**ASSIGNMENT\_5**

**NAME**: D. SAIMANIKANTA

**REG NO**: 192372348

**COURSE**:DATA BASE MANAGEMENT

SYSTEM

**CODE**:CSA0593

**SCENERIO:**

**Design a database for storing and processing data generated by social media users in real-time. Requirements: Model tables to handle users, posts, comments, likes, and shares. Implement partitioning based on time intervals (e.g., hourly) to efficiently manage large volumes of user-generated content. Write queries to generate reports on user engagement trends, detect spikes in activity, and visualize data by user demographics. Create views or materialized views for summarizing engagement metrics and improving performance of analytics queries**

Designing a database to store and process real-time data generated by social media users involves careful consideration of table structure, partitioning strategies, and indexing to handle large volumes of data efficiently. Below is a suggested approach to design this system.

**1. Database Schema Design**

We will design the schema with tables for Users, Posts, Comments, Likes, and Shares. These tables will be related using foreign keys, and we will partition certain tables based on time intervals (e.g., hourly partitions) to manage large volumes of data.

**Users Table**

Stores information about the users of the social media platform.

|  |
| --- |
| CREATE TABLE Users (  user\_id SERIAL PRIMARY KEY,  username VARCHAR(255) UNIQUE,  email VARCHAR(255) UNIQUE,  gender VARCHAR(50), *-- e.g., "Male", "Female", "Non-Binary"*  age INT,  location VARCHAR(255),  created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP |

**Posts Table**

**Stores information about the posts created by users. This table will be partitioned by time, i.e., by created\_at (hourly).**

|  |
| --- |
| CREATE TABLE posts (  post\_id BIGINT PRIMARY KEY,  user\_id BIGINT NOT NULL,  content TEXT,  created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  post\_type VARCHAR(50), *-- Type of post (text, image, video, etc.)*  visibility VARCHAR(20), *-- Public, private, etc.*  FOREIGN KEY (user\_id) REFERENCES users(user\_id)  )  PARTITION BY RANGE (created\_at) (  PARTITION p\_2024\_11\_29\_00 VALUES LESS THAN ('2024-11-29 01:00:00'),  PARTITION p\_2024\_11\_29\_01 VALUES LESS THAN ('2024-11-29 02:00:00'),  *-- Add partitions for* |

**Comments Table**

Stores comments made on posts. This will also be partitioned by created\_at to handle high activity.

.

|  |
| --- |
| CREATE TABLE comments (  comment\_id BIGINT PRIMARY KEY,  post\_id BIGINT NOT NULL,  user\_id BIGINT NOT NULL,  content TEXT,  created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  FOREIGN KEY (post\_id) REFERENCES posts(post\_id),  FOREIGN KEY (user\_id) REFERENCES users(user\_id)  )  PARTITION BY RANGE (created\_at) (  PARTITION c\_2024\_11\_29\_00 VALUES LESS THAN ('2024-11-29 01:00:00'),  PARTITION c\_2024\_11\_29\_01 VALUES LESS THAN ('2024-11-29 02:00:00'),  *-- Add partitions for each hour or use* |

**Likes Table**

Stores likes on posts by users. We will also partition this table by time for similar reasons.

|  |
| --- |
| CREATE TABLE likes (  like\_id BIGINT PRIMARY KEY,  post\_id BIGINT NOT NULL,  user\_id BIGINT NOT NULL,  created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  FOREIGN KEY (post\_id) REFERENCES posts(post\_id),  FOREIGN KEY (user\_id) REFERENCES users(user\_id)  )  PARTITION BY RANGE (created\_at) (  PARTITION l\_2024\_11\_29\_00 VALUES LESS THAN ('2024-11-29 01:00:00'),  PARTITION l\_2024\_11\_29\_01 VALUES LESS THAN ('2024-11-29 02:00:00'),  *-- Add partitions for each hour or use dynamic partitioning*  ); |

**Shares Table**

Stores information about shares of posts by users.

|  |
| --- |
| TABLE shares (  share\_id BIGINT PRIMARY KEY,  post\_id BIGINT NOT NULL,  user\_id BIGINT NOT NULL,  created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  FOREIGN KEY (post\_id) REFERENCES posts(post\_id),  FOREIGN KEY (user\_id) REFERENCES users(user\_id)  )  PARTITION BY RANGE (created\_at) (  PARTITION s\_2024\_11\_29\_00 VALUES LESS THAN ('2024-11-29 01:00:00'),  PARTITION s\_2024\_11\_29\_01 VALUES LESS THAN ('2024-11-29 02:00:00'),  *-- Add partitions for each hour or use dynamic* |

**2. Indexes for Performance**

To ensure optimal query performance, we will create indexes on commonly queried columns, such as user\_id, post\_id, and created\_a.

|  |
| --- |
| *-- Indexes on posts for faster retrieval by user*  CREATE INDEX idx\_posts\_user\_id ON posts (user\_id);  CREATE INDEX idx\_posts\_created\_at ON posts (created\_at);  *-- Indexes on comments for faster retrieval by post and user*  CREATE INDEX idx\_comments\_post\_id ON comments (post\_id);  CREATE INDEX idx\_comments\_user\_id ON comments (user\_id);  CREATE INDEX idx\_comments\_created\_at ON comments (created\_at);  *-- Indexes on likes for faster retrieval by post and user*  CREATE INDEX idx\_likes\_post\_id ON likes (post\_id);  CREATE INDEX idx\_likes\_user\_id ON likes (user\_id);  CREATE INDEX idx\_likes\_created\_at ON likes (created\_at);  *-- Indexes on shares for faster retrieval by post and user*  CREATE INDEX idx\_shares\_post\_id ON shares (post\_id);  CREATE INDEX idx\_shares\_user\_id ON shares (user\_id);  CREATE INDEX idx\_shares\_created\_at ON shares (created\_at); |

This query will give us the number of readings recorded for each sensor, grouped by day.

**3. Views for Reporting and Analytics**

We will create views or materialized views to summarize engagement metrics and improve query performance for reporting purposes.

**Engagement Summary View**

A view to summarize the engagement for each post (likes, comments, and shares).

|  |
| --- |
| **CREATE VIEW engagement\_summary AS**  **SELECT**  **p.post\_id,**  **p.user\_id,**  **COUNT(l.like\_id) AS total\_likes,**  **COUNT(c.comment\_id) AS total\_comments,**  **COUNT(s.share\_id) AS total\_shares**  **FROM**  **posts p**  **LEFT JOIN**  **likes l ON p.post\_id = l.post\_id**  **LEFT JOIN**  **comments c ON p.post\_id = c.post\_id**  **LEFT JOIN**  **shares s ON p.post\_id = s.post\_id**  **GROUP BY**  **p.post\_id, p.user\_id;** |

**Detecting Spikes in Activity**

To detect spikes in activity, we can write a query that identifies time periods where engagement metrics (like likes, comments, and shares) deviate significantly from the average. For simplicity, we could check for periods with significantly higher engagement than average.

|  |
| --- |
| WITH avg\_engagement AS (  SELECT  hour,  AVG(total\_likes) AS avg\_likes,  AVG(total\_comments) AS avg\_comments,  AVG(total\_shares) AS avg\_shares  FROM hourly\_engagement  GROUP BY hour  )  SELECT  h.hour,  h.total\_likes,  h.total\_comments,  h.total\_shares  FROM  hourly\_engagement h  JOIN  avg\_engagement a ON h.hour = a.hour  WHERE  h.total\_likes > a.avg\_likes \* 1.5  OR h.total\_comments > a.avg\_comments \* 1.5  OR h.total\_shares > a.avg\_shares \* 1.5  ORDER BY h.hour; |

This query provides the average temperature for each location (e.g., area, city) and orders them by temperature.

**Partition Management**

For real-time data processing, you'll want to implement automated partition management, which can be done via scheduling jobs that create new partitions every hour or day. This helps to manage the data lifecycle effectively, ensuring that old data is archived or purged as needed, and the system continues to perform optimally.

This schema and approach should enable efficient storage, querying, and reporting on large volumes of real-time social media user data, while also providing flexibility to detect trends and spikes in activity.

**CONCLUSION:**

In conclusion, the database design for handling large volumes of real-time social media data must prioritize scalability, efficient querying, and timely reporting. By partitioning tables based on time intervals (such as hourly or daily), the system ensures that the data is well-organized for performance optimization, particularly when dealing with high-frequency user interactions like posts, comments, likes, and shares. The use of indexed and partitioned tables allows for faster retrieval and aggregation of engagement metrics, making it easier to detect trends and spikes in activity.

In addition, creating views or materialized views to summarize key metrics (like user engagement, likes per post, or share frequency) ensures that analytical queries run efficiently by precomputing results that are commonly used for reporting. This approach minimizes computational load during peak activity periods and provides insights into user behavior and engagement across various demographics.

By implementing these design principles, the database will be capable of handling large amounts of real-time data while providing quick, actionable insights into user engagement trends, ensuring both scalability and performance for social media analytics.