COMVERSE

ADVANCE DATABASE SYSTEM
FINAL PROJECT DOCUMENTATION

Prepared by:
Ferdinand T. Corbin Jr.
Carlxen Brieyl P. Duran
Rynel Jazmin L. Fetalvero
Alvhin C. Solo

TABLE OF CONTENT

I. Introduction

I.I Project Overview

Modern applications, especially those involving user interaction and real-time data, rely heavily on efficient and well-structured databases. This project introduces COMVERSE, an academic e-commerce system created for Computer Science students. Rather than focusing on user interface design or front-end development, this study puts the spotlight on what powers the system behind the scenes, the database.

COMVERSE uses MySQL as its core database engine. The main objective is to design and implement a well-structured relational database capable of handling inventory, user accounts, and purchase transactions. The project demonstrates how proper database design ensures accuracy, efficiency, and reliability in real-world applications. The expected outcome is a clean and functional system that showcases best practices in database development within the context of academic e-commerce.

I.II Learning Objectives

This project encourages students to look beyond the visual side of application development and understand the vital role of the backend. By working on COMVERSE, students will gain practical experience in designing and managing data systems. Through this project, students will:

- Learn how to create and normalize relational database schemas.
- Practice writing SQL queries to manage data effectively.
- Apply concepts such as primary keys, foreign keys, and constraints to ensure data integrity.
- Understand the role of transactions and indexing in improving system performance.
- Appreciate how structured data supports scalable and secure applications.

II. Background

II.I Introduction to Advanced Database Systems

Databases are essential components of nearly all modern applications. As data becomes larger and more complex, advanced database systems provide powerful tools for managing, processing, and securing information efficiently. These systems go beyond basic storage, offering features such as query optimization, concurrency control, and robust security.

Relational databases remain one of the most dependable technologies for handling structured data. Core concepts like normalization, referential integrity, and ACID (Atomicity, Consistency, Isolation, Durability) compliance are foundational in industries such as finance, healthcare, and education. These principles ensure that data remains consistent, reliable, and easy to query, enabling seamless analysis and visualization particularly when uncovering connections across complex datasets.

In online e-commerce systems, having a solid database foundation is crucial to enhancing user experience and streamlining operations. The design and functionality of

COMVERSE are heavily inspired by existing successful platforms such as Shopee and the DLSU Online Merchandise system.

Shopee, a leading Southeast Asian e-commerce platform, emphasizes a seamless shopping experience with categorized product listings, real-time inventory updates, and secured user transactions. COMVERSE adopts similar features by ensuring organized product data, real-time availability, and user-centered navigation.

La Salle Merchandise Shop, though more limited in scale, provides a practical local context. It showcases how school-affiliated e-commerce platforms manage student orders, product promotions, and inventory using simplified systems. COMVERSE builds on this by integrating real-time stock validation, user account management, and order tracking, making it suitable for both academic and entrepreneurial scenarios.

When developing such systems, several challenges must be addressed. These include the scalability required to handle growing volumes of data, the implementation of security measures to safeguard customer information, and proper data normalization to reduce redundancy and improve data integrity. COMVERSE is designed to meet these challenges using sound relational database principles.

II.II Chosen Advanced Database Model/Technology

The database technology chosen for COMVERSE is **MySQL**, a robust and widely-used relational database management system (RDBMS). MySQL structures data into organized tables and enforces relationships through primary and foreign keys, ensuring consistency and logical data organization. It is valued for its performance, reliability, and user-friendliness, making it ideal for both academic and professional use.

In the context of COMVERSE, MySQL plays a central role in managing critical data operations such as storing product listings, handling user profiles, and recording transactions. With its support for structured query language (SQL) and features like indexing, foreign key constraints, and transaction handling, MySQL offers a dependable foundation for building a secure and scalable backend.

By implementing MySQL in this academic project, COMVERSE not only provides practical hands-on experience with a professional-grade database system but also reinforces key theoretical concepts covered in advanced database courses.

III. System Design

III.I System Architecture

The overall system architecture is composed of three main layers: the user interface, the business logic layer, and the database layer:

The **user interface** is built using JavaFX with the help of SceneBuilder, allowing users to interact visually with the system. Users can register, log in, browse products, manage their cart, place orders, and provide shipping and payment information through a user-friendly interface.

The **business logic layer**, implemented in Java using Visual Studio Code (VSC), acts as a controller that manages the flow of data between the UI and the database. It handles

user input, validates data, and executes operations such as retrieving product lists or processing orders.

The **database layer** is managed using MySQL, which stores all application data based on the structure defined in the Entity Relationship Diagram (ERD). The system includes several interrelated tables such as User, Products, Categories, Cart, Order, Shipping, and Payment.

When a user interacts with the system, their data is stored in the User table. Products are displayed from the Products table, which is linked to Categories. Users can add items to a Cart, which can later be converted into an Order. Each order is associated with a user and products and may include additional details such as shipping information (stored in the Shipping table) and payment method (stored in the Payment table). These components work together to provide a functional ecommerce-like platform that supports product browsing, order processing, and delivery management.

III.II Design Diagrams

The **COMVERSE MERCHANDISE** system is designed to support an online merchandise platform tailored for student users. The system allows students to browse products, manage their cart, place orders, make payments, and arrange for shipping. At its core, the system uses a relational database structure, illustrated in the Entity-Relationship Diagram (ERD), which outlines the main entities, their attributes, and the relationships between them.

The central entity in the system is the **User**, identified by a unique Student_Number, and containing personal details such as email, password, name, department, and course.

	User	
PK	Student_Number	varchar(7)
	Email	varchar(100)
	Password	varchar(255)
	First_Name	varchar(50)
	Last_Name	varchar(50)
	Department	varchar(100)
	Course	varchar(100)

Users can place multiple **Orders**, which are linked to both the **Cart** and the **Products** being purchased.

	Products	
PK	Product_Id	varchar(10)

FK	Category_Id	varchar(100)
	Product_Name	varchar(20)
	Image_URL	varchar(255)
	Amount	decimal (10, 2)
	Quantity	int

	Order	
PK	Order_Id	varchar(100)
	Transaction_Id	varchar(100)
FK	CartOrder_Id	varchar(250)
FK	Student_Number	varchar(7)
	Image_URL	varchar(255)
	Order_Date	Date (mm-dd-yyyy)
	Quantity	varchar(150)
	Total_Amount	decimal (10, 2)
	User_Marked_Received	tinyint(1)
	Received_Date	timestamp

Each order records the product selected, the quantity, total amount, and order date. Products are organized into **Categories**, allowing for better navigation and filtering of items, and each product belongs to one category.

	Categories	
PK	Category_Id	varchar(100)
	Category_Name	varchar(20)

Before placing an order, users add items to their **Cart**, which temporarily stores selected products along with their subtotal amounts.

	Cart	
PK	CartOrder_Id	varchar(100)

FK	Product_Id	varchar(20)
FK	Student_Number	varchar(7)
	Image_URL	varchar(255)
	Quantity	int
	Amount	int

Once finalized, the cart contents become part of an order. After placing an order, the user proceeds to the **Payment** process, where the system records the payment method used.

	Payment	
PK	Payment_Id	varchar(100)
	Transaction_Id	varchar(100)
FK	Student_Number	varchar(7)
	Payment_Amount	decimal (10, 2)
	Payment_Date	Date (mm-dd-yyyy)

The **Shipping** entity captures delivery information, including the shipping method, address, and cost, all associated with both the order and the user.

	Shipping	
PK	Shipping_Id	varchar(100)
	Transaction_Id	varchar(100)
FK	Student_Number	varchar(7)
	Shipping_Address	varchar(255)
	Shipping_Date	Date (mm-dd-yyyy)
	Shipping_Amount	int

The following figure follows the comprehensive workframe of the COMVERSE' Entity-Relationship Diagram.

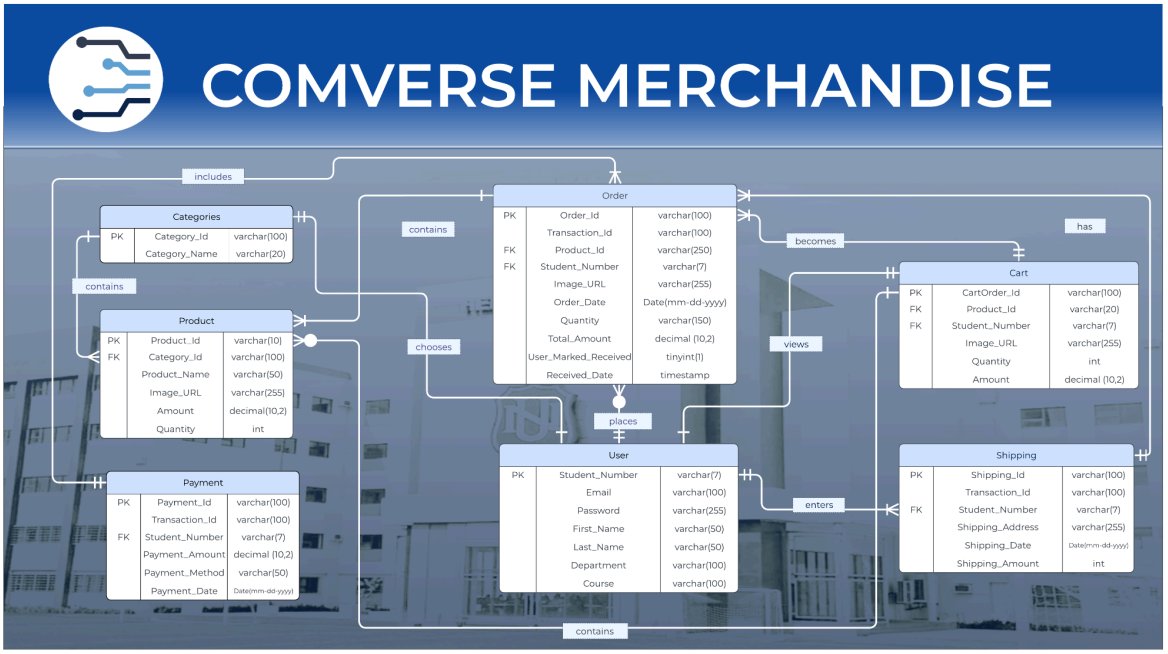

Figure 1. COMVERSE Entity-Relationship Diagram (ERD)

IV. Implementation

IV.I Technology Stack

COMVERSE was developed using MySQL as the primary database management system (DBMS), chosen for its reliability, simplicity, and wide adoption in the industry. As an open-source RDBMS, MySQL provided the necessary tools to model a structured and normalized database for an academic e-commerce platform.

Other supporting technologies included:

- **JavaFx**for the basic front-end interface.
- **MySQL** for database administration and query execution.
- **Visual Studio Code (VS Code)** as the primary code editor.

This combination of technologies ensured seamless integration between the application and the database, allowing real-time data transactions and user interactions.

IV.II Implementation Process

The implementation of the Comverse Ordering System database was done using MySQL, following a logical progression of creating tables, establishing relationships, and automating ID generation using triggers. The steps below include challenges and the exact SQL queries used.

Step 1: Database Initialization

```sql
CREATE DATABASE comverse;
USE comverse;
```

- **Creates** a database named comverse.
- **Activates** comverse as the default database for the session.

Step 2: User Table Creation and Automation

```sql
-- STUDENT TABLE ------------------------------------------------------------
CREATE TABLE users (
    student_number VARCHAR(7) PRIMARY KEY,
    email VARCHAR(100) UNIQUE NOT NULL CHECK (email LIKE '%@gmail.com'),
    password VARCHAR(255) NOT NULL,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    department VARCHAR(100) NOT NULL,
    course VARCHAR(100) NOT NULL
);
```

- Stores student/user accounts.
- student_number is **auto-generated** via a **trigger** with a format like CV-001, CV-002, etc.
- Email must end in **@gmail.com** (CHECK (email LIKE '%@gmail.com')).

Trigger: `before_insert_users`

```sql
DELIMITER $$
CREATE TRIGGER before_insert_users
BEFORE INSERT ON users
FOR EACH ROW
BEGIN
    DECLARE next_student_number INT;
    SELECT IFNULL(MAX(CAST(SUBSTRING(student_number, 5, 3) AS UNSIGNED)), 0) + 1
    INTO next_student_number FROM users;
    SET NEW.student_number = CONCAT('CV-', LPAD(next_student_number, 3, '0'));
END$$
DELIMITER ;
```

- Automatically assigns a unique student_number like CV-001, CV-002, etc., by:
- Extracting the numeric part from existing IDs.

- Padding the next number with 0s.

Step 3: Category Table

```
CREATE TABLE category (
    category_id VARCHAR(100) PRIMARY KEY,
    category_name VARCHAR(20) UNIQUE NOT NULL
);
```

- Stores **product categories** like SHIRT, CAP, etc.
- Category IDs like C001 - SHIRT are manually inserted.
- BEFORE_INSERT_PRODUCT to auto-generate IDs like COM-001.

```
INSERT INTO category (category_id, category_name) VALUES
('C001 - SHIRT', 'SHIRT'),
('C002 - CAP', 'CAP'),
('C003 - BAG', 'BAG'),
('C004 - JACKET', 'JACKET'),
('C005 - LACE', 'LACE');
```

- Inserts predefined categories.

Step 4: Product Table

```
CREATE TABLE product (
    product_id VARCHAR(10) PRIMARY KEY,
    category_id VARCHAR(100),
    product_name VARCHAR(50),
    image_url VARCHAR(255),
    amount DECIMAL(10,2),
    quantity int,
    FOREIGN KEY (category_id) REFERENCES category(category_id)
);
```

Stores products with:

- Name, category, image URL, price, and stock.
- category_id links to category.

Trigger: `before_insert_product`

```
DELIMITER $$
CREATE TRIGGER before_insert_product
BEFORE INSERT ON product
FOR EACH ROW
BEGIN
    DECLARE next_product_id INT;
    SELECT IFNULL(MAX(CAST(SUBSTRING(product_id, 5, 3) AS UNSIGNED)), 0) + 1
    INTO next_product_id FROM product;
    SET NEW.product_id = CONCAT('COM-', LPAD(next_product_id, 3, '0'));
END$$
DELIMITER ;
```

- Auto-generates product_id like COM-001, COM-002.

Step 5: Cart Table

```
CREATE TABLE cart (
    cart_id VARCHAR(100) PRIMARY KEY,
    product_id VARCHAR(20) NOT NULL,
    student_number VARCHAR(7) NOT NULL,
    image_url VARCHAR(255) NOT NULL,
    quantity INT NOT NULL,
    amount DECIMAL(10,2) NOT NULL,
    FOREIGN KEY (product_id) REFERENCES product(product_id),
    FOREIGN KEY (student_number) REFERENCES users(student_number)
);
```

- Stores items that users add to their cart.
- Includes product_id, student_number, quantity, amount.

Trigger: `before_insert_cart`

```
DELIMITER $$
CREATE TRIGGER before_insert_cart
BEFORE INSERT ON cart
FOR EACH ROW
BEGIN
    DECLARE next_cart_id INT;
    SELECT IFNULL(MAX(CAST(SUBSTRING(cart_id, 6) AS UNSIGNED)), 0) + 1
    INTO next_cart_id FROM cart;
    SET NEW.cart_id = CONCAT('CART-', LPAD(next_cart_id, 3, '0'));
END$$
DELIMITER ;
```

- Auto-generates cart_id like CART-001, CART-002.

Step 6: Order Table

```
CREATE TABLE orders (
    order_id VARCHAR(100) PRIMARY KEY,
    transaction_id VARCHAR(100) NOT NULL,
    product_id VARCHAR(250) NOT NULL,
    student_number VARCHAR(7) NOT NULL,
    image_url VARCHAR(255) NOT NULL,
    order_date DATE NOT NULL,
    quantity VARCHAR(150) NOT NULL,
    total_amount DECIMAL(10, 2) NOT NULL,
    FOREIGN KEY (product_id) REFERENCES product(product_id),
    FOREIGN KEY (student_number) REFERENCES users(student_number)
);
```

```
ALTER TABLE orders ADD COLUMN user_marked_received TINYINT(1) DEFAULT 0;
ALTER TABLE orders ADD COLUMN received_date TIMESTAMP NULL;
```

- Stores finalized orders:
  - Tied to a transaction, product, user, and total cost.

- Has extra columns:
  - user_marked_received (for delivery confirmation).
  - received_date (when the user marks it received).

Trigger: `before_insert_orders`

```
DELIMITER $$
CREATE TRIGGER before_insert_orders
BEFORE INSERT ON orders
FOR EACH ROW
BEGIN
    DECLARE next_order_id INT;
    SELECT IFNULL(MAX(CAST(SUBSTRING(order_id, 5) AS UNSIGNED)), 0) + 1
    INTO next_order_id FROM orders;
    SET NEW.order_id = CONCAT('ORD-', LPAD(next_order_id, 3, '0'));
    IF NEW.order_date IS NULL THEN
        SET NEW.order_date = CURDATE();
    END IF;
END$$
DELIMITER ;
```

- Auto-generates order_id like ORD-001.
- Automatically sets the order_date to today if not provided.

Step 7: Shipping Table

```
CREATE TABLE shipping (
    shipping_id VARCHAR(100) PRIMARY KEY,
    transaction_id VARCHAR(100) NOT NULL,
    student_number VARCHAR(7) NOT NULL,
    shipping_address VARCHAR(255) NOT NULL,
    shipping_date DATE NOT NULL,
    shipping_amount DECIMAL(10,2) NOT NULL,
    FOREIGN KEY (student_number) REFERENCES users(student_number)
);
```

- Stores shipping details like:
    - Address, shipping date, and flat rate of ₱40.
- Linked to the user and a transaction.

Trigger: `before_insert_shipping`

```
CREATE TRIGGER before_insert_shipping
BEFORE INSERT ON shipping
FOR EACH ROW
BEGIN
    DECLARE next_shipping_id INT;
    SELECT IFNULL(MAX(CAST(SUBSTRING(shipping_id, 6) AS UNSIGNED)), 0) + 1
    INTO next_shipping_id FROM shipping;
    SET NEW.shipping_id = CONCAT('SHIP-', LPAD(next_shipping_id, 3, '0'));
    IF NEW.shipping_date IS NULL THEN
        SET NEW.shipping_date = CURDATE();
    END IF;
    SET NEW.shipping_amount = 40;
END$$
DELIMITER ;
```

- Auto-generates shipping_id like SHIP-001.
- Sets default shipping_date to today.
- Always sets shipping_amount to 40.

Step 8: Payment Table

```
CREATE TABLE payment (
    payment_id VARCHAR(100) PRIMARY KEY,
    transaction_id VARCHAR(100) NOT NULL,
    student_number VARCHAR(7) NOT NULL,
    payment_amount DECIMAL(10,2) NOT NULL,
    payment_method VARCHAR(50) NOT NULL,
    payment_date DATE NOT NULL,
    FOREIGN KEY (student_number) REFERENCES users(student_number)
);
```

- Stores payment details:
    - Method (e.g., GCash, Credit Card), date, amount, etc.
- Linked to the student and the transaction.

Trigger: `before_insert_payment`

```
DELIMITER $$
CREATE TRIGGER before_insert_payment
BEFORE INSERT ON payment
FOR EACH ROW
BEGIN
    DECLARE next_payment_id INT;
    SELECT IFNULL(MAX(CAST(SUBSTRING(payment_id, 6) AS UNSIGNED)), 0) + 1
    INTO next_payment_id FROM payment;
    SET NEW.payment_id = CONCAT('PAY-', LPAD(next_payment_id, 3, '0'));
    IF NEW.payment_date IS NULL THEN
        SET NEW.payment_date = CURDATE();
    END IF;
END$$
DELIMITER ;
```

- Auto-generates payment_id like PAY-001.
- Automatically sets payment_date to today if not provided.

V. Evaluation

V.I Evaluation Criteria

Criteria	5 (Excellent)	4 (Good)	3 (Satisfactory)	2 (Needs Improvement)	1 (Unsatisfactory)	Score
Software Selection & Use (25%)	Selects the most appropriate software solutions for data creation, modification, and querying based on data model and project requirements. Demonstrates proficiency in using chosen software for all CRUD operations (Create, Read, Update, Delete).	Selects suitable software solutions for data manipulation. <Competently performs CRUD operations with some minor software-specific issues.	Selects software solutions that may not be the optimal choice for the data model. Performs basic CRUD operations but may require assistance with more complex tasks.	Selects software solutions that are not well-suited for data manipulation tasks. Struggles to perform CRUD operations effectively.	Does not demonstrate any understanding of software solutions for data manipulation.	
Understanding of Data Models (25%)	Clearly demonstrates a strong understanding of different data models (e.g., relational, hierarchical, network). Applies data model concepts effectively in software selection and query construction.	Shows a good understanding of data models and can apply them to software selection and basic query formulation.	Has a basic understanding of data models, but may encounter difficulties in applying them to software and queries.	Limited understanding of data models, leading to inappropriate software selection and difficulties with queries.	Does not demonstrate any understanding of data models.	
SQL Implementation (25%)	Writes well-structured and efficient SQL queries to perform complex data manipulation tasks on the chosen SQL storage system. Leverages advanced SQL features (e.g., joins, subqueries) appropriately.	Writes correct SQL queries to perform essential data manipulation tasks. May require assistance with more complex queries or advanced features.	Writes basic SQL queries with some syntax errors or inefficiencies. Limited use of advanced features.	Writes SQL queries with significant errors or struggles to perform basic data manipulation tasks.	Does not demonstrate any understanding of SQL or its use with storage systems.	
Error Handling & Optimization (25%)	Implements robust error handling mechanisms to identify and address potential issues during data manipulation. Optimizes queries for performance by utilizing appropriate techniques (e.g., indexing, proper joins).	Includes basic error handling in queries. Attempts to optimize queries but may not be entirely successful.	Limited use of error handling. Does not consider query optimization techniques.	Queries are prone to errors and may not function as intended. No attempt at optimization.	Does not implement error handling or query optimization.	
TOTAL: 100%						

Database Design Evaluation – Likert Scale Questionnaire

Instructions: Please rate the following statements based on your level of agreement.

Scale:
1 – Strongly Disagree
2 – Disagree
3 – Neutral
4 – Agree
5 – Strongly Agree

#	Statement	1	2	3	4	5
1	The database schema accurately reflects the requirements of the system.					
2	The entity-relationship diagram (ERD) is clear and logically structured.					
3	Tables are properly normalized to avoid redundancy.					
4	The use of primary and foreign keys is consistent and effective.					
5	Indexing is used appropriately to optimize performance.					
6	Data types chosen for each field are appropriate and efficient.					

7	Constraints (e.g., NOT NULL, UNIQUE) are properly implemented.					
8	The database design is scalable for future growth or changes.					
9	Security considerations (e.g., access control) were incorporated.					
10	The design supports efficient data retrieval and reporting.					

V.II Testing Methodology

To see how well our E-Commerce system performs, we used a mix of testing methods. We started with functionality testing, where we manually tried out different features like signing up, logging in, browsing items, and placing orders—just like a regular user would. Then we did integration testing to make sure all the features worked together smoothly.

For performance testing, we simulated what would happen if many users were shopping at the same time, and we paid close attention to how fast the pages loaded and how responsive the system was. We also conducted usability testing by asking people to try out the site and give feedback on how easy or difficult it was to use. Lastly, we ran some basic security checks to confirm that unauthorized users couldn't access admin pages and that user data, especially passwords, were properly secured.

V.III Evaluation Results

Strengths:

- Users can successfully register, log in, browse products, and complete purchases.
- Admin has full control over user data, product inventory, and order management through functional CRUD operations.
- The system ensures a smooth and responsive user experience across different sections (login, profile, shopping, and checkout).
- Clear separation of user and admin roles maintains security and usability. The interface is user-friendly and intuitive, making navigation easy even for first-time users.

Weaknesses:

- The system lacks encryption or advanced security measures for sensitive user data.

Insights Gained from Testing:

- Performing tests across different user roles helped identify small inconsistencies in behavior and access control.
- CRUD operations, especially on the admin side, improve workflow efficiency and system control.
- Users responded positively to a simple and fast shopping process.
- Testing helped prioritize which features to polish and which areas need better optimization or stronger validation rules.

VI. Discussion

VI.I Software Solutions for Data Manipulation

For our E-Commerce system, we focused on implementing CRUD (Create, Read, Update, Delete) operations primarily on the admin side. These features allow the admin to manage user data such as viewing, editing, and deleting registered accounts through simple button actions. The system uses Java as the main programming language and MySQL as the database to handle all the data operations.

Software Solutions Used:

- Java (Swing GUI) Used to design the interface where admins can perform CRUD operations. Button clicks trigger specific logic for each operation.
- MySQL Acts as the backend where user data is stored and updated. All data entered or modified through the interface is reflected in the database.

Strengths:

- Admins can efficiently manage user data through a simple, responsive interface.
- The current setup lays a solid foundation for expanding CRUD to other parts of the system like products or orders.

Limitations:

- No data encryption or security validation is implemented yet for sensitive information.

VI.II Implementation of SQL Storage Systems

The use of MYSQL in the merchandise applications serves as the foundation for the whole database system although there are things that need to be considered such as handling multiple records or data like the Order, User, Product, and Shipping. Several transactions still need to be properly handled, especially in terms of insertions and deletions. Some of the challenges experienced are mostly related to deletion and insertion in clearing orders or deleting payment manually.

VII. Conclusion

VII.I Project Summary

The COMVERSE project is a platform to process order, deliver, and manage orders of the students within the organization which makes it easier since everything is done online to avoid long queues. The key accomplishment of this project is to also show the effectiveness of using a database model by the use of MySQL along with Java language for its interface development.

VII.II Limitations and Future Improvements

The merchandise system is only for users and admin. Future improvement for the application can also be for delivery. Aside from the type of user, other improvements can also be for a real time update of the products such as updates from prices, sizes, and colors since it displays specifically for one color per item.

VII.III Contribution to Advanced Database Systems

This project is essential for the students within the organization. This can also help in promoting not just the organization but the university as a whole through the products presented on the website. This can be a good example and contribution in the field of advanced database systems, since it is also shown in this project that advertising and selling can work together in an orderly manner.

VIII. References

Abbas, N., & Farah, J. (2023, November 10). *Optimizing E-commerce Databases: A Comparative Analysis of SQL and NoSQL Solutions*. https://doi.org/10.13140/RG.2.2.22028.53121

Usman, M. (2022, December 16). *Designing The Relational Database for eCommerce Website — The Complete Guide*. Medium. https://medium.com/@usmananwaar_de/designing-the-relational-database-for-ecommerce-website-the-complete-guide-cbf905ee0cf0

Building an E-commerce Database with SQL - Datatas. (2024, November 24). Datatas. https://datatas.com/building-an-e-commerce-database-with-sql/

Shopee Philippines. (n.d.). *Shop online with Shopee Philippines*. https://shopee.ph

La Salle Merchandise Shop (n.d.). *Animo Nation*. https://animonation.com/