DATABASE PROJECT

SOCIAL MEDIA MANAGEMENT
RAIHAN HOSSAIN RAKIB
2007005

Overview

Project Name: SOCIAL MEDIA MANAGEMENT

Target Date: May 7, 2024

SUBMITTED TO:	SUBMITTED BY
NAZIA JAHAN KHAN CHOWDHURY ASSISTANT PROFESSOR DEPT OF CSE,KUET	**MD RAIHAN HOSSAIN RAKIB** ROLL:2007005 DEPT:CSE
MD SHAHIDUL SALIM LECTURER DEPT OF CSE, KUET	BATCH:2K20

➢Introduction:

In the contemporary digital landscape, managing social media presence efficiently has become imperative for businesses and individuals alike. To address this need, we embarked on the development of a Social Media Management Database Project. This project aims to provide a comprehensive solution for organizing, scheduling, analyzing, and optimizing social media content across various platforms.

➢Objectives:

- **Centralized Data Management:** Create a centralized database to store all social media-related data including posts, users, engagement metrics, and scheduling information.
- **Content Scheduling:** Implement features to schedule posts for publishing on multiple social media platforms at predetermined times.
- **Analytics Tracking:** Enable tracking and analysis of key performance indicators (KPIs) such as likes, comments, shares, and follower growth.
- **User Engagement Monitoring:** Develop tools to monitor user interactions and engagement with social media content.

➢ Overview of the Database Schema

1. **USERS**:
    - This table stores information about users registered on the platform.
    - Attributes:
        - `USER_ID`: A unique identifier for each user.
        - `USERNAME`: The username chosen by the user.
        - `USER_MOBILE`: The mobile number of the user.
        - `USER_EMAIL`: The email address of the user.
        - `USER_ADDRESS`: The address of the user.

- Primary Key: `USER_ID`

2. **LOGINS**:
    - This table is responsible for storing login credentials for users.
    - Attributes:
        - `LOGIN_ID`: A unique identifier for each login record.
        - `LOGIN_USERNAME`: The username associated with the login.
        - `USER_PASSWORD`: The password associated with the login.
    - Primary Key: `LOGIN_ID`
    - Foreign Key: `LOGIN_ID` references `USER_ID` in the `USERS` table with cascade delete enabled, ensuring that if a user account is deleted, associated login records are also removed.
3. **FRIENDS**:
    - This table represents the connections or friendships between users.
    - Attributes:
        - `FRIEND_ID`: A unique identifier for each friendship record.
        - `USER_ID`: The ID of the user who initiated the friendship request.
    - Primary Key: `FRIEND_ID`
    - Foreign Key: `USER_ID` references `USER_ID` in the `USERS` table with cascade delete enabled, ensuring that if a user account is deleted, associated friendship records are also removed.
4. **POSTS**:
    - This table stores posts created by users.
    - Attributes:
        - `POST_ID`: A unique identifier for each post.
        - `POST_CONTENT`: The content of the post.
        - `POST_DATE`: The date when the post was made.
        - `USER_ID`: The ID of the user who made the post.
    - Primary Key: `POST_ID`
    - Foreign Key: `USER_ID` references `USER_ID` in the `USERS` table with cascade delete enabled, ensuring that if a user account is deleted, associated posts are also removed.
5. **COMMENTS**:
    - This table stores comments made by users on posts.
    - Attributes:
        - `COMMENT_ID`: A unique identifier for each comment.
        - `USER_ID`: The ID of the user who made the comment.
        - `POST_ID`: The ID of the post on which the comment was made.
        - `COMMENT_DATE`: The date when the comment was made.
        - `COMMENT_CONTENT`: The content of the comment.

- Primary Key: `COMMENT_ID`
- Foreign Key: `POST_ID` references `POST_ID` in the `POSTS` table with cascade delete enabled, ensuring that if a post is deleted, associated comments are also removed.

➢ Table Relationships

```
-- TABLE CREATION

CREATE TABLE USERS(
    USER_ID NUMBER(10) NOT NULL,
    USERNAME VARCHAR2(20),
    USER_MOBILE NUMBER(11),
    USER_EMAIL VARCHAR2(50),
    USER_ADDRESS VARCHAR2(50),
    PRIMARY KEY(USER_ID)
);

CREATE TABLE LOGINS(
    LOGIN_ID NUMBER(10) NOT NULL,
    LOGIN_USERNAME VARCHAR2(20),
    USER_PASSWORD VARCHAR2(20),
    PRIMARY KEY(LOGIN_ID),
    FOREIGN KEY(LOGIN_ID) REFERENCES USERS(USER_ID) ON DELETE CASCADE
);
CREATE TABLE FRIENDS (
    FRIEND_ID NUMBER(10) NOT NULL,
    USER_ID NUMBER(10) NOT NULL,
    PRIMARY KEY (FRIEND_ID),
    FOREIGN KEY (USER_ID) REFERENCES USERS(USER_ID) ON DELETE CASCADE
  );
```

```
CREATE TABLE POSTS (
    POST_ID NUMBER(12) NOT NULL,
    POST_CONTENT VARCHAR2(50),
    POST_DATE DATE,
    USER_ID NUMBER(10) NOT NULL,
    PRIMARY KEY(POST_ID),
    FOREIGN KEY(USER_ID) REFERENCES USERS(USER_ID) ON DELETE CASCADE
);
CREATE TABLE COMMENTS (
    COMMENT_ID NUMBER(10) NOT NULL,
    USER_ID NUMBER(10) NOT NULL,
    POST_ID NUMBER(12) NOT NULL,
    COMMENT_DATE DATE,
    COMMENT_CONTENT VARCHAR2(50),
    PRIMARY KEY(COMMENT_ID),
    FOREIGN KEY(POST_ID) REFERENCES POSTS(POST_ID) ON DELETE CASCADE
);
```

➢ Schema Diagram

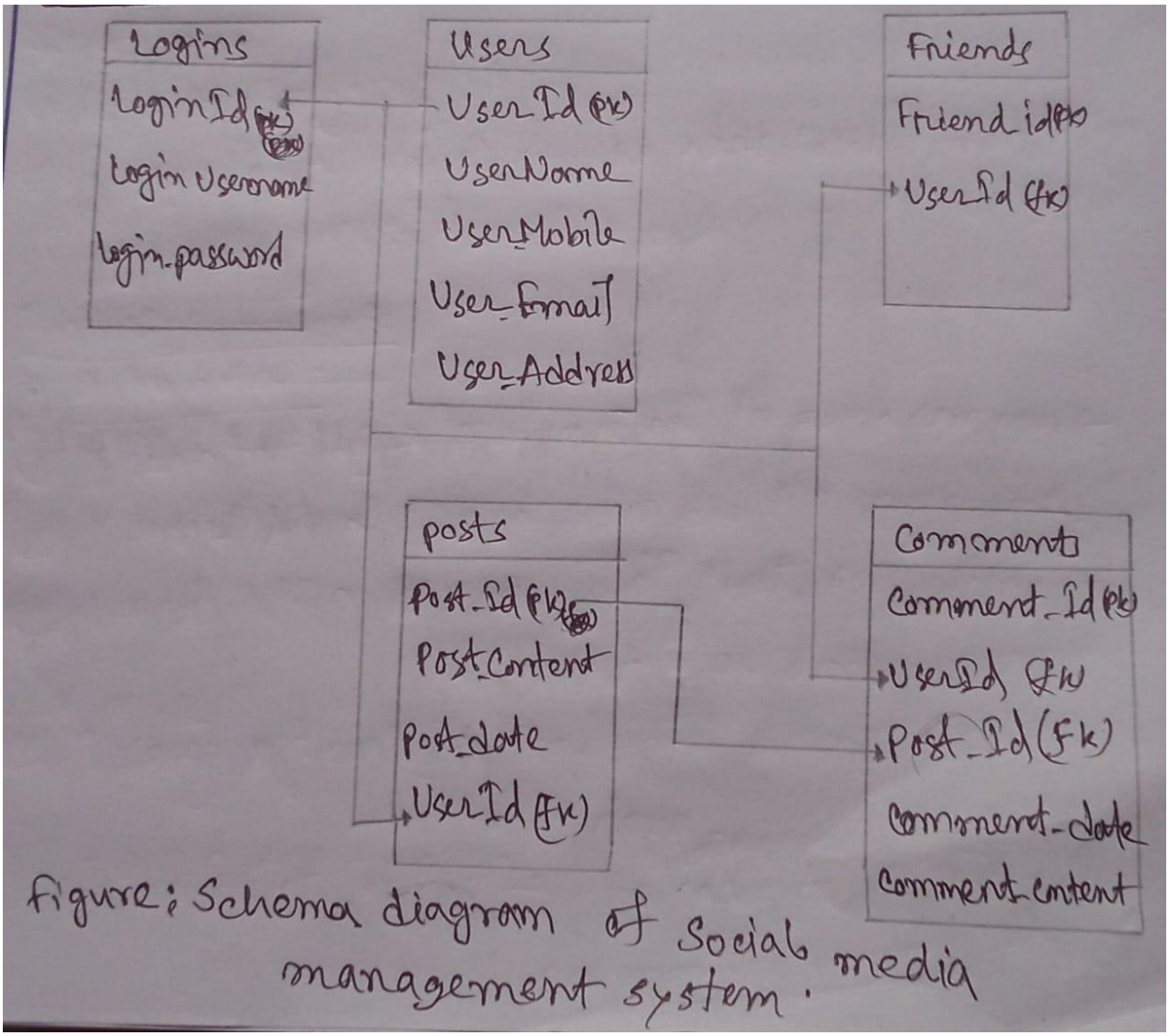

figure: Schema diagram of Social media management system.

➢ Entity-Relationship Diagram (ERD)

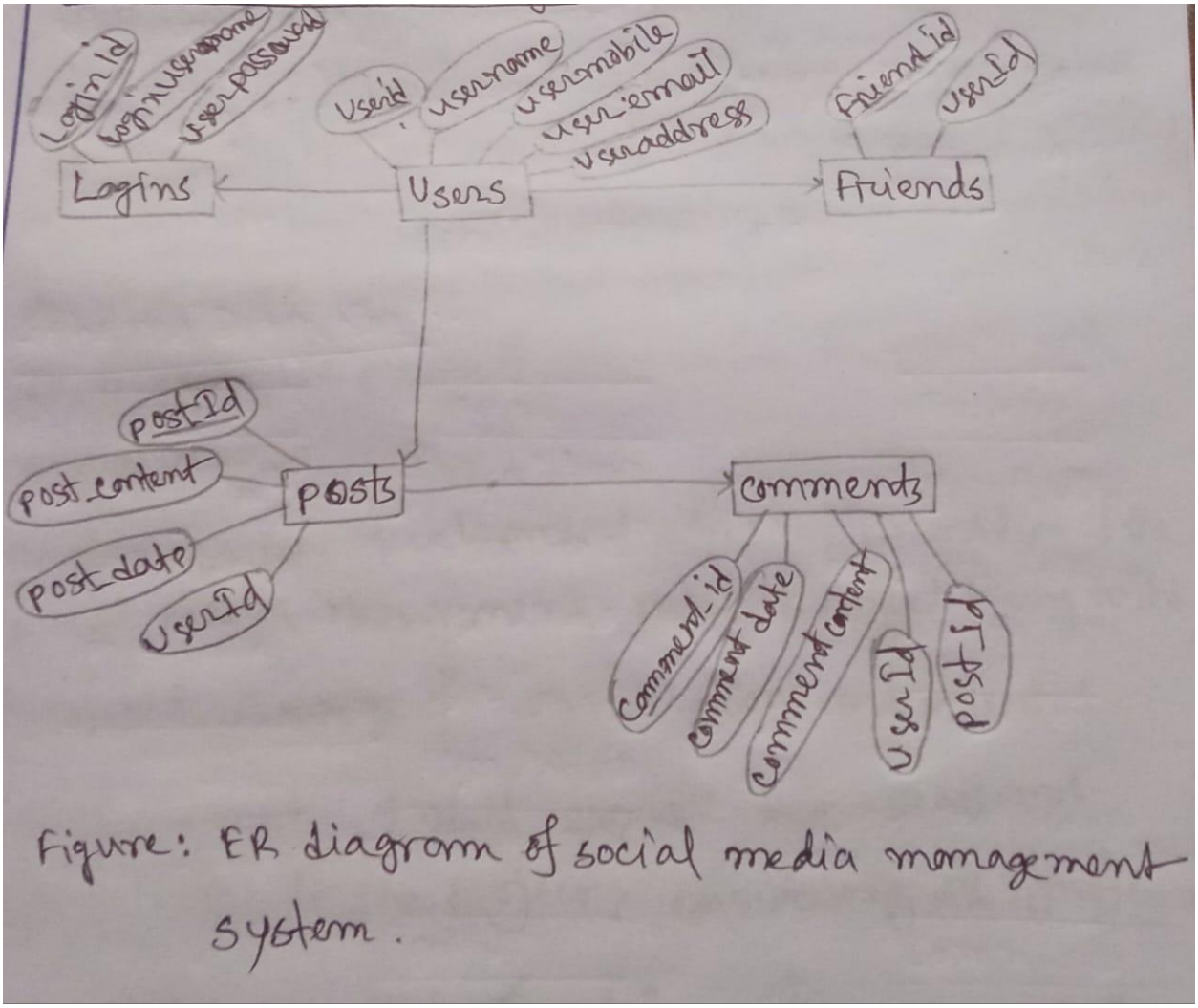

Figure: ER diagram of social media management system.

SQL Queries and Functionality

```
-- CHECKING THE EXISTING THE TABLE IN DATABASE
SELECT TABLE_NAME FROM USER_TABLES;

-- ADD COLUMN
ALTER TABLE LOGINS ADD EMAIL CHAR(20);

-- MODIFY COLUMN
ALTER TABLE LOGINS MODIFY EMAIL VARCHAR(25);

--RENAME COLUMN
ALTER TABLE LOGINS RENAME COLUMN EMAIL TO
USER_EMAIL;

-- DROP COLUMN
ALTER TABLE LOGINS DROP COLUMN USER_EMAIL;

-- DESCRIBE TABLE
DESCRIBE POSTS;
```

```
--SHOW POSTS TABLE

SELECT POST_ID, POST_CONTENT FROM POSTS;

--SHOW USER INFO

SELECT * FROM USERS WHERE USER_ID=102005;

--NESTED SUBQUERY

SELECT * FROM USERS WHERE USER_ID=(SELECT USER_ID
FROM POSTS WHERE POST_ID=98000104);

--UPDATE DATA IN USERS TABLE

UPDATE USERS SET USERNAME='RAKIB' WHERE
USER_ID=102005;

--ADD DATA IN USERS TABLE

INSERT INTO USERS (USER_ID, USERNAME, USER_MOBILE,
USER_EMAIL, USER_ADDRESS) VALUES (102011, 'ARIFUL',
01007232604, 'sifat@gmail.com', 'DKAKA');

--DELETE ROW FROM USERS TABLE

DELETE FROM USERS WHERE USER_ID=102011;
```

--UNION

```
SELECT USERNAME FROM USERS WHERE USERNAME LIKE 'R%' UNION SELECT USERNAME FROM USERS WHERE USERNAME LIKE 'S%';
```

--AGGREGATE FUNCTION

```
SELECT COUNT(*) FROM USERS;

SELECT COUNT(USER_ADDRESS) AS NUMBER_OF_ADDRESS FROM USERS;

SELECT COUNT(DISTINCT USER_ADDRESS) AS NUMBER_OF_ADDRESS FROM USERS;

SELECT MAX(USER_ID) FROM USERS;

SELECT MIN(USER_ID) FROM USERS;
```

--GROUP BY

```
SELECT COUNT(*) FROM USERS GROUP BY USER_ADDRESS;
```

--INTERSECT

```
SELECT * FROM USERS WHERE USERNAME LIKE 'R%' AND USER_ADDRESS LIKE 'D%';
```

--STRING OPERATION

```
SELECT * FROM USERS WHERE USER_ADDRESS LIKE '_____';
```

--NATURAL JOIN

```
SELECT * FROM USERS NATURAL JOIN POSTS;
```

--WITH CLAUSE

```
WITH MAX_ID(VAL) AS (SELECT MAX(USER_ID) FROM
USERS)

SELECT * FROM USERS,MAX_ID WHERE
USERS.USER_ID=MAX_ID.VAL;
```

--VIEW

```
CREATE VIEW USER_DETAILS AS SELECT USER_ID,
USERNAME FROM USERS;
```

-- PL/SQL

```
SET SERVEROUTPUT ON

DECLARE

FRIEND_ID FRIENDS.FRIEND_ID%TYPE;

USER_ID FRIENDS.USER_ID%TYPE;

BEGIN
```

```
SELECT FRIEND_ID,USER_ID INTO FRIEND_ID, USER_ID
FROM FRIENDS WHERE FRIEND_ID=102105;

DBMS_OUTPUT.PUT_LINE('FRIEND_ID: '||FRIEND_ID||'
USER_ID: '||USER_ID);

END;

/
```

--INSERT AND SET DEFAULT VALUE

```
SET SERVEROUTPUT ON

DECLARE

FRIEND_ID FRIENDS.FRIEND_ID%TYPE:=102110;

USER_ID FRIENDS.USER_ID%TYPE:=102001;

BEGIN

INSERT INTO FRIENDS VALUES(FRIEND_ID,USER_ID);

END;

/
```

--ROW TYPE

```
SET SERVEROUTPUT ON

DECLARE

FRIENDS_ROW FRIENDS%ROWTYPE;

BEGIN
```

```
SELECT FRIEND_ID,USER_ID INTO
FRIENDS_ROW.FRIEND_ID,FRIENDS_ROW.USER_ID FROM
FRIENDS WHERE FRIEND_ID=102105;

END;

/
```

--CURSOR AND ROW COUNT

```
SET SERVEROUTPUT ON

DECLARE

CURSOR FRIENDS_CURSOR IS SELECT * FROM FRIENDS;

FRIENDS_ROW FRIENDS%ROWTYPE;

BEGIN

OPEN FRIENDS_CURSOR;

FETCH FRIENDS_CURSOR INTO FRIENDS_ROW.FRIEND_ID
,FRIENDS_ROW.USER_ID;

WHILE FRIENDS_CURSOR%FOUND LOOP

DBMS_OUTPUT.PUT_LINE('FRIEND_ID:
'||FRIENDS_ROW.FRIEND_ID||' USER_ID:
'||FRIENDS_ROW.USER_ID);

DBMS_OUTPUT.PUT_LINE('ROW COUNT:
'||FRIENDS_CURSOR%ROWCOUNT);

FETCH FRIENDS_CURSOR INTO FRIENDS_ROW.FRIEND_ID
,FRIENDS_ROW.USER_ID;

END LOOP;
```

```
CLOSE FRIENDS_CURSOR;
END;
/
```

--TRIGGER

```
CREATE OR REPLACE TRIGGER DELETE_USER_DATA
BEFORE DELETE ON USERS
FOR EACH ROW
BEGIN
   DELETE FROM LOGINS WHERE LOGIN_ID = :OLD.USER_ID;
   DELETE FROM FRIENDS WHERE USER_ID = :OLD.USER_ID;
   DELETE FROM POSTS WHERE USER_ID = :OLD.USER_ID;
   DELETE FROM COMMENTS WHERE USER_ID =
:OLD.USER_ID;
END;
/
```

--PROCEDURE

```
CREATE OR REPLACE PROCEDURE SHOW_USER_POSTS(
   P_USER_ID IN NUMBER,
   P_POST_ID OUT NUMBER,
```

```
   P_POST_CONTENT OUT VARCHAR2,
   P_POST_DATE OUT DATE
)
AS
 T_SHOW CHAR(30);
BEGIN
 T_SHOW := 'FROM PROCEDURE: ';

   FOR POST_REC IN (SELECT POST_ID, POST_CONTENT,
POST_DATE FROM POSTS WHERE USER_ID = P_USER_ID)
   LOOP
      P_POST_ID := POST_REC.POST_ID;
       P_POST_CONTENT :=POST_REC.POST_CONTENT;
      P_POST_DATE := POST_REC.POST_DATE;

      DBMS_OUTPUT.PUT_LINE('POST_ID: '|| P_POST_ID ||'
POST_CONTENT: ' || P_POST_CONTENT || ' POST_DATE: ' ||
P_POST_DATE );
   END LOOP;
   END;
/

SET SERVEROUTPUT ON
```

```
DECLARE
USER_ID USERS.USER_ID%TYPE:=102002;
POST_ID NUMBER;
POST_CONTENT VARCHAR2(50);
POST_DATE DATE;

BEGIN

SHOW_USER_POSTS(USER_ID,POST_ID,POST_CONTENT,POST_DATE);
END;
/
```

--DELETE PROCEDURE

```
DROP PROCEDURE SHOW_USER_POSTS;
```

--FUNCTION

```
SET SERVEROUTPUT ON
CREATE OR REPLACE FUNCTION GET_POST_CONTENT(
P_POST_ID IN NUMBER) RETURN VARCHAR2
AS
   V_POST_CONTENT POSTS.POST_CONTENT%TYPE;
```

```
BEGIN
    SELECT POST_CONTENT INTO V_POST_CONTENT FROM
POSTS WHERE POST_ID = P_POST_ID;

    RETURN V_POST_CONTENT;
END;
/

SET SERVEROUTPUT ON
DECLARE
  POST_ID NUMBER(10) :=98000104;
  POST_CONTENT VARCHAR2(50);
BEGIN
  POST_CONTENT := GET_POST_CONTENT(POST_ID);
   DBMS_OUTPUT.PUT_LINE('POST CONTENT: ' ||
POST_CONTENT);
END;
/

--DELETE FUNCTION
DROP FUNCTION GET_COMMENT_CONTENT;
```

➢Future Enhancements:

Future enhancements to the database project may include integrating additional social media platforms, implementing advanced analytics algorithms, incorporating artificial intelligence for content optimization, and enhancing security features to protect user data and privacy.

➢ Conclusion:

The Social Media Management Database Project aims to streamline and optimize social media management workflows, empowering businesses and individuals to effectively leverage the power of social media for growth and engagement. By providing a centralized platform for managing social media content, scheduling posts, tracking analytics, and monitoring user engagement, this project seeks to enhance digital communication strategies and foster organizational success in the dynamic realm of social media.