Social Media Platform Database Management System

Course Title: Database Management System Lab
Course Code: CSE-2424
Session: Spring-2025
Semester: 4th

Submitted By:
Name: Tahasina Tasnim Afra
Student ID: C233456

Instructor:
Ms. Mysha Sarin Kabisha
Assistant Lecturer
Department of Computer Science and Engineering
International Islamic University Chittagong

Submission Date: July 13, 2025

Contents

1 Abstract

This project presents the design and development of a database management system (DBMS) for a simplified social media platform, modeled after features commonly seen in applications like Instagram. The aim is to apply core DBMS concepts such as entity-relationship modeling, normalization, relational schema creation, and SQL querying to manage data related to users, posts, comments, likes, replies, follows, and user profiles. The system follows a normalized database design up to Third Normal Form (3NF), ensuring minimal redundancy and improved data consistency. It uses strong, weak, and associative entities to accurately model real-world relationships such as user-follow-user and user-like-post interactions. For the backend, Oracle 10g Express Edition is used as the database engine, with SQL queries executed through PHP. The web interface is developed using HTML, CSS, PHP, and XAMPP as the local server environment. The OCI8 extension is used to connect PHP with the Oracle database. Users can input SQL queries through a textbox on the web page and view dynamic results. This project demonstrates real-world implementation of database principles in a social media context.

2 Introduction

2.1 Background

In the digital age, social media platforms like Instagram, Facebook, and Twitter have transformed how people connect, share, and communicate. These platforms manage vast amounts of user-generated data, including profiles, posts, comments, likes, and follower relationships. As user bases and interactions grow, organizing this data efficiently becomes a significant challenge. A robust Database Management System (DBMS) is essential to store, retrieve, and manage this data while ensuring integrity, minimizing redundancy, and supporting scalability. Without a well-designed database, platforms risk data inconsistencies and performance issues. This project addresses these challenges by developing a DBMS for a social media platform, modeling real-world interactions and providing a foundation for efficient data management.

2.2 Objective

The primary objective of this project is to design a database system that accurately models a simplified social media platform using real-world relationships such as posts, comments, likes, follows, and user profiles. The system is intended to be efficient, normalized, and user-friendly.

- Design and implement a fully normalized database (up to 3NF) for managing users, posts, comments, likes, replies, and follow relationships.
- Apply entity-relationship modeling using strong, weak, and associative entities to reflect real-world social media behavior.
- Develop a web-based front-end using HTML, CSS, and PHP, connected to the Oracle 10g XE database via XAMPP and OCI8.
- Provide a functional SQL query interface for retrieving, analyzing, and displaying live data.
- Enable meaningful SQL-based insights such as total likes, followers, post counts, and active users using GROUP BY, subqueries, and joins.

2.3 Scope

This project focuses on the backend database structure and query interface for a simplified social media platform. It emphasizes data organization, integrity, and interaction through SQL, but intentionally avoids frontend-heavy or advanced application logic.

Includes:

- Database design with an Entity-Relationship Diagram (ERD) and normalization up to Third Normal Form (3NF).
- Creation of Oracle XE 10g tables for core components: users, posts, comments, replies, likes, follows, and user profiles.
- A lightweight, PHP-based web interface (index.php and results.php) developed using HTML/CSS and run on XAMPP, allowing users to submit and view SQL query results.

Excludes:

- User authentication or session management.
- Handling or storing image/media files within the database.
- Implementation of real-time features such as notifications, live chats, or push updates.

3 Requirement Analysis

3.1 Functional Requirements

The Social Media Platform DBMS is designed to support key user interactions and data operations, emulating the core features of a simplified social media environment. The system must support the following functionalities:

- **User Account Management**: Users can register and manage their accounts, with credentials and personal details stored in the `users` table (e.g., username, email, password).
- **Content Creation**: Users can create and publish posts containing a title and body text. Each post is stored in the `posts` table, along with relevant metadata such as timestamp and visibility status.
- **Engagement Features**: Users can engage with posts by liking (`likes` table), commenting (`comments` table), and replying to comments (`reply` table).
- **Social Connections**: Users can follow other users. These relationships are recorded in the `follows` table, enabling features like personalized feeds.
- **Profile Updates**: Users can update their bios and profile pictures. These details are managed in the `user_profile` table, with support for version tracking.
- **Data Retrieval and Querying**: A PHP-based web interface (`index.php`) allows users to execute SQL queries. Query results (e.g., popular posts, user connections) are displayed via `results.php`.

3.2 Non-Functional Requirements

To ensure usability, efficiency, and maintainability, the system must satisfy the following non-functional requirements:

- **Performance**: SQL queries should return results within 1 second for datasets containing up to 10,000 records, utilizing Oracle XE 10g's query optimization features.
- **Data Integrity**: The system must enforce database constraints such as unique usernames, referential integrity via foreign keys (in `posts`, `comments`, `likes`, `follows`, `reply`) to prevent anomalies and ensure consistency.
- **Scalability**: The database is designed using Third Normal Form (3NF), enabling future feature additions and table expansions with minimal impact on the existing structure.
- **User Interface**: The web interface (`index.php`, `results.php`) should incorporate an Instagram-inspired design, including a gradient-themed header, intuitive SQL input field, and visually organized result display using HTML/CSS.
- **Reliability**: The system should operate smoothly under concurrent user interactions, supported by XAMPP and Oracle OCI8 for stable server-database communication.
- **Ease of Use**: The interface must be user-friendly and accessible to non-technical users, requiring minimal training to execute queries and interpret output.

4 Conceptual Model (ERD)

The Entity-Relationship Diagram (ERD) models the Social Media Platform DBMS, capturing the structure and relationships of its data. The ERD includes seven entities: three strong entities (`users`, `posts`, `comments`), two associative entities (`likes`, `follows`), and two weak entities (`user_profile`, `reply`). Designed using Draw.io, the ERD ensures a clear representation of the platform's core functionalities, such as user interactions and content management.

Entities and Attributes:

- **Users (Strong Entity)**: Represents platform users. Attributes: `id`, `username`, `email`, `password`, `created_at`.
- **Posts (Strong Entity)**: Represents user-generated content. Attributes: `id`, `user_id`, `title`, `body`, `status`, `created_at`.
- **Comments (Strong Entity)**: Represents user comments on posts. Attributes: `id`, `post_id`, `user_id`, `comment_text`, `created_at`.
- **Likes (Associative Entity)**: Models the many-to-many relationship between users and posts for likes. Attributes: `id`, `post_id`, `user_id`, `created_at`.
- **Follows (Associative Entity)**: Models the many-to-many relationship between users for following. Attributes: `id`, `following_user_id`, `followed_user_id`, `created_at`.
- **User_profile (Weak Entity)**: Represents user profile versions, dependent on `users`. Attributes: `user_id`, `profile_version`, `profile_picture`, `bio`, `updated_at`.
- **Reply (Weak Entity)**: Represents replies to comments, dependent on `comments`. Attributes: `comment_id`, `reply_id`, `user_id`, `reply_text`, `created_at`.

Relationships:

- **Users ↔ Posts (One-to-Many)**: A user can create multiple posts, but each post is created by one user (`user_id` in `posts` references `users.id`).
- **Posts ↔ Comments (One-to-Many)**: A post can have multiple comments, but each comment belongs to one post (`post_id` in `comments` references `posts.id`).
- **Users ↔ Comments (One-to-Many)**: A user can post multiple comments, but each comment is made by one user (`user_id` in `comments` references `users.id`).
- **Users ↔ Posts via Likes (Many-to-Many)**: A user can like multiple posts, and a post can be liked by multiple users, managed by the `likes` associative entity (`post_id`, `user_id` reference `posts.id`, `users.id`).
- **Users ↔ Users via Follows (Many-to-Many)**: A user can follow multiple users, and a user can be followed by multiple users, managed by the `follows` associative entity (`following_user_id`, `followed_user_id` reference `users.id`).
- **Users ↔ User_profile (One-to-One)**: Each profile belongs to one user (`user_id` in `user_profile` references `users.id`). The weak entity, `user_profile` depends on `users`.
- **Comments ↔ Reply (One-to-Many)**: A comment can have multiple replies, but each reply belongs to one comment (`comment_id` in `reply` references `comments.id`). As a weak entity, `reply` depends on `comments`.
- **Users ↔ Reply (One-to-Many)**: A user can post multiple replies, but each reply is made by one user (`user_id` in `reply` references `users.id`).

ERD Visualization: The ERD, created using Draw.io, visually represents these entities and relationships, with rectangles for strong entities, diamonds for relationships, double rectangles for weak entities, and lines indicating cardinalities (e.g., 1:N, M:N). The diagram is attached below (see Figure 1).

5 Normalization Process

Normalization ensures the Social Media Platform DBMS is efficient, minimizes redundancy, and maintains data integrity. The data is normalized from an Unnormalized Form (UNF) to the Third Normal Form (3NF) through systematic steps.

5.1 Unnormalized Form (UNF)

All data is initially stored in a single table, `SocialMediaData`, encompassing attributes for users, posts, comments, likes, follows, profiles, and replies. This structure causes redundancy, data anomalies, and violates atomicity principles.

Example Schema:

```
SocialMediaData(user_id, username, email, password, created_at, post_id, title,
body, status, post_date, comment_id, comment_text, commented_at, like_id, liked_at,
follow_id, following_user_id, followed_user_id, follow_date, user_profile_id,
biography, profile_picture, updated_at, reply_id, reply_text)
```

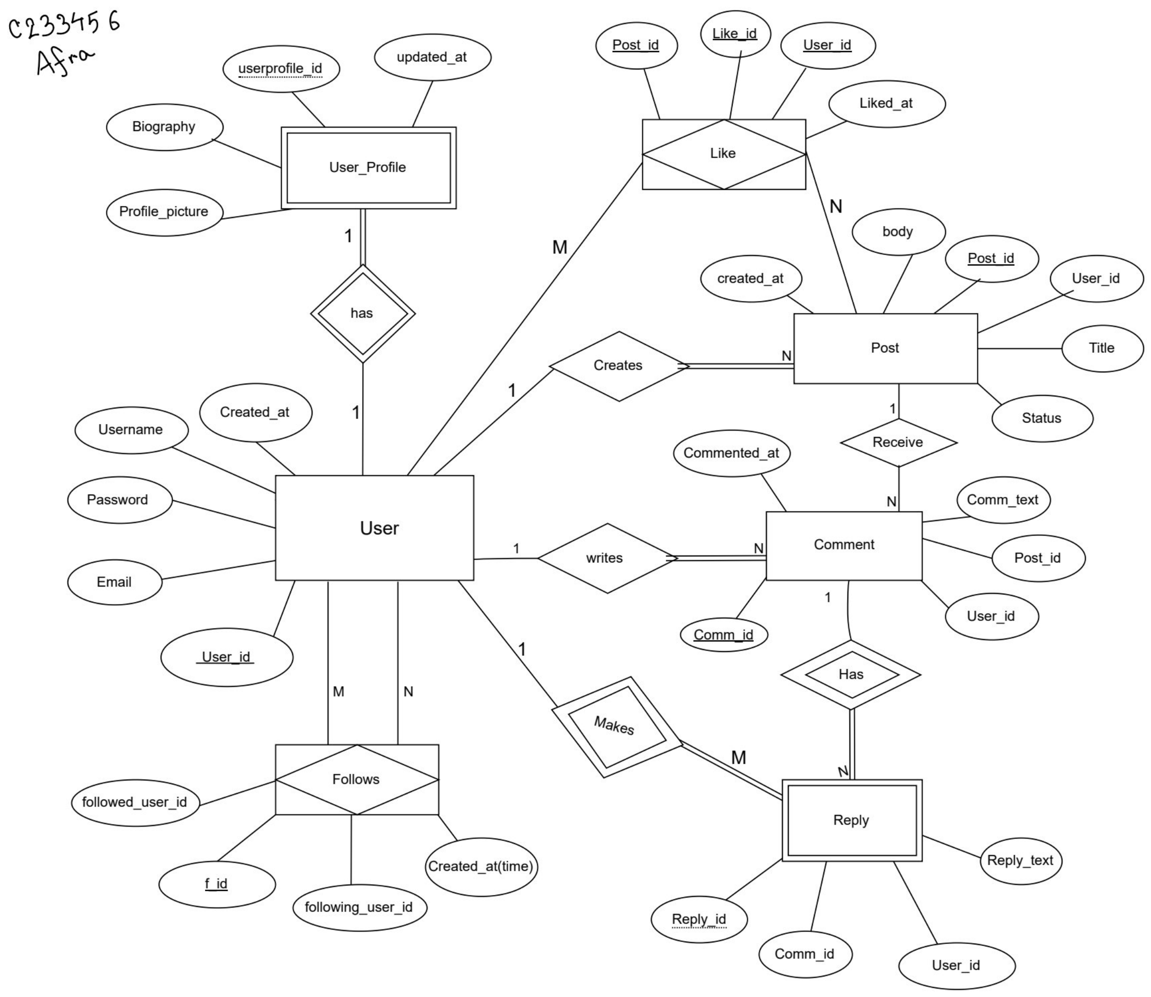


Figure 1: Social Media Database Management System ERD

user_id	username	post_id	comment_text	liked_at	reply_text
1	alice	101	Nice post!	2023-07-01	Thanks!
1	alice	102	Great idea	2023-07-02	I agree
2	bob	103	Interesting	2023-07-01	-

Table 1: Sample Unnormalized Data in `SocialMediaData` Table

Sample UNF Data:

Issues Identified:

- User information (like `username` "alice") is repeated for every post and comment.
- Repeating groups (e.g., multiple comments or replies per user) are stored in the same row.
- Some attributes may contain non-atomic values (e.g., multiple likes or replies in one field).
- Leads to insertion, update, and deletion anomalies.

Next Step: To resolve these issues, the data is decomposed into multiple atomic relations during First Normal Form (1NF), as discussed in the following section.

5.2 First Normal Form (1NF)

To achieve First Normal Form, all attributes must be atomic and no repeating groups should exist. Starting from the unnormalized data, the table is decomposed into multiple relations that represent distinct entities and their attributes.

Key Transformations:

- Eliminate repeating groups by separating user, post, comment, like, follow, profile, and reply information into different tables.
- Assign unique primary keys to each table, such as `user_id`, `post_id`, and composite keys where necessary.
- Ensure all attributes contain atomic values — no multi-valued or nested fields remain.

This process results in a set of normalized tables that conform to 1NF by organizing data into atomic, uniquely identifiable records.

5.3 Second Normal Form (2NF)

2NF requires 1NF and that non-key attributes fully depend on the entire primary key, eliminating partial dependencies.

Transformations:

- For `reply` (composite key: `comment_id, reply_id`), `user_id` and `reply_text` depend fully on the composite key.
- Tables with single-column primary keys (`users`, `posts`, `comments`, `likes`, `follows`, `user_profile`) have no partial dependencies.

Outcome: All tables satisfy 2NF, with non-key attributes fully dependent on primary keys.

5.4 Third Normal Form (3NF)

3NF requires 2NF and no transitive dependencies (non-key attributes depending on other non-key attributes).

Transformations:

- In `users`, `username`, `email`, `password`, `created_at` depend only on `user_id`.
- In `posts`, `title`, `body`, `status`, `post_date` depend on `post_id`; `user_id` is a foreign key.
- In `user_profile`, `biography`, `profile_picture`, `updated_at` depend on `user_profile_id`.
- No non-key attributes depend on other non-key attributes.

Outcome: All tables are in 3NF, ensuring minimal redundancy and data integrity.

6 Final Relational Schema

The final relational schema for the Social Media Platform DBMS is in 3NF, ensuring minimal redundancy, data integrity, and scalability. Below are the tables with their attributes, primary keys (PK), foreign keys (FK), and constraints, supporting user interactions like posting, commenting, liking, and following.

- **users(user_id, username, email, password, created_at)**
  - **PK**: `user_id`
  - **Constraints**: `username` is UNIQUE, `email` is UNIQUE
- **posts(post_id, user_id, title, body, status, created_at)**
  - **PK**: `post_id`
  - **FK**: `user_id` → `users(user_id)`
- **comments(comment_id, post_id, user_id, comment_text, commented_at)**
  - **PK**: `comment_id`
  - **FKs**:
    - `post_id` → `posts(post_id)`
    - `user_id` → `users(user_id)`
- **likes(like_id, post_id, user_id, liked_at)**
  - **PK**: `like_id`
  - **FKs**:
    - `post_id` → `posts(post_id)`
    - `user_id` → `users(user_id)`

- **follows(follow_id, following_user_id, followed_user_id, created_at)**
  - **PK**: `follow_id`
  - **FKs**:
    - `following_user_id` → `users(user_id)`
    - `followed_user_id` → `users(user_id)`
- **user_profile(user_id, biography, profile_picture, updated_at)**
  - **PK**: `user_id`
  - **FK**: `user_id` → `users(user_id)`
- **reply(comment_id, reply_id, user_id, reply_text)**
  - **PK**: `comment_id, reply_id`
  - **FKs**:
    - `comment_id` → `comments(comment_id)`
    - `user_id` → `users(user_id)`

Note: The `user_profile` table is a weak entity, with `user_id` as both its primary key and foreign key referencing `users(user_id)`, ensuring a one-to-one relationship. No additional `user_profile_id` is needed.

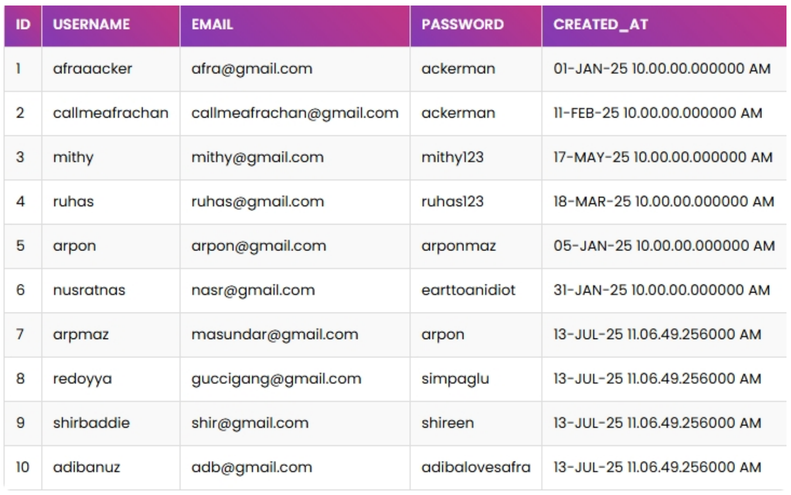
7 Table Creation and Sample Data

7.1 Table Structures (DDL)

```
CREATE TABLE users (
    id NUMBER PRIMARY KEY,
    username VARCHAR2(50) UNIQUE NOT NULL,
    email VARCHAR2(100) UNIQUE NOT NULL,
    password VARCHAR2(100) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
CREATE TABLE user_profile (
    user_id NUMBER NOT NULL,
    profile_version NUMBER NOT NULL,
    profile_picture VARCHAR2(200),
    bio CLOB,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT pk_user_profile PRIMARY KEY (user_id,
       profile_version),
    CONSTRAINT fk_user_profile FOREIGN KEY (user_id) REFERENCES
       users(id)
);
CREATE TABLE follows (
    id NUMBER PRIMARY KEY,
    following_user_id NUMBER NOT NULL,
    followed_user_id NUMBER NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT fk_following_user FOREIGN KEY (following_user_id)
       REFERENCES users(id),
    CONSTRAINT fk_followed_user FOREIGN KEY (followed_user_id)
       REFERENCES users(id)
);
CREATE TABLE posts (
    id NUMBER PRIMARY KEY,
    user_id NUMBER NOT NULL,
    title VARCHAR2(100),
    body CLOB,
    status VARCHAR2(20),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT fk_user_posts FOREIGN KEY (user_id) REFERENCES
       users(id)
);
CREATE TABLE likes (
    id NUMBER PRIMARY KEY,
    post_id NUMBER NOT NULL,
    user_id NUMBER NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT fk_post_likes FOREIGN KEY (post_id) REFERENCES
       posts(id),
    CONSTRAINT fk_user_likes FOREIGN KEY (user_id) REFERENCES
       users(id)
);
CREATE TABLE comments (
    id NUMBER PRIMARY KEY,
    post_id NUMBER NOT NULL,
    user_id NUMBER NOT NULL,
    comment_text CLOB,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT fk_post_comments FOREIGN KEY (post_id) REFERENCES
       posts(id),
    CONSTRAINT fk_user_comments FOREIGN KEY (user_id) REFERENCES
       users(id)
);
CREATE TABLE reply (
    comment_id NUMBER NOT NULL,
    reply_id NUMBER NOT NULL,
    user_id NUMBER NOT NULL,
    reply_text CLOB,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT pk_reply PRIMARY KEY (comment_id, reply_id),
    CONSTRAINT fk_reply_comment FOREIGN KEY (comment_id)
       REFERENCES comments(id),
    CONSTRAINT fk_reply_user FOREIGN KEY (user_id) REFERENCES
       users(id)
);
```

7.2 Sample Data (DML)

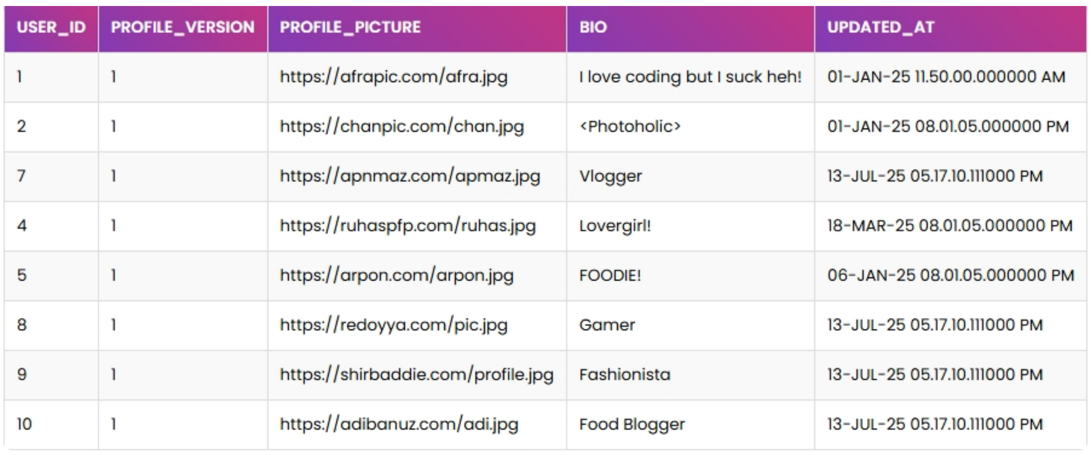
```
INSERT ALL
    INTO users (id, username, email, password, created_at)
  VALUES (1, 'afraaacker', 'afra@gmail.com', 'ackerman',
     CURRENT_TIMESTAMP)
  INTO users (id, username, email, password, created_at)
  VALUES (2, 'callmeafrachan', 'callmeafrachan@gmail.com', '
     ackerman', CURRENT_TIMESTAMP)
  INTO users (id, username, email, password, created_at)
  VALUES (3, 'mithy', 'mithy@gmail.com', 'mithy123',
     CURRENT_TIMESTAMP)
  INTO users (id, username, email, password, created_at)
  VALUES (4, 'ruhas', 'ruhas@gmail.com', 'ruhas123',
     CURRENT_TIMESTAMP)
  INTO users (id, username, email, password, created_at)
  VALUES (5, 'arpon', 'arpon@gmail.com', 'arponmaz',
     CURRENT_TIMESTAMP)
  INTO users (id, username, email, password, created_at)
  VALUES (6, 'nusratnas', 'nasr@gmail.com', 'earttoanidiot',
     CURRENT_TIMESTAMP)
  INTO users (id, username, email, password, created_at)
  VALUES (7, 'arpmaz', 'masundar@gmail.com', 'arpon',
     CURRENT_TIMESTAMP)
  INTO users (id, username, email, password, created_at)
  VALUES (8, 'redoyya', 'guccigang@gmail.com', 'simpaglu',
     CURRENT_TIMESTAMP)
  INTO users (id, username, email, password, created_at)
  VALUES (9, 'shirbaddie', 'shir@gmail.com', 'shireen',
     CURRENT_TIMESTAMP)
  INTO users (id, username, email, password, created_at)
  VALUES (10, 'adibanuz', 'adb@gmail.com', 'adibalovesafra',
     CURRENT_TIMESTAMP)
SELECT * FROM dual;
```



ID	USERNAME	EMAIL	PASSWORD	CREATED_AT
1	afraaacker	afra@gmail.com	ackerman	01-JAN-25 10.00.00.000000 AM
2	callmeafrachan	callmeafrachan@gmail.com	ackerman	11-FEB-25 10.00.00.000000 AM
3	mithy	mithy@gmail.com	mithy123	17-MAY-25 10.00.00.000000 AM
4	ruhas	ruhas@gmail.com	ruhas123	18-MAR-25 10.00.00.000000 AM
5	arpon	arpon@gmail.com	arponmaz	05-JAN-25 10.00.00.000000 AM
6	nusratnas	nasr@gmail.com	earttoanidiot	31-JAN-25 10.00.00.000000 AM
7	arpmaz	masundar@gmail.com	arpon	13-JUL-25 11.06.49.256000 AM
8	redoyya	guccigang@gmail.com	simpaglu	13-JUL-25 11.06.49.256000 AM
9	shirbaddie	shir@gmail.com	shireen	13-JUL-25 11.06.49.256000 AM
10	adibanuz	adb@gmail.com	adibalovesafra	13-JUL-25 11.06.49.256000 AM

Figure 2: Users table after inserting sample data.

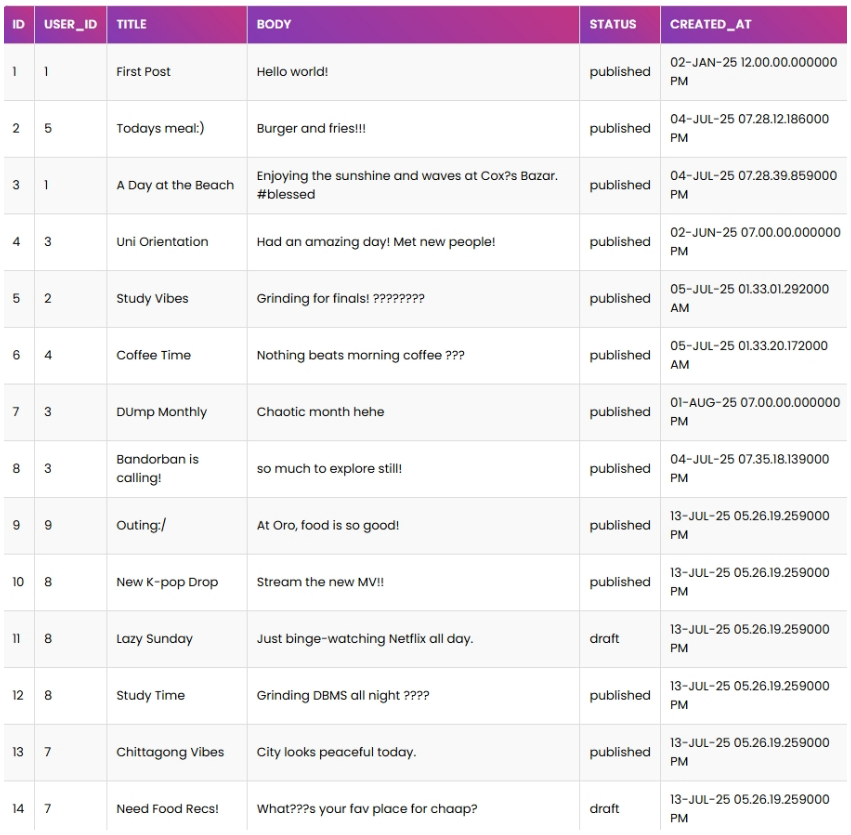
```
INSERT ALL
    INTO user_profile (user_id, profile_version, profile_picture,
        biography, updated_at)
  VALUES (1, 1, ’https://afrapic.com/afra.jpg’, ’I␣love␣coding␣
     but␣I␣suck␣heh!’, CURRENT_TIMESTAMP)
  INTO user_profile (user_id, profile_version, profile_picture,
     biography, updated_at)
  VALUES (2, 1, ’https://chanpic.com/chan.jpg’, ’<Photoholic>’,
     CURRENT_TIMESTAMP)
  INTO user_profile (user_id, profile_version, profile_picture,
     biography, updated_at)
  VALUES (7, 1, ’https://apnmaz.com/apmaz.jpg’, ’Vlogger’,
     CURRENT_TIMESTAMP)
  INTO user_profile (user_id, profile_version, profile_picture,
     biography, updated_at)
  VALUES (4, 1, ’https://ruhaspfp.com/ruhas.jpg’, ’Lovergirl!’,
     CURRENT_TIMESTAMP)
  INTO user_profile (user_id, profile_version, profile_picture,
     biography, updated_at)
  VALUES (5, 1, ’https://arpon.com/arpon.jpg’, ’FOODIE!’,
     CURRENT_TIMESTAMP)
  INTO user_profile (user_id, profile_version, profile_picture,
     biography, updated_at)
  VALUES (8, 1, ’https://redoyya.com/pic.jpg’, ’Gamer’,
     CURRENT_TIMESTAMP)
  INTO user_profile (user_id, profile_version, profile_picture,
     biography, updated_at)
  VALUES (9, 1, ’https://shirbaddie.com/profile.jpg’, ’
     Fashionista’, CURRENT_TIMESTAMP)
  INTO user_profile (user_id, profile_version, profile_picture,
     biography, updated_at)
  VALUES (10, 1, ’https://adibanuz.com/adi.jpg’, ’Food␣Blogger’,
     CURRENT_TIMESTAMP)
SELECT * FROM dual;
```



USER_ID	PROFILE_VERSION	PROFILE_PICTURE	BIO	UPDATED_AT
1	1	https://afrapic.com/afra.jpg	I love coding but I suck heh!	01-JAN-25 11.50.00.000000 AM
2	1	https://chanpic.com/chan.jpg	<Photoholic>	01-JAN-25 08.01.05.000000 PM
7	1	https://apnmaz.com/apmaz.jpg	Vlogger	13-JUL-25 05.17.10.111000 PM
4	1	https://ruhaspfp.com/ruhas.jpg	Lovergirl!	18-MAR-25 08.01.05.000000 PM
5	1	https://arpon.com/arpon.jpg	FOODIE!	06-JAN-25 08.01.05.000000 PM
8	1	https://redoyya.com/pic.jpg	Gamer	13-JUL-25 05.17.10.111000 PM
9	1	https://shirbaddie.com/profile.jpg	Fashionista	13-JUL-25 05.17.10.111000 PM
10	1	https://adibanuz.com/adi.jpg	Food Blogger	13-JUL-25 05.17.10.111000 PM

Figure 3: User Profile table after inserting sample data.

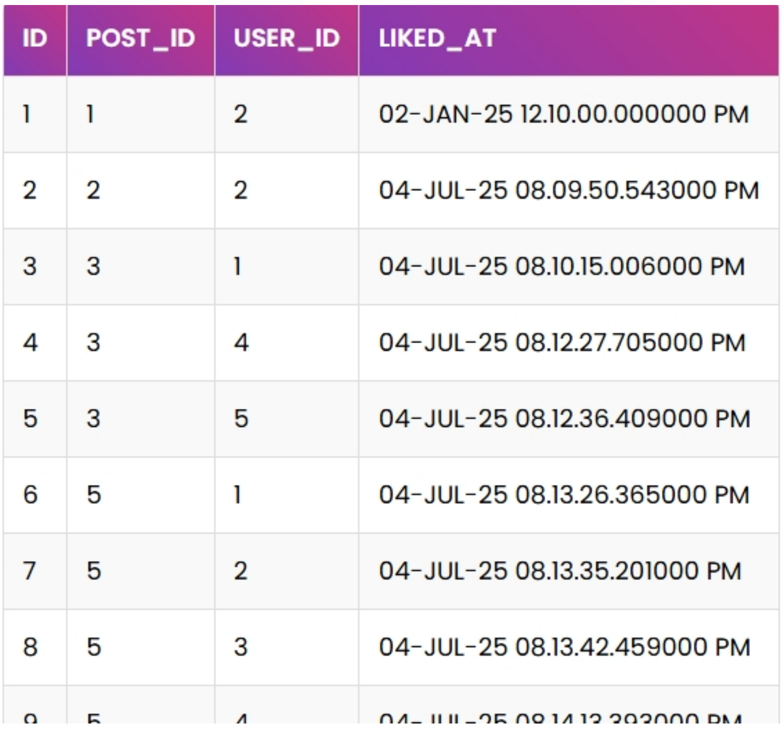
```sql
INSERT ALL
    INTO posts (id, user_id, title, body, status, created_at)
    VALUES (1, 1, 'First␣Post', 'Hello␣world!', 'published', CURRENT_TIMESTAMP)
    INTO posts (id, user_id, title, body, status, created_at)
    VALUES (2, 5, 'Todays␣meal:)', 'Burger␣and␣fries!!!', 'published', CURRENT_TIMESTAMP)
    INTO posts (id, user_id, title, body, status, created_at)
    VALUES (3, 1, 'A␣Day␣at␣the␣Beach', 'Enjoying␣the␣sunshine␣and␣waves␣at␣Cox''s␣Bazar.␣#blessed', 'published', CURRENT_TIMESTAMP)
    INTO posts (id, user_id, title, body, status, created_at)
    VALUES (4, 3, 'Uni␣Orientation', 'Had␣an␣amazing␣day!␣Met␣new␣people!', 'published', CURRENT_TIMESTAMP)
    INTO posts (id, user_id, title, body, status, created_at)
    VALUES (5, 2, 'Study␣Vibes', 'Grinding␣for␣finals!', 'published', CURRENT_TIMESTAMP)
    INTO posts (id, user_id, title, body, status, created_at)
    VALUES (6, 4, 'Coffee␣Time', 'Nothing␣beats␣morning␣coffee␣', 'published', CURRENT_TIMESTAMP)
    INTO posts (id, user_id, title, body, status, created_at)
    VALUES (7, 3, 'DUmp␣Monthly', 'Chaotic␣month␣hehe', 'published', CURRENT_TIMESTAMP)
    INTO posts (id, user_id, title, body, status, created_at)
    VALUES (8, 3, 'Bandorban␣is␣calling!', 'so␣much␣to␣explore␣still!', 'published', CURRENT_TIMESTAMP)
    INTO posts (id, user_id, title, body, status, created_at)
    VALUES (9, 9, 'Outing:/', 'At␣Oro,␣food␣is␣so␣good!', 'published', CURRENT_TIMESTAMP)
    INTO posts (id, user_id, title, body, status, created_at)
    VALUES (10, 8, 'New␣K-pop␣Drop', 'Stream␣the␣new␣MV!!', 'published', CURRENT_TIMESTAMP)
    INTO posts (id, user_id, title, body, status, created_at)
    VALUES (11, 8, 'Lazy␣Sunday', 'Just␣binge-watching␣Netflix␣all␣day.', 'draft', CURRENT_TIMESTAMP)
    INTO posts (id, user_id, title, body, status, created_at)
    VALUES (12, 8, 'Study␣Time', 'Grinding␣DBMS␣all␣night', 'published', CURRENT_TIMESTAMP)
    INTO posts (id, user_id, title, body, status, created_at)
    VALUES (13, 7, 'Chittagong␣Vibes', 'City␣looks␣peaceful␣today.', 'published', CURRENT_TIMESTAMP)
    INTO posts (id, user_id, title, body, status, created_at)
    VALUES (14, 7, 'Need␣Food␣Recs!', 'What''s␣your␣fav␣place␣for␣chaap?', 'draft', CURRENT_TIMESTAMP)
SELECT * FROM dual;
```



ID	USER_ID	TITLE	BODY	STATUS	CREATED_AT
1	1	First Post	Hello world!	published	02-JAN-25 12.00.00.000000 PM
2	5	Todays meal:)	Burger and fries!!!	published	04-JUL-25 07.28.12.186000 PM
3	1	A Day at the Beach	Enjoying the sunshine and waves at Cox?s Bazar. #blessed	published	04-JUL-25 07.28.39.859000 PM
4	3	Uni Orientation	Had an amazing day! Met new people!	published	02-JUN-25 07.00.00.000000 PM
5	2	Study Vibes	Grinding for finals! ????????	published	05-JUL-25 01.33.01.292000 AM
6	4	Coffee Time	Nothing beats morning coffee ???	published	05-JUL-25 01.33.20.172000 AM
7	3	DUmp Monthly	Chaotic month hehe	published	01-AUG-25 07.00.00.000000 PM
8	3	Bandorban is calling!	so much to explore still!	published	04-JUL-25 07.35.18.139000 PM
9	9	Outing:/	At Oro, food is so good!	published	13-JUL-25 05.26.19.259000 PM
10	8	New K-pop Drop	Stream the new MV!!	published	13-JUL-25 05.26.19.259000 PM
11	8	Lazy Sunday	Just binge-watching Netflix all day.	draft	13-JUL-25 05.26.19.259000 PM
12	8	Study Time	Grinding DBMS all night ????	published	13-JUL-25 05.26.19.259000 PM
13	7	Chittagong Vibes	City looks peaceful today.	published	13-JUL-25 05.26.19.259000 PM
14	7	Need Food Recs!	What???s your fav place for chaap?	draft	13-JUL-25 05.26.19.259000 PM

Figure 4: Posts table after inserting sample data.

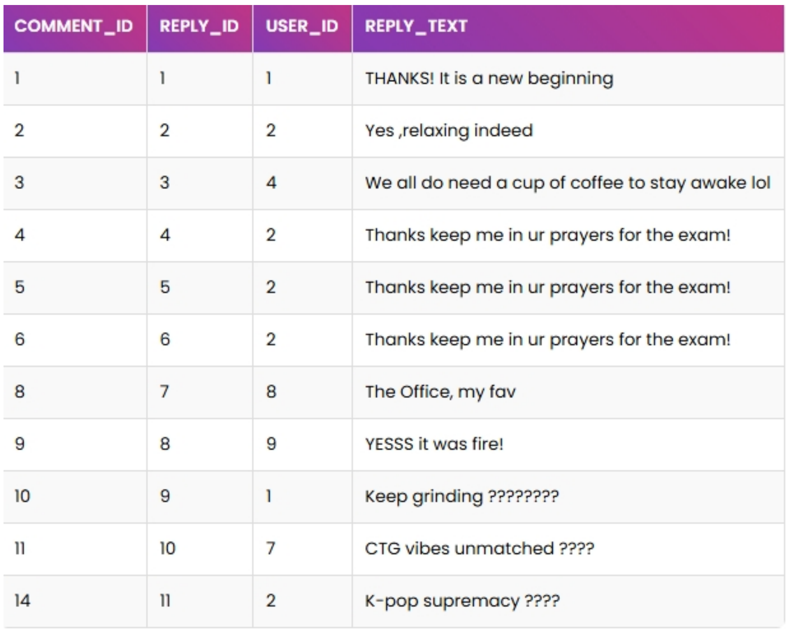
```
INSERT ALL
    INTO likes (id, post_id, user_id, liked_at)
    VALUES (1, 1, 2, CURRENT_TIMESTAMP)
    INTO likes (id, post_id, user_id, liked_at)
    VALUES (2, 2, 2, CURRENT_TIMESTAMP)
    INTO likes (id, post_id, user_id, liked_at)
    VALUES (3, 3, 1, CURRENT_TIMESTAMP)
    INTO likes (id, post_id, user_id, liked_at)
    VALUES (4, 3, 4, CURRENT_TIMESTAMP)
    INTO likes (id, post_id, user_id, liked_at)
    VALUES (5, 3, 5, CURRENT_TIMESTAMP)
    INTO likes (id, post_id, user_id, liked_at)
    VALUES (6, 5, 1, CURRENT_TIMESTAMP)
    INTO likes (id, post_id, user_id, liked_at)
    VALUES (7, 5, 2, CURRENT_TIMESTAMP)
    INTO likes (id, post_id, user_id, liked_at)
    VALUES (8, 5, 3, CURRENT_TIMESTAMP)
    INTO likes (id, post_id, user_id, liked_at)
    VALUES (9, 5, 4, CURRENT_TIMESTAMP)
    INTO likes (id, post_id, user_id, liked_at)
    VALUES (10, 5, 5, CURRENT_TIMESTAMP)
    INTO likes (id, post_id, user_id, liked_at)
    VALUES (11, 4, 2, CURRENT_TIMESTAMP)
    INTO likes (id, post_id, user_id, liked_at)
    VALUES (12, 6, 2, CURRENT_TIMESTAMP)
    INTO likes (id, post_id, user_id, liked_at)
    VALUES (13, 9, 8, CURRENT_TIMESTAMP)
    INTO likes (id, post_id, user_id, liked_at)
    VALUES (14, 10, 7, CURRENT_TIMESTAMP)
    INTO likes (id, post_id, user_id, liked_at)
    VALUES (15, 11, 9, CURRENT_TIMESTAMP)
    INTO likes (id, post_id, user_id, liked_at)
    VALUES (16, 12, 1, CURRENT_TIMESTAMP)
    INTO likes (id, post_id, user_id, liked_at)
    VALUES (17, 13, 2, CURRENT_TIMESTAMP)
    INTO likes (id, post_id, user_id, liked_at)
    VALUES (18, 14, 8, CURRENT_TIMESTAMP)
SELECT * FROM dual;
```



ID	POST_ID	USER_ID	LIKED_AT
1	1	2	02-JAN-25 12.10.00.000000 PM
2	2	2	04-JUL-25 08.09.50.543000 PM
3	3	1	04-JUL-25 08.10.15.006000 PM
4	3	4	04-JUL-25 08.12.27.705000 PM
5	3	5	04-JUL-25 08.12.36.409000 PM
6	5	1	04-JUL-25 08.13.26.365000 PM
7	5	2	04-JUL-25 08.13.35.201000 PM
8	5	3	04-JUL-25 08.13.42.459000 PM
9	5	4	04-JUL-25 08.14.13.393000 PM

Figure 5: Likes table after inserting sample data.

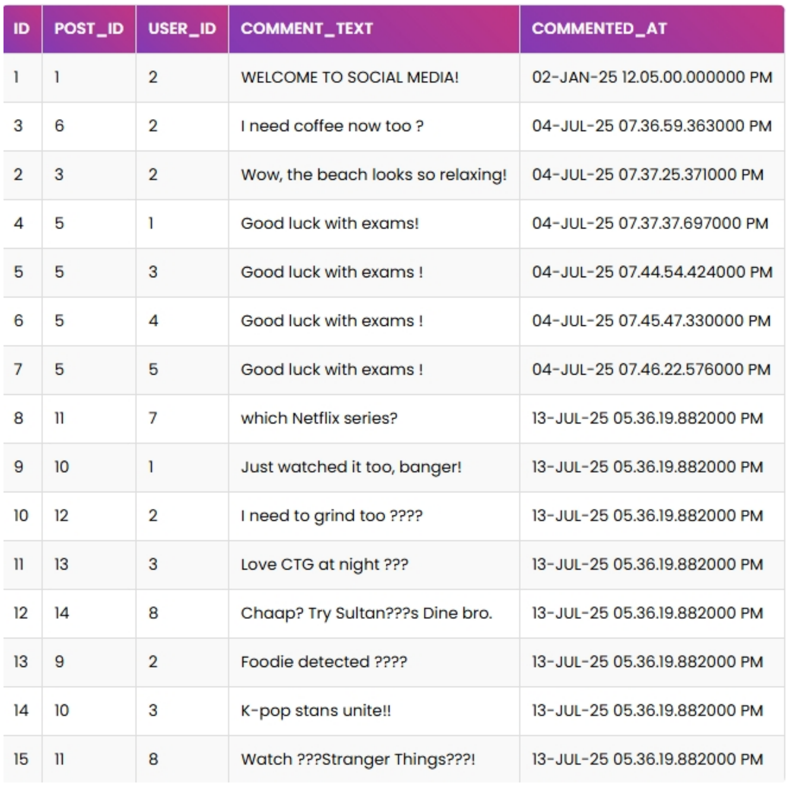
```sql
INSERT ALL
    INTO comments (id, post_id, user_id, comment_text, commented_at) VALUES (1, 1, 2, 'WELCOME␣TO␣SOCIAL␣MEDIA!', CURRENT_TIMESTAMP)
    INTO comments (id, post_id, user_id, comment_text, commented_at) VALUES (3, 6, 2, 'I␣need␣coffee␣now␣too␣?', CURRENT_TIMESTAMP)
    INTO comments (id, post_id, user_id, comment_text, commented_at) VALUES (2, 3, 2, 'Wow,␣the␣beach␣looks␣so␣relaxing!', CURRENT_TIMESTAMP)
    INTO comments (id, post_id, user_id, comment_text, commented_at) VALUES (4, 5, 1, 'Good␣luck␣with␣exams!', CURRENT_TIMESTAMP)
    INTO comments (id, post_id, user_id, comment_text, commented_at) VALUES (5, 5, 3, 'Good␣luck␣with␣exams␣!', CURRENT_TIMESTAMP)
    INTO comments (id, post_id, user_id, comment_text, commented_at) VALUES (6, 5, 4, 'Good␣luck␣with␣exams␣!', CURRENT_TIMESTAMP)
    INTO comments (id, post_id, user_id, comment_text, commented_at) VALUES (7, 5, 5, 'Good␣luck␣with␣exams␣!', CURRENT_TIMESTAMP)
    INTO comments (id, post_id, user_id, comment_text, commented_at) VALUES (8, 11, 7, 'which␣Netflix␣series?', CURRENT_TIMESTAMP)
    INTO comments (id, post_id, user_id, comment_text, commented_at) VALUES (9, 10, 1, 'Just␣watched␣it␣too,␣banger!', CURRENT_TIMESTAMP)
    INTO comments (id, post_id, user_id, comment_text, commented_at) VALUES (10, 12, 2, 'I␣need␣to␣grind␣too␣????', CURRENT_TIMESTAMP)
    INTO comments (id, post_id, user_id, comment_text, commented_at) VALUES (11, 13, 3, 'Love␣CTG␣at␣night␣???', CURRENT_TIMESTAMP)
    INTO comments (id, post_id, user_id, comment_text, commented_at) VALUES (12, 14, 8, 'Chaap...Try␣Sultans␣Dine␣bro', CURRENT_TIMESTAMP)
    INTO comments (id, post_id, user_id, comment_text, commented_at) VALUES (13, 9, 2, 'Foodie␣detected␣????', CURRENT_TIMESTAMP)
    INTO comments (id, post_id, user_id, comment_text, commented_at) VALUES (14, 10, 3, 'Kpop␣stans␣unite', CURRENT_TIMESTAMP)
    INTO comments (id, post_id, user_id, comment_text, commented_at) VALUES (15, 11, 8, 'Watch␣Stranger␣Things', CURRENT_TIMESTAMP)
SELECT * FROM dual;
```



COMMENT_ID	REPLY_ID	USER_ID	REPLY_TEXT
1	1	1	THANKS! It is a new beginning
2	2	2	Yes ,relaxing indeed
3	3	4	We all do need a cup of coffee to stay awake lol
4	4	2	Thanks keep me in ur prayers for the exam!
5	5	2	Thanks keep me in ur prayers for the exam!
6	6	2	Thanks keep me in ur prayers for the exam!
8	7	8	The Office, my fav
9	8	9	YESSS it was fire!
10	9	1	Keep grinding ????????
11	10	7	CTG vibes unmatched ????
14	11	2	K-pop supremacy ????

Figure 6: Comments table after inserting sample data.

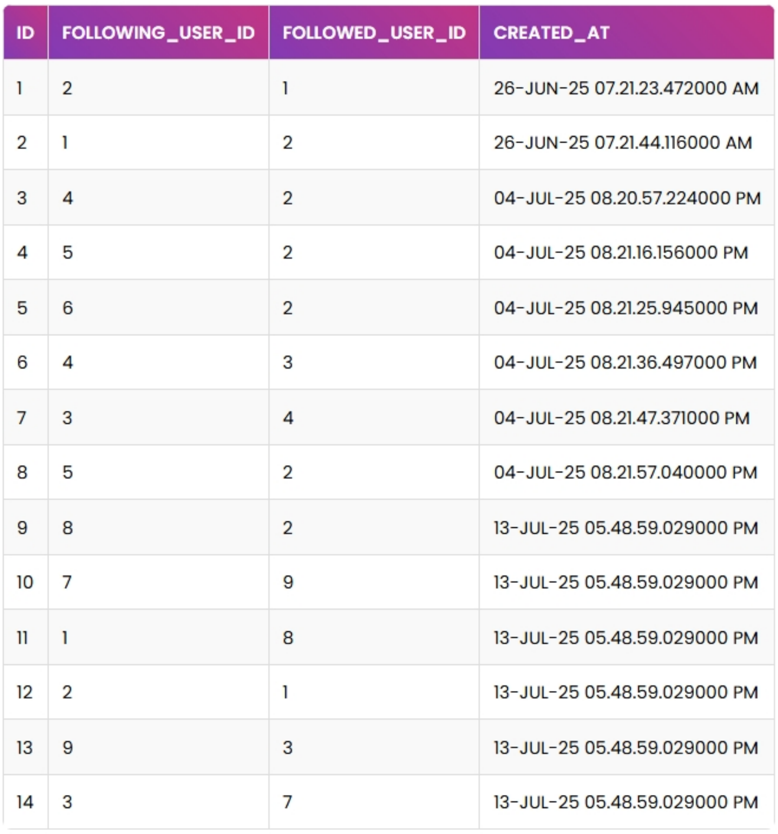
```sql
INSERT ALL
    INTO reply (comment_id, reply_id, user_id, reply_text) VALUES
        (1, 1, 1, 'THANKS! It is a new beginning')
    INTO reply (comment_id, reply_id, user_id, reply_text) VALUES
        (2, 2, 2, 'Yes ,relaxing indeed')
    INTO reply (comment_id, reply_id, user_id, reply_text) VALUES
        (3, 3, 4, 'We all do need a cup of coffee to stay awake 
       lol')
    INTO reply (comment_id, reply_id, user_id, reply_text) VALUES
        (4, 4, 2, 'Thanks keep me in ur prayers for the exam!')
    INTO reply (comment_id, reply_id, user_id, reply_text) VALUES
        (5, 5, 2, 'Thanks keep me in ur prayers for the exam!')
    INTO reply (comment_id, reply_id, user_id, reply_text) VALUES
        (6, 6, 2, 'Thanks keep me in ur prayers for the exam!')
    INTO reply (comment_id, reply_id, user_id, reply_text) VALUES
        (8, 7, 8, 'The Office, my fav')
    INTO reply (comment_id, reply_id, user_id, reply_text) VALUES
        (9, 8, 9, 'YESSS it was fire!')
    INTO reply (comment_id, reply_id, user_id, reply_text) VALUES
        (10, 9, 1, 'Keep grinding ????????')
    INTO reply (comment_id, reply_id, user_id, reply_text) VALUES
        (11, 10, 7, 'CTG vibes unmatched ????')
    INTO reply (comment_id, reply_id, user_id, reply_text) VALUES
        (14, 11, 2, 'K-pop supremacy ????')
SELECT * FROM dual;
```



ID	POST_ID	USER_ID	COMMENT_TEXT	COMMENTED_AT
1	1	2	WELCOME TO SOCIAL MEDIA!	02-JAN-25 12.05.00.000000 PM
3	6	2	I need coffee now too ?	04-JUL-25 07.36.59.363000 PM
2	3	2	Wow, the beach looks so relaxing!	04-JUL-25 07.37.25.371000 PM
4	5	1	Good luck with exams!	04-JUL-25 07.37.37.697000 PM
5	5	3	Good luck with exams !	04-JUL-25 07.44.54.424000 PM
6	5	4	Good luck with exams !	04-JUL-25 07.45.47.330000 PM
7	5	5	Good luck with exams !	04-JUL-25 07.46.22.576000 PM
8	11	7	which Netflix series?	13-JUL-25 05.36.19.882000 PM
9	10	1	Just watched it too, banger!	13-JUL-25 05.36.19.882000 PM
10	12	2	I need to grind too ????	13-JUL-25 05.36.19.882000 PM
11	13	3	Love CTG at night ???	13-JUL-25 05.36.19.882000 PM
12	14	8	Chaap? Try Sultan???s Dine bro.	13-JUL-25 05.36.19.882000 PM
13	9	2	Foodie detected ????	13-JUL-25 05.36.19.882000 PM
14	10	3	K-pop stans unite!!	13-JUL-25 05.36.19.882000 PM
15	11	8	Watch ???Stranger Things???!	13-JUL-25 05.36.19.882000 PM

Figure 7: Reply table after inserting sample data.

```
INSERT ALL
    INTO follows (id, following_user_id, followed_user_id,
       created_at)
    VALUES (1, 2, 1, CURRENT_TIMESTAMP)
    INTO follows (id, following_user_id, followed_user_id,
       created_at)
    VALUES (2, 1, 2, CURRENT_TIMESTAMP)
    INTO follows (id, following_user_id, followed_user_id,
       created_at)
    VALUES (3, 4, 2, CURRENT_TIMESTAMP)
    INTO follows (id, following_user_id, followed_user_id,
       created_at)
    VALUES (4, 5, 2, CURRENT_TIMESTAMP)
    INTO follows (id, following_user_id, followed_user_id,
       created_at)
    VALUES (5, 6, 2, CURRENT_TIMESTAMP)
    INTO follows (id, following_user_id, followed_user_id,
       created_at)
    VALUES (6, 4, 3, CURRENT_TIMESTAMP)
    INTO follows (id, following_user_id, followed_user_id,
       created_at)
    VALUES (7, 3, 4, CURRENT_TIMESTAMP)
    INTO follows (id, following_user_id, followed_user_id,
       created_at)
    VALUES (8, 5, 2, CURRENT_TIMESTAMP)
    INTO follows (id, following_user_id, followed_user_id,
       created_at)
    VALUES (9, 8, 2, CURRENT_TIMESTAMP)
    INTO follows (id, following_user_id, followed_user_id,
       created_at)
    VALUES (10, 7, 9, CURRENT_TIMESTAMP)
    INTO follows (id, following_user_id, followed_user_id,
       created_at)
    VALUES (11, 1, 8, CURRENT_TIMESTAMP)
    INTO follows (id, following_user_id, followed_user_id,
       created_at)
    VALUES (12, 2, 1, CURRENT_TIMESTAMP)
    INTO follows (id, following_user_id, followed_user_id,
       created_at)
    VALUES (13, 9, 3, CURRENT_TIMESTAMP)
    INTO follows (id, following_user_id, followed_user_id,
       created_at)
    VALUES (14, 3, 7, CURRENT_TIMESTAMP)
SELECT * FROM dual;
```



ID	FOLLOWING_USER_ID	FOLLOWED_USER_ID	CREATED_AT
1	2	1	26-JUN-25 07.21.23.472000 AM
2	1	2	26-JUN-25 07.21.44.116000 AM
3	4	2	04-JUL-25 08.20.57.224000 PM
4	5	2	04-JUL-25 08.21.16.156000 PM
5	6	2	04-JUL-25 08.21.25.945000 PM
6	4	3	04-JUL-25 08.21.36.497000 PM
7	3	4	04-JUL-25 08.21.47.371000 PM
8	5	2	04-JUL-25 08.21.57.040000 PM
9	8	2	13-JUL-25 05.48.59.029000 PM
10	7	9	13-JUL-25 05.48.59.029000 PM
11	1	8	13-JUL-25 05.48.59.029000 PM
12	2	1	13-JUL-25 05.48.59.029000 PM
13	9	3	13-JUL-25 05.48.59.029000 PM
14	3	7	13-JUL-25 05.48.59.029000 PM

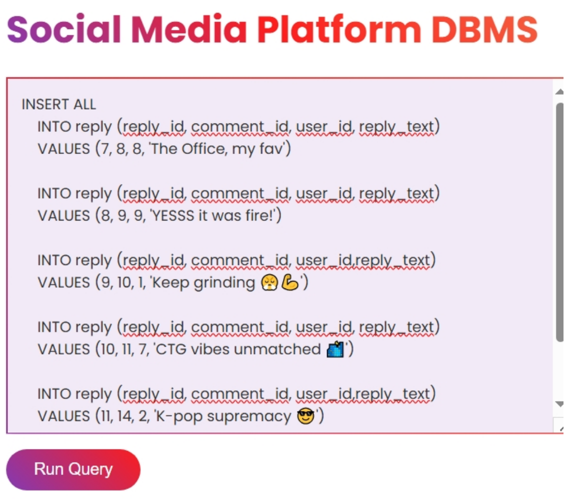
Figure 8: Follows table after inserting sample data.

8 User Interface Design

The Social Media Platform DBMS features a PHP-based web interface for interacting with the Oracle XE 10g database. The interface, styled with an Instagram-themed design, includes two main components:

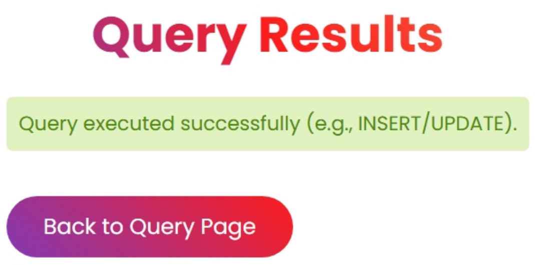
- **Query Input Page** (`index.php`): Users enter SQL queries in a textarea, with a gradient header and a submit button for a visually appealing, user-friendly experience.
- **Query Results Page** (`results.php`): Displays query results in a colorful HTML table, shows the executed query, and includes a "Back to Query Page" button for navigation.

Screenshots of the interface are provided below to illustrate the design and functionality.



```
INSERT ALL
    INTO reply (reply_id, comment_id, user_id, reply_text)
    VALUES (7, 8, 8, 'The Office, my fav')

    INTO reply (reply_id, comment_id, user_id, reply_text)
    VALUES (8, 9, 9, 'YESSS it was fire!')

    INTO reply (reply_id, comment_id, user_id,reply_text)
    VALUES (9, 10, 1, 'Keep grinding 😤💪')

    INTO reply (reply_id, comment_id, user_id, reply_text)
    VALUES (10, 11, 7, 'CTG vibes unmatched 🏙')

    INTO reply (reply_id, comment_id, user_id,reply_text)
    VALUES (11, 14, 2, 'K-pop supremacy 😎')
```

Figure 9: Query Input Page (`index.php`) showing the Instagram-themed UI.



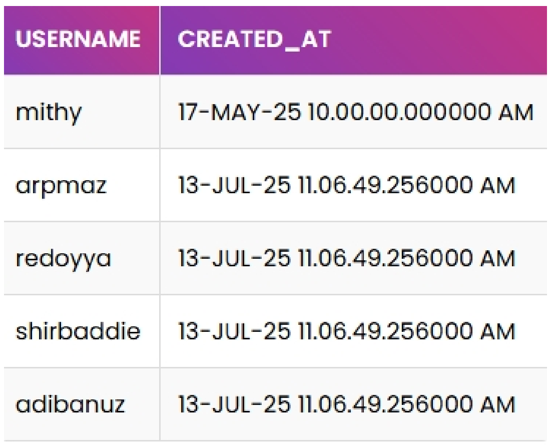
Query executed successfully (e.g., INSERT/UPDATE).

Figure 10: Query Results Page (`results.php`) displaying results in a colorful table.

9 DML Queries

9.1 Single Table Queries

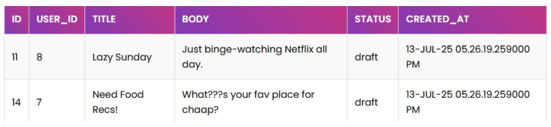
1. `SELECT username ,created_at FROM users WHERE created_at > TO_TIMESTAMP('2025-05-01','YYYY-MM-DD');`



USERNAME	CREATED_AT
mithy	17-MAY-25 10.00.00.000000 AM
arpmaz	13-JUL-25 11.06.49.256000 AM
redoyya	13-JUL-25 11.06.49.256000 AM
shirbaddie	13-JUL-25 11.06.49.256000 AM
adibanuz	13-JUL-25 11.06.49.256000 AM

Figure 11: Lists users created after May 1, 2025

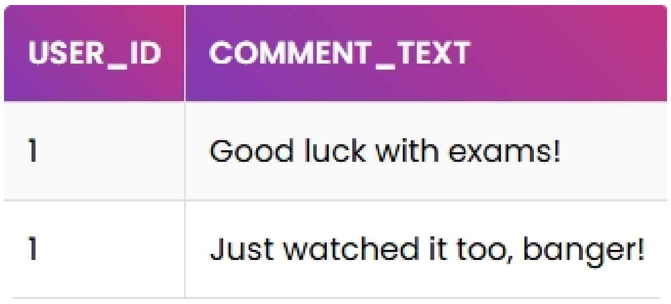
2. `SELECT * FROM posts WHERE status = ’draft’;`



ID	USER_ID	TITLE	BODY	STATUS	CREATED_AT
11	8	Lazy Sunday	Just binge-watching Netflix all day.	draft	13-JUL-25 05.26.19.259000 PM
14	7	Need Food Recs!	What???s your fav place for chaap?	draft	13-JUL-25 05.26.19.259000 PM

Figure 12: Finds posts that are drafts

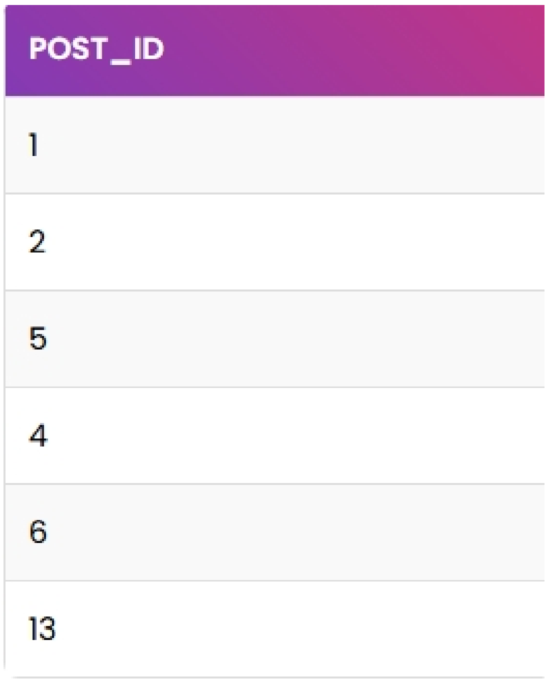
3. `SELECT user_id, comment_text FROM comments WHERE user_id = 1;`



USER_ID	COMMENT_TEXT
1	Good luck with exams!
1	Just watched it too, banger!

Figure 13: Shows comments by user with ID 1

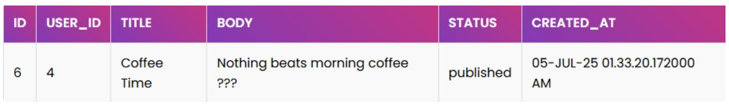
4. `SELECT post_id FROM likes WHERE user_id = 2;`



POST_ID
1
2
5
4
6
13

Figure 14: Lists posts liked by user with ID 2

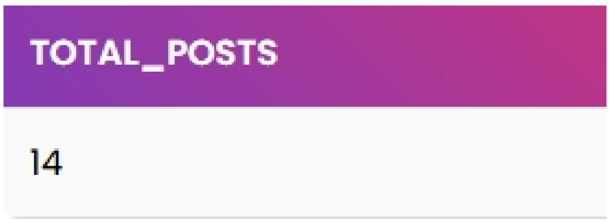
5. `SELECT * FROM posts WHERE LOWER(body) LIKE '%coffee%';`



ID	USER_ID	TITLE	BODY	STATUS	CREATED_AT
6	4	Coffee Time	Nothing beats morning coffee ???	published	05-JUL-25 01.33.20.172000 AM

Figure 15: Finds posts that contain the word 'coffee'

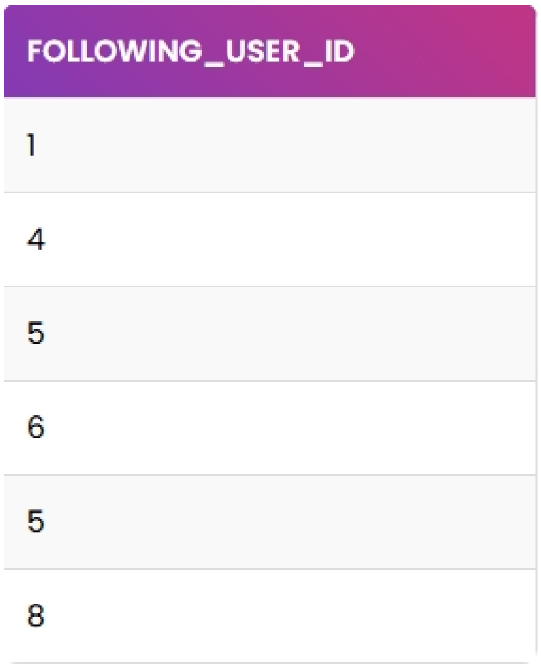
6. `SELECT COUNT(*) AS total_posts FROM posts;`



TOTAL_POSTS
14

Figure 16: Counts the total number of posts

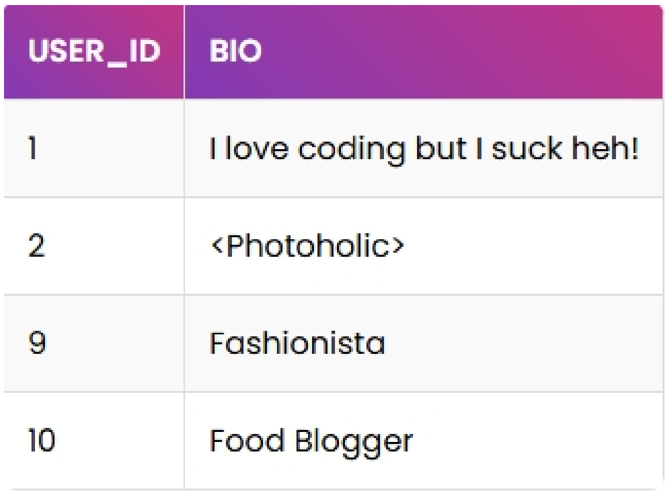
7. `SELECT following_user_id FROM follows WHERE followed_user_id = 2;`



FOLLOWING_USER_ID
1
4
5
6
5
8

Figure 17: Shows users following user with ID 2

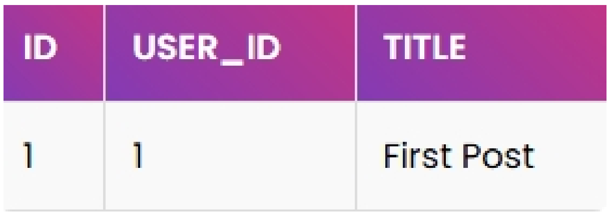
8. `SELECT user_id ,bio FROM user_profile WHERE LENGTH(biography) > 10;`



USER_ID	BIO
1	I love coding but I suck heh!
2	<Photoholic>
9	Fashionista
10	Food Blogger

Figure 18: Shows user bios with more than 10 characters

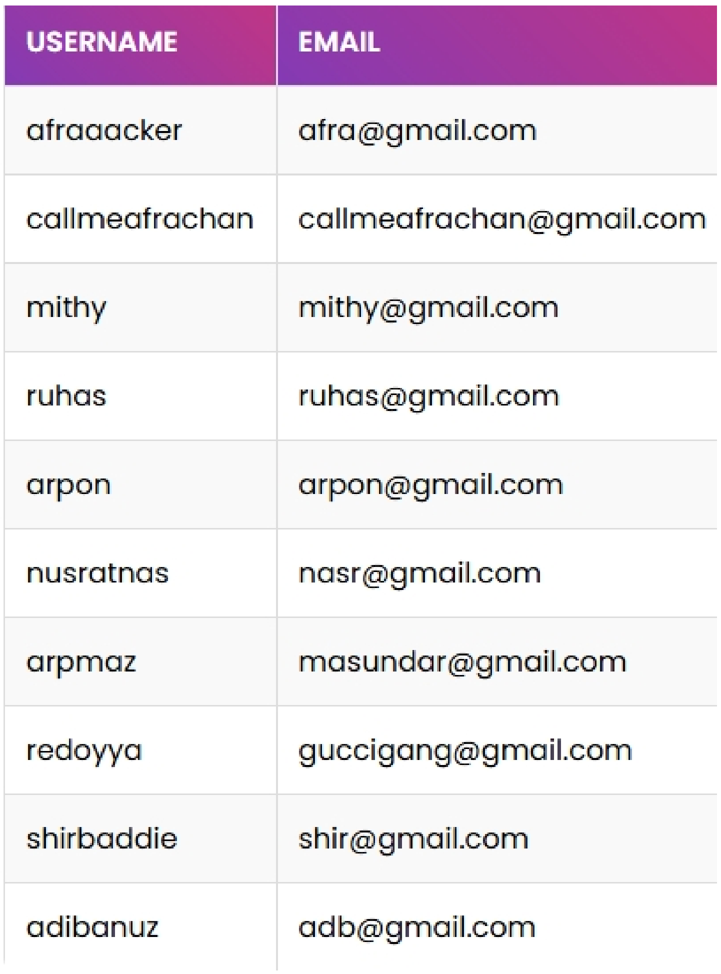
9. `SELECT id,user_id,title FROM posts WHERE post_date < TO_DATE('2025-06-02', 'YYYY-MM-DD');`



ID	USER_ID	TITLE
1	1	First Post

Figure 19: Lists posts created before June 2, 2025

10. `SELECT username,email FROM users;`



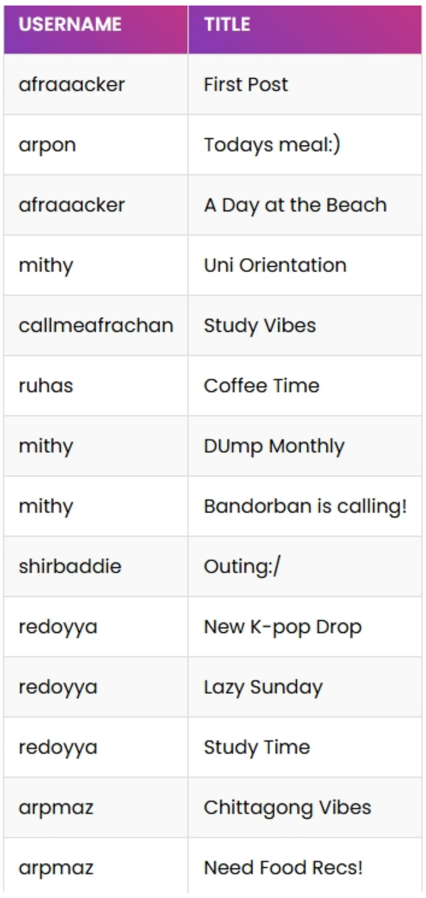
USERNAME	EMAIL
afraaacker	afra@gmail.com
callmeafrachan	callmeafrachan@gmail.com
mithy	mithy@gmail.com
ruhas	ruhas@gmail.com
arpon	arpon@gmail.com
nusratnas	nasr@gmail.com
arpmaz	masundar@gmail.com
redoyya	guccigang@gmail.com
shirbaddie	shir@gmail.com
adibanuz	adb@gmail.com

Figure 20: Find usernames and emails of all users

9.2 Multi-Table Queries

1. **INNER JOIN – Display usernames with their post titles**

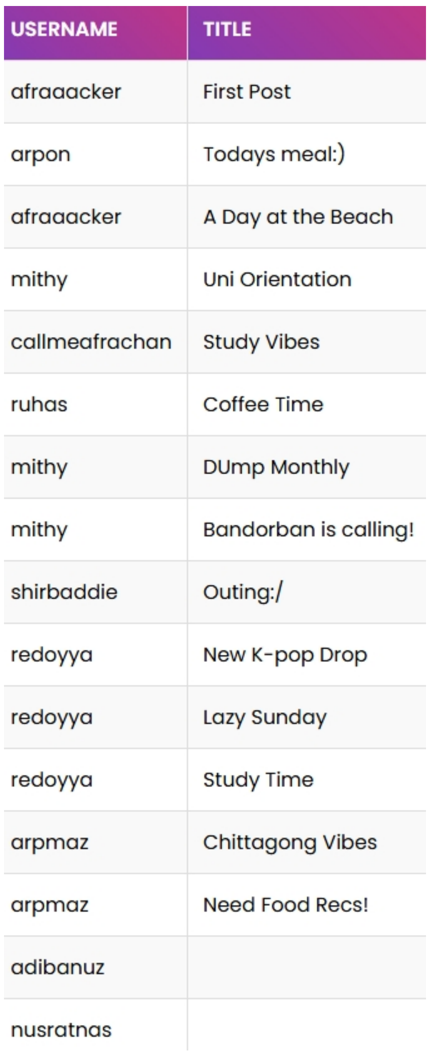
```
SELECT u.username, p.title
FROM users u
INNER JOIN posts p ON u.id = p.user_id;
```



USERNAME	TITLE
afraaacker	First Post
arpon	Todays meal:)
afraaacker	A Day at the Beach
mithy	Uni Orientation
callmeafrachan	Study Vibes
ruhas	Coffee Time
mithy	DUmp Monthly
mithy	Bandorban is calling!
shirbaddie	Outing:/
redoyya	New K-pop Drop
redoyya	Lazy Sunday
redoyya	Study Time
arpmaz	Chittagong Vibes
arpmaz	Need Food Recs!

2. **LEFT JOIN – Show all users and their posts (if any)**

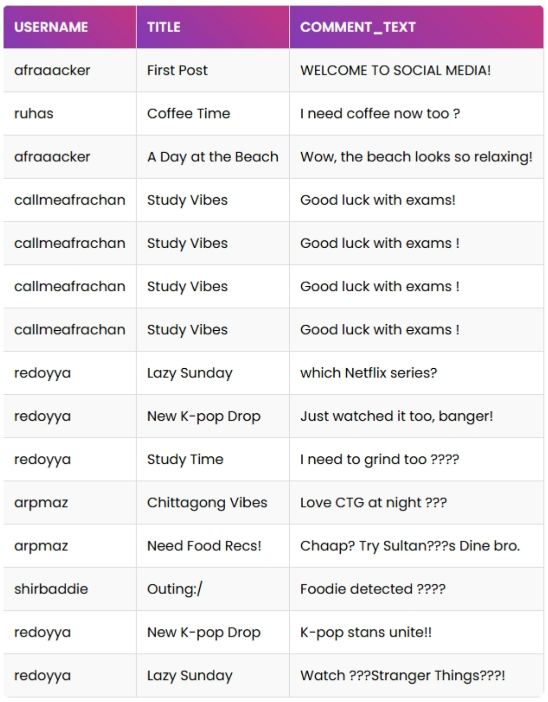
```
SELECT u.username, p.title
FROM users u
LEFT JOIN posts p ON u.id = p.user_id;
```



USERNAME	TITLE
afraaacker	First Post
arpon	Todays meal:)
afraaacker	A Day at the Beach
mithy	Uni Orientation
callmeafrachan	Study Vibes
ruhas	Coffee Time
mithy	DUmp Monthly
mithy	Bandorban is calling!
shirbaddie	Outing:/
redoyya	New K-pop Drop
redoyya	Lazy Sunday
redoyya	Study Time
arpmaz	Chittagong Vibes
arpmaz	Need Food Recs!
adibanuz	
nusratnas	

3. **Connecting 3 tables with JOIN**

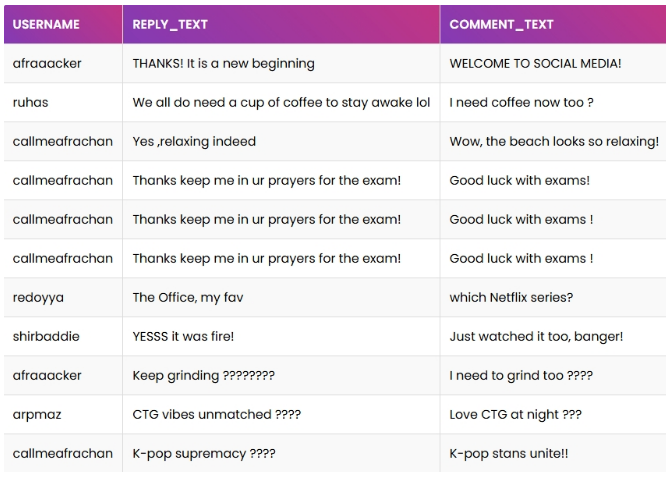
```
ELECT u.username,p.title,c.comment_text
FROM users u
JOIN posts p ON u.id=p.user_id
JOIN comments c ON p.id=c.post_id;
```



USERNAME	TITLE	COMMENT_TEXT
afraaacker	First Post	WELCOME TO SOCIAL MEDIA!
ruhas	Coffee Time	I need coffee now too ?
afraaacker	A Day at the Beach	Wow, the beach looks so relaxing!
callmeafrachan	Study Vibes	Good luck with exams!
callmeafrachan	Study Vibes	Good luck with exams !
callmeafrachan	Study Vibes	Good luck with exams !
callmeafrachan	Study Vibes	Good luck with exams !
redoyya	Lazy Sunday	which Netflix series?
redoyya	New K-pop Drop	Just watched it too, banger!
redoyya	Study Time	I need to grind too ????
arpmaz	Chittagong Vibes	Love CTG at night ???
arpmaz	Need Food Recs!	Chaap? Try Sultan???s Dine bro.
shirbaddie	Outing:/	Foodie detected ????
redoyya	New K-pop Drop	K-pop stans unite!!
redoyya	Lazy Sunday	Watch ???Stranger Things???!

4. **Replies by the users**

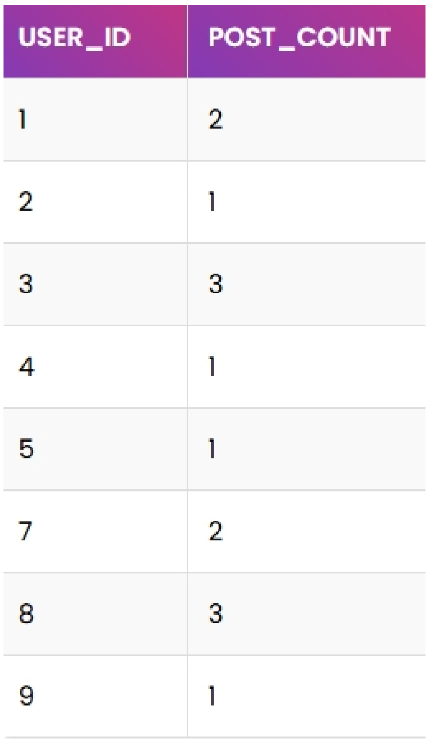
```
SELECT u.username,
r.reply_text,
c.comment_text
FROM reply r
JOIN comments c ON r.comment_id=c.id
JOIN users u ON r.user_id=u.id;
```



USERNAME	REPLY_TEXT	COMMENT_TEXT
afraaacker	THANKS! It is a new beginning	WELCOME TO SOCIAL MEDIA!
ruhas	We all do need a cup of coffee to stay awake lol	I need coffee now too ?
callmeafrachan	Yes ,relaxing indeed	Wow, the beach looks so relaxing!
callmeafrachan	Thanks keep me in ur prayers for the exam!	Good luck with exams!
callmeafrachan	Thanks keep me in ur prayers for the exam!	Good luck with exams !
callmeafrachan	Thanks keep me in ur prayers for the exam!	Good luck with exams !
redoyya	The Office, my fav	which Netflix series?
shirbaddie	YESSS it was fire!	Just watched it too, banger!
afraaacker	Keep grinding ????????	I need to grind too ????
arpmaz	CTG vibes unmatched ????	Love CTG at night ???
callmeafrachan	K-pop supremacy ????	K-pop stans unite!!

5. **Posts count by each user**

```
SELECT user_id, COUNT(*) AS post_count
FROM posts
GROUP BY user_id
HAVING COUNT(*) > 0
ORDER BY user_id ASC;
```

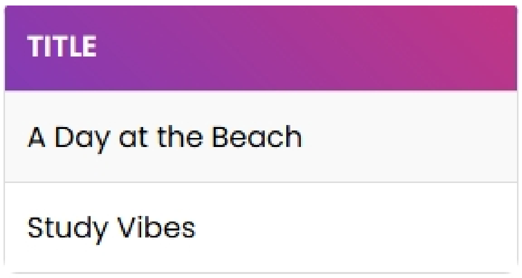


USER_ID	POST_COUNT
1	2
2	1
3	3
4	1
5	1
7	2
8	3
9	1

9.3 Subqueries

1. **Lists titles of posts with more than 2 likes**

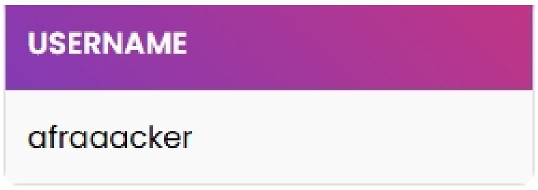
```
SELECT title
FROM posts
WHERE id IN (
    SELECT post_id
    FROM likes
    GROUP BY post_id
    HAVING COUNT(*) > 2
);
```



TITLE
A Day at the Beach
Study Vibes

2. **Finds username of user who posted 'First Post'**

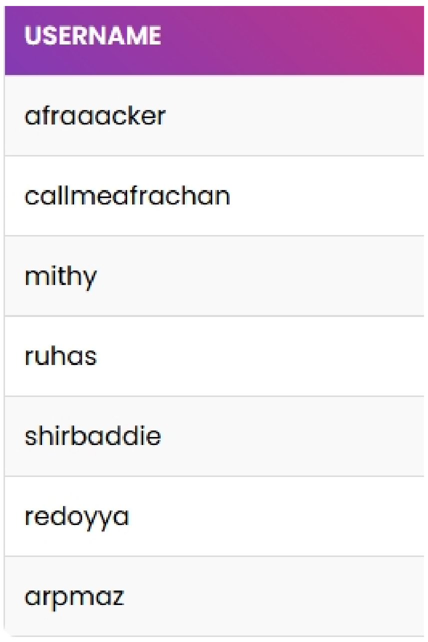
```
SELECT username
FROM users
WHERE id = (
    SELECT user_id
    FROM posts
    WHERE title = 'First Post'
);
```



USERNAME
afraacker

3. **Lists users who are followed (using ANY)**

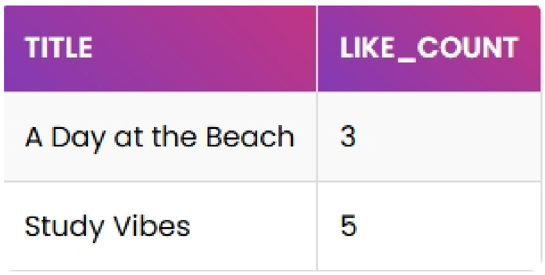
```
SELECT username
FROM users
WHERE id = ANY (
    SELECT followed_user_id
    FROM follows
);
```



USERNAME
afraacker
callmeafrachan
mithy
ruhas
shirbaddie
redoyya
arpmaz

4. **Shows post titles and like counts for posts with more than 2 likes**

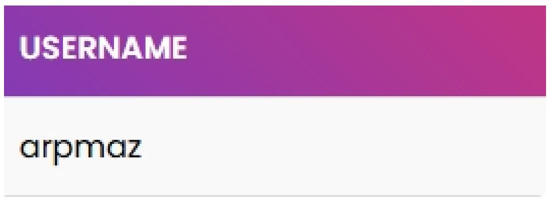
```
SELECT p.title, (
    SELECT COUNT(*)
    FROM likes l
    WHERE l.post_id = p.id
) AS like_count
FROM posts p
WHERE (
    SELECT COUNT(*)
    FROM likes l
    WHERE l.post_id = p.id
) > 2;
```



TITLE	LIKE_COUNT
A Day at the Beach	3
Study Vibes	5

5. **Lists users following 'shirbaddie'**

```
SELECT username
FROM users
WHERE id IN (
    SELECT following_user_id
    FROM follows
    WHERE followed_user_id = (
        SELECT id
        FROM users
        WHERE username = 'shirbaddie'
    )
);
```



USERNAME
arpmaz

10 Challenges Faced

While building the Social Media Platform DBMS, we faced a few important challenges that helped us learn and improve:

- **Foreign Key Constraint Errors**: In Oracle XE 10g, foreign key constraints caused errors when data was inserted before related tables were created. We fixed this by inserting records in the correct order, starting with the `users` table and then adding related data to `posts`, `comments`, and `reply`.
- **Normalization Decisions**: Creating the `user_profile` table to avoid repetition and follow one-to-one relationships took careful planning. We finalized the design in 3NF, which helped remove unnecessary duplication and kept the structure clean.
- **PHP-Oracle Connectivity**: Connecting the PHP files to Oracle XE using OCI8 was difficult at first due to client setup issues. After updating the TNS settings and fixing the connection string, we were able to connect the interface successfully.
- **User Interface Design**: Making a clean, Instagram-themed interface using HTML and CSS was challenging. Different browsers showed styles differently, so we had to test and

adjust the design to make it look consistent everywhere.

These issues helped us understand database rules, structure planning, and how web interfaces work with backend systems.

11 Conclusion

This project provided valuable hands-on experience in designing and implementing a functional database system with a connected web interface. Throughout the development process, we gained a solid understanding of how to normalize data effectively to eliminate redundancy and ensure a structured database design. Working with Oracle XE 10g allowed us to practice creating relational tables, enforcing constraints, and managing data in a real-world context. We also explored PHP integration with Oracle to execute SQL queries through a user-friendly web interface. In addition to technical skills, we learned to troubleshoot practical issues such as connectivity errors and layout design challenges. Overall, this project deepened our understanding of how social media platforms handle data and demonstrated the interplay between front-end and back-end components, preparing us for more advanced systems development in the future.

12 References

The following sources supported the development of the Social Media Platform DBMS, guiding database design, software installation, and PHP-Oracle integration.

1. A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts*, 6th ed. McGraw-Hill, 2010. [Used for normalization and schema design principles.]

2. Oracle Instant Client 19.26 Download, `https://www.oracle.com/database/technologies/instant-client-downloads.html`, accessed July 2025. [Source for Oracle Instant Client to enable OCI8 connectivity.]

3. XAMPP Download, `https://www.apachefriends.org/download.html`, accessed July 2025. [Source for XAMPP to host the PHP web interface.]

4. Oracle Database Express Edition 10g Download, `https://www.oracle.com/database/technologies/xe-downloads.html`, accessed July 2025. [Source for Oracle XE 10g database.]

5. “PHP and Oracle Tutorial,” YouTube, `https://youtu.be/f8NBmBkheC8`, accessed July 2025. [Followed for configuring OCI8 and building the PHP web interface.]

13 Appendix

The GitHub repository hosts all supplementary materials for the Social Media Platform DBMS, including the complete SQL code (`social_media_dbms.sql` with DDL, DML, and queries) and screenshots (`screenshots` folder with web interface and query outputs), complementing Sections 8, 9, and 10.

To set up and run the project:

1. Install Oracle XE 10g with credentials `SYSTEM/afra@XE`.

2. Install XAMPP and place project files in `C:\xampp\htdocs\social_media_dbms`.
3. Install Oracle Instant Client 19.26 and set its path.
4. Enable OCI8 in `php.ini`.
5. Run `social_media_dbms.sql` from GitHub in SQL*Plus.
6. Start XAMPP and access `http://localhost/social_media_dbms/index.php` to test queries.

GitHub Repository: `https://github.com/afrahackerman/DBMS_pro_SocialMediaPlatform`