**ASSIGNMENT\_2**

**NAME**: D. SAIMANIKANTA

**REG NO**: 192372348

**COURSE**:DATA BASE MANAGEMENT

SYSTEM

**CODE**:CSA0593

**SCENERIO**:

**Design a database to manage support tickets, customers, agents, and resolutions.**

**- Model tables for tickets, customers, agents, and resolutions.**

**- Write stored procedures for opening, updating, and resolving tickets.**

**- Implement triggers to update ticket statuses and agent assignments.**

**- Write SQL queries to analyze ticket resolution times and customer satisfactio**

**1. Database Schema Design**

We'll design tables to store data related to customers, agents, support tickets, and resolutions.

**Table 1: Customers**

This table stores customer information.

|  |
| --- |
| CREATE TABLE Customers (  customer\_iIDSERIAL PRIMARY KEY,  first\_name VARCHAR(255),  last\_name VARCHAR(255),  email VARCHAR(255) UNIQUE,  phone VARCHAR(50),  created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  ); |

**Agents**

This table stores agent information.

|  |
| --- |
| CREATE TABLE Agents (  agent\_id SERIAL PRIMARY KEY,  first\_name VARCHAR(255),  last\_name VARCHAR(255),  email VARCHAR(255) UNIQUE,  status VARCHAR(50), *-- e.g., 'active', 'on\_leave'*  created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP |

**Table 3: Tickets**

|  |
| --- |
| CREATE TABLE Tickets (  ticket\_id SERIAL PRIMARY KEY,  customer\_id INT REFERENCES Customers(customer\_id),  agent\_id INT REFERENCES Agents(agent\_id), *-- Agent assigned to the ticket*  title VARCHAR(255),  description TEXT,  status VARCHAR(50), *-- e.g., 'open', 'in\_progress', 'resolved'*  priority VARCHAR(50), *-- e.g., 'low', 'medium', 'high'*  created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,  updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP |

This table stores the support tickets submitted by customers.

**Table 4: Resolutions**

This table stores details about how each ticket was resolved.

|  |
| --- |
| CREATE TABLE Resolutions (  resolution\_id SERIAL PRIMARY KEY,  ticket\_id INT REFERENCES Tickets(ticket\_id),  resolution\_description TEXT,  resolved\_by INT REFERENCES Agents(agent\_id),  resolution\_time TIMESTAMP DEFAULT CURRENT\_TIMESTAMP |

**2. Stored Procedures**

**Opening a Ticket**

This stored procedure will be used when a customer submits a new support ticket.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE open\_ticket(  p\_customer\_id INT,  p\_agent\_id INT,  title VARCHAR,  p\_description TEXT,  p\_priority VARCHAR)  LANGUAGE plpgsql  AS $$  BEGIN  INSERT INTO Tickets(customer\_id, agent\_id, title, description, priority, status)  END; |

|  |  |
| --- | --- |
| |  | | --- | |  |   SELECT t.ticket\_id,  Tatitlek,  EXTRACT(EPOCH FROM (r.resolution\_time - t.created\_at)) / 3600 AS resolution\_hours  FROM Tickets t  JOIN Resolutions r ON t.ticket\_id = r.ticket\_id  WHERE t.status = 'resolved'; |

**Updating a Ticket**

This stored procedure will be used when a ticket’s details need to be updated (e.g., changing the status or description).

|  |
| --- |
| CREATE OR REPLACE PROCEDURE update\_ticket(  p\_ticket\_id INT,  p\_status VARCHAR,  p\_description TEXT)  LANGUAGE plpgsql  AS $$  BEGIN  UPDATE Tickets  SET status = p\_status,  description = p\_description,  updated\_at = CURRENT\_TIMESTAMP  WHERE ticket\_id = p\_ticket\_id;  END; |

**Resolving a Ticket**

This stored procedure will handle ticket resolution

Copy code

**4. SQL Queries for Reporting and Analytics**

**1. Ticket Resolution Time**

To calculate the resolution time for each ticket, we will calculate the difference between the time the ticket was created and the time it was resolved.

|  |
| --- |
| SELECT t.ticket\_id,  t.title,  EXTRACT(EPOCH FROM (r.resolution\_time - t.created\_at)) / 3600 AS resolution\_hours  FROM Tickets t  JOIN Resolutions r ON t.ticket\_id = r.ticket\_id  WHERE t.status = 'resolved'; |

**Customer Satisfaction (Hypothetical)**

Assuming that customer satisfaction ratings are collected after a ticket is resolved, a customer\_satisfaction table could store customer feedback. The query below calculates the average customer satisfaction for resolved tickets.

sql

Copy code

|  |
| --- |
| SELECT t.ticket\_id,  AVG(cs.rating) AS avg\_satisfaction  FROM Tickets t  JOIN CustomerSatisfaction cs ON t.ticket\_id = cs.ticket\_id  WHERE t.status = 'resolved'  GROUP BY t.ticket\_id; |

**Agent Performance Report**

This query shows the number of tickets assigned to each agent and their resolution time on average.

|  |
| --- |
| SELECT a.agent\_id,  a.first\_name,  a.last\_name,  COUNT(t.ticket\_id) AS assigned\_tickets,  AVG(EXTRACT(EPOCH FROM (r.resolution\_time - t.created\_at)) / 3600) AS avg\_resolution\_time\_hours  FROM Agents a  JOIN Tickets t ON a.agent\_id = t.agent\_id  JOIN Resolutions r ON t.ticket\_id = r.ticket\_id  WHERE t.status = 'resol |

**Materialized Views**

To improve the performance of frequent reporting queries (e.g., resolution times or agent performance), materialized views can be created:

sql

Copy code

|  |
| --- |
| CREATE MATERIALIZED VIEW agent\_performance\_report AS  SELECT a.agent\_id,  a.first\_name,  a.last\_name,  COUNT(t.ticket\_id) AS assigned\_tickets,  AVG(EXTRACT(EPOCH FROM (r.resolution\_time - t.created\_at)) / 3600) AS avg\_resolution\_time\_hours  FROM Agents a  JOIN Tickets t ON a.agent\_id = t.agent\_id  JOIN Resolutions r ON t.ticket\_id = r.ticket\_id  WHERE t.status = 'resolved'  GROUP BY a.agent\_id; |

**Conclusion**

The proposed system design includes the following components:

* **Tables**: For customers, agents, tickets, and resolutions.
* **Stored Procedures**: For opening, updating, and resolving tickets.
* **Triggers**: To automatically update ticket statuses and agent assignments.
* **SQL Queries**: To analyze ticket resolution times and agent performance.
* **Partitioning**: To efficiently manage large volumes of ticket data.
* **Materialized Views**: To improve performance for frequent reports.

This design ensures efficient management of support tickets and helps in analyzing key metrics for performance improvement.