**ST. PAULS UNIVERSITY**

**UNIT CODE : BCS 4103**

**UNIT TITLE: ADVANCED DATABASE SYSTEMS**

**REPORT ON OPTIMIZING E-COMMERCE DATABASE PERFOMANCE WITH POSTGRESQL AND NODE.JS**

**Githublink https://github.com/Group-A-SPU-Advance-database-2024**

1. **BOBITLMR175622- KHULWAEL DRUSILLA**
2. **BOBITLMR145922- KIPKEMOI EVANS**
3. **BOBITLMR190122- KELVIN GICHUHI**

**INTRODUCTON**

The goal of this project is to improve ShopEase's database system performance, as it has been experiencing noticeable slowdowns during periods of high shopping demand. Complex queries and frequent updates proved difficult for the outdated system to handle, which caused response times to lag and unhappy users. We have used Node.js and Express.js for backend development, and PostgreSQL for management of databases in order to overcome these difficulties. Furthermore, we've included Swagger for thorough API documentation.

**PROJECT OVERVIEW**

The main objectives of this project is to enhance the overall effectiveness of ShopEase's backend operations and streamlining the database system's performance. The project consists of four primary tasks:

1. Schema Design and Data Population.
2. Backend API Development
3. API Documentation using Swagger
4. Query Optimization

This project is intended to provide strong database structures, effective query processing, and comprehensive documentation in order to improve the inadequacies of the current system. Below we will broadly explain each task specific implementation.

**Schema Design and Data Population.**

For an e-commerce platform such as ShopEase, the establishment of a reliable and effective database system begins with the schema design and data population. These procedures entail loading realistic data into the database to replicate real-world usage and organizing the database to maximize data retrieval, storage, and management.

**SCHEMA DESIGN**

A database's boundaries, relationships, and structure are defined through the process of schema design. Data integrity, effective query efficiency, and scalability are all dependent on a well-designed schema.

Key Components of Schema Design

1. Tables and Relationships: Create tables to symbolize various entities, like orders, products, users, and their connections. For example:  
   The order information is stored in the orders table and the order\_items table store the products included in each order.

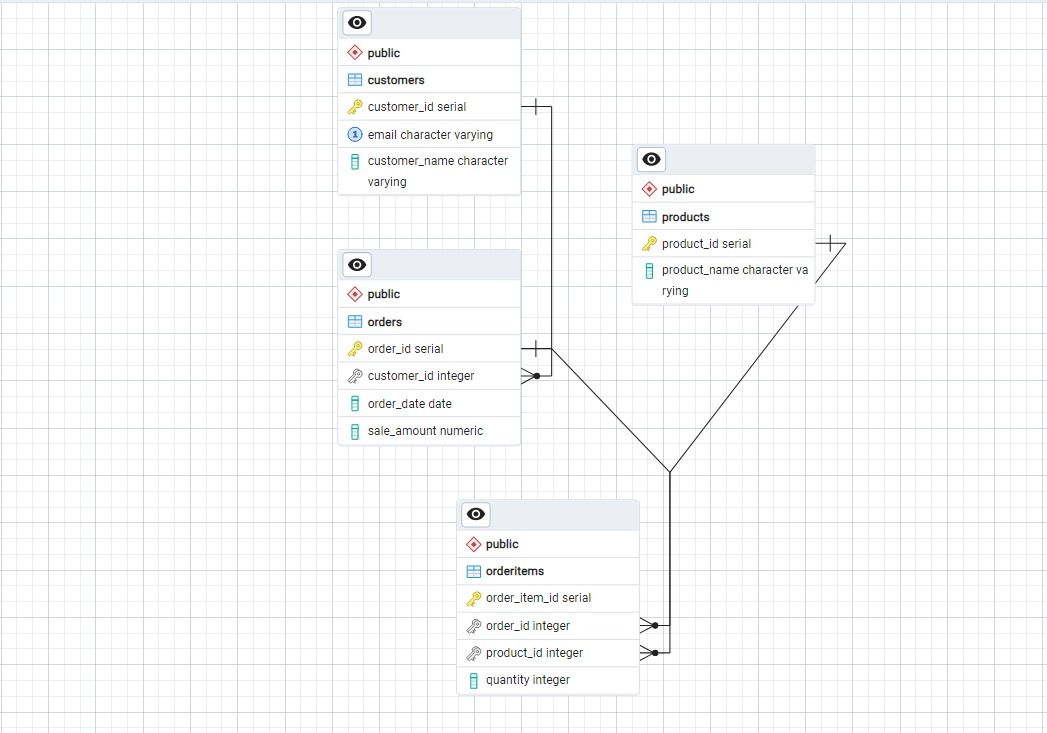
User information is stored in the customers table.

Product details are stored in the products table.

1. Types of Data and Restrictions: To guarantee data integrity, provide constraints and select the proper data types for each column. For example:  
   For text fields containing names and emails, use VARCHAR.  
   For the price and amount fields, use INT.  
   To identify each record individually, define the primary keys.  
   To preserve referential integrity between connected tables, define foreign keys.
2. Indexes and Partitioning: To maximize query efficiency and effectively manage massive datasets we applied indexing and partitioning techniques. For example:  
   Creating indexes on fields that are often accessed, such customer\_id in the orders table.  
   Dividing big data, such as orders, according to dates to enhance query efficiency and control.
3. Preserved Methods and Inducers: Using triggers to automate repetitive activities and enforce business rules, and stored procedures for complex and often used queries.

ERD DIAGRAM

The relationships between the different entities in the ShopEase database are shown in the following Entity-Relationship Diagram (ERD), which offers a visual depiction of the database structure:



**DATA POPULATION**

Data population refers to inserting sample data into the database to simulate real-world usage.

**IMPORTANCE OF DATA POPULATION**

1. Performance Evaluation It assists in testing query speed and locating possible bottlenecks by adding a sizable amount of sample data to the database.
2. Functionality testing- it verifies that, under realistic circumstances, all database operations including inserts, updates, and deletes, perform as anticipated.
3. Query optimization: Offers a practical dataset for index, partitioning, and query optimization.
4. System simulation: Assists in simulating real-world events, such as periods of high traffic, to guarantee the database can manage heavy loads.

**BACKEND API DEVELOPMENT**

The functionality and performance of an e-commerce platform such as ShopEase depend heavily on backend API development. As part of this development, server-side logic is created to enable client applications—such as web and mobile apps—to communicate with the database and carry out tasks like handling user accounts, processing orders, and accessing product data.

**Node.js and Express.js**

Since Node.js and Express.js are asynchronous, scalable, and have a large developer community, they are common choices for creating backend APIs. The procedures needed to create a backend API for the ShopEase e-commerce platform are listed below;

*Setting up the project*

1. Initialize the Node.js Project- Creating a new Node.js project and install the necessary dependencies.

This is done by first creating a new package.json file by running the npm init on the command line on the project directory.

Then run npm install express body-parser pg dotenv to install necessary dependencies.

1. Set up the project structure- To make the project easier to maintain, arrange it into folders and files. For example;

ShopEase

|-----config/db.js

|------controllers/productController.js

|------routes/productRoutes.js

|------app.js

|-----.env

1. Database Configuration- Using the pg module, create a configuration file to establish the database connection.
2. Integrating routes – Node.js and Express.js are all about routes. Routes can be defined using the app.get(), app.post(), app.put( ) and app.delete( ) methods. The app.get( ) function accepts two parameters, the path and the callback function (what happens when the GET request is called). The function also has called the request body parameter(has details like request query, string, parameters, body and HTTP headers and the response body which has information to be sent.

**Middleware setup**

Middleware setup- Middleware is designed to manage sessions, parse request bodies, and defend against cross-site request forgery attacks.

The essential middleware consists of:  
1. Body-parser: Used to parse request bodies encoded with URLs and JSON.

2. Used cookie-parser for parsing cookies.  
3. Helmet: For configuring different HTTP headers to improve security.  
4. We used connect-pg-simple and express-session to manage user sessions.

**CRUD API Development**

CRUD API Development - The essential web development processes known as CRUD (Create, Read, Update, Delete) let clients communicate with the database. For the e-commerce platform to work and execute well, these procedures must be implemented effectively.

Below is an overview of the four primary HTTP methods that are associated with the CRUD operations i.e. the backbone of how the application will interact with the data;

1. POST is used to create the new data on the database (CREATE).
2. GET: Is used for reading and retrieving data from the database(READ)
3. PUT is used for updating the existing data on the database. (UPDATE).
4. DELETE is used for removing from the database. (DELETE).

**The API endpoints we defined:**

POST/users to create the users,

GET/users- Finds all users

GET/users/:id- finds a specific user

DELETE/users/:id- deletes a specific user.

PATCH/ users/:id- updates a specific user.

**API Documentation with Swagger**

ShopEase has several endpoints for different functions thus requires thorough documentation of the APIs. The documentation is appropriate to guarantee that developers, both internal and external, can comprehend and utilize the APIs efficiently. We used swagger to make sure it’s easier to explore and test endpoints as it generates interactive API documentation.

Below is an example of Swagger-based API documentation for ShopEase  
We will go over how to set up Swagger in a Node.js and Express.js project and show how to document different endpoints in order to show how it can be used to document the ShopEase API.

*Setting up Swagger*

1. Install Swagger dependencies by running the npm install swagger-jsdoc swagger-ui-express.
2. Configure Swagger - To set up Swagger in your Express.js application and describe the API standards, create a configuration file.
3. Integrate swagger with express- To use the Swagger configuration, make changes to your main application file.

**QUERY OPTIMIZATION**

We used triggers and stored procedures to improve the efficiency of complex and often used queries.

1. Preserved Methods  
   Tasks like figuring out the overall sales for a certain time period were accomplished using stored processes.
2. Initiators  
   Automating processes such as adjusting inventory levels in response to new orders was done with the use of triggers.

**RECOMMENDATIONS**

The project's results have led to the following suggestions for future enhancements and scalability:

1. Continuous Performance Monitoring: To continuously monitor database performance and spot bottlenecks, put monitoring tools into place.
2. Load Testing: To make sure the system can manage a lot of traffic and concurrent users, load test it frequently.
3. Advanced Security Measures: Look into cutting-edge security techniques include end-to-end encryption for sensitive data and two-factor authentication (2FA).
4. Microservices design: As the platform expands, think about switching to a microservices design for improved scalability and maintainability.
5. Automated Testing: To guarantee code quality and lower the possibility of errors in production, use automated testing for both frontend and backend components.

**CONCLUSION**

ShopEase's e-commerce platform's performance problems were effectively resolved by the project through the implementation of an optimized database schema, effective backend APIs, extensive security measures, and thorough API documentation. A more dependable and user-friendly e-commerce experience is the result of these enhancements, which also provide faster query performance, improved scalability, and more security.

**References**

1. <https://excelbianalytics.com/>
2. Ponelat, J. S., & Rosenstock, L. L. (2022). *Designing APIs with Swagger and OpenAPI*. Simon and Schuster.
3. Larsson, J., & Åkermark, L. (2021). The Value Of Implementing API-First As A Methodology When Developing Apis.
4. Martins, P., Tomé, P., Wanzeller, C., Sá, F., & Abbasi, M. (2021). Comparing oracle and postgresql, performance and optimization. In *Trends and Applications in Information Systems and Technologies: Volume 2 9* (pp. 481-490). Springer International Publishing.
5. Nascimento, R., Brito, A., Hora, A., & Figueiredo, E. (2020, February). JavaScript API deprecation in the wild: A first assessment. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)* (pp. 567-571). IEEE.
6. Qvarnström, E., & Jonsson, M. (2022). A performance comparison on REST-APIs in Express. js, Flask and ASP. NET Core.
7. <https://www.twilio.com/docs/usage/tutorials/how-to-set-up-your-node-js-and-express-development-environment>
8. <https://www.freecodecamp.org/news/create-crud-api-project/>