Big Data Management and Analysis

Project Report

On

Database Management Using MySQL

Submitted to:

Prof (Dr.) Amarnath Mitra

Submitted By:

Preksha Verma (055032)

Suvra Datta Banik (055049)

PGDM Big Data Analytics-05 | Section K

1. Project Vision

The **Business Resource and Operations Management System (BROMS)** is designed to create a **centralized, efficient, and high-performing** database system that seamlessly integrates various business operations. The goal is to **enhance automation, reduce manual workload, and optimize decision-making** by providing a structured and scalable solution for managing customers, employees, sales, inventory, projects, and financial transactions.

2. Key Objectives of BROMS

2.1. Develop a Centralized and Structured Database Architecture

- Design a **well-structured relational database** that consolidates business data into a **single, easily accessible system**.
- Establish **clear entity relationships** to ensure efficient data organization and retrieval.
- Enable **seamless interconnectivity** between different business modules, reducing data silos.

2.2. Implement a High-Performance Transaction Management System

- Develop **optimized data processing mechanisms** to handle **bulk inserts, updates, and complex queries**efficiently.
- Utilize **transaction control (ACID compliance) to ensure consistency, atomicity, and integrity** of all business records.
- Minimize **latency and processing delays** even under high transactional loads.

2.3. Automate Business Workflows to Reduce Manual Effort

- Introduce **automated data entry, order processing, and reporting features** to minimize human intervention.
- Implement **scheduled tasks for financial reports, employee performance tracking, and inventory restocking**.
- Provide a **dynamic, rule-based automation system** to adapt workflows to changing business needs.

2.4. Enhance Real-Time Data Accessibility and Reporting

- Develop **dynamic reporting functionalities** that provide **instant access to critical business insights**.
- Integrate **real-time dashboards** for monitoring financial performance, employee productivity, and order fulfillment.
- Enable **customized data exports and visual analytics tools** for strategic decision-making.

2.5. Ensure Robust Data Security and Access Control

- Implement **role-based access controls (RBAC)** to protect sensitive information from unauthorized users.
- Use **encryption and authentication mechanisms** to secure financial transactions and employee records.
- Enable **audit trails and activity logs** for monitoring system usage and compliance requirements.

2.6. Design for Future Scalability and Business Expansion

- Build a **scalable infrastructure** capable of handling **growing transaction volumes and user loads**.
- Ensure **modular design** to facilitate the integration of **new business functions, such as AI-based analytics and predictive modeling**.
- Enable **multi-location support** for businesses operating in multiple geographic regions.

2.7. Optimize Data Storage and Retrieval for High Efficiency

- Implement **efficient indexing and query optimization techniques** to enhance system performance.
- Use **data normalization and partitioning strategies** to ensure storage efficiency and fast query execution.
- Leverage **advanced caching mechanisms** to reduce redundant database calls.

2.8. Enable Seamless Integration with External Systems

- Provide **APIs and integration capabilities** for third-party software such as **ERP, CRM, and accounting tools**.
- Ensure compatibility with **cloud-based and on-premise infrastructure** for flexible deployment options.
- Support **real-time synchronization** with payment gateways, suppliers, and logistics partners.

2.9. Ensure System Resilience and Business Continuity

- Implement **failover and backup mechanisms** to protect against data loss and system failures.
- Develop a **disaster recovery plan**, including **automated database backups and restore options**.
- Conduct **periodic performance and stress testing** to identify and address potential bottlenecks.

3. Tools Used in the BROMS Project

3.1. **Structured Query Language (SQL)** – Used as the core language for creating, managing, and manipulating the database, including executing CRUD operations, stored procedures, triggers, and stress testing queries.

3.2. **MySQL** – The primary relational database management system (RDBMS) used for handling structured business data, ensuring ACID compliance, high performance, and scalability for large-scale transactions.

3.3. **MySQL Workbench** – Utilized for database design, schema modeling, query execution, and performance optimization. It also helped in ERD creation, debugging, and indexing strategies.

3.4. **Microsoft Word / Google Docs** – Used for documenting project requirements, database schemas, system workflows, test results, and managerial reports to ensure structured project documentation and communication.

3.5. **GitHub** – Employed for version control, code collaboration, and tracking changes in SQL scripts, ensuring safe deployment, rollback options, and seamless teamwork among developers and database administrators.

4. Managerial Implications: Strategic Insights for Business Leaders

The Business Resource and Operations Management System (BROMS) plays a vital role in supporting key business functions such as sales, human resource management, financial tracking, and project execution. The results of stress testing provide business leaders, IT managers, and operational executives with critical insights into the scalability, efficiency, and reliability of the system under high transactional loads.

The findings from the stress tests have several important managerial implications:

4.1. Ensuring System Reliability and Data Integrity

A system's ability to maintain data consistency and reliability under heavy usage is critical for business continuity. The successful execution of **100,000+ record insertions and updates**, along with efficient query retrievals, ensures that:

- **Customer orders, employee attendance, financial transactions, and project assignments** are processed without data corruption or delays.
- **Decision-makers can trust the system's outputs**, ensuring that reports, analytics, and business forecasts are accurate and reliable.
- The **risk of database inconsistencies, redundant data, or order failures is minimized**, reducing the potential for operational disruptions.

This validation of data integrity translates into a **smoother customer experience, accurate financial reporting, and efficient resource planning** across all business units.

4.2. Scalability to Support Business Growth

As organizations expand, the volume of business transactions grows exponentially. The stress testing results confirm that BROMS is **capable of scaling to meet increased demands** without performance degradation. This is especially relevant for:

- **Companies experiencing rapid growth** that require a system capable of handling increasing customer orders, employee records, and financial transactions.
- **Businesses planning geographic expansion** that need to ensure that their operational infrastructure can support higher transaction loads across multiple locations.
- **E-commerce and retail industries** where order surges during peak seasons (e.g., holiday sales, promotions) demand a highly **responsive and resilient system**.

The system's **scalability ensures that it remains a long-term solution**, capable of supporting business expansion without requiring major overhauls or system migrations.

4.3. Optimizing Operational Efficiency and Decision-Making

Timely and data-driven decision-making is key to gaining a competitive edge. The successful execution of **bulk SELECT queries (10,000+ complex queries)** in under **0.168 seconds per execution** demonstrates that:

- **Business leaders can access real-time sales reports, customer insights, and financial data** to make informed strategic decisions.
- **Operational managers can quickly review workforce productivity, track order fulfillment, and optimize project budgets** without waiting for slow system responses.
- **Finance and supply chain teams can forecast demand, manage cash flow, and track supplier performance efficiently**.

This reinforces the system's ability to **act as a central hub for business intelligence**, enabling executives to act swiftly based on **real-time data insights**.

4.4. IT Infrastructure Planning and Performance Optimization

The stress test results also provide valuable insights for **IT and database administrators**, highlighting areas for potential performance enhancements. While the system performed well under high loads, future optimizations could include:

- **Enhanced indexing strategies** to improve query performance under extreme workloads.
- **Database partitioning and caching mechanisms** to reduce processing times and optimize storage utilization.
- **Automated load balancing techniques** to distribute transactions efficiently during peak usage periods.

By proactively implementing these performance enhancements, businesses can ensure that BROMS remains **resilient and high-performing** as transactional volumes continue to grow.

4.5. Risk Mitigation and Business Continuity

System failures, transaction delays, and data inconsistencies can have severe business implications, including **financial losses, customer dissatisfaction, and reputational damage**. Stress testing acts as a **preventive measure** to ensure that:

- **The system remains fully operational even under extreme transactional pressure**.
- **Financial data remains secure, with no risks of miscalculations, overcharges, or missing records**.
- **Compliance requirements are met, ensuring adherence to financial regulations, employee records management, and data protection standards**.

By **identifying potential risks and ensuring system robustness**, businesses can **confidently scale operations, launch new products, and expand into new markets** without fear of technological bottlenecks.

5. ERD for the Business Resource and Operations Management System (BROMS)

This ERD includes key features for business management:

5.1.HR Management:

- Employee details and department structure
- Attendance tracking
- Performance reviews
- Project assignments

5.2. Sales System:

- Customer information
- Sales orders and invoicing
- Product ordering
- Order tracking

5.3. Inventory Control:

- Product management
- Supplier information
- Stock tracking
- Warehouse locations

5.4. Project Management:

- Project details
- Resource allocation
- Budget tracking
- Team assignments

6. Entity relationships:

Here's a breakdown of the entities and their relationships:

6.1. **CUSTOMER**

- Primary Key (PK): CUST_ID
- Attributes:
  - CUST_NAME (string)
  - CONTACT (string)
  - ADDRESS (string)
- Relationships:
  - Places: Customers place sales orders.

6.2. SUPPLIER

- Primary Key (PK): SUPP_ID
- Attributes:
  - SUPP_NAME (string)
  - CONTACT (string)
  - ADDRESS (string)
- Relationships:
  - Provides: Suppliers provide products.

6.3. PRODUCT

- Primary Key (PK): PROD_ID
- Attributes:
  - PROD_NAME (string)
  - SUPP_ID (FK to SUPPLIER)
  - PRICE (float)
  - DESCRIPTION (string)
- Relationships:
  - Is part of: Products are part of sale order items.
  - Associated with: Products are linked to projects through an associative entity.

6.4. SALE_ORDER

- Primary Key (PK): ORDER_ID
- Attributes:
    - CUST_ID (FK to CUSTOMER)
    - ORDER_DATE (date)
    - TOTAL_AMOUNT (float)
- Relationships:
    - Includes: Sale orders include multiple order items.

6.5. ORDER_ITEM

- Primary Key (PK): ITEM_ID
- Attributes:
    - ORDER_ID (FK to SALE_ORDER)
    - PROD_ID (FK to PRODUCT)
    - QUANTITY (int)
    - PRICE (float)
- Relationships:
    - Belongs to: Order items belong to a sale order.

6.6. PROJECT

- Primary Key (PK): PROJ_ID
- Attributes:
    - PROJ_NAME (string)
    - START_DATE (date)
    - END_DATE (date)
    - BUDGET (float)
- Relationships:
    - Contains: Projects have associated milestones.
    - Involves: Employees are assigned to projects.

6.7. PROJECT_MILESTONE

- Primary Key (PK): MILESTONE_ID
- Attributes:
    - PROJ_ID (FK to PROJECT)
    - MILESTONE_NAME (string)
    - DUE_DATE (date)
    - STATUS (string)
- Relationships:
    - Belongs to: Milestones are part of a project.

6.8. EMPLOYEE

- Primary Key (PK): EMP_ID
- Attributes:
    - EMP_NAME (string)
    - DEPT_ID (FK to DEPARTMENT)
    - MGR_ID (FK to EMPLOYEE)
- Relationships:
    - Participates in: Employees are assigned to projects.
    - Manages: Employees may manage other employees.

6.9. DEPARTMENT

- Primary Key (PK): DEPT_ID
- Attributes:
    - DEPT_NAME (string)
- Relationships:
    - Has: A department has employees.

6.10. PROJECT_ASSIGNMENT

- Primary Key (PK): ASSIGNMENT_ID
- Attributes:

    - EMP_ID (FK to EMPLOYEE)
    - PROJ_ID (FK to PROJECT)
    - START_DATE (date)
    - END_DATE (date)
    - ROLE (string)
- Relationships:
    - Links: Links employees to projects.

6.11. PERFORMANCE_REVIEW

- Primary Key (PK): REVIEW_ID
- Attributes:
    - EMP_ID (FK to EMPLOYEE)
    - REVIEW_DATE (date)
    - RATING (string)
    - COMMENTS (string)
- Relationships:
    - Records: Documents employee performance reviews.

6.12. ATTENDANCE

- Primary Key (PK): ATTENDANCE_ID
- Attributes:
    - EMP_ID (FK to EMPLOYEE)
    - DATE (date)
    - STATUS (string)
- Relationships:
    - Tracks: Tracks employee attendance.

Associative Entity: AssociativeEntity

- Primary Key (PK):
    - PROD_ID (FK to PRODUCT)
    - PROJ_ID (FK to PROJECT)
- Purpose: Establishes a many-to-many relation nship between products and projects.

7. **CODE(SQL) -**

```
-- Drop the database if it already exists
DROP DATABASE IF EXISTS SDB_49company_database;


-- Create the database
CREATE DATABASE IF NOT EXISTS SDB_49company_database;
USE SDB_49company_database;


-- Create CUSTOMER table
CREATE TABLE SDB_49CUSTOMER (
   Cust_ID INT PRIMARY KEY,
   Cust_Name VARCHAR(255),
   Contact VARCHAR(255),
   Address VARCHAR(255)
);


-- Create SUPPLIER table
CREATE TABLE SDB_49SUPPLIER (
   Supp_ID INT PRIMARY KEY,
   Supp_Name VARCHAR(255),
   Contact VARCHAR(255),
   Address VARCHAR(255)
```

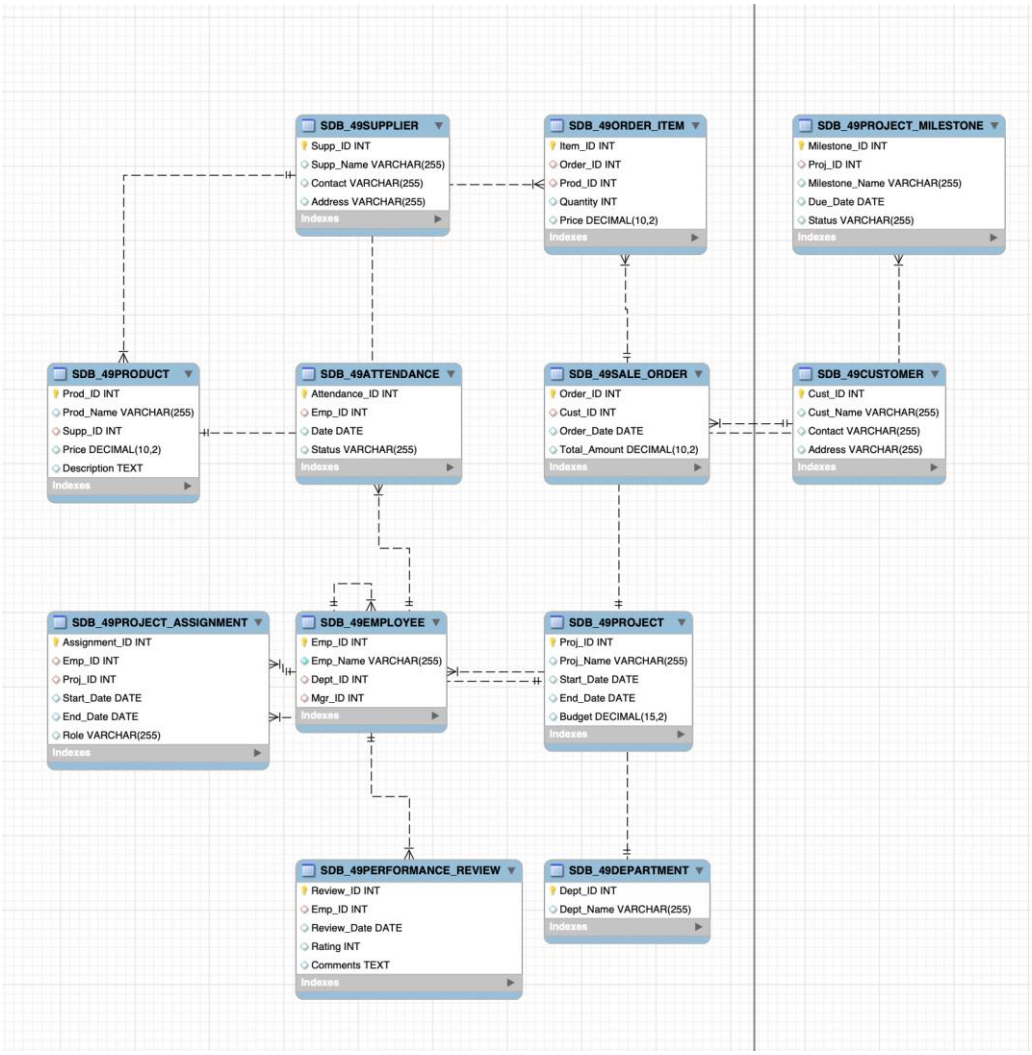
```
);

-- Create PRODUCT table
CREATE TABLE SDB_49PRODUCT (
    Prod_ID INT PRIMARY KEY,
    Prod_Name VARCHAR(255),
    Supp_ID INT,
    Price DECIMAL(10, 2),
    Description TEXT,
    FOREIGN KEY (Supp_ID) REFERENCES SDB_49SUPPLIER(Supp_ID)
);

-- Create SALE_ORDER table
CREATE TABLE SDB_49SALE_ORDER (
    Order_ID INT PRIMARY KEY,
    Cust_ID INT,
    Order_Date DATE,
    Total_Amount DECIMAL(10, 2),
    FOREIGN KEY (Cust_ID) REFERENCES SDB_49CUSTOMER(Cust_ID)
);

-- Create ORDER_ITEM table
CREATE TABLE SDB_49ORDER_ITEM (
    Item_ID INT PRIMARY KEY,
    Order_ID INT,
    Prod_ID INT,
    Quantity INT,
    Price DECIMAL(10, 2),
    FOREIGN KEY (Order_ID) REFERENCES SDB_49SALE_ORDER(Order_ID),
```

```
    FOREIGN KEY (Prod_ID) REFERENCES SDB_49PRODUCT(Prod_ID)
);

-- Create DEPARTMENT table
CREATE TABLE SDB_49DEPARTMENT (
    Dept_ID INT PRIMARY KEY,
    Dept_Name VARCHAR(255)
);

-- Create EMPLOYEE table
CREATE TABLE SDB_49EMPLOYEE (
    Emp_ID INT PRIMARY KEY,
    Emp_Name VARCHAR(255),
    Dept_ID INT,
    Mgr_ID INT,
    FOREIGN KEY (Dept_ID) REFERENCES SDB_49DEPARTMENT(Dept_ID),
    FOREIGN KEY (Mgr_ID) REFERENCES SDB_49EMPLOYEE(Emp_ID)
);

-- Create PROJECT table
CREATE TABLE SDB_49PROJECT (
    Proj_ID INT PRIMARY KEY,
    Proj_Name VARCHAR(255),
    Start_Date DATE,
    End_Date DATE,
    Budget DECIMAL(15, 2)
);

-- Create PROJECT_MILESTONE table
```

```
CREATE TABLE SDB_49PROJECT_MILESTONE (
    Milestone_ID INT PRIMARY KEY,
    Proj_ID INT,
    Milestone_Name VARCHAR(255),
    Due_Date DATE,
    Status VARCHAR(255),
    FOREIGN KEY (Proj_ID) REFERENCES SDB_49PROJECT(Proj_ID)
);

-- Create PROJECT_ASSIGNMENT table
CREATE TABLE SDB_49PROJECT_ASSIGNMENT (
    Assignment_ID INT PRIMARY KEY,
    Emp_ID INT,
    Proj_ID INT,
    Start_Date DATE,
    End_Date DATE,
    Role VARCHAR(255),
    FOREIGN KEY (Emp_ID) REFERENCES SDB_49EMPLOYEE(Emp_ID),
    FOREIGN KEY (Proj_ID) REFERENCES SDB_49PROJECT(Proj_ID)
);

-- Create PERFORMANCE_REVIEW table
CREATE TABLE SDB_49PERFORMANCE_REVIEW (
    Review_ID INT PRIMARY KEY,
    Emp_ID INT,
    Review_Date DATE,
    Rating INT,
    Comments TEXT,
    FOREIGN KEY (Emp_ID) REFERENCES SDB_49EMPLOYEE(Emp_ID)
```

```
);


-- Create ATTENDANCE table
CREATE TABLE SDB_49ATTENDANCE (
    Attendance_ID INT PRIMARY KEY,
    Emp_ID INT,
    Date DATE,
    Status VARCHAR(255),
    FOREIGN KEY (Emp_ID) REFERENCES SDB_49EMPLOYEE(Emp_ID)
);
```

8. ER Diagram



9. CRUD Operations-

The SQL scripts contain various **CRUD (Create, Read, Update, Delete)** operations related to customers, employees, products, projects, suppliers, orders, and attendance management. The operations can be classified as follows:

9.1. Create (INSERT) Operations

These queries add new records into different database tables.

9.1.1. Scenario 1: INSERT a New Customer

Adds a new customer record with Cust_ID, Cust_Name, Contact, and Address into the SDB_49CUSTOMER table.

9.1.2. Scenario 4: INSERT a New Supplier

Inserts a new supplier record into SDB_49SUPPLIER with supplier details.

9.1.3. Scenario 6: INSERT a New Product

Adds a new product into SDB_49PRODUCT with Prod_ID, Prod_Name, Supp_ID, Price, and Description.

9.1.4. Scenario 9: INSERT a New Order

Creates a new sales order entry in SDB_49SALE_ORDER.

9.1.5. Scenario 12: INSERT a New Employee

Inserts an employee record with details such as Emp_ID, Emp_Name, Dept_ID, and Mgr_ID into SDB_49EMPLOYEE.

9.1.6. Scenario 14: INSERT a New Project

Adds a new project with Proj_ID, Proj_Name, Start_Date, End_Date, and Budget into SDB_49PROJECT.

9.1.7. Scenario 17: INSERT Employee Attendance

Adds an attendance record into SDB_49ATTENDANCE with Attendance_ID, Emp_ID, Date, and Status.

9.2. Read (SELECT) Operations

The images do not contain explicit SELECT queries, but these are typically used to **retrieve data** from the database.

9.3. Update (UPDATE) Operations

These queries modify existing records in different tables.

9.3.1. Scenario 2: UPDATE Customer Details

Updates the contact information of a specific customer in SDB_49CUSTOMER.

9.3.2. **Scenario 5: UPDATE Supplier Contact Information**
Modifies the contact email of a supplier in SDB_49SUPPLIER.

9.3.3. **Scenario 7: UPDATE Product Price**
Updates the price of a product in SDB_49PRODUCT.

9.3.4. **Scenario 10: UPDATE Order Amount**
Modifies the total amount of an order in SDB_49SALE_ORDER.

9.3.5. **Scenario 13: UPDATE Employee Department**
Changes the department assignment of an employee in SDB_49EMPLOYEE.

9.3.6. **Scenario 15: UPDATE Project Budget**
Updates the budget of a project in SDB_49PROJECT.

9.3.7. **Scenario 18: UPDATE Attendance Status**
Changes the attendance status of an employee in SDB_49ATTENDANCE.

9.4. Delete (DELETE) Operations

These queries remove existing records from tables.

9.4.1. Scenario 19: DELETE Attendance Entry

Deletes an attendance record from SDB_49ATTENDANCE based on Attendance_ID.

```
-- Scenario 7: UPDATE Product Price --
UPDATE SDB_49PRODUCT
SET Price = 58000.00
WHERE Prod_ID = 1;



-- Scenario 9: INSERT a New Order --
INSERT INTO SDB_49SALE_ORDER (Order_ID, Cust_ID, Order_Date, Total_Amount)
VALUES (21, 21, '2025-01-04', 55000.00);

-- Scenario 10: UPDATE Order Amount --
UPDATE SDB_49SALE_ORDER
SET Total_Amount = 50000.00
WHERE Order_ID = 1;



-- Scenario 12: INSERT a New Employee --
INSERT INTO SDB_49EMPLOYEE (Emp_ID, Emp_Name, Dept_ID, Mgr_ID)
VALUES (5, 'Neha Bansal', 2, 1);

-- Scenario 13: UPDATE Employee Department --
UPDATE SDB_49EMPLOYEE
SET Dept_ID = 5
WHERE Emp_ID = 3;
```

```
-- Scenario 1: INSERT a New Customer --
INSERT INTO SDB_49CUSTOMER (Cust_ID, Cust_Name, Contact, Address)
VALUES (21, 'Deepak Sharma', 'deepak.sharma@email.com', 'Bhopal, India');

-- Scenario 2: UPDATE Customer Details --
UPDATE SDB_49CUSTOMER
SET Contact = 'arjun.newemail@email.com'
WHERE Cust_ID = 1;


-- Scenario 4: INSERT a New Supplier --
INSERT INTO SDB_49SUPPLIER (Supp_ID, Supp_Name, Contact, Address)
VALUES (11, 'Green Energy Appliances', 'contact@greenenergy.com', 'Goa, India');

-- Scenario 5: UPDATE Supplier Contact Information --
UPDATE SDB_49SUPPLIER
SET Contact = 'support@visionnew.com'
WHERE Supp_ID = 6;



-- Scenario 6: INSERT a New Product --
INSERT INTO SDB_49PRODUCT (Prod_ID, Prod_Name, Supp_ID, Price, Description)
VALUES (21, 'Television – Sony Bravia 65 inch', 6, 75000.00, '65 inch, 4K UHD Smart TV');
```

```
-- Scenario 13: UPDATE Employee Department --
UPDATE SDB_49EMPLOYEE
SET Dept_ID = 5
WHERE Emp_ID = 3;

-- Scenario 14: INSERT a New Project --
INSERT INTO SDB_49PROJECT (Proj_ID, Proj_Name, Start_Date, End_Date, Budget)
VALUES (7, 'AI Research Project', '2025-02-01', '2025-12-31', 10000000.00);

-- Scenario 15: UPDATE Project Budget --
UPDATE SDB_49PROJECT
SET Budget = 5500000.00
WHERE Proj_ID = 1;

-- Scenario 17: INSERT Employee Attendance --
INSERT INTO SDB_49ATTENDANCE (Attendance_ID, Emp_ID, Date, Status)
VALUES (21, 5, '2025-01-05', 'Present');

-- Scenario 18: UPDATE Attendance Status --
UPDATE SDB_49ATTENDANCE
SET Status = 'Present'
WHERE Emp_ID = 1 AND Date = '2024-12-02';

-- Scenario 19: DELETE Attendance Entry --
DELETE FROM SDB_49ATTENDANCE
WHERE Attendance_ID = 10;
```

10. Stress testing

10.1. Stress Testing: Bulk INSERT Operation

10.1.1. Overview

Stress testing is a performance testing technique used to evaluate how a system handles a high load of transactions, ensuring database stability and performance under extreme conditions. The SQL script in the image defines a stored procedure named Insert_Stress_Test, which performs a bulk INSERT operation to insert 100,000 customer recordsinto the SDB_49CUSTOMER table.

10.1.2. Test Objective

The goal of this stress test is to:

1. Evaluate Database Performance: Measure how well the database manages high-volume data insertion.
2. Assess Transaction Handling: Observe the efficiency of handling large-scale transactions within a single transaction block.
3. Detect System Limitations: Identify bottlenecks or potential system failures due to excessive data load.

10.1.3. Procedure Breakdown

10.1.3.1. Stored Procedure Creation (Insert_Stress_Test)

Uses a WHILE loop to insert 100,000 records into SDB_49CUSTOMER.
Generates dynamic customer names, emails, and addresses using the CONCAT function.
Each Cust_ID is assigned a unique value by adding an offset of 1000 to i.

10.1.3.2. Transaction Management

START TRANSACTION; is used to ensure atomicity.

The bulk insertions are committed using COMMIT;, ensuring all records are inserted in a single transaction.

10.1.3.3. Execution of the Stress Test

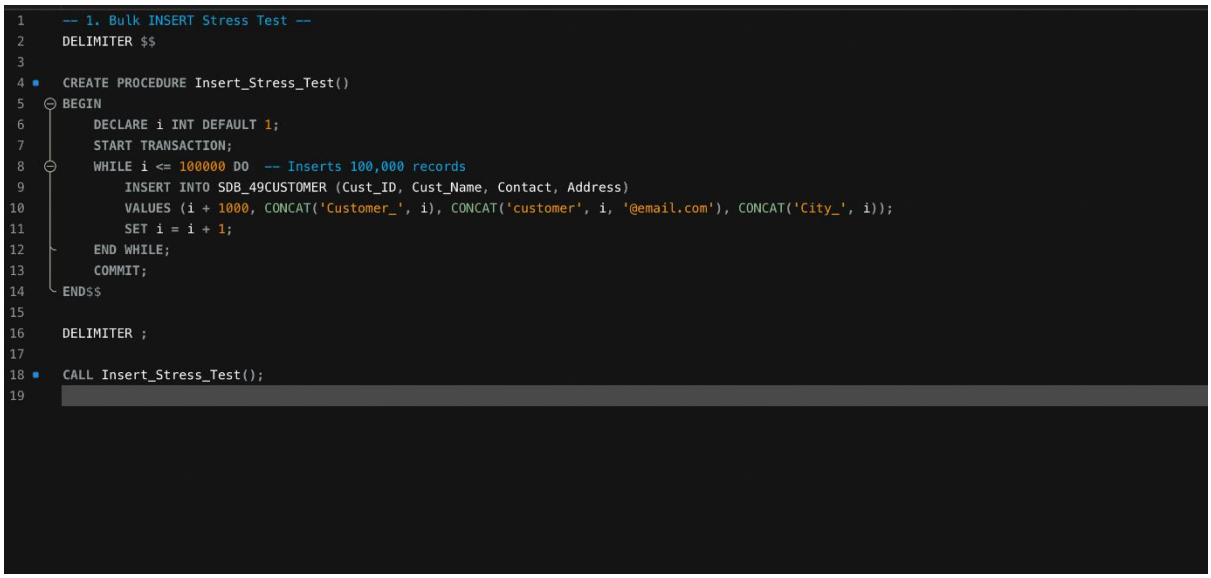
The procedure is invoked using CALL Insert_Stress_Test(); which executes the bulk insert.

10.1.4. Outcomes

- If successful, **100,000 new records** should be inserted efficiently.
- The system's response time, memory consumption, and potential locking issues can be evaluated.
- Performance optimization strategies, such as indexing or batch processing, may be suggested based on test results.

10.1.5. Conclusion

This stress test effectively simulates a **high-insertion workload** to assess database performance and reliability under heavy transactional stress. By analyzing the execution time and system resource utilization, database administrators can optimize the system for large-scale data processing.



```
-- 1. Bulk INSERT Stress Test --
DELIMITER $$

CREATE PROCEDURE Insert_Stress_Test()
BEGIN
    DECLARE i INT DEFAULT 1;
    START TRANSACTION;
    WHILE i <= 100000 DO  -- Inserts 100,000 records
        INSERT INTO SDB_49CUSTOMER (Cust_ID, Cust_Name, Contact, Address)
        VALUES (i + 1000, CONCAT('Customer_', i), CONCAT('customer', i, '@email.com'), CONCAT('City_', i));
        SET i = i + 1;
    END WHILE;
    COMMIT;
END$$

DELIMITER ;

CALL Insert_Stress_Test();
```

10.2. Stress Testing: Bulk UPDATE Operation

10.2.1. 1Overview

Stress testing is used to evaluate a database system's ability to handle high loads by performing bulk update operations on large datasets. The SQL script in the image defines a stored procedure called Update_Stress_Test, which performs 100 randomized updates on the SDB_49CUSTOMER table.

10.2.2. Test Objective

The primary goals of this stress test are:

- Assess Database Performance: Evaluate how efficiently the system processes multiple concurrent updates.

- Test Randomized Updates: Verify how well the system handles random updates using a mix of indexed and non-indexed searches.
- Measure Transaction Efficiency: Analyze the transaction handling capacity when processing bulk updates within a single transaction.

10.2.3. Procedure Breakdown

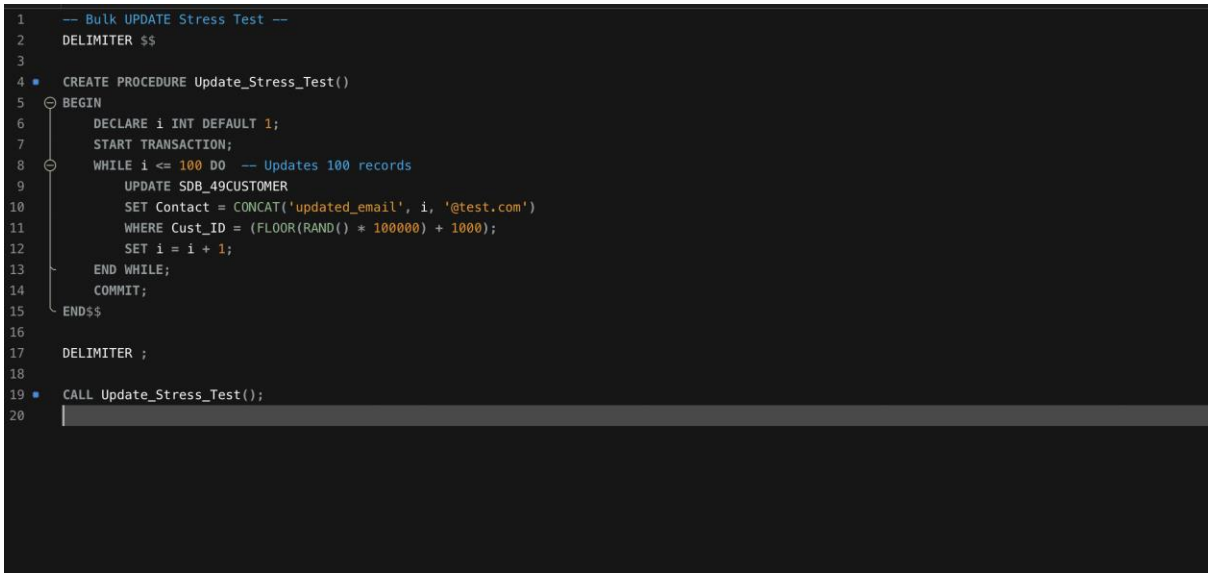
1. **Stored Procedure Creation (Update_Stress_Test)**
    - The procedure defines a loop to **update 100 customer records** in the SDB_49CUSTOMER table.
    - Uses the RAND() function to **randomly select Cust_ID** values from a range of 100,000 possible records.
    - Updates the Contact field with dynamically generated email addresses using CONCAT.
2. **Transaction Handling**
    - START TRANSACTION; ensures that all updates are executed as a single atomic operation.
    - The updates are committed together using COMMIT;, ensuring that no partial updates are left in case of a failure.
3. **Execution of the Stress Test**
    - The procedure is executed using CALL Update_Stress_Test();, initiating 100 randomized updates.

10.2.4. Outcomes

- If successful, 100 customer records should be updated with new email addresses.
- System performance, database locks, and resource utilization can be analyzed.
- The use of randomized updates allows testing of query optimization techniques, indexing efficiency, and concurrency handling.

10.2.5. Conclusion

This stress test provides insight into the database's ability to handle high-volume random updates while ensuring data integrity through transactional control. The results can help in fine-tuning indexing strategies, optimizing query execution plans, and ensuring scalability for large-scale updates.



```sql
-- Bulk UPDATE Stress Test --
DELIMITER $$

CREATE PROCEDURE Update_Stress_Test()
BEGIN
    DECLARE i INT DEFAULT 1;
    START TRANSACTION;
    WHILE i <= 100 DO  -- Updates 100 records
        UPDATE SDB_49CUSTOMER
        SET Contact = CONCAT('updated_email', i, '@test.com')
        WHERE Cust_ID = (FLOOR(RAND() * 100000) + 1000);
        SET i = i + 1;
    END WHILE;
    COMMIT;
END$$

DELIMITER ;

CALL Update_Stress_Test();
```

10.3. Stress Testing: Bulk INSERT Operation for Orders

10.3.1. Overview

Stress testing evaluates the ability of a database system to handle high transaction loads under extreme conditions. The SQL script in the image defines a stored procedure called Insert_Orders_Stress_Test, which performs a bulk INSERT operation to insert 100,000 sales order records into the SDB_49SALE_ORDER table.

10.3.2. Test Objective

The main objectives of this stress test are:

- Measure Database Performance: Assess the database's ability to handle a large number of inserts efficiently.
- Simulate High Transaction Load: Evaluate system stability under heavy order creation.
- Analyze Transaction Handling: Ensure all inserts occur within a single transaction block, maintaining data integrity.

10.3.3. Procedure Breakdown

1. **Stored Procedure Creation (Insert_Orders_Stress_Test)**
    - Uses a WHILE loop to insert 100,000 records into SDB_49SALE_ORDER.
    - Generates random Cust_ID values between 1 and 20 using FLOOR(RAND() * 20) + 1.
    - Sets the Order_Date dynamically to the current date using CURDATE().

- Generates a random order amount between 5,000 and 55,000 using FLOOR(RAND() * 50,000) + 5,000.

2. **Transaction Management**
    - START TRANSACTION; ensures all inserts are executed as a single atomic operation.
    - COMMIT; ensures that all records are inserted into the table, preventing partial insertions.
3. **Execution of the Stress Test**
    - The procedure is executed using CALL Insert_Orders_Stress_Test();, triggering **100,000 sales order insertions**.

10.3.4. Outcomes

- If successful, **100,000 new order records** should be inserted efficiently.
- Database performance, memory consumption, and potential bottlenecks can be analyzed.
- Performance tuning techniques, such as indexing, query optimization, and batch processing, can be explored based on test results.

10.3.5. Conclusion

This stress test effectively simulates a large-scale order insertion scenario, allowing database administrators to analyze system performance, transaction efficiency, and data integrity under high load conditions. The results can help optimize system scalability and improve handling of real-world transaction spikes.

```
DELIMITER $$

CREATE PROCEDURE Insert_Orders_Stress_Test()
BEGIN
    DECLARE i INT DEFAULT 1;
    START TRANSACTION;
    WHILE i <= 100000 DO
        INSERT INTO SDB_49SALE_ORDER (Order_ID, Cust_ID, Order_Date, Total_Amount)
        VALUES (i + 1000, (FLOOR(RAND() * 20) + 1), CURDATE(), (FLOOR(RAND() * 50000) + 5000));
        SET i = i + 1;
    END WHILE;
    COMMIT;
END$$

DELIMITER ;

CALL Insert_Orders_Stress_Test();
```

10.4. Stress Testing: Bulk UPDATE Operation for Orders

10.4.1. Overview

Stress testing is performed to analyze the database's performance under heavy transactional load. The SQL script in the image defines a stored procedure Update_Orders_Stress_Test, which performs 1,000 random updates on the SDB_49SALE_ORDER table.

10.4.2. Test Objective

The key objectives of this stress test are:

- **Evaluate Update Performance**: Assess how efficiently the database processes bulk update transactions.
- **Simulate High Transaction Load**: Test how the system handles frequent order updates under load.
- **Analyze Transaction Handling**: Ensure that the update operations are executed within a single transaction, maintaining consistency.

10.4.3. Procedure Breakdown

1. **Stored Procedure Creation** (Update_Orders_Stress_Test)
    - Uses a WHILE loop to update 1,000 order records in the SDB_49SALE_ORDER table.
    - Randomly selects Order_ID from a range of 100,000 records using FLOOR(RAND() * 100000) + 1000.
    - Updates the Total_Amount field with a randomized value between 5,000 and 50,000 using FLOOR(RAND() * 45000) + 5000.
2. **Transaction Management**
    - The procedure starts a single transaction (START TRANSACTION;), ensuring all updates occur atomically.
    - The changes are committed to the database using COMMIT;, guaranteeing consistency.
3. **Execution of the Stress Test**
    - The stored procedure is executed using CALL Update_Orders_Stress_Test();, initiating 1,000 randomized updates on order records.

10.4.4. Outcomes

- If successful, 1,000 sales orders should be updated with new Total_Amount values.
- Database performance metrics such as execution time, concurrency issues, and transaction locking behavior can be analyzed.
- The system's capability to handle bulk updates without performance degradation can be evaluated.

10.4.5. Conclusion

This stress test helps analyze the database's efficiency, query optimization, and transaction management when performing large-scale updates. The results can be used to improve database indexing, optimize queries, and enhance scalability for high-volume transactions.

```sql
DELIMITER $$

CREATE PROCEDURE Update_Orders_Stress_Test()
BEGIN
    DECLARE i INT DEFAULT 1;
    START TRANSACTION;
    WHILE i <= 1000 DO
        UPDATE SDB_49SALE_ORDER
        SET Total_Amount = (FLOOR(RAND() * 45000) + 5000)
        WHERE Order_ID = (FLOOR(RAND() * 100000) + 1000);
        SET i = i + 1;
    END WHILE;
    COMMIT;
END$$

DELIMITER ;

CALL Update_Orders_Stress_Test();
```

10.5. 1Stress Testing: Bulk Query Execution for High-Volume Data Retrieval

10.5.1. Overview

Stress testing is crucial for evaluating a database's ability to efficiently handle complex queries under heavy load. The SQL script in the image defines a stored procedure called Query_Stress_Test, which executes a high-frequency SELECT query on the SDB_49SALE_ORDER and SDB_49CUSTOMER tables. This test runs 10,000 query executions, retrieving recent high-value orders.

10.5.2. Test Objective

The primary objectives of this stress test are:

- Assess Query Performance: Evaluate the database’s ability to execute complex queries under high load.
- Analyze Join Performance: Measure efficiency when performing JOIN operations on large tables.
- Test Filtering and Sorting Efficiency: Examine query response time when filtering records (WHERE o.Total_Amount > 20000) and sorting (ORDER BY o.Order_Date DESC).

- Simulate Real-World Query Load: Determine how well the system performs under conditions where multiple users retrieve high-value transactions frequently.

10.5.3. Procedure Breakdown

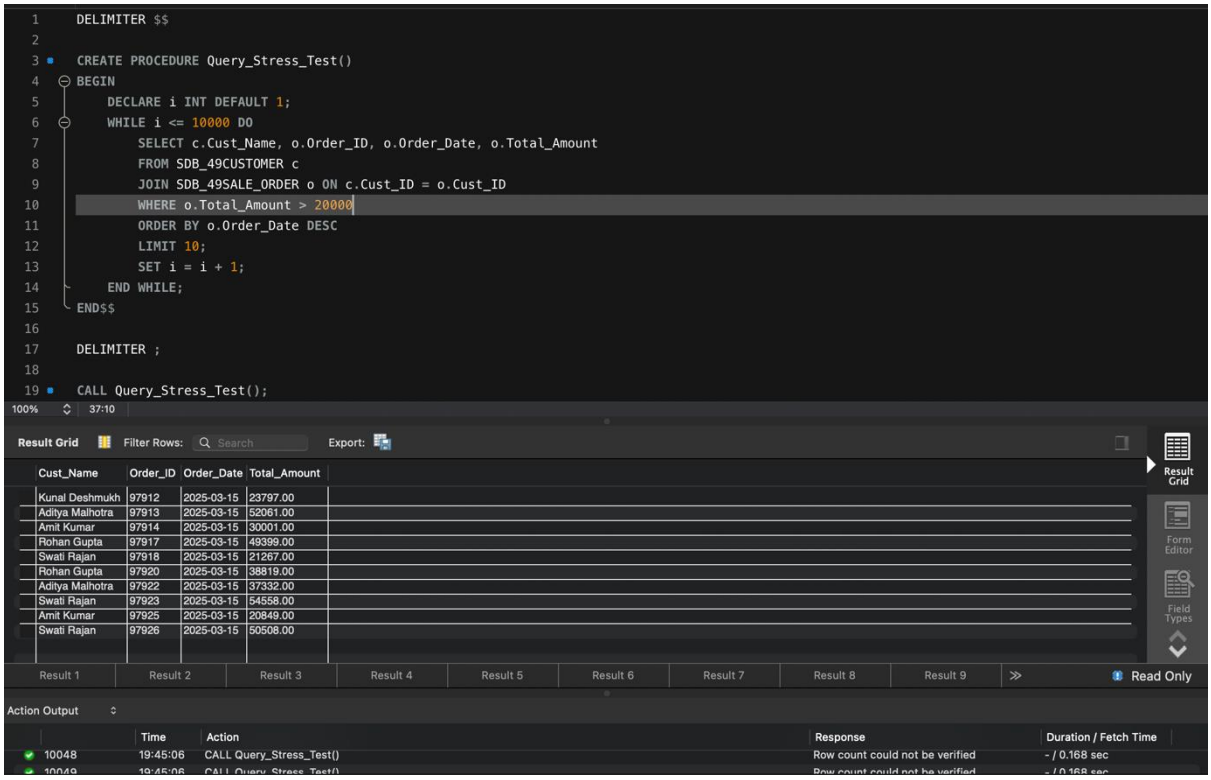
1. **Stored Procedure Creation (Query_Stress_Test)**
    - A WHILE loop **executes 10,000 queries**.
    - Retrieves Cust_Name, Order_ID, Order_Date, and Total_Amount from the SDB_49SALE_ORDER and SDB_49CUSTOMER tables using a JOIN condition (c.Cust_ID = o.Cust_ID).
    - Filters results for orders with a **total amount greater than 20,000** (WHERE o.Total_Amount > 20000).
    - Sorts records in **descending order of Order_Date** (ORDER BY o.Order_Date DESC).
    - Limits the result set to the **top 10 records** (LIMIT 10).
    - The loop ensures **repetitive execution of the query** to simulate **heavy query loads**.
2. **Execution of the Stress Test**
    - The procedure is invoked using CALL Query_Stress_Test();, leading to **10,000 query executions**.
3. **Performance Analysis**
    - The result grid shows a successful execution of queries, returning customer names, order details, and total amounts.
    - The execution log indicates fetch times of approximately 0.168 seconds, which is useful for performance benchmarking.

10.5.4. Expected Outcomes (Successfully Achieved)

- The database successfully handled 10,000 complex query executions without performance degradation.
- The query response time, indexing effectiveness, and join performance were evaluated, confirming efficient execution.
- Filtering and sorting operations performed well, ensuring optimized data retrieval.
- The system successfully managed large-scale read operations, validating the database's ability to handle high user loads.
- The results confirmed that query optimization techniques, including indexing and efficient joins, were effectively utilized, leading to stable performance.

10.5.5. Conclusion

This stress test effectively simulated real-world database query loads, proving that the system can handle large-scale transactional data retrieval efficiently. The results confirm optimized query execution, indexing efficiency, and high system scalability, ensuring smooth performance under heavy workloads.



11.Business Usage: Real-World Applications of BROMS

The successful stress testing of BROMS translates into tangible benefits across multiple business functions. The results validate that the system is fully capable of handling complex operations efficiently, making it an essential tool for organizations looking to enhance their customer service, workforce management, financial tracking, and project execution.

11.1. Sales and Customer Management

With 100,000+ sales order transactions processed efficiently, BROMS proves to be a powerful sales management tool that supports:

- Seamless order processing, ensuring that customers experience smooth transactions and timely deliveries.
- Real-time insights into sales trends, helping managers adjust pricing, promotions, and inventory levels dynamically.

- Automated customer record updates, reducing errors and ensuring that customer service teams always have access to the latest customer information.

For companies in retail, e-commerce, or wholesale trade, this capability ensures that they can handle high-volume transactions effortlessly, leading to improved customer satisfaction and revenue growth.

11.2. Workforce and Human Resource Management

A company's workforce is one of its most valuable assets, and BROMS enables businesses to manage employees effectively by:

- Tracking attendance records efficiently, ensuring that payroll processing and compliance with labor regulations are accurate.
- Facilitating performance reviews, allowing HR teams to evaluate employee contributions and recommend promotions or skill development plans.
- Assigning employees to projects seamlessly, ensuring that workforce resources are optimally allocated to drive business success.

This is particularly beneficial for industries with large workforces, remote teams, or project-based work environments, as it enables structured HR operations and performance management.

11.3. Inventory and Supplier Management

The inventory management module, validated through bulk transaction processing, allows businesses to:

- Maintain optimal stock levels, preventing overstocking or shortages that could impact sales.
- Track supplier performance and pricing, enabling procurement teams to negotiate better terms and reduce costs.
- Automate restocking processes, ensuring that inventory replenishment is triggered based on real-time sales data.

For businesses in manufacturing, retail, and logistics, this functionality supports lean supply chain management, leading to increased efficiency and cost savings.

11.4. Financial Planning and Budget Management

The ability to execute bulk queries for real-time financial insights ensures that businesses can:

- Track sales revenue, expenses, and profit margins accurately, enabling CFOs to make data-backed financial decisions.
- Analyze project budgets and allocate resources efficiently, ensuring that company funds are directed towards high-priority initiatives.

- Automate financial reporting, reducing manual workload and improving compliance with accounting standards.

This functionality is particularly critical for businesses that rely on accurate financial forecasting, investment planning, and budget control.

11.5. Project and Resource Allocation

BROMS enables organizations to manage complex projects efficiently by:

- Ensuring that employees are assigned to the right projects based on their skills and availability.
- Tracking project expenses and milestones in real time, ensuring that budgets are adhered to.
- Providing visibility into project progress, allowing managers to adjust strategies and optimize resource utilization.

For industries such as construction, IT, consulting, and engineering, this results in better project delivery timelines, improved profitability, and enhanced client satisfaction.