VT2024 DT003G Datateknik GR (A), Databaser, 7,5 hp (distans)

Projekt, Hoomam Azrooni, hoaz1600

Beskriv den organisation som kommer att använda din databas.

Denna databas hör till en programmering forumwebbplats. I detta forum kommer användare att kunna skapa egna konton.

Forumet är delat i olika sektioner beroende på diskussionsämnena, varje sektion har ett namn och beskrivning av vad sektionen handlar om. Användarna kan delta i diskussioner och bidra med sina idéer genom att skapa inlägg i forumets diverse sektioner. Förutom inläggets text ska varje inlägg ha en beskrivande rubrik. Datumet när inlägget har publicerats ska också framgå.

Användarna kan kommentera sina egna och andras inlägg. Förutom kommentartexten ska också kommentardatumet visas i samband med texten. På samma sätt som användarna kan kommentera inlägg så kommer dem också att kunna svara på kommentarerna som finns under inläggen och här också visas samma mönster som ovan med svartext och datum.

Denna forumwebbplats har mer funktionalitet än att bara skriva inlägg, kommentarer och svar. Användarna kommer att kunna rösta på de inläggen som de gillar. På samma princip kommer användarna att kunna gilla kommentarer som är skrivna under diverse inlägg på forumet.

Detta forum har två användare som sköter forumet en administratör och en moderator.

Sammanfattning av informationskraven som ska stödjas av databasen.

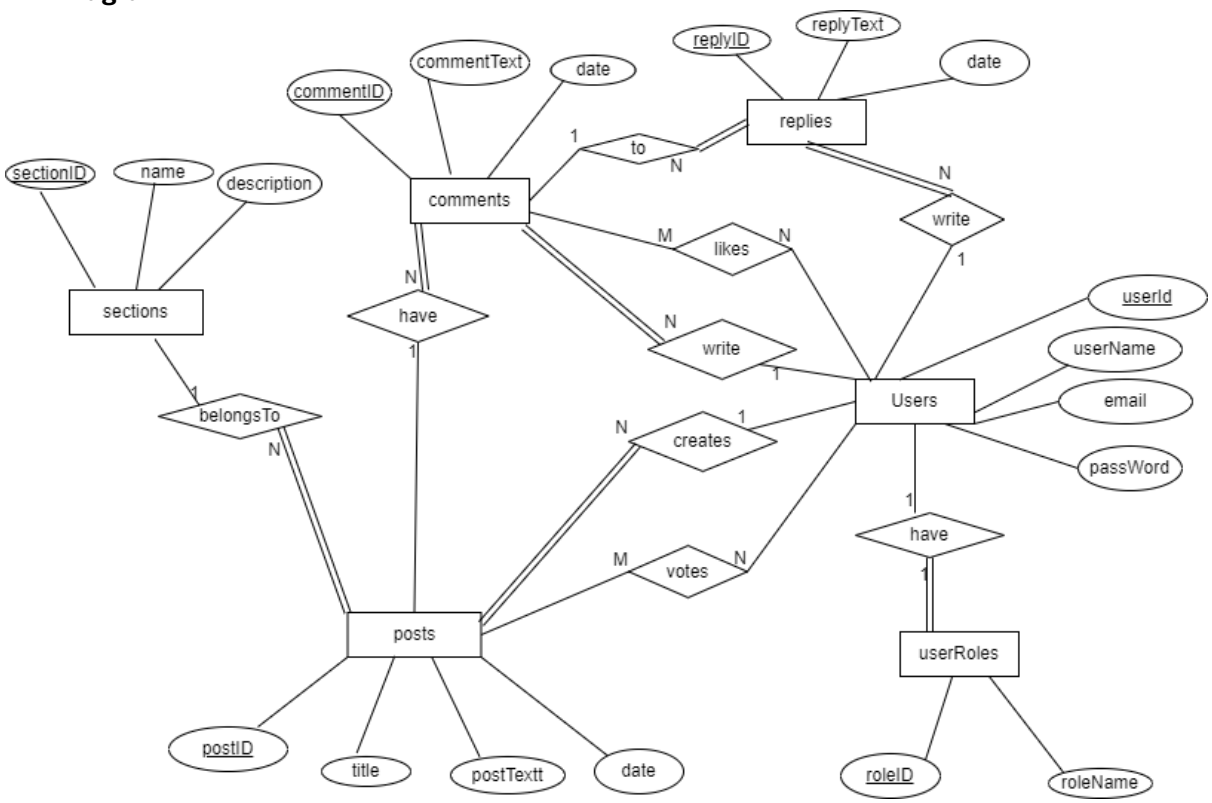
För att skapa ett konto en användare måste ange användarnamn, lösenord och e-postadress. Användarnamnet måste vara unikt dvs att systemet/databasen inte ska acceptera duplicerade användarnamn. E-postadressen ska anges i ett giltigt format.

Själva Inläggen består av två komponenter en rubrik och en text. Datumet när inlägget har skapats ska också vara med och detta (att datumet ska framgå) gäller även kommentarer och svar.

Alla inlägg måste höra till någon sektion.

Inläggen och kommentarerna kan få röster (Votes) och gillas (Likes) av flera användare men var och en kommentar kan få endast en Like per användare och var och ett inlägg kan få endast en röst per användare

ER-Diagram



Översättning av ER-diagrammet:

Users(userID(PK), username (U), passWord, email)

Posts(postID(PK), userID(FK), sectionID(FK), title, postText, date)

Comments(commentID(PK), postID(FK), userID(FK), commentText, date)

Replies(replyID(PK), commentID(FK), userID(FK), replyText, replyDate)

Sections(sectionID(PK), name, description)

userRoles(roleID(PK), userID(FK) , roleName)

likes(commentID(PK)(FK), userID(PK)(FK))

votes(postID(PK)(FK), userID(PK)(FK))

Databas implementering:

Databasen skapades med hjälp av DBMS mariaDB. Databasens namn är forum_hoaz1600.

Första steget var att skapa databasen med hjälp av följande kod:

```
CREATE DATABASE IF NOT EXISTS forum_hoaz1600;
```

Vilket innebär skapa databasen om den inte redan finns med namnet forum_hoaz1600

Sedan har gets till DBMS kommandot att använda databasen genom följande kod:

```
USE forum_hoaz1600;
```

Safe updates har inaktiverats:

```
/* Disable safe updates */
SET SQL_SAFE_UPDATES=0;
```

Nu är det klart för att skapa tabeller.

För att skapa tabellerna har kommandot CREATE TABLE <tabellnamn> (kolumnnamn datatyp);

Använts.

```
CREATE TABLE Users (
    userID      INTEGER NOT NULL,
    username    VARCHAR(20) NOT NULL,
    PASSWORD    VARCHAR(50) NOT NULL,
    email       VARCHAR(50) NOT NULL);

CREATE TABLE Posts (
    postID      INTEGER NOT NULL,
    userID      INTEGER NOT NULL,
    sectionID   INTEGER NOT NULL,
    title       VARCHAR(100) NOT NULL,
    postText    VARCHAR(2000) NOT NULL,
    date        DATE NOT NULL);

CREATE TABLE Comments (
    commentID       INTEGER NOT NULL,
    postID          INTEGER NOT NULL,
    userID          INTEGER NOT NULL,
    commentText     VARCHAR(500) NOT NULL,
    date            DATE NOT NULL);

CREATE TABLE Replies (
    replyID         INTEGER NOT NULL,
    commentID       INTEGER NOT NULL,
    userID          INTEGER NOT NULL,
    replyText       VARCHAR(500) NOT NULL,
    replyDate       DATE NOT NULL);

CREATE TABLE Sections (
    sectionID       INTEGER NOT NULL,
    name            VARCHAR(50) NOT NULL,
    description     VARCHAR(200) NOT NULL);

CREATE TABLE userRoles (
    roleID          INTEGER NOT NULL,
    userID          INTEGER NOT NULL,
    roleName        VARCHAR(20) NOT NULL );

CREATE TABLE likes (
```

```
    commentID       INTEGER NOT NULL,
    userID          INTEGER NOT NULL);

CREATE TABLE votes (
    postID          INTEGER NOT NULL,
    userID          INTEGER NOT NULL);
```

En kontroll genomfördes för kontrollera att alla tabeller skapades:

Show tables;

```
+--------------------------+
| Tables_in_forum_hoaz1600 |
+--------------------------+
| comments                 |
| likes                    |
| posts                    |
| replies                  |
| sections                 |
| userroles                |
| users                    |
| votes                    |
+--------------------------+
```

Nu när tabellerna är skapade så har constraints eller begränsningar på svenska angetts. För att ange vilka constraints som gäller har kommandot ALTER TABEL <tabellnamn> använts.

För att sätta primärnycklar (Primary keys) har följande kod använts:

```
ALTER TABLE Users ADD CONSTRAINT user_pk PRIMARY KEY (userID);
ALTER TABLE Posts ADD CONSTRAINT post_pk PRIMARY KEY (postID);
ALTER TABLE Comments ADD CONSTRAINT comment_pk PRIMARY KEY (commentID);
ALTER TABLE Replies ADD CONSTRAINT reply_pk PRIMARY KEY (replyID);
ALTER TABLE Sections ADD CONSTRAINT section_pk PRIMARY KEY (sectionID);
ALTER TABLE userRoles ADD CONSTRAINT userRole_pk PRIMARY KEY (roleID);
ALTER TABLE likes ADD CONSTRAINT like_pk PRIMARY KEY (commentID, userID);
ALTER TABLE votes ADD CONSTRAINT vote_pk PRIMARY KEY (postID, userID);
```

Och för att ange främmandenycklarna (Foreign key) har följande kod använts:

```
ALTER TABLE Posts ADD CONSTRAINT Posts_userID_FK FOREIGN KEY (userID)
REFERENCES Users(userID) ON UPDATE CASCADE ON DELETE CASCADE;
ALTER TABLE Posts ADD CONSTRAINT Posts_sectionID_FK FOREIGN KEY (sectionID)
REFERENCES Sections(sectionID) ON UPDATE CASCADE ON DELETE CASCADE;
ALTER TABLE Comments ADD CONSTRAINT Comments_postID_FK FOREIGN KEY (postID)
REFERENCES Posts(postID) ON UPDATE CASCADE ON DELETE CASCADE;
ALTER TABLE Comments ADD CONSTRAINT Comments_userID_FK FOREIGN KEY (userID)
REFERENCES Users(userID) ON UPDATE CASCADE ON DELETE CASCADE;
```

```
ALTER TABLE Replies ADD CONSTRAINT Replies_commentID_FK FOREIGN KEY
(commentID) REFERENCES Comments(commentID) ON UPDATE CASCADE ON DELETE
CASCADE;
ALTER TABLE Replies ADD CONSTRAINT Replies_userID_FK FOREIGN KEY (userID)
REFERENCES Users(userID) ON UPDATE CASCADE ON DELETE CASCADE;
ALTER TABLE userRoles ADD CONSTRAINT userRoles_userID_FK FOREIGN KEY (userID)
REFERENCES Users(userID) ON UPDATE CASCADE ON DELETE CASCADE;
ALTER TABLE likes ADD CONSTRAINT likes_commentID_FK FOREIGN KEY (commentID)
REFERENCES Comments(commentID) ON UPDATE CASCADE ON DELETE CASCADE;
ALTER TABLE likes ADD CONSTRAINT likes_userID_FK FOREIGN KEY (userID)
REFERENCES Users(userID) ON UPDATE CASCADE ON DELETE CASCADE;
ALTER TABLE votes ADD CONSTRAINT votes_postID_FK FOREIGN KEY (postID)
REFERENCES Posts(postID) ON UPDATE CASCADE ON DELETE CASCADE;
ALTER TABLE votes ADD CONSTRAINT votes_userID_FK FOREIGN KEY (userID)
REFERENCES Users(userID) ON UPDATE CASCADE ON DELETE CASCADE;
```

I ovan kod har ON UPDATE CSCADE och ON DELETE CASCADE använts, dessa säkerställer att ändringar eller borttagningar i den refererade tabellen återspeglas i den relaterade tabellen.

För att se till att inmatade E-post adresser vid registrering av nya användare följer lämpligt format dvs t.ex exempel@exempel.com så har följande constraint använts:

```
ALTER TABLE Users ADD CONSTRAINT email_format CHECK (email LIKE '%_@__%.%');
```

För att säkerställa att alla användarnamn är unika, dvs två användare eller fler inte kan ha ett och samma användarnamn så har följande kod använts:

```
ALTER TABLE Users ADD UNIQUE (username);
```

DESCRIBE TABELLNAMN; t.ex. DESCRIBE USERS; har körts för varje tabell i det här stadiet för att ha en överblick över hur tabellerna ser ut, PRI, MUL , UNI osv.

Nu när alla begränsningar (constraints) var deklarerade så var det dags att testa om dessa begränsningar fungerar. För att åstadkomma detta så har testdata skapats och använts enligt följande:

```
INSERT INTO Users VALUES (1, 'alice', 'password_1', 'alice@example.com');
```

En användare har skapats, sedan en till användare med samma användarID (userID) har försökts att skapa:

```
INSERT INTO Users VALUES (1, 'bob', 'password_2', 'bob@example.com');
```

Denna kod generar följande fel:

ERROR 1062 (23000): Duplicate entry '1' for key 'PRIMARY'

Vilket innebär att entitets integritet upprätthålls av databasen.

Ett test att lägga till ett inlägg med ett obefintligt userID har skett enligt följande:

```
INSERT INTO Posts VALUES (10, 200, 1, 'Invalid Post', 'skriver ett inlägg med
obefintligt användar ID', '2024-03-01');
```

Denna kod genererar följande error: Cannot add or update a child row: a foreign key constraint fails (`forum_hoaz1600`.`posts`, CONSTRAINT `Posts_userID_FK` FOREIGN KEY (`userID`) REFERENCES `users` (`userID`) ON DELETE CASCADE ON UPDATE CASCADE)

Vilket bekräftar att referensintegritets constraint fungerar korrekt.

Mer tester har gjorts. Ett test att lägga till användare med felaktigt E-post format:

```
INSERT INTO Users (userID, username, PASSWORD, email) VALUES (5, 'eve',
'losenord', 'felaktig_email');
```

Detta resulterade till följande fel: ERROR 4025 (23000): CONSTRAINT `email_format` failed for `forum_hoaz1600`.`users`

Vilket innebär att E-post format constraint fungerar.

Med samma princip har användarnamns unikhet testats:

Skapa en användare:

```
INSERT INTO Users VALUES (11, 'Gustav', 'secure_pw1', 'gustav@example.com');
```

Och skapa en till användare med samma användarnamn:

```
INSERT INTO Users VALUES (23, 'Gustav', 'secure_pw1', 'gustav@example.com');
```

Generear error: ERROR 1062 (23000): Duplicate entry 'Gustav' for key 'username'

Vilket innebär att unikheten för username upprätthålls.

Nu när testerna av integritetsvillkor och de uppsatta begränsningarna är klara så var det dags att skapa och implementera exempeldata. Exempeldatan har förts in i tabellerna genom att använda INSERT INTO <tabellnamn> VALUES (värdena som ska in); följande kod har använts för att föra in data i tabellerna:

```
/* Insättningar i tabellen users */
INSERT INTO Users VALUES (1, 'alice', 'password_1', 'alice@example.com');
INSERT INTO Users VALUES (2, 'Elsa', 'strong_pw2', 'elsa@example.com');
INSERT INTO Users VALUES (3, 'Oscar', 'secret_pw3', 'oscar@example.com');
INSERT INTO Users VALUES (4, 'Ingrid', 'hidden_pw4', 'ingrid@example.com');
INSERT INTO Users VALUES (5, 'Axel', 'classified_pw5', 'axel@example.com');
INSERT INTO Users VALUES (6, 'Astrid', 'encrypted_pw6', 'astrid@example.com');
INSERT INTO Users VALUES (7, 'Viktor', 'guarded_pw7', 'viktor@example.com');
INSERT INTO Users VALUES (8, 'Freja', 'shielded_pw8', 'freja@example.com');
INSERT INTO Users VALUES (9, 'Liam', 'locked_pw9', 'liam@example.com');
INSERT INTO Users VALUES (10, 'Maja', 'protected_pw10', 'maja@example.com');
```

```sql
INSERT INTO Users VALUES (11, 'Gustav', 'secure_pw1', 'gustav@example.com');

/* Sektioner */
INSERT INTO Sections VALUES (11, 'Python Developers', 'A space for Python
enthusiasts to discuss topics related to this versatile programming
 language, share code snippets, and help each other solve problems.');
INSERT INTO Sections VALUES (12, 'Web Development', 'This section is dedicated
to web developers of all levels. Discuss HTML, CSS, JavaScript,
 and more. Share your projects and get feedback.');
INSERT INTO Sections VALUES (13, 'Data Science & AI', 'Explore the world of
data analytics, machine learning, and artificial intelligence.
Share your insights, projects, and learn from professionals.');

/* Posts */
INSERT INTO Posts VALUES (1, 3, 11, 'Python Debugging Tips', 'Having trouble
with Python? Check out these debugging techniques.', STR_TO_DATE('17-Nov-
2023', '%e-%M-%Y'));
INSERT INTO Posts VALUES (2, 3, 12, 'Web Development Trends', 'Exploring the
latest trends in web development.', STR_TO_DATE('18-Nov-2023', '%e-%M-%Y'));
INSERT INTO Posts VALUES (3, 4, 11, 'Data Analysis med Pandas', 'Det är bra
att lära sig Pandas för data manipulation.', STR_TO_DATE('19-Jan-2024', '%e-
%M-%Y'));
INSERT INTO Posts VALUES (4, 4, 12, 'Responsive Web Design', 'Creating
responsive web layouts using CSS media queries.', STR_TO_DATE('20-Nov-2023',
'%e-%M-%Y'));
INSERT INTO Posts VALUES (5, 4, 13, 'Introduction to Machine Learning',
'Getting started with ML algorithms and models.', STR_TO_DATE('21-Nov-2023',
'%e-%M-%Y'));
INSERT INTO Posts VALUES (6, 5, 11, 'Python vs. JavaScript', 'Comparing Python
and JavaScript for backend development.', STR_TO_DATE('22-Nov-2023', '%e-%M-
%Y'));
INSERT INTO Posts VALUES (7, 6, 12, 'CSS Flexbox Basics', 'Understanding the
fundamentals of CSS Flexbox layout.', STR_TO_DATE('23-Nov-2023', '%e-%M-%Y'));
INSERT INTO Posts VALUES (8, 7, 13, 'Linear Regression Explained', 'A
beginner-friendly guide to linear regression.', STR_TO_DATE('24-Nov-2023',
'%e-%M-%Y'));
INSERT INTO Posts VALUES (9, 7, 13, 'Teknologins framtid', 'Tror ni att vi
kommer att behövas om ett par år?', STR_TO_DATE('25-Nov-2023', '%e-%M-%Y'));

/* Comments */
INSERT INTO Comments VALUES (1, 1, 5, 'Interesting Python debugging tips!',
STR_TO_DATE('17-Nov-2023', '%e-%M-%Y'));
INSERT INTO Comments VALUES (2, 2, 6, 'Web development trends are evolving
rapidly.', STR_TO_DATE('18-Nov-2023', '%e-%M-%Y'));
INSERT INTO Comments VALUES (3, 3, 7, 'Pandas is a powerful library for data
analysis.', STR_TO_DATE('20-Jan-2024', '%e-%M-%Y'));
INSERT INTO Comments VALUES (4, 4, 8, 'Responsive web design is crucial for
user experience.', STR_TO_DATE('20-Nov-2023', '%e-%M-%Y'));
```

```sql
INSERT INTO Comments VALUES (5, 5, 9, 'Machine learning opens up exciting possibilities.', STR_TO_DATE('21-Nov-2023', '%e-%M-%Y'));
INSERT INTO Comments VALUES (6, 6, 10, 'Python vs. JavaScript debate!', STR_TO_DATE('22-Nov-2023', '%e-%M-%Y'));
INSERT INTO Comments VALUES (7, 7, 11, 'Flexbox is a game-changer for web layouts.', STR_TO_DATE('23-Nov-2023', '%e-%M-%Y'));
INSERT INTO Comments VALUES (8, 8, 5, 'Linear regression basics explained.', STR_TO_DATE('24-Nov-2023', '%e-%M-%Y'));
INSERT INTO Comments VALUES (9, 7, 6, 'CSS is not my cup of tea anyway', STR_TO_DATE('23-Nov-2023', '%e-%M-%Y'));

/* Replies */
INSERT INTO Replies VALUES (1, 8, 10, 'Grundläggande förklaring av linjär regression.', STR_TO_DATE('27-Nov-2023', '%e-%M-%Y'));
INSERT INTO Replies VALUES (2, 7, 11, 'Flexbox är verkligen en game changer.', STR_TO_DATE('28-Nov-2023', '%e-%M-%Y'));
INSERT INTO Replies VALUES (3, 6, 1, 'Python vs. JavaScript-debatten fortsätter!.', STR_TO_DATE('29-Nov-2023', '%e-%M-%Y'));
INSERT INTO Replies VALUES (4, 5, 2, 'Maskininlärning öppnar upp för spännande möjligheter.', STR_TO_DATE('30-Nov-2023', '%e-%M-%Y'));
INSERT INTO Replies VALUES (5, 4, 3, 'Responsiv design är avgörande för användarupplevelsen.', STR_TO_DATE('21-Dec-2023', '%e-%M-%Y'));
INSERT INTO Replies VALUES (6, 3, 4, 'Pandas är en game changer för dataanalys', STR_TO_DATE('22-Jan-2024', '%e-%M-%Y'));
INSERT INTO Replies VALUES (7, 2, 5, 'Absolut! Webbutvecklingstrender är fascinerande', STR_TO_DATE('23-Dec-2023', '%e-%M-%Y'));
INSERT INTO Replies VALUES (8, 1, 6, 'Jag håller med Python felsökning kan vara lurigt ibland.', STR_TO_DATE('24-Dec-2023', '%e-%M-%Y'));

/* UserRoles */
INSERT INTO userRoles VALUES (1, 3, 'Admin');
INSERT INTO userRoles VALUES (2, 5, 'Mod');

/* Likes */
INSERT INTO likes VALUES (1, 5);
INSERT INTO likes VALUES (2, 6);
INSERT INTO likes VALUES (3, 7);
INSERT INTO likes VALUES (4, 7);
INSERT INTO likes VALUES (4, 8);
INSERT INTO likes VALUES (4, 9);
INSERT INTO likes VALUES (5, 9);
INSERT INTO likes VALUES (6, 10);
INSERT INTO likes VALUES (7, 11);
INSERT INTO likes VALUES (8, 4);
INSERT INTO likes VALUES (1, 3);
INSERT INTO likes VALUES (2, 1);
INSERT INTO likes VALUES (3, 2);
```

```sql
/* Votes */
INSERT INTO votes VALUES (4, 7);
INSERT INTO votes VALUES (4, 8);
INSERT INTO votes VALUES (4, 9);
INSERT INTO votes VALUES (5, 6);
INSERT INTO votes VALUES (5, 1);
INSERT INTO votes VALUES (1, 11);
```

Nu när data är inmatad i tabellerna är det möjligt att kontrollera om databasen och dess tabeller följer normalformerna 1-3.

SELECT * FROM TABELLNAMN; har körts för varje tabell för att skriva ut tabellerna i terminalen och genomföra NF kontrollen.

SELECT * FROM users;
SELECT * FROM sections;
SELECT * FROM posts;
SELECT * FROM comments;
SELECT * FROM replies;
SELECT * FROM likes;
SELECT * FROM votes;
SELECT * FROM userroles;

1NF: Alla värden i tabellerna är atomära dvs att det inte finns mer än ett värde per ruta. Det finns primär nyckel i varje tabell. Tabellerna innehåller inga upprepade rader eller kolumner, och varje kolumn innehåller en datatyp eller ett värde. Med detta sagt så bedöms att tabellerna följer den första normalformen.

2NF: Endast 2 av databasens tabeller har fler än 1 primary key PK. Dessa är Votes och Likes. Dessa två tabeller har inga flera attribut i sig än primär nycklarna vilket leder till att risken för partiellt beroende (Partial dependency) är uteslutet, med andra ord det finns inga attribut i dessa tabeller som inte har fullständigt funktionell beroende av hela PK uppsättningen. Detta leder till att databastabellerna inte bryter mot den andra normalformen.

3NF: I databastabellerna finns det inget icke-nyckelattribut som är fullständigt funktionellt beroende av något annat icke-nyckelattribut. Vilket leder till att databasens tabeller inte bryter mot 3NF.

En liten kommentar angående 3NF när det gäller tabellen Posts:

```
MariaDB [forum_hoaz1600]> select * from posts;
+--------+--------+-----------+---------------------------------+-----------------------------------------------------------------+------------+
| postID | userID | sectionID | title                           | postText                                                        | date       |
+--------+--------+-----------+---------------------------------+-----------------------------------------------------------------+------------+
|      1 |      3 |        11 | Python Debugging Tips           | Having trouble with Python? Check out these debugging techniques. | 2023-11-17 |
|      2 |      3 |        12 | Web Development Trends          | Exploring the latest trends in web development.                 | 2023-11-18 |
|      3 |      4 |        11 | Data Analysis med Pandas        | Det är bra att lära sig Pandas för data manipulation.           | 2024-01-19 |
|      4 |      4 |        12 | Responsive Web Design           | Creating responsive web layouts using CSS media queries.        | 2023-11-20 |
|      5 |      4 |        13 | Introduction to Machine Learning | Getting started with ML algorithms and models.                  | 2023-11-21 |
|      6 |      5 |        11 | Python vs. JavaScript           | Comparing Python and JavaScript for backend development.        | 2023-11-22 |
|      7 |      6 |        12 | CSS Flexbox Basics              | Understanding the fundamentals of CSS Flexbox layout.           | 2023-11-23 |
|      8 |      7 |        13 | Linear Regression Explained     | A beginner-friendly guide to linear regression.                 | 2023-11-24 |
+--------+--------+-----------+---------------------------------+-----------------------------------------------------------------+------------+
```

Title och PostText är två separata komponenter som är beroende endast av PK som är postID. Title bestämmer inte PostText. PostText har inget funktionellt beroende av Title. Teknisktmässigt så kan en användare skriva titeln om något ämne och inläggstexten om något helt annat, men

det är inte så smart då blir inlägget inte så populärt och kanske får inga röster från andra användare. Både title och postText måste fyllas i för att skapa ett inlägg. Det som försöks att förmedlas här är att även denna tabell är granskat med resultat att den inte bryter mot 3NF.

Databasen är designad med fokus på att inte bryta mot normalformerna från början vilket har lett till att ingen normalisering i efterhand har behövts.

SQL-frågor

```
/* En SELECT sats med jämförelse operatorer som > ,<  != eller liknande
Välj postID från tabellen POSTS där datumet är större (eller senare) än 2024-
01-01 */
SELECT POSTID FROM POSTS WHERE DATE > '2024-01-01';

/* En SELECT med LIKE mot ett textfält.
Välj Kommentarer från tabellen commentText some innehåller ordet "Flexbox" */
SELECT COMMENTTEXT FROM COMMENTS WHERE COMMENTTEXT LIKE '%Flexbox%';

/* En fråga med IN, GROUP BY och HAVING.
Lista ut titlarna för de inlägg som har fått 2 röster eller fler */
SELECT title FROM Posts WHERE postID IN (SELECT postID FROM votes GROUP BY
postID HAVING COUNT(*) >= 2);

/* En fråga med EXIST
Välj kommenratexten från tabellen kommentarer där användaren har kommenterat
sitt eget inlägg.
Här har jag använt Alias p och c för att referera till posts och comments. */
SELECT commentText FROM Comments c WHERE EXISTS (SELECT postID FROM Posts p
WHERE p.postID = c.postID AND p.userID = c.userID);

/* En fråga med GROUP BY.
För de användare som har kommenterat inlägg lista ut antal kommentarer som är
gjorda av varje användare (av varje userID). */
SELECT COUNT(commentID), userID FROM comments GROUP BY userID;

/* En fråga med HAVING
Lista ut antalet inlägg som är gjorda av varje användare och inkludera endast
användare som har gjort mer än 2 inlägg */
SELECT COUNT(postID), userID FROM posts GROUP BY userID HAVING COUNT(postID) >
2;

/* En fråga med ORDER BY
Lista ut inläggtexterna och sortera dem enligt textlängden från texten med
högst antal tecken till texten med lägst antal tecken (descending) */
SELECT posttext FROM posts ORDER BY CHAR_LENGTH(posttext) DESC;

/* En fråga med LEFT JOIN
lista ut alla inlägg tillsammans med eventuella tillhörande kommentarer. Om
ett inlägg inte har några kommentarer ska det ändå
inkluderas i resultatet. */
```

```sql
SELECT Posts.postID, Posts.postText, Comments.commentText FROM Posts LEFT JOIN Comments ON Posts.postID = Comments.postID;

/* En fråga med MAX()
Lista ut inläggtitlen av det inlägget som har det högsta dvs senaste datumet */
SELECT title FROM posts WHERE DATE = (SELECT MAX(DATE) FROM posts);

/* En selektiv DELETE
Ta bort inlägg som inte har några kommentarer alls */
DELETE p FROM posts p LEFT JOIN comments c ON p.postID = c.postID WHERE c.postID IS NULL;


/* En selective UPDATE
Uppdatera tabellen Users så att du byter namn till Anna för användaren med användar ID 1 */
UPDATE Users SET username = 'Anna' WHERE userID = 1;



/* En Auto-increment som gör att ett unikt nummer genereras automatiskt när en ny insert infogas i en tabell. */
ALTER TABLE Users MODIFY COLUMN userID INTEGER NOT NULL AUTO_INCREMENT;
INSERT INTO Users (username, PASSWORD, email) VALUES ('Jesper', 'secure_pw1', 'jesse@example.com'); -- Ny user utan att ange userID, den får ett ID per automatik från auto increment
```

Funktioner

```sql
/* 1- Funktion som visar det totala antalet kommentarer för ett angivet inlägg */
DELIMITER //
CREATE FUNCTION commentCount(postNum INT) RETURNS INT
BEGIN
    DECLARE commentCount INT;
    SELECT COUNT(*) INTO commentCount FROM Comments WHERE postID = postNum;
    RETURN commentCount;
END //
DELIMITER ;


SELECT commentCount(4); -- function anrop i query

-------------------------------------------------------------------------------
/* 2- Funktion som visar det totala antalet kommentarer gjorda av en specifik användare */
DELIMITER //
```

```
CREATE FUNCTION numberOfComments(inputUserId INT) RETURNS INT
BEGIN
    DECLARE commentCount INT;
    SELECT COUNT(*) INTO commentCount FROM Comments WHERE userID =
inputUserId;
    RETURN commentCount;
END;
//
DELIMITER ;

SELECT numberOfComments(5);
-------------------------------------------------------------------------
/* 3- Function som ändrar/uppdaterar E-post adress för angiven användare*/
DELIMITER //

CREATE FUNCTION changeEmail(inputUserId INT, newEmail VARCHAR(50)) RETURNS
VARCHAR(40)
BEGIN
    UPDATE Users
    SET email = newEmail
    WHERE userID = inputUserId;
    RETURN 'Klart!';
END;
//
DELIMITER ;

SELECT changeEmail(2, 'test@mejl.com');   -- Testar funktionen

-------------------------------------------------------------------------
/* 4- Funktion som returnerar anvädarnamnet som är kopplat till det inmatade
ID  */
DELIMITER //

CREATE FUNCTION showUsername(imputID INT) RETURNS VARCHAR(20)
BEGIN
    DECLARE nameQuery VARCHAR(20);
    SELECT username INTO nameQuery FROM Users WHERE userID = imputID;
    RETURN nameQuery;
END;
//
DELIMITER ;

SELECT showUsername(4);
-------------------------------------------------------------------------
/* 5- Funktion som beräknar den genomsnittliga längden på inlägg (i form av
antalet tecken) som gjorts av en
specifik användare */
```

```
DELIMITER //

CREATE FUNCTION averagePostLength(inputID INT) RETURNS DECIMAL
BEGIN
    DECLARE totalLength INT;
    DECLARE postCount INT;
    DECLARE avgLength DECIMAL;

    -- Beräkna den totala längden (antal tecken) av inlägg som är gjorda av
användaren
    SELECT SUM(CHAR_LENGTH(postText)) INTO totalLength FROM Posts WHERE userID
= inputID;

    -- räkna antal inlägg gjorda av användaren
    SELECT COUNT(*) INTO postCount FROM Posts WHERE userID = inputID;

    -- Beräkna medelvärdet av inläggslängden
    IF postCount > 0 THEN
        SET avgLength = totalLength / postCount;
    ELSE
        SET avgLength = 0;
    END IF;

    RETURN avgLength;
END;
//
DELIMITER ;

SELECT averagePostLength(1);
```

Procedurer

```
/* 1- Procedure AddComment lägger till kommentar till ett inlägg med det
aktuella datumet (dvs utan att mata in datumet manuellt) */
ALTER TABLE Comments MODIFY COLUMN commentID INTEGER NOT NULL AUTO_INCREMENT;
-- Detta är för att skippa mata in commentID och få ett auto genererat

DELIMITER //
CREATE PROCEDURE AddComment(IN postId INT, IN userId INT, IN commentText
VARCHAR(500))

BEGIN
    INSERT INTO Comments (postID, userID, commentText, date)
    VALUES (postId, userId, commentText, NOW());
END //
DELIMITER ;
```

```
CALL AddComment(3, 2, 'Det här är ju bra grejer');

-------------------------------------------------------------------------------
----------
/* 2- En procerdur som visar kommenratexten från tabellen kommentarer där
användaren har kommenterat sitt eget inlägg. */
DELIMITER //

CREATE PROCEDURE showOwnComment()
BEGIN
    SELECT postID, userID, commentText FROM Comments c WHERE EXISTS
    (SELECT postID FROM Posts p WHERE p.postID = c.postID AND p.userID =
c.userID);
END;
//
DELIMITER ;

CALL showOwnComment();
-------------------------------------------------------------------------------
-----------------
-- 3- En procedure som visar antalet inlägg en användare har gjort
DELIMITER //

CREATE PROCEDURE totalUserPosts(IN inputUserId INT, OUT totalPosts INT)
BEGIN
    SELECT COUNT(*) INTO totalPosts FROM Posts WHERE userID = inputUserId;
END;
//
DELIMITER ;

CALL totalUserPosts(3, @totalPosts); -- Antalet inlägg som är skrivna av
användare med ID = 3
SELECT @totalPosts;
-------------------------------------------------------------------------------
----------------------
/* 4- Procedure som visar kommentartexten på valt inlägg */
DELIMITER //

CREATE PROCEDURE postComments(IN commentedPost INT)
BEGIN
    SELECT commentText FROM Comments WHERE postID = commentedPost;
END;
//
DELIMITER ;

CALL postComments(7);  -- Här anrops det proceduren med postID 7 och då får vi
texterna på 2st kommentarer på inlägget
```

```
-------------------------------------------------------------------------------
--------------------
/* 5- En procedure som listar ut inlägget som har fått flest röster */
DELIMITER //

CREATE PROCEDURE mostVoted()
BEGIN

    SELECT Posts.postID, Posts.title, Posts.postText, Posts.date,
COUNT(votes.postID) AS numVotes
    FROM Posts
    LEFT JOIN votes ON Posts.postID = votes.postID
    GROUP BY Posts.postID
    ORDER BY numVotes DESC
    LIMIT 1;
END;
//
DELIMITER ;

CALL mostVoted();
```

Trigger

```
/* 1- Trigger som anger det aktuella datumet vid postskapandet så att man kan
skapa post utan att manuellt ange date där date
stoppas in i insertion av triggern */

DELIMITER //
CREATE TRIGGER inserPostCurrentDate
BEFORE INSERT ON Posts
FOR EACH ROW
BEGIN
    SET NEW.date = CURDATE();
END;
//
DELIMITER ;


INSERT INTO Posts (postID, userID, sectionID, title, postText) VALUES (14, 1,
11, 'Trigger test', 'Här testar vi triggern'); -- Testa trigger

-------------------------------------------------------------------------------
-----------------------------------------

/* 2- Trigger som uppdaterar latest.activity kolumn i Users när en användare
skapar ett inlägg. Efter inlägget är skapat så updaterar triggern
latest_activity kolumnen med datumet då inlägget skapades*/
```

```
-- skapar denna kolumn för triggers skull. Obs kolumnen är ej granskad mot
Normal formerna.
ALTER TABLE Users ADD COLUMN latest_activity DATE;


DELIMITER //

CREATE TRIGGER userLatestActivity
AFTER INSERT ON Posts
FOR EACH ROW
BEGIN
    UPDATE Users
    SET latest_activity = NOW()
    WHERE userID = NEW.userID;
END;
//
DELIMITER ;

INSERT INTO Posts (postID, userID, sectionID, title, postText) VALUES (13, 2,
12, 'Trigger test igen', 'Här testar vi en till triggern');

------------------------------------------------------------------------------
--------------------
/* 3- En trigger som inte tillåter att en användare med userRole att tas bort
från databasen */
DELIMITER //

CREATE TRIGGER noDeleteUserRole
BEFORE DELETE ON Users
FOR EACH ROW
BEGIN
    DECLARE roleCount INT;

    -- Kollar ifall användaren har någon roll i userRole tabellen
    SELECT COUNT(*) INTO roleCount
    FROM userRoles
    WHERE userID = OLD.userID;

    -- om användaren har roll blir roleCount värde 1 och då bryts borttagning
och ett felmeddelande genereras
    IF roleCount > 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Du får inte ta bort en användare som är Admin
eller moderator';
    END IF;
END;
//
DELIMITER ;
```

```
--testa att tabort användaren vars ID = 3 som är Admin
DELETE FROM Users WHERE userID = 3;
-------------------------------------------------------------------------------
/* 4- Trigger som uppdaterar inläggetsdatum till aktuelldatum när inläggstext
ändras/uppdateras */
DELIMITER //

CREATE TRIGGER postEditedDate
BEFORE UPDATE ON Posts
FOR EACH ROW
BEGIN
    IF OLD.postText != NEW.postText THEN
        SET NEW.date = CURDATE();
    END IF;
END;
//

DELIMITER ;

UPDATE Posts SET postText = 'ändra inläggtext för att testa postEditeDate
trigger' WHERE postID = 1; -- test av att ändra texten i ett inlägg
-------------------------------------------------------------------------------
/* 5- Trigger som ger inlägg skrivare/skapare 1 reputation poäng i
userReputation kolumn i Users tabell varje gång deras inlägg får en röst
(Vote) */
--Obs kolumnen är ej granskad mot Normal formerna.
ALTER TABLE Users ADD COLUMN userReputation INTEGER DEFAULT 0; -- skapar
kolumn userReputation för triggers skull

DELIMITER //

CREATE TRIGGER reputation
AFTER INSERT ON votes
FOR EACH ROW
BEGIN
    DECLARE postCreator INT;

    -- Avgör userID på den som skapade inlägget som har fått en röst och
lagrar det i variabeln postCreator
    SELECT userID INTO postCreator
    FROM Posts
    WHERE postID = NEW.postID;

    -- lägger till 1 poäng till inläggets skaparen
    IF postCreator IS NOT NULL THEN
        UPDATE Users
        SET userReputation = userReputation + 1
        WHERE userID = postCreator;
```

```
    END IF;
END;
//

DELIMITER ;


INSERT INTO votes VALUES (2, 10);  -- Lägga till en röst (vote) för att testa
triggern
```

Views

```
/* En vy som visar alla inlägg (både title och postText) med antal röster och
kommentarer som är kopplade till var och ett inlägg */
CREATE VIEW postOverview AS
SELECT
    p.title,
    p.postText,
    COUNT(DISTINCT v.userID) AS voteCount,
    COUNT(DISTINCT c.commentID) AS commentCount
FROM
    Posts p
LEFT JOIN
    Votes v ON p.postID = v.postID
LEFT JOIN
    Comments c ON p.postID = c.postID
GROUP BY
    p.postID, p.userID, p.sectionID, p.title, p.postText;

    SELECT * FROM postOverview; -- Vyanrop


/* En vy som visar alla användare (användar ID och namn) med antalet imlägg
och inläggkommentarer som de har fått */


CREATE VIEW userContribution AS
SELECT
    u.userID,
    u.userName,
    COUNT(DISTINCT p.postID) AS totalPosts,
    COUNT(DISTINCT c.commentID) AS totalComments
FROM
    Users u
LEFT JOIN
    Posts p ON u.userID = p.userID
LEFT JOIN
    Comments c ON p.postID = c.postID
GROUP BY
```

```
    u.userID, u.username;

SELECT * FROM userContribution; -- Anropa vyn
```