Instagram Data Model

In this project, I will design a data model for Instagram using PostgreSQL. I will create different tables and build the data model, generate SQL create table statements, insert data, updating data, and add analytic statements using where condition, orderby, group by, having clause, along with different aggregation functions. I will also cover subquery, window function, CTE, case statement, date casting, and working with dates.

Designing Data Model

We will have 5 tables stored in a database.

Users

COLUMN	TYPE	CONSTRAINTS
user_id	SERIAL	PRIMARY KEY
name	VARCHAR(50)	NOT NULL
email	VARCHAR(50)	UNIQUE NOT NULL
phone_number	VARCHAR(20)	UNIQUE

Posts

COLUMN	TYPE	CONSTRAINTS
post_id	SERIAL	PRIMARY KEY
user_id	INTEGER	NOT NULL FK (user_id) REFERENCE (user_id)
caption	TEXT	
image_url	VARCHAR(200)	
created_at	TIMESTAMP	DEFAULT CURRENT_TIME STAMP

Comments

COLUMN	TYPE	CONSTRAINTS
comment_id	SERIAL	PRIMARY KEY
post_id	INTEGER	NOT NULL FK (post_id) REFERENCE (post_id)
user_id	INTEGER	NOT NULL FK (user_id) REFERENCE (user_id)
comment_text	TEXT	
created_at	TIMESTAMP	DEFAULT CURRENT_TIME STAMP

Likes

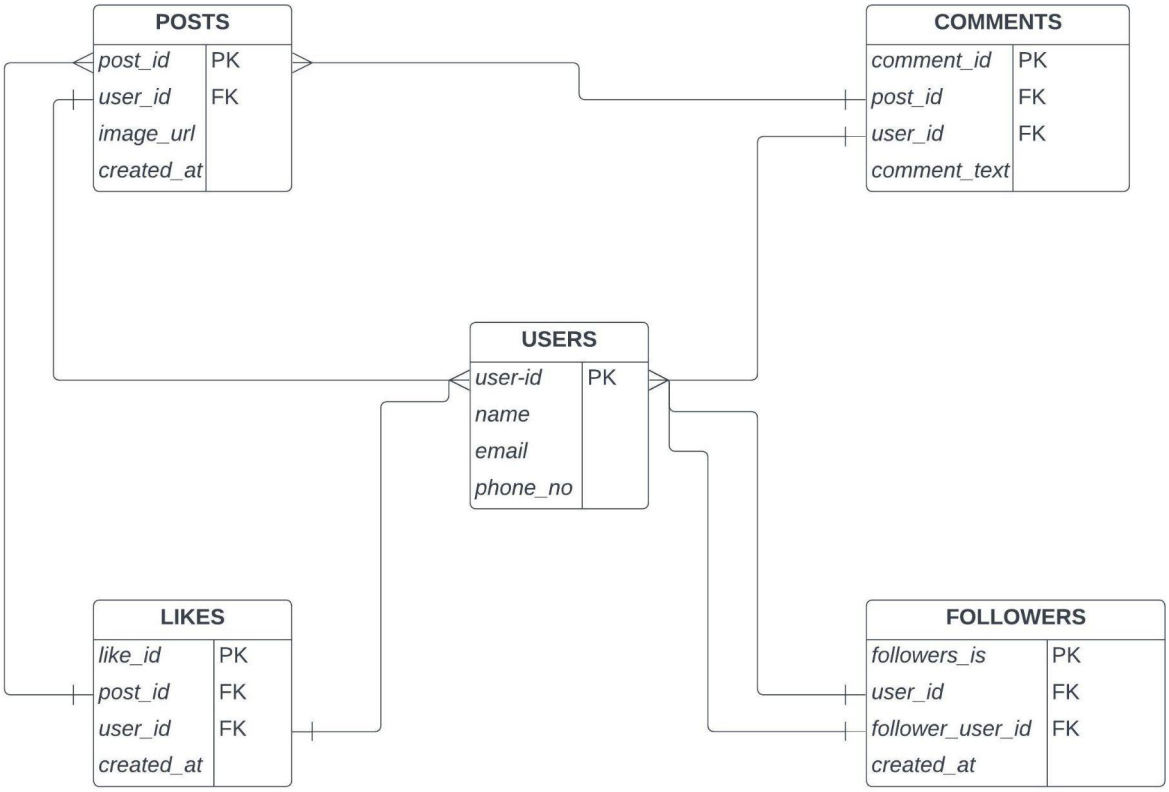
COLUMN	TYPE	CONSTRAINTS
like_id	SERIAL	PRIMARY KEY
post_id	INTEGER	NOT NULL FK (post_id) REFERENCE (post_id)
user_id	INTEGER	NOT NULL FK (user_id) REFERENCE (user_id)
Created_at	TIMESTAMP	DEFAULT CURRENT_TIME STAMP

Followers

COLUMN	TYPE	CONSTRAINTS
follower_id	SERIAL	PRIMARY KEY
user_id	INTEGER	NOT NULL FK (user_id)

follower_user_id	INTEGER	NOT NULL FK (follower_user_id) REFERENCE (user_id)
created_at	TIMESTAMP	DEFAULT CURRENT TIME_STAMP

Instagram Data Model



Generally, the social networks are not built using the relational database systems, but here I am just trying to replicate this on a smaller system.

SQL Queries

Creating tables

```
CREATE TABLE Users (
    user_id SERIAL PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    phone_number VARCHAR(20) UNIQUE
);

CREATE TABLE Posts (
    post_id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL,
    caption TEXT,
    image_url VARCHAR(200),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES Users(user_id)
);

CREATE TABLE Comments (
    comment_id SERIAL PRIMARY KEY,
    post_id INTEGER NOT NULL,
    user_id INTEGER NOT NULL,
    comment_text TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (post_id) REFERENCES Posts(post_id),
    FOREIGN KEY (user_id) REFERENCES Users(user_id)
);

CREATE TABLE Likes (
    like_id SERIAL PRIMARY KEY,
    post_id INTEGER NOT NULL,
    user_id INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (post_id) REFERENCES Posts(post_id),
    FOREIGN KEY (user_id) REFERENCES Users(user_id)
);

CREATE TABLE Followers (
```

```
    follower_id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL,
    follower_user_id INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (user_id) REFERENCES Users(user_id),
    FOREIGN KEY (follower_user_id) REFERENCES Users(user_id)
);
```

Inserting Data

```
-- Inserting into Users table
INSERT INTO Users (name, email, phone_number)
VALUES
    ('John Smith', 'johnsmith@gmail.com', '1234567890'),
    ('Jane Doe', 'janedoe@yahoo.com', '0987654321'),
    ('Bob Johnson', 'bjohnson@gmail.com', '1112223333'),
    ('Alice Brown', 'abrown@yahoo.com', NULL),
    ('Mike Davis', 'mdavis@gmail.com', '5556667777');

-- Inserting into Posts table
INSERT INTO Posts (user_id, caption, image_url)
VALUES
    (1, 'Beautiful sunset', '<https://www.example.com/sunset.jpg>'),
    (2, 'My new puppy', '<https://www.example.com/puppy.jpg>'),
    (3, 'Delicious pizza', '<https://www.example.com/pizza.jpg>'),
    (4, 'Throwback to my vacation',
'<https://www.example.com/vacation.jpg>'),
    (5, 'Amazing concert', '<https://www.example.com/concert.jpg>');

-- Inserting into Comments table
INSERT INTO Comments (post_id, user_id, comment_text)
VALUES
    (1, 2, 'Wow! Stunning.'),
    (1, 3, 'Beautiful colors.'),
    (2, 1, 'What a cutie!'),
    (2, 4, 'Aww, I want one.'),
    (3, 5, 'Yum!'),
```

```
    (4, 1, 'Looks like an awesome trip.'),
    (5, 3, 'Wish I was there!');

-- Inserting into Likes table
INSERT INTO Likes (post_id, user_id)
VALUES
    (1, 2),
    (1, 4),
    (2, 1),
    (2, 3),
    (3, 5),
    (4, 1),
    (4, 2),
    (4, 3),
    (5, 4),
    (5, 5);

-- Inserting into Followers table
INSERT INTO Followers (user_id, follower_user_id)
VALUES
    (1, 2),
    (2, 1),
    (1, 3),
    (3, 1),
    (1, 4),
    (4, 1),
    (1, 5),
    (5, 1);
```

Updating Data

```
-- Updating the caption of post_id 3
UPDATE Posts
SET caption = 'Best pizza ever'
WHERE post_id = 3;
```

Where Condition

```
-- Selecting all the posts where user_id is 1
SELECT *
FROM Posts
WHERE user_id = 1;
```

Order By

```
-- Selecting all the posts and ordering them by created_at in descending
order

SELECT *
FROM Posts
ORDER BY created_at DESC;
```

Group By and Having

```
-- Counting the number of likes for each post and showing only the posts
with more than 2 likes

SELECT Posts.post_id, COUNT(Likes.like_id) AS num_likes
FROM Posts
LEFT JOIN Likes ON Posts.post_id = Likes.post_id
GROUP BY Posts.post_id
HAVING COUNT(Likes.like_id) > 2;
```

Aggregation Functions

```
-- Finding the total number of likes for all posts
SELECT SUM(num_likes) AS total_likes
FROM (
    SELECT COUNT(Likes.like_id) AS num_likes
    FROM Posts
    LEFT JOIN Likes ON Posts.post_id = Likes.post_id
    GROUP BY Posts.post_id
```

```
) AS likes_by_post;
```

```
-- Finding how many likes does each post have

SELECT Posts.caption, COUNT(Likes.like_id) AS num_likes
FROM Posts
LEFT JOIN Likes ON Posts.post_id = Likes.post_id
GROUP BY Posts.post_id;
```

```
-- Finding average number of likes per post
SELECT AVG(num_likes) AS avg_likes
FROM (
    SELECT COUNT(Likes.like_id) AS num_likes
    FROM Posts
    LEFT JOIN Likes ON Posts.post_id = Likes.post_id
    GROUP BY Posts.post_id
) AS likes_by_post;
```

```
-- Finding which user has the most number of followers
SELECT Users.name, COUNT(Followers.follower_id) AS num_followers
FROM Users
LEFT JOIN Followers ON Users.user_id = Followers.user_id
GROUP BY Users.user_id
ORDER BY num_followers DESC
LIMIT 1;
```

Joins

```
-- Which users have like post_id 2

SELECT Users.name
FROM Users
JOIN Likes ON Users.user_id = Likes.user_id
WHERE Likes.post_id = 2;
```

```sql
-- Which posts have no comment

SELECT Posts.caption
FROM Posts
LEFT JOIN Comments ON Posts.post_id = Comments.post_id
WHERE Comments.comment_id IS NULL;
```

```sql
-- Which posts were created by users who have no followers

SELECT Posts.caption
FROM Posts
LEFT JOIN Comments ON Posts.post_id = Comments.post_id
WHERE Comments.comment_id IS NULL;
```

Sub Query

```sql
-- Finding all the users who have commented on post_id 1
SELECT name
FROM Users
WHERE user_id IN (
    SELECT user_id
    FROM Comments
    WHERE post_id = 1
);
```

CTE

```sql
-- Finding all the posts and their comments using a Common Table Expression

WITH post_comments AS (
    SELECT Posts.post_id, Posts.caption, Comments.comment_text
    FROM Posts
    LEFT JOIN Comments ON Posts.post_id = Comments.post_id
)
SELECT *
FROM post_comments;
```

```sql
-- Ranking posts based on number of likes
SELECT post_id, num_likes, RANK() OVER (ORDER BY num_likes DESC) AS rank
FROM (
    SELECT Posts.post_id, COUNT(Likes.like_id) AS num_likes
    FROM Posts
    LEFT JOIN Likes ON Posts.post_id = Likes.post_id
    GROUP BY Posts.post_id
) AS likes_by_post;
```

```sql
-- Finding all the posts and comments using CTE
WITH post_comments AS (
    SELECT Posts.caption, Comments.comment_text
    FROM Posts
    LEFT JOIN Comments ON Posts.post_id = Comments.post_id
)
SELECT *
FROM post_comments;
```

Case Statement

```sql
-- Categorizing the posts based on the number of likes
SELECT
    post_id,
    CASE
        WHEN num_likes = 0 THEN 'No likes'
        WHEN num_likes < 5 THEN 'Few likes'
        WHEN num_likes < 10 THEN 'Some likes'
        ELSE 'Lots of likes'
    END AS like_category
FROM (
    SELECT Posts.post_id, COUNT(Likes.like_id) AS num_likes
    FROM Posts
    LEFT JOIN Likes ON Posts.post_id = Likes.post_id
    GROUP BY Posts.post_id
) AS likes_by_post;
```

Date Casting and Working with Dates

```
-- Finding all the posts created in the last month
SELECT *
FROM Posts
WHERE created_at >= CAST(DATE_TRUNC('month', CURRENT_TIMESTAMP - INTERVAL
'1 month') AS DATE);
```