName, MediaName, Username, TimePosted),
  FOREIGN KEY (ProjectName)             REFERENCES Project(ProjectName),
  FOREIGN KEY (MediaName, Username, TimePosted)
    REFERENCES Post(MediaName, Username, TimePosted)
);
```

Table: `PostAnalysis`

Captures the result of an analysis field applied to a post within a project.
Ternary relationship (Post–Field–Project) with a `Value` attribute.
Supports partial analysis (not all fields must be filled for every post).
FK constraints ensure analysis entries are only stored for valid field-project-post assignments.

```
CREATE TABLE PostAnalysis (
  ProjectName VARCHAR(100),
  FieldName   VARCHAR(50),
  MediaName   VARCHAR(50),
  Username    VARCHAR(40),
  TimePosted  DATETIME,
  Value       TEXT,
  PRIMARY KEY (ProjectName, FieldName, MediaName, Username, TimePosted),
  FOREIGN KEY (ProjectName, FieldName)
    REFERENCES Field(ProjectName, FieldName),
  FOREIGN KEY (MediaName, Username, TimePosted)
    REFERENCES Post(MediaName, Username, TimePosted)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  FOREIGN KEY (ProjectName, MediaName, Username, TimePosted)
```

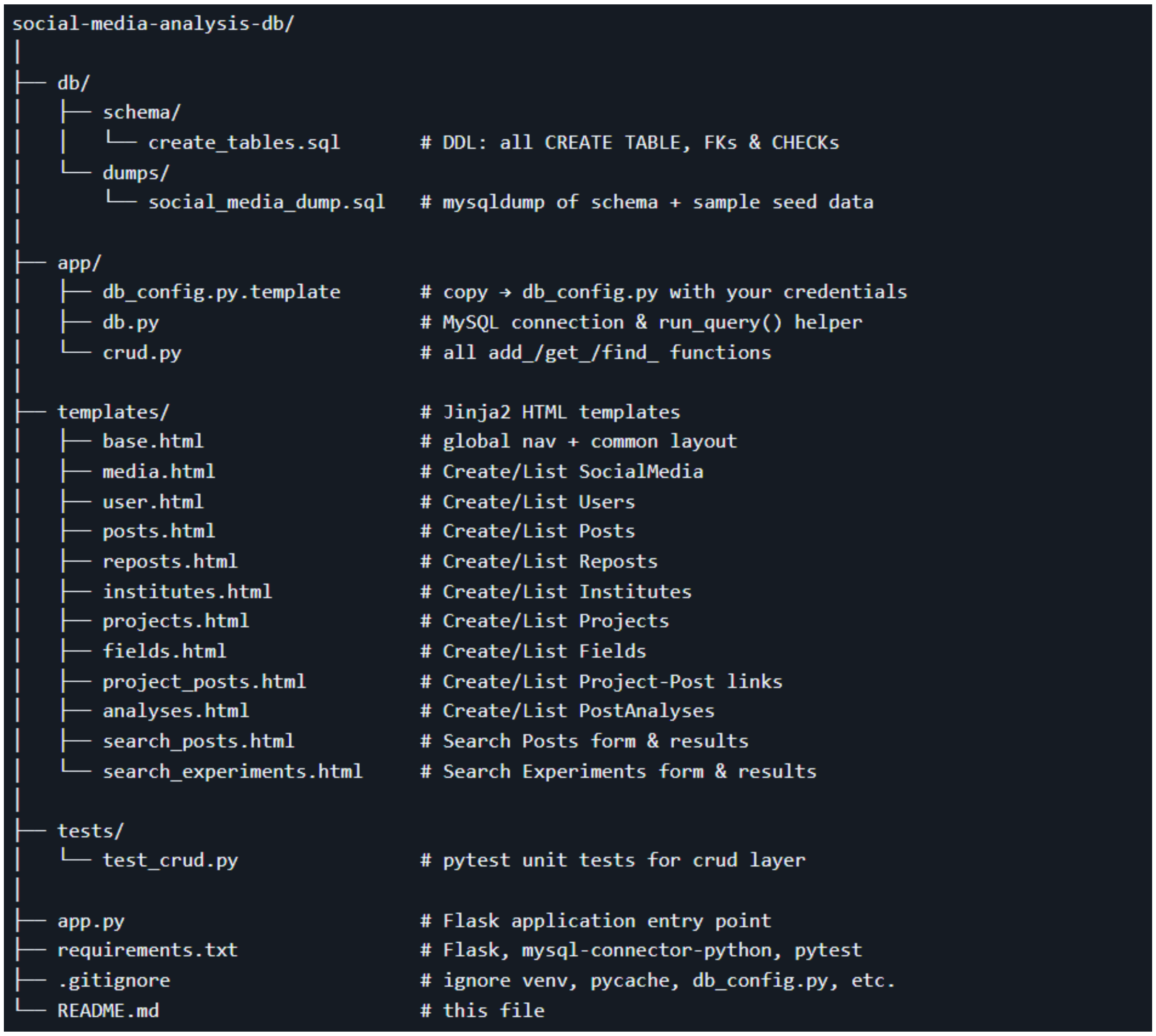
```
    REFERENCES ProjectPost(ProjectName, MediaName, Username, TimePosted)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

4. Implementation

4.1 System Architecture

The system is a web-based database application built using Python's Flask framework and a MySQL relational backend. It follows a modular architecture that separates database logic, application routes, and interface templates to ensure maintainability and clarity.

Folder Structure and Core Components



```
social-media-analysis-db/
│
├── db/
│   ├── schema/
│   │   └── create_tables.sql       # DDL: all CREATE TABLE, FKs & CHECKs
│   └── dumps/
│       └── social_media_dump.sql   # mysqldump of schema + sample seed data
│
├── app/
│   ├── db_config.py.template       # copy → db_config.py with your credentials
│   ├── db.py                       # MySQL connection & run_query() helper
│   └── crud.py                     # all add_/get_/find_ functions
│
├── templates/                      # Jinja2 HTML templates
│   ├── base.html                   # global nav + common layout
│   ├── media.html                  # Create/List SocialMedia
│   ├── user.html                   # Create/List Users
│   ├── posts.html                  # Create/List Posts
│   ├── reposts.html                # Create/List Reposts
│   ├── institutes.html             # Create/List Institutes
│   ├── projects.html               # Create/List Projects
│   ├── fields.html                 # Create/List Fields
│   ├── project_posts.html          # Create/List Project-Post links
│   ├── analyses.html               # Create/List PostAnalyses
│   ├── search_posts.html           # Search Posts form & results
│   └── search_experiments.html     # Search Experiments form & results
│
├── tests/
│   └── test_crud.py                # pytest unit tests for crud layer
│
├── app.py                          # Flask application entry point
├── requirements.txt                # Flask, mysql-connector-python, pytest
├── .gitignore                      # ignore venv, pycache, db_config.py, etc.
└── README.md                       # this file
```

Figure 2: Directory Tree

- `app.py`
  The main entry point for the Flask application. It initializes the app, registers routes, and launches the server at runtime. When executed, it starts the web service on `http://localhost:5000`.

- `app/`
  Contains the core backend logic and database utility files:

  - `db.py`: Manages the database connection and engine setup using MySQL credentials from a configuration file.
  - `db_config.py`: Stores connection parameters (username, password, host, and database name). A template file is provided for easy setup.
  - `crud.py`: Implements Create, Read, Update, and Delete (CRUD) operations for all key entities (users, posts, projects, etc.). This file encapsulates the SQL logic for interacting with the database in a reusable format.

- `templates/`
  Contains all Jinja2-based HTML templates used for rendering entity-specific forms and tables. Each entity in the database has a corresponding view (e.g., `posts.html`, `users.html`, `projects.html`), along with matching `*_edit.html` files for data entry and modification. Shared layout components (e.g., navigation bar) are centralized in `base.html`.

- `db/schema/`
  Contains the SQL script `create_tables.sql` used to initialize the database. This script defines all necessary tables, constraints, and keys.

Application Features

The application allows users to perform a variety of data management and exploration tasks through an interactive web interface:

- **Media and User Management**
  Users can add new social media platforms and create user accounts tied to specific platforms. Each user entry includes optional demographic attributes and a verified status flag.

- **Post and Repost Entry**
  Users can create original posts with rich metadata, including timestamp, text content, and optional location and multimedia indicators. Reposts are entered with their own timestamp and are validated to ensure reposts occur only after the original post was published.

- **Project and Field Definition**
  Research projects can be created with a defined manager, affiliated institute, and time window. Within each project, users can define a custom set of fields that represent the dimensions along which posts will be analyzed.
- **Post Assignment and Manual Analysis**
  Posts can be assigned to one or more projects. Once assigned, analysts can enter string-based analysis results for any of the defined fields. Partial analysis entries are supported, meaning not all fields need to be filled out simultaneously.
- **Search and Query Tools**
  The system supports search functionality by platform, user, and timestamp. It also allows users to view field coverage within each project—i.e., how many assigned posts have been annotated with values for each defined field.

Data Validation and Integrity

- Reposts are only permitted when their `RepostTime` is equal to or later than the original post's timestamp. This is enforced both in the SQL schema (via a `CHECK` constraint) and application logic.
- Foreign key constraints ensure that all data associations (e.g., users, posts, projects, fields) are consistent.
- Application logic prevents duplicate entries and allows flexibility in entering optional or partial data.
- Output pages display associated metadata (e.g., field names, project affiliations) clearly in all views for user clarity.

This architecture ensures a clean separation between data management, business logic, and user interface, while offering a complete and user-friendly interface for conducting manual social media post analysis.

4.2 Data Integrity and Validation

Ensuring data integrity was a central focus of both the database schema design and the application logic. A combination of SQL-level constraints and Flask-based validation routines was used to enforce consistency, prevent logical errors, and allow flexible but controlled data entry.

Relational Integrity via Primary and Foreign Keys

- **Primary Keys (PKs)**
  Every table in the database has a clearly defined primary key to ensure entity uniqueness. Composite keys are used where necessary—for example, the `User` table uses `(MediaName, Username)` to uniquely identify users within the context of a specific platform, and the `Post` table uses `(MediaName, Username, TimePosted)` to prevent duplicate posts by the same user at the same time.

- **Foreign Keys (FKs)**
  Foreign key constraints maintain referential integrity between related tables:

  – Each `User` is required to belong to a valid `SocialMedia` platform.
  – Each `Post` must be associated with an existing `User`.
  – `Reposts` reference valid original posts and reposting users.
  – `Project` records must be tied to a valid `Institute`.
  – `Field` records must belong to a valid `Project`.
  – `PostAnalysis` results are only valid for posts properly assigned to a project and matching an existing field definition.

  These constraints prevent orphaned records, enforce consistent relationships, and provide a strong foundation for query reliability.

Application-Level Validation

Additional validation logic was implemented in the Flask application to enforce rules not easily captured by database constraints or to provide user-friendly error handling:

- **Unique Usernames per Platform**
  The application ensures that no two users share the same `(MediaName, Username)` combination. Attempts to enter duplicates are caught before SQL execution and returned with a user-facing error message.

- **Repost Timestamp Validation**
  To maintain chronological consistency, the application checks that all reposts occur at the same time or after the original post. This is enforced both in the SQL schema (via the `CHECK (RepostTime >= OrigTime)` constraint) and within the backend logic, which rejects invalid repost entries with clear feedback.

- **Support for Partial Data Entry**
  The system is designed to accommodate gradual data collection. For example:

  – Fields such as `CountryOfBirth`, `Dislikes`, or `HasMultimedia` in the `User` and `Post` tables are optional.

  - Analysis values in `PostAnalysis` are not required for all fields when a post is first assigned to a project.
  - This supports real-world research workflows where analysts may progressively annotate content over time.

Together, these integrity and validation mechanisms ensure that the data remains logically sound, cleanly structured, and aligned with real-world user behavior.

---

5. Application Features

The application provides a structured, form-based web interface that enables users to perform core data entry tasks and execute targeted queries across social media posts and research projects. All interaction occurs through clearly labeled HTML pages generated from Jinja2 templates.

5.1 Data Entry Features

Data entry in the system is fully form-driven and segmented into logical modules accessible through the navigation bar. Each entity has a dedicated view for both creating new records and editing existing ones.

- **Create and Edit Users**
  Users can be added by selecting a media platform and entering a username, along with optional demographic fields such as first/last name, age, gender, country of birth/residence, and verification status. Duplicate usernames per platform are disallowed.
  View used: `user.html`, `user_edit.html`
- **Add Social Media Platforms**
  New platforms (e.g., "Twitter", "Instagram") can be added to the system via a simple text-entry form.
  View used: `media.html`, `media_edit.html`
- **Create and View Posts**
  Original posts are added with timestamp, text content, and optional location fields. Likes, dislikes, and multimedia flags are also supported. Posts are automatically associated with the current user and media.
  View used: `posts.html`, `posts_edit.html`
- **Record Reposts**
  Reposts are created by selecting an existing post and entering the reposting user and repost time. Reposts are validated to ensure they occur on or after the original post's timestamp.
  View used: `reposts.html`, `repost_edit.html`
- **Create Projects and Define Fields**
  Users can define new projects with a name, manager information, start/end dates, and institute. Within each project, analytic fields (e.g., "Sentiment", "Category") can be added dynamically.
  Views used: `projects.html`, `projects_edit.html`, `fields.html`, `fields_edit.html`

- **Assign Posts to Projects**
  Posts can be assigned to one or more projects via a dropdown selection interface. This allows the same post to be reused across different research contexts.
  View used: `project_posts.html`, `project_posts_edit.html`
- **Enter Analysis Results**
  Once a post has been assigned to a project, users may input string-valued results for any defined fields. Partial results are allowed—fields may be left blank and completed later. The system validates that each result corresponds to an assigned post and valid field.
  View used: `analyses.html`, `analyses_edit.html`

5.2 Query Features

The application includes built-in search tools to help researchers locate and explore relevant data across the system. These searches are executed via form inputs and rendered as structured tables for readability.

- **Search Posts**
  Users can search for posts using multiple filters:
  - Media platform
  - Username
  - Timestamp (range)
  - Optional keyword search in the post content
    Matching results include the full post text and associated metadata.
    View used: `search_posts.html`
- **Explore Projects and Analysis Progress**
  Researchers can view all posts assigned to a specific project. For each field, the system calculates and displays the percentage of assigned posts that have been analyzed (i.e., have a non-null value). This helps track progress and identify gaps.
  View used: `search_experiments.html`

All query views are read-only and optimized for clarity and usability. They are designed to assist users in managing the research process and ensuring adequate coverage of all analysis fields.

---

6. Example Usage

This section illustrates a basic use case of the system, demonstrating how a researcher would interact with the interface to create entities, annotate a post, and view analysis progress. It provides a high-level overview of how the system supports the analytical workflow.

Example: Analyzing a Social Media Post

1. **Create a User and a Post**
   From the **"Users"** tab, a new user is added with the platform "Twitter" and username "jdoe". Demographic information is optionally filled in.
   Next, from the **"Posts"** tab, an original post authored by "jdoe" is added with timestamp, location, and content:
   "The event was amazing! Everyone should've seen it."

2. **Create a Project and Define Fields**
   In the **"Projects"** section, a new project is created:

   - Name: *Event Sentiment Study*
   - Manager: Jane Smith
   - Affiliated Institute: *Research Lab A*
   - Fields defined: `"Sentiment"`, `"Topic"`
     These fields allow analysts to manually classify the post.

3. **Assign Post to Project and Annotate**
   From the **"Project Posts"** tab, the post is assigned to the *Event Sentiment Study* project.
   Then, from the **"Analyses"** tab, the user selects the post and enters:

   - `Sentiment`: *Positive*
   - `Topic`: *Event Experience*

4. **Query and View Analysis Progress**
   From the **"Search Experiments"** tab, selecting the *Event Sentiment Study* project displays a list of assigned posts and shows the percentage for each field.
   In this example, both fields for one post are filled, resulting in 100% coverage.

This example shows how the system enables manual, structured analysis of social media content within a project-based research framework. Each step is accessible through a user-friendly form interface and enforces the data relationships defined in the schema.

---

7. Testing

Testing for this project was conducted using the `pytest` framework to ensure the correctness of CRUD operations and the enforcement of relational integrity. Unit tests are located in the file `tests/test_crud.py` and are designed to verify that the backend functions behave as expected in isolation and in sequence.

Each test uses a shared fixture (`cleanup_db`) that runs before and after every test case. This fixture deletes all rows from all tables to ensure that each test is performed on a clean slate, avoiding interference between tests and maintaining independence.

What's Tested

The test suite verifies the following key functionalities:

- **User and Platform Insertion**
  The `test_add_media_and_user()` test confirms that users can be successfully created under a valid media platform and that demographic information is stored and retrieved correctly.
- **Post and Repost Handling**
  The `test_add_post_and_repost()` test ensures that original posts can be added and retrieved, and that reposts are only created when valid original posts and reposting users exist. The repost timestamp is tested to confirm logical consistency.
- **Project Creation and Field Definition**
  The `test_project_and_field_and_analysis()` test validates the creation of institutes, projects, and project-specific fields. It confirms that the `Project` and `Field` tables store expected values and that constraints (e.g., valid institute linkage) are respected.
- **Post Assignment and Analysis Entry**
  Within the same test, a user is created, a post is added, and it is assigned to a project via `add_project_post()`. A `PostAnalysis` entry is then inserted using `add_post_analysis()`, and the test confirms that the analysis value is correctly recorded and queryable.

Running the Tests

To run the full test suite, navigate to the project directory and execute the following command in the terminal:

```
pytest -q
```

This will run all test cases in a minimal format, showing only passed or failed assertions. All current tests pass successfully, confirming that the core application logic performs as intended.

Coverage and Scope

The test suite provides coverage for:

- Basic CRUD operations for all major entities
- Edge-case validation
- Relational consistency using primary and foreign key constraints
- Correct sequencing of inserts to support dependencies (e.g., project before field, user before post)

Additional UI-level and integration testing was conducted manually within the application, but the current tests provide strong backend confidence and align well with the goals and constraints of the assignment.

---

8. Installation and User Manual

This section provides a step-by-step guide for installing, configuring, and running the system. It is written for junior computer science students who may have no prior experience with databases or web frameworks.

All instructions assume the user is starting from a clean local machine and aims to run the application locally using the tools provided in the project.

8.1 System Requirements

Before beginning the installation, ensure the following software is installed and functioning on your system:

Python 3.x

The application backend is written in Python and requires Python 3.7 or newer. You can verify whether Python is installed and check the version by running:

```
python --version
```

If Python is not installed, it can be downloaded from:
https://www.python.org/downloads/

---

pip (Python Package Installer)

Python's package manager `pip` is used to install the required dependencies from the `requirements.txt` file. It is included with most Python installations.

To verify pip is installed:

```
pip --version
```

If not installed, follow instructions at:
https://pip.pypa.io/en/stable/installation/

MySQL Server

The system uses a MySQL (or MariaDB) server for database storage. Any recent version of MySQL Server should be compatible (e.g., 8.x). The database must be accessible locally.

If MySQL is not installed, it can be downloaded from:
https://dev.mysql.com/downloads/mysql/

After installation, ensure that: - You know your MySQL **root password** (or have permissions to create users and databases). - The MySQL service is running locally (check with your system's services or task manager).

You will use MySQL to: - Create a new database user - Create the schema from provided SQL files - Load test data if desired

---

Optional: MySQL Client (Command-Line or GUI)

Although not strictly required, having a MySQL client will help for running SQL scripts, inspecting tables, and troubleshooting. Options include:

- **MySQL CLI**: `mysql` command from terminal
- **DBeaver** (recommended): Free GUI tool to view and interact with databases
  https://dbeaver.io/download/

Once these system requirements are met, proceed to the next section: **9.2 Setup Instructions**, where we will walk through cloning the repository, configuring credentials, and launching the web application.

8.2 Setup Instructions

The following instructions guide the user through setting up the social media analysis database application on a local machine. This includes cloning the project, installing dependencies, configuring the database, and running the Flask web server.

All commands should be executed from a terminal or command prompt. Steps marked as "SQL" should be run in a MySQL client such as the MySQL CLI or a GUI like DBeaver.

Step 1: Clone the GitHub Repository

Clone the project files into a local directory:

```
git clone https://github.com/b1akp1astik/social-media-analysis-db
cd social-media-analysis-db
```

This creates a new folder named `social-media-analysis-db` containing the full source code.

Step 2: Install Python Dependencies

Ensure you are in the root of the project directory, then install all required Python libraries using `pip`:

```
pip install -r requirements.txt
```

This will install packages such as `Flask`, `mysql-connector-python`, and `pytest`.

Step 3: Set Up MySQL User (SQL)

Open your MySQL client and log in as a user with sufficient privileges (e.g., `root`). Then run the following command to create a database user for this project:

```sql
CREATE USER 'cs5330'@'localhost' IDENTIFIED BY 'cs5330';
```

Step 4: Create and Initialize the Database (SQL)

1. **Create a database for the project**:

```
CREATE DATABASE social_media_db;
```

2. **Switch to the new database**:

```
USE social_media_db;
```

3. **Run the schema creation script**:

If using the MySQL CLI:

```
SOURCE db/schema/create_tables.sql;
```

This will create all necessary tables with primary keys, foreign keys, constraints, and default values.

4. Grant user privelages to app:

```
GRANT ALL PRIVILEGES ON social_media.* TO 'cs5330'@'localhost';
FLUSH PRIVILEGES;
```

This will allow the app to interface with the database.

5. *(Optional)*: Load example data to test functionality:

```
SOURCE db/dumps/social_media_dump.sql;
```

This will insert predefined records into `SocialMedia`, `User`, `Post`, and other tables to support test scenarios.

Step 5: Configure Database Credentials

The application reads database connection settings from a configuration file. To set this up:

1. Copy the provided template file:

```
cp app/db_config.py.template app/db_config.py
```

2. Open `app/db_config.py` in a text editor and edit the fields:

```
DB_USER = 'cs5330'
DB_PASSWORD = 'cs5330'
DB_HOST = 'localhost'
DB_NAME = 'social_media_db'
```

Ensure these values match the credentials and database name you created in previous steps.

Step 6: Launch the Application

Start the Flask development server by running:

```
python app.py
```

If configured correctly, the application will start a local web server and output:

```
Running on http://127.0.0.1:5000/
```

Step 7: Access the Application

Open a web browser and navigate to:

```
http://127.0.0.1:5000/
```

You should see the home page of the Social Media Analysis Database. From here, you can begin adding users, posts, projects, and analysis data via the interface.

After completing these steps, the application is ready for use in a local development environment. For further guidance on how to use each interface, see the next section: **9.3 How to Use the App**.

8.3 How to Use the App

Once the application is running and accessible via `http://127.0.0.1:5000/`, users can interact with the system entirely through the web browser. The interface is organized around a horizontal navigation bar at the top of the page, with separate pages (views) for each major entity or task.

Each page contains a table listing all existing records for that entity and a form at the bottom or side of the page for entering new records. All data entry is performed using web forms—no SQL knowledge is required.

A. Managing Core Data

Use the following tabs in the navigation bar to enter and manage data:

Media

- Add social media platforms (e.g., "Twitter", "Instagram").
- Each platform must have a unique name.
- Required before adding any users.

Users

- Add new users by selecting a platform and providing a unique username.
- Optional demographic fields include first/last name, age, gender, country of birth/residence, and verification status.
- Users are identified by `(MediaName, Username)`.

Posts

- Create original posts by selecting an existing user and entering:
  - Timestamp (must be unique per user)
  - Text content (required)
  - Optional fields: city, state, country, number of likes/dislikes, multimedia flag
- Posts are uniquely identified by the author and timestamp.

Reposts

- Record reposts of existing posts.
- Select the original post and the reposting user.
- Enter a repost timestamp that must be the same or later than the original.
- Reposts are stored separately but linked to both the original post and reposting user.

B. Managing Projects and Fields

Projects

- Define a new research project by entering:
  - Project name (unique)
  - Project manager’s first and last name
  - Start and end dates
  - Affiliated institute (must already exist)

Fields

- After creating a project, define analysis fields such as `"Sentiment"` or `"Category"` that will be used to annotate posts.
- Fields are unique within each project and are used later for assigning values to posts.

Institutes

- Add institutes before creating projects.
- Each institute must have a unique name.

C. Assignments and Analysis

Project Posts

- Assign existing posts to one or more projects.
- This enables those posts to be annotated with field values.
- Each assignment links a post to a project and prepares it for manual analysis.

Analyses

- For each assigned post, enter analysis values for any defined fields in that project.
- All values are strings (e.g., `"Positive"`, `"News"`, `"Satirical"`).
- Partial results are allowed: not all fields need to be filled at once.

D. Search and Query Tools

Search Posts

- Use this tab to search for posts by:
  - Username
  - Media platform
  - Date/time range
  - Optional keyword in the post's text
- Matching posts are displayed in a table along with author and content metadata.

Search Experiments

- View the contents of a selected project.
- Lists all assigned posts and displays the **field coverage percentage** for each defined field—i.e., what percent of posts have a non-null value for that field.
- This helps track analysis progress.

After completing entries in each section, data is automatically saved and displayed in the corresponding table. The application supports editing or deleting entries through the interface, and error messages are shown when constraints are violated (e.g., duplicate keys or invalid repost times).

For best results, users should proceed in the following order:

1. Add media platforms and institutes
2. Add users and posts
3. Create projects and fields
4. Assign posts to projects

5. Enter field-level analysis values
6. Use search tools to review data and track project progress

---

9. Challenges and Lessons Learned

Throughout the development of this project, several challenges emerged that led to meaningful technical and conceptual learning. These included database modeling decisions, enforcing application logic, and learning new frameworks.

Learning Flask and Jinja Templates

This was our first time working with Flask and Jinja2. Understanding how to connect form inputs to backend processing, structure route handlers, and display query results using template inheritance required experimentation. Mastering this improved our ability to build interactive, consistent web interfaces that reflected underlying data accurately.

Managing Partial and Duplicate Data

Supporting real-world workflows like partial analysis results and optional user fields meant that we needed to allow nulls in many places while still maintaining referential integrity. At the same time, we had to prevent data duplication (e.g., the same user being added twice to the same platform, or the same post being assigned to a project multiple times). Handling this required the right balance of database constraints and backend validation.

Validation Oversights and Corrections

During testing, we identified several important validation gaps:

- **Future-Dated Posts**: The system initially allowed users to create posts with timestamps in the future, which was unrealistic and unintended. While not enforced at the SQL level, this was later corrected in the application logic.
- **Invalid Reposts**: The application allowed users to create reposts that occurred before the original post was published, violating logical time flow. This was corrected both with a `CHECK` constraint in the schema and additional Flask-side validation.
- **Cross-Project Analysis**: A user could mistakenly enter analysis results for a post that was not actually assigned to the selected project. To fix this, we enforced that any `PostAnalysis` entry must also exist in the `ProjectPost` table and added backend checks to restrict such mismatches.

These cases reinforced the importance of validating assumptions not just in schema design but also in user interactions. They also taught us how to handle edge cases through thoughtful application-layer safeguards in addition to database-level constraints.

Overall, these challenges deepened our understanding of database design, data validation, and full-stack application architecture. They highlighted how technical details, if left unchecked, can lead to logically inconsistent data and user confusion—and how iterative development can resolve them.

---

10. Conclusion

This project resulted in a fully functional, relational database application for managing and analyzing social media posts across multiple platforms and research projects. The system provides robust support for original posts, reposts, users, and institutes, as well as for defining and tracking analysis results within customizable projects.

Built using MySQL and Flask, the system emphasizes both flexibility and data integrity. Researchers can define project-specific fields, assign posts across contexts, and manually annotate content—all within a clean, form-driven interface. Partial data entry is supported throughout, aligning the system with real-world analysis workflows.

The application's schema is normalized and constraint-driven, with composite keys, foreign keys, and validation checks that prevent inconsistent or duplicate data. While some features—such as user authentication or advanced data typing for fields—are outside the scope of the current implementation, the system was intentionally designed to be modular and extensible.

Future improvements could include:

- Support for typed or enumerated field values (e.g., numeric scores or category options)
- Role-based access control for multiple users
- Integration of basic analytics, reporting, or export tools
- Implement more intuitive table displays in app
- Incorporate reposts in the post display
- Incorporate display filters for each page

In its current form, the system serves as a solid foundation for structured, manual social media content analysis and demonstrates a strong understanding of database design, application development, and user-centered interface structure.

11. Contributions

List group member roles and tasks:

- Jason Zeng: Python backend, CRUD logic, Pytest Test Suite
- Harley Gribble: Schema design, MyQSL Database set up, Frontend templates
- Matthew Virginia: validation logic, Flask UI integration
- All: demo, and report
