-- 10. Social Media Analytics Dashboard 📱

-- Skills Used: SQL Aggregations, Window Functions, Data Warehousing

-- • Tables: Users, Posts, Likes, Comments, Shares

-- • Features:

-- o Track user engagement metrics.

-- o Generate reports on trending posts, top influencers.

-- o Analyze user activity over time.

-- • Advanced: Use BigQuery or Snowflake for large-scale analytics.

-- 1. Users

CREATE TABLE Users (

user\_id INT PRIMARY KEY,

username VARCHAR(50),

email VARCHAR(100),

signup\_date DATE

);

-- 2. Posts

CREATE TABLE Posts (

post\_id INT PRIMARY KEY,

user\_id INT,

content TEXT,

post\_date DATETIME,

FOREIGN KEY (user\_id) REFERENCES Users(user\_id)

);

-- 3. Likes

CREATE TABLE Likes (

like\_id INT PRIMARY KEY,

post\_id INT,

user\_id INT,

liked\_at DATETIME,

FOREIGN KEY (post\_id) REFERENCES Posts(post\_id),

FOREIGN KEY (user\_id) REFERENCES Users(user\_id)

);

-- 4. Comments

CREATE TABLE Comments (

comment\_id INT PRIMARY KEY,

post\_id INT,

user\_id INT,

comment TEXT,

commented\_at DATETIME,

FOREIGN KEY (post\_id) REFERENCES Posts(post\_id),

FOREIGN KEY (user\_id) REFERENCES Users(user\_id)

);

-- 5. Shares

CREATE TABLE Shares (

share\_id INT PRIMARY KEY,

post\_id INT,

user\_id INT,

shared\_at DATETIME,

FOREIGN KEY (post\_id) REFERENCES Posts(post\_id),

FOREIGN KEY (user\_id) REFERENCES Users(user\_id)

);

-- 1. User Engagement Metrics

SELECT

u.user\_id,

u.username,

COUNT(DISTINCT p.post\_id) AS total\_posts,

COUNT(DISTINCT l.like\_id) AS total\_likes,

COUNT(DISTINCT c.comment\_id) AS total\_comments,

COUNT(DISTINCT s.share\_id) AS total\_shares

FROM Users u

LEFT JOIN Posts p ON u.user\_id = p.user\_id

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 u.user\_id, u.username;

-- 2. Trending Posts (last 7 days)

SELECT

p.post\_id,

u.username,

p.content,

COUNT(DISTINCT l.like\_id) + COUNT(DISTINCT c.comment\_id) + COUNT(DISTINCT s.share\_id) AS engagement\_score

FROM Posts p

JOIN Users u ON p.user\_id = u.user\_id

LEFT JOIN Likes l ON p.post\_id = l.post\_id AND l.liked\_at >= NOW() - INTERVAL 7 DAY

LEFT JOIN Comments c ON p.post\_id = c.post\_id AND c.commented\_at >= NOW() - INTERVAL 7 DAY

LEFT JOIN Shares s ON p.post\_id = s.post\_id AND s.shared\_at >= NOW() - INTERVAL 7 DAY

WHERE p.post\_date >= NOW() - INTERVAL 7 DAY

GROUP BY p.post\_id, u.username, p.content

ORDER BY engagement\_score DESC

LIMIT 10;

-- 3. Top Influencers (Window Function)

SELECT

user\_id,

username,

total\_engagement,

RANK() OVER (ORDER BY total\_engagement DESC) AS influencer\_rank

FROM (

SELECT

u.user\_id,

u.username,

COUNT(l.like\_id) + COUNT(c.comment\_id) + COUNT(s.share\_id) AS total\_engagement

FROM Users u

LEFT JOIN Posts p ON u.user\_id = p.user\_id

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 u.user\_id, u.username

) AS engagement\_data;

INSERT INTO Users (user\_id, username, email, signup\_date) VALUES

(1, 'akash123', 'akash@example.com', '2023-01-15'),

(2, 'jane\_doe', 'jane@example.com', '2023-03-10'),

(3, 'techguy', 'tech@example.com', '2023-06-25'),

(4, 'naturelover', 'nature@example.com', '2023-08-01');

INSERT INTO Posts (post\_id, user\_id, content, post\_date) VALUES

(101, 1, 'Just finished my new SQL project!', '2025-04-01 10:00:00'),

(102, 2, 'Loving the new features on this platform.', '2025-04-03 14:25:00'),

(103, 3, 'Check out my latest tech review!', '2025-04-04 09:10:00'),

(104, 1, 'SQL tips and tricks coming soon...', '2025-04-05 18:20:00'),

(105, 4, 'Nature walks are the best therapy.', '2025-04-06 07:50:00');

INSERT INTO Likes (like\_id, post\_id, user\_id, liked\_at) VALUES

(201, 101, 2, '2025-04-01 11:00:00'),

(202, 101, 3, '2025-04-01 11:10:00'),

(203, 102, 1, '2025-04-03 15:00:00'),

(204, 103, 2, '2025-04-04 10:00:00'),

(205, 104, 3, '2025-04-05 20:00:00'),

(206, 105, 1, '2025-04-06 09:00:00');

INSERT INTO Comments (comment\_id, post\_id, user\_id, comment, commented\_at) VALUES

(301, 101, 2, 'Awesome project, Akash!', '2025-04-01 12:00:00'),

(302, 102, 3, 'Totally agree!', '2025-04-03 16:00:00'),

(303, 103, 1, 'Great review!', '2025-04-04 11:00:00'),

(304, 105, 2, 'So peaceful!', '2025-04-06 08:15:00');

INSERT INTO Shares (share\_id, post\_id, user\_id, shared\_at) VALUES

(401, 101, 3, '2025-04-01 13:00:00'),

(402, 103, 2, '2025-04-04 12:30:00'),

(403, 104, 4, '2025-04-05 21:00:00'),

(404, 105, 1, '2025-04-06 10:00:00');

SELECT \* FROM Users;

SELECT \* FROM Posts;

SELECT \* FROM Likes;

SELECT \* FROM Comments;

SELECT \* FROM Shares;

-- 2. E-Commerce Database System 🛒

-- Skills Used: SQL Queries, Normalization, Transactions

-- • Tables: Users, Orders, Products, Payments, Shipments

-- • Features:

-- o Users can place orders.

-- o Payment processing and order status tracking.

-- o Generate reports for monthly sales, top-selling products.

-- • Advanced: Implement indexing and query optimization for performance.

-- 1. Create the Database

drop database if exists ecommerce\_db;

create database ecommerce\_db;

use ecommerce\_db;

-- 2. Create Tables

CREATE TABLE Users (

user\_id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

password VARCHAR(255) NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE Products (

product\_id INT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

description TEXT,

price DECIMAL(10,2) NOT NULL,

stock INT NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE Orders (

order\_id INT AUTO\_INCREMENT PRIMARY KEY,

user\_id INT,

order\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

status ENUM('Pending', 'Shipped', 'Delivered', 'Cancelled') DEFAULT 'Pending',

total\_amount DECIMAL(10,2) NOT NULL,

FOREIGN KEY (user\_id) REFERENCES Users(user\_id) ON DELETE CASCADE

);

CREATE TABLE OrderDetails (

order\_detail\_id INT AUTO\_INCREMENT PRIMARY KEY,

order\_id INT,

product\_id INT,

quantity INT NOT NULL,

subtotal DECIMAL(10,2) NOT NULL,

FOREIGN KEY (order\_id) REFERENCES Orders(order\_id) ON DELETE CASCADE,

FOREIGN KEY (product\_id) REFERENCES Products(product\_id) ON DELETE CASCADE

);

CREATE TABLE Payments (

payment\_id INT AUTO\_INCREMENT PRIMARY KEY,

order\_id INT,

payment\_method ENUM('Credit Card', 'PayPal', 'Bank Transfer') NOT NULL,

payment\_status ENUM('Pending', 'Completed', 'Failed') DEFAULT 'Pending',

transaction\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (order\_id) REFERENCES Orders(order\_id) ON DELETE CASCADE

);

CREATE TABLE Shipments (

shipment\_id INT AUTO\_INCREMENT PRIMARY KEY,

order\_id INT,

shipment\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

tracking\_number VARCHAR(50) UNIQUE,

carrier VARCHAR(50),

status ENUM('In Transit', 'Delivered', 'Failed') DEFAULT 'In Transit',

FOREIGN KEY (order\_id) REFERENCES Orders(order\_id) ON DELETE CASCADE

);

-- 3. Sample Queries

-- Insert Users

INSERT INTO Users (name, email, password) VALUES

('John Doe', 'john@example.com', 'hashed\_password'),

('Jane Smith', 'jane@example.com', 'hashed\_password');

-- Insert Products

INSERT INTO Products (name, description, price, stock) VALUES

('Laptop', 'Gaming Laptop', 1200.00, 10),

('Smartphone', 'Latest Model', 800.00, 15);

-- Place an Order

INSERT INTO Orders (user\_id, total\_amount) VALUES (1, 2000.00);

SET @order\_id = LAST\_INSERT\_ID();

INSERT INTO OrderDetails (order\_id, product\_id, quantity, subtotal) VALUES (@order\_id, 1, 1, 1200.00), (@order\_id, 2, 1, 800.00);

-- Process Payment

INSERT INTO Payments (order\_id, payment\_method, payment\_status) VALUES (@order\_id, 'Credit Card', 'Completed');

-- Ship Order

INSERT INTO Shipments (order\_id, tracking\_number, carrier, status) VALUES (@order\_id, 'TRACK12345', 'FedEx', 'In Transit');

-- Reports

-- Monthly Sales Report

SELECT DATE\_FORMAT(order\_date, '%Y-%m') AS month, SUM(total\_amount) AS total\_sales

FROM Orders WHERE status = 'Delivered' GROUP BY month;

-- Top-Selling Products

SELECT p.name, SUM(od.quantity) AS total\_sold

FROM OrderDetails od

JOIN Products p ON od.product\_id = p.product\_id

GROUP BY p.product\_id

ORDER BY total\_sold DESC LIMIT 5;

-- 4. Performance Optimization

CREATE INDEX idx\_orders\_user ON Orders(user\_id);

CREATE INDEX idx\_orderdetails\_product ON OrderDetails(product\_id);

CREATE INDEX idx\_payments\_order ON Payments(order\_id);

CREATE INDEX idx\_shipments\_order ON Shipments(order\_id);

-- 8. Customer Relationship Management (CRM) System 📊

-- Skills Used: SQL Queries, Data Warehousing, Reporting

-- • Tables: Customers, Sales, Support Tickets, Feedback

-- • Features:

-- o Track customer interactions and purchases.

-- o Analyze customer feedback and sales trends.

-- o Generate reports for sales performance.

-- • Advanced: Implement data warehouse concepts for analytical reporting.

-- Tools & Technologies:

-- Database: PostgreSQL / MySQL / SQL Server

-- ETL Tool (Optional for DWH): Apache Airflow / Python Scripts

-- BI Tool (Optional for reports): Power BI / Tableau / Excel

-- Data Warehouse Concepts: Star Schema, Fact & Dimension Tables

CREATE DATABASE CRM\_System;

USE CRM\_System;

-- Database Tables:

-- 1. Customers

CREATE TABLE Customers (

customer\_id INT PRIMARY KEY,

name VARCHAR(100),

email VARCHAR(100),

phone VARCHAR(20),

location VARCHAR(100),

created\_at DATE

);

-- 2. Sales

CREATE TABLE Sales (

sale\_id INT PRIMARY KEY,

customer\_id INT,

product VARCHAR(100),

amount DECIMAL(10, 2),

sale\_date DATE,

FOREIGN KEY (customer\_id) REFERENCES Customers(customer\_id)

);

-- 3. Support\_Tickets

CREATE TABLE Support\_Tickets (

ticket\_id INT PRIMARY KEY,

customer\_id INT,

issue VARCHAR(255),

status VARCHAR(20),

created\_at DATE,

resolved\_at DATE,

FOREIGN KEY (customer\_id) REFERENCES Customers(customer\_id)

);

-- 4. Feedback

CREATE TABLE Feedback (

feedback\_id INT PRIMARY KEY,

customer\_id INT,

rating INT CHECK (rating BETWEEN 1 AND 5),

comments TEXT,

submitted\_at DATE,

FOREIGN KEY (customer\_id) REFERENCES Customers(customer\_id)

);

INSERT INTO Customers (customer\_id, name, email, phone, location, created\_at) VALUES

(1, 'Alice Johnson', 'alice@example.com', '9876543210', 'New York', '2024-01-10'),

(2, 'Bob Smith', 'bob@example.com', '9123456780', 'Los Angeles', '2024-02-15'),

(3, 'Charlie Lee', 'charlie@example.com', '9988776655', 'Chicago', '2024-03-20'),

(4, 'Diana Rose', 'diana@example.com', '9090909090', 'Houston', '2024-04-05'),

(5, 'Ethan Brown', 'ethan@example.com', '9234567890', 'Phoenix', '2024-05-18');

INSERT INTO Sales (sale\_id, customer\_id, product, amount, sale\_date) VALUES

(101, 1, 'Laptop', 1200.00, '2024-02-01'),

(102, 2, 'Smartphone', 800.00, '2024-03-05'),

(103, 3, 'Tablet', 450.50, '2024-04-10'),

(104, 1, 'Headphones', 150.75, '2024-04-15'),

(105, 4, 'Smartwatch', 299.99, '2024-05-22');

INSERT INTO Support\_Tickets (ticket\_id, customer\_id, issue, status, created\_at, resolved\_at) VALUES

(201, 1, 'Laptop not charging', 'Resolved', '2024-02-05', '2024-02-06'),

(202, 2, 'Smartphone screen cracked', 'Open', '2024-03-06', NULL),

(203, 3, 'Tablet not turning on', 'Resolved', '2024-04-11', '2024-04-13'),

(204, 4, 'Smartwatch sync issue', 'Pending', '2024-05-23', NULL),

(205, 1, 'Headphones sound issue', 'Resolved', '2024-04-16', '2024-04-18');

INSERT INTO Feedback (feedback\_id, customer\_id, rating, comments, submitted\_at) VALUES

(301, 1, 5, 'Excellent service!', '2024-02-07'),

(302, 2, 3, 'Product is okay, but delivery was late.', '2024-03-10'),

(303, 3, 4, 'Tablet is good but battery could be better.', '2024-04-15'),

(304, 4, 2, 'Not happy with support response time.', '2024-05-25'),

(305, 5, 5, 'Amazing experience, great value!', '2024-06-01');

-- Features Implementation (SQL Queries):

-- 1. Track Customer Interactions and Purchases

-- Get recent interactions for a customer

SELECT c.name, s.product, s.amount, s.sale\_date, t.issue, t.status

FROM Customers c

LEFT JOIN Sales s ON c.customer\_id = s.customer\_id

LEFT JOIN Support\_Tickets t ON c.customer\_id = t.customer\_id

WHERE c.customer\_id = 101;

-- 2. Analyze Customer Feedback and Sales Trends

-- 1. Average Customer Feedback by Location

SELECT location, AVG(rating) AS avg\_rating

FROM Customers c

JOIN Feedback f ON c.customer\_id = f.customer\_id

GROUP BY location;

-- 2. Monthly Sales Trend

SELECT DATE\_FORMAT(sale\_date, '%Y-%m-01') AS month, SUM(amount) AS total\_sales

FROM Sales

GROUP BY month

ORDER BY month;

-- 3. Generate Sales Performance Reports

-- Top 5 customers by total purchase

SELECT c.name, SUM(s.amount) AS total\_spent

FROM Customers c

JOIN Sales s ON c.customer\_id = s.customer\_id

GROUP BY c.name

ORDER BY total\_spent DESC

LIMIT 5;