Cambodia Academy of Digital Technology

Inventory Database Management System

Subject: Database Administration

Lecturer: Thear Sophal

Team 01

Members

Chum Sothyrak

Ngovseng Povpanha

Soveth Roathbunna

Ith Youdy

1. Introduction

1.1. Project Objectives and Scope

This document outlines the development and implementation of the *Inventory Management Database System*. The system is designed to support efficient data management related to product inventories, sales transactions, and overall business records. It also provides functionalities for user access control, data integrity, and performance optimization.

1.2. Database Scope: Identification of Core Entities and Relationships

The primary objectives of this project are:

- To build a robust, scalable, and secure inventory database system.
- To implement core functionalities such as product management, transaction recording, and user access control.
- To simulate a real-world scenario by handling large-scale data and evaluating query performance.
- To implement a practical backup and recovery mechanism to prevent data loss.

Scope Included:

- Full database lifecycle: Design, Implementation, Population, and Optimization.
- Role-Based Access Control (RBAC).
- Daily automated backup and manual recovery capabilities.
- Web-based user and role management interface.

1.3. Team Members and Roles

- Chum Sothyrak: Database Design & Implementation and Query Optimization
- Soveth Roathbunna: Data Population and Query Performance Testing
- Ith Youdy: Database Server Deployment, Backup & Recovery and Scheduling
- Ngovseng Povpanha: User Access Control & Frontend and Backend Implementation

2. **Database Design**

2.1. Entity-Relationship (ER) Diagram

By drafting, we generate several entities which are needed in our system. The strong entities are Product, User and Sale, together generating several important functions such as store product info & qty, make sales (check-in & check-out) and application user involvement. All the entities are connected by their relationship and defined as (1 to 1, 1 to M, M to M).

2.2. Relational Schema

The relational schema was derived from the ERD and consists of the following tables:

- **Product** (PK id, name, description, category)
- **Product_Variant** (PK id, color, size, price, FK product.id)

- **Sales** (PK id, status, total_price, FK user.id)
- **Sales_Records** (PK id, qty, price_each, FK sale.id, FK product_variant.id)
- **Stock_Transaction** (PK id, qty, type, FK product_variant.id)
- **Users** (PK id,name, email,password,role)

Each table includes defined attributes, primary keys, and foreign keys to establish integrity and linkage between entities.

3. **Database Implementation**

By using a relational schema, all the tables are constructed and stored in the database using the SQL language. The primary keys are identified and remarked to be auto increment and not null, which makes the tables more specific and easier to populate. In several tables, the foreign keys are implemented inside the tables to make the table's connection. The 'Alter' keyword is used in rare cases to fix the table for defining errors.

4. **Data Population**

The Big Data concept is used in our database project where we store millions of records in required tables. By using python and faker libraries, we can generate the number of records we want.

- Product Table 100 records
- Product Variants 1M records
- Sales 300 records
- Sales Records 1M records
- Stock Transaction 2M records
- Users 1M records

5. **User Management and Access Control**

5.1. Role-Based Access Control (RBAC) Implementation

Database Roles are divided into 4 mains and their accessibility such as

- Database Admin: full access to system
- Developer: Create, Alter, Select, Insert, Update on all tables
- Analysist: Read only (Select) on all tables
- Backup Admin: Read only (Select) on all tables

Within all four default roles, the admin or other user with the permission can customize by creating and updating roles such as interns.

5.2. Web-based Application for User and Role Management

We built a front-end application for users to justify user control and management by displaying it on a virtual screen. The front-end is built using React JS framework and connects to the database by backend using Node JS Library.

Features

- Creation of New Users
- Define roles
- Grant Access for User with roles
- Displaying all the users and roles

6. **Backup and Recovery Strategy**

6.1. Important Backup & Recovery

Backup & Recovery is a method which copying the database and restoring it back when needed. It is very important for an organization to secure when losing data at any time.

6.2. Implementation

In this project, we use this method to store a copy and recover it when needed. By using python, we built two scripts, one for backup copy and another for recovery. Inside the backup script, we built a function which can copy the database by generating it into a Copy_script. On the other hand, if the database is dropped or deleted, we can use the recovery script to generate the database back.

6.3. Scheduling

Task Scheduler are used for daily backup. We set to run the backup copy script every 24 hours in the local machine using task scheduler. It will notify when it is the time for backup and run automatically.

7. **Query Development and Optimization**

We built 10 queries in SQL script without using indexing method and the same queries using indexing. Those queries are based on tables with millions of records. Using Indexing, each table's columns are all indexed to ensure when the user wants to query based on the attribute.

8. **Database Server Deployment**

In this project, the database is hosted globally using Ngrok. To use this application for hosting, we need to host the database server in one computer as a server and convert the local host port to TCP port. Another computer can use the public port to get the database, but they need the user accounts in which those accounts are created and stored in the server (user access & control).

9. **Performance Comparison**

9.1. Comparison Methodology

All the queries are compared to each other.

- Using python script to execute the queries
- The run times are compared
- Run all and compared those 10 queries

9.2. Results and Analysis

SQL Query	*No Indexing*	*Indexing*
`SELECT id, name, category FROM product WHERE id >= 1 AND id <= 1000`	0.0009 seconds	0.0005 seconds
`SELECT name FROM user WHERE role = 'staff'`	7.0773 seconds	6.1714 seconds
`SELECT id,product_id, color, size, price,stock_qty FROM product_variant WHERE product_id >50`	0.0084 seconds	0.0054 seconds
`SELECT id, user_id, status, total_price FROM sales WHERE user_id = 50 AND status = 'completed'`	0.0075 seconds	0.0063 seconds
`SELECT id, sale_id, variant_id, qty, price_each FROM sales_record WHERE sale_id = 200`	0.0066 seconds	0.0053 seconds
`SELECT id, variant_id, qty, type FROM stock_transaction WHERE id >100`	2.2743 seconds	2.1757 seconds
`SELECT id, name FROM user WHERE role = 'admin' AND id > 5000`	8.7854 seconds	8.1189 seconds
`SELECT id, user_id, status,total_price FROM sales WHERE user_id > 5000 AND Total_price > 200`	2.7511 seconds	2.5857 seconds
`SELECT id, product_id, color, size, price, stock_qty FROM product_variant WHERE product_id >50 && stock_qty > 10`	0.0063 seconds	0.0055 seconds

10. **Conclusion**

10.1. Summary of Project Achievements

The project successfully achieved its goal of designing and implementing a comprehensive Inventory Management Database System. It included core functionalities such as product and sales tracking, user role management, and performance optimization. A relational database schema was created, and millions of records were populated using Python and the Faker library to simulate a real-world business environment. Additionally, we implemented Role-Based Access Control (RBAC), built a web-based user and role management interface using React and Node.js, developed automated backup and recovery scripts, and conducted query performance optimization through indexing. These accomplishments resulted in a reliable, scalable, and efficient system ready for practical use.

10.2. Challenges Faced and Solutions Implemented

Throughout the project, we encountered several technical and operational challenges. Data generation for millions of records initially led to memory issues and long insertion times. This was addressed by modifying our Python scripts to use batch inserts and optimize resource usage. Query performance was also a challenge, as large data volumes caused slow execution and timeouts. We solved this by introducing indexing on frequently queried columns, which drastically reduced response times. Role conflicts and permission overlap were resolved by refining our RBAC logic, and secure access over the internet was achieved through controlled deployment using Ngrok and proper user isolation.

10.3. Lessons Learned

This project provided valuable insights into real-world database development. We learned how crucial indexing is for optimizing query performance on large datasets. We also understood the importance of role-based access control in maintaining security and operational hierarchy within a database environment. Developing a user-friendly web interface taught us the value of frontend-backend integration for managing complex backend systems. Additionally, we realized that automated backup and recovery mechanisms are vital to ensure data protection and system resilience in case of failure. These lessons are applicable to future projects and professional scenarios alike.

10.4. Future Enhancements and Recommendations

To further improve the system, several enhancements are recommended. Implementing logging and audit trails would provide visibility into user activities and support security auditing. Introducing JWT-based authentication would enhance access control for the web interface. Moving the database to a cloud-based platform such as AWS RDS or Azure SQL would offer better scalability, security, and availability. Additionally, creating a real-time analytics dashboard would provide insights into inventory and sales trends. Developing a mobile-friendly interface or Progressive Web App (PWA) would also make the system more accessible for on-the-go users. These improvements would significantly increase the system’s utility, security, and scalability.