RMIT UNIVERSITY

Database Design and Implementation Report

ISYS2099 | Database Applications

Lecturer: Mr. Tri Dang Tran

Vo Tuong Minh - s3877562
Do Le Long An – s3963207
Phan Thanh Loi – s3864188
Tran Minh Nhat - s3924826

Table of Contents

I. Introduction..2

II. Project Description and Implementation Details 2

III. Conclusion 12

IV. Reference 13

V. Appendix..14

I. Introduction

Ecommerce systems have become a pivotal tool in the digital era when online marketplaces are blooming and rapidly developing. Lazada Group, a Southeast Asia’s leading platform, besides serving as a marketplace for sellers, also makes sure that the marketplace is sufficient and efficient enough for its customer and itself. Thus, a model that provides users with the fastest, most convenient delivery called FBL – Fulfillment by Lazada was developed and implemented to the real world.

Admitting the requisite of this application, the course ISYS2099 has provided a database project, which asked us to come up with, design, and implement a simplified FBL model for storing, querying, and managing data effectively and efficiently. In order to achieve final solution, this project requires to complete some goals including database design, performance analysis, data consistency, and data security.

Therefore, this project will involve some prior frontend and backend techniques such as NodeJS, MySQL, Mongo DB, etc. Eventually, their global implementation details will be discussed in this document hinged on the four goals mentioned previously.

II. Project Description and Implementation Details

1. Database Design

The database design phase is a crucial initialization in this project which takes part in creating a structured and efficient data warehouse solution to meet the requirements of the simplified FBL model. In this document, this phase outlines and describe the process of designing the database, including data analysis, entity-relationship diagrams (ERD), relational schema design, constraints, relationships, and considerations for non-relational documents and documents’ structure.

a. Data analysis

The scenario of this project required us to come up with a simplified Fulfill-By-Lazada (FBL) model design. Thus, we have conducted a brief analysis to understand the nature of the data to be stored that contributes to this model. Initially, the primary entities interacting within the model consist of user groups, including buyers, sellers, and warehouse administrators. In addition, product inventory updates due to Inbound Orders and Buyer Orders would be automated without involving warehouse administrators for simplicity. Another assumption is that the warehouse admin should be able to manage the data in warehouse, but they could not modify seller and buyer information, nor view their password hash. According to our transient findings of conditions and assumptions for business process, we were able to lay the first stone of defining relationships or any constraints between entities. In this process, our group managed to grasp the basic understanding to create a conceptual model which provides us with more context for our database and backend programming later on.

b. ERD and Relational Schema (including relationships and constraints)

Please see **Appendix 1 and 2** for visual aids

Entities and Their Attributes:

- **wh_admin**: username (PK), refresh_token, password_hash.
- **lazada_user**: username (PK), salt, refresh_token, password_hash.
- **buyer**: username (PK, FK).
- **seller**: username (PK, FK), shop_name (Unique).
- **warehouse**: id (PK) warehouse_name (Unique), volume, province, city, district, street, street_number.
- **product_category**: category_name (PK), parent (FK)
- **product_attribute**: attribute_name (PK), attribute_type, required.
- **product_category_attribute_association**: category (PK, FK), attribute (PK, FK).
- **product**: id (PK), title, image, product_description, category (FK), price, width, length, height, seller (FK).
- **stockpile**: product_id (PK, FK), warehouse_id (PK, FK), quantity.
- **inbound_order**: id (PK), quantity, product_id (FK), created_date, created_time, fulfilled_date, fulfilled_time, seller (FK).
- **buyer_order**: id (PK), quantity, product_id (FK), created_date, created_time, order_status, fulfilled_date, fulfilled_time, buyer (FK).

Table Relationships:

lazada_user and buyer, and seller: One-to-One relationship showing that each Lazada user can either be a buyer, or a seller.

product and product_category: Products belong to just one category, yet each category contains various associated products, resulting in the one-to-many connection between them.

product_category and product_category_attribute_association: Category x number of associated attributes = 1, while category y number of shared attributes between them >1.

product_attribute and product_category_attribute_association: Due to many-to-many connection, numerous classification assignments accompany every merchandise trait.

seller and product: One-to-Many relationship as each seller can have multiple products, but each product is associated with only one seller.

product and stockpile: One-to-Many relationship showing that each product can have multiple stockpile records in different warehouses, but each stockpile is associated with only one product.

warehouse and stockpile: One-to-Many relationship showing that each warehouse can have multiple stockpile records, but each stockpile is associated with one warehouse.

product and inbound_order: One-to-Many relationship showing that each product can be associated with multiple inbound orders, but each inbound order is for one product.

seller and inbound_order: One-to-Many relationship showing that each seller can have multiple inbound orders, but each inbound order belongs to one seller.

product and buyer_order: One-to-Many relationship showing that each product can be associated with multiple buyer orders, but each buyer order is for one product.

buyer and buyer_order: One-to-Many relationship showing that each buyer can have multiple buyer orders, but each buyer order belongs to one buyer.

Unique Constraints:

lazada_user_pk, seller_pk, buyer_pk: Ensure each username is unique in the respective tables.

seller_shop_name_un: Ensure each shop_name in seller is unique.

warehouse_warehouse_name_un: Ensure each warehouse_name in warehouse is unique.

product_category_pk: Ensure each category_name in product_category is unique.

product_attribute_pk: Ensure each attribute_name in product_attribute is unique.

product_category_attribute_association_pk: Ensure unique combinations of category and attribute.

Foreign Key Constraints:

seller_username_fk, buyer_username_fk: Connect username in seller/buyer to lazada_user.

product_category_parent_fk: Connect parent in product_category to category_name.

product_category_attribute_association_category_fk: Connect category to product_category.

product_category_attribute_association_attribute_fk: Connect attribute to product_attribute.

product_category_fk: Connect category in product to product_category.

product_seller_fk: Connect seller in product to seller.

stockpile_product_fk: Connect product_id in stockpile to id in product.

stockpile_warehouse_fk: Connect warehouse_id in stockpile to id in warehouse.

inbound_order_product_id_fk: Connect product_id in inbound_order to id in product.

inbound_order_seller_fk: Connect seller in inbound_order to username in seller.

buyer_order_product_id_fk: Connec product_id in buyer_order to id in product.

buyer_order_buyer_fk: Connect buyer in buyer_order to username in buyer.

Check Constraints:

chk_warehouse: Guarantee volume in warehouse is greater than 0.

chk_product: Guarantee valid values for price, width, length, and height in product.

chk_stockpile: Guarantee quantity in stockpile is greater than 0.

chk_inbound_order_quantity: Guarantee quantity in inbound_order is greater than 0.

chk_inbound_order_datetime: Guarantee valid datetime relationships in **inbound_order.**

chk_buyer_order_quantity: Guarantee quantity in buyer_order is greater than 0.

chk_buyer_order_datetime: Guarantee valid datetime relationships in buyer_order.

chk_buyer_order_status: Guarantee valid order statuses in buyer_order.

Justifications: These constraints help our database maintain data integrity, enforce uniqueness, and validate data relationships across the tables in the database schema.

c. Non-relational documents and their structure

For non-relational documents, our group decided to use Mongoose particularly for the category, category_attribute, and category_attribute_association tables. The non-relational collection used Mongoose, a MongoDB object modeling tool, in which the documents of product_id and attribute_name fields are stored accordingly. According to the NoSQL requirement for warehouse admin, we concluded that MySQL itself could not help us store the data in attribute_value column with various data structures (string, Boolean, number, etc.) as it only allows a column to have a single datatype, so we stored them as datatype values, and by combining the use of Mongoose framework, it should contribute a great help to this issue due to its allowance of dynamic datatypes for documents in a field. Thus, it increased the data efficiency and assisted us significantly when it comes to storing and interacting with the data in this warehouse section.

2. Performance Analysis

a. Query Optimization (Index and Partition)

Please note that no partitioning is possible since all of our tables contain foreign keys to each other's. Partitioning with foreign key is not supported for InnoDB tables by MySQL Community Server (*MySQL :: MySQL 8.0 Reference Manual :: 24.6 Restrictions and Limitations on Partitioning*, n.d.).

Warehouse table

These secondary indices are sufficient to aid the warehouse admin in efficient retrieval of warehouse data:

- unidx_warehouse_warehouse_name: to gain faster retrieval of warehouses using their (unique) name - warehouse_name column.
- idx_warehouse_volume: to gain faster retrieval of warehouses using their volume - volume column.
- idx_warehouse_address: to gain faster retrieval of warehouses using their address - province, city, district, street, and street_number columns.

Product table

In addition to basic indices, this table also requires a full-text index on the product_description column to aid efficient data retrieval by the warehouse admin, seller, and buyer:

- idx_product_title: to gain faster of products using their title - title column.
- idx_fulltext_product_product_description: to gain faster **full-text** search and filtering of products using their description - product_description column.
- idx_product_category: to gain faster retrieval of products using their category - category column.
- idx_product_price: to gain faster retrieval of products using their price - price column.
- idx_product_dimensions: to gain faster retrieval of products using their 3 dimensions- width, length, and height columns.
- idx_product_seller: to gain faster retrieval of products using their seller - seller column.

Inbound_order table

These secondary indices are sufficient to aid the seller in efficient retrieval of inbound order data:

- idx_inbound_order_quantity: to gain faster retrieval of inbound orders using their quantity - quantity column.

- idx_inbound_order_product_id: to gain faster retrieval of inbound orders using their associated product - product_id column.
- idx_inbound_order_created_datetime: to gain faster retrieval of inbound orders using their created date and time - created_date and created_time columns.
- idx_inbound_order_fulfilled_datetime: to gain faster retrieval of inbound orders using their fulfilled date and time - fulfilled_date and fulfilled_time columns.
- idx_inbound_order_seller: to gain faster retrieval of inbound orders using their seller - seller column.

Buyer_order table

These secondary indices are sufficient to aid the buyer in efficient retrieval of buyer order data:

- idx_buyer_order_quantity: to gain faster retrieval of buyer orders using their quantity - quantity column.
- idx_buyer_order_product_id: to gain faster retrieval of buyer orders using their associated product - product_id column.
- idx_buyer_order_created_datetime: to gain faster retrieval of buyer orders using their created date and time - created_date and created_time columns.
- idx_buyer_order_order_status: to gain faster retrieval of buyer orders using their status - order_status column.
- idx_buyer_order_fulfilled_datetime: to gain faster retrieval of buyer orders using their fulfilled date and time - fulfilled_date and fulfilled_time columns.
- idx_buyer_order_buyer: to gain faster retrieval of buyer orders using their seller seller column.

Other tables

The structure of the remaining tables is simple enough for their primary index (automatically generated by MySQL on the primary key column) to be good enough for database performance.

b. Concurrent Access

Without the database system exerting some control over what gets updated by who and when, all kinds of data integrity and consistency issues can arise. Concurrent database access refers to the situation where the database is being accessed from more than one connection (user) at a time (Concurrent Database Access, n.d.). Therefore, to ensure the optimal performance of concurrent access in our SQL database, we have implemented SQL transactions using START TRANSACTIONS, COMMIT, ROLLBACK statement. The implication of these statements provided atomicity, which is vital for managing concurrent access.

Eventually, ensuring that a series of database operations either all succeed, or all fail.

Another point to consider is the effects of isolation levels on concurrent access. To be more specific, the FOR SHARE and FOR UPDATE are used accordingly in every SELECT statement in a transaction to control data visibility and locking behavior. For instance, we used the shared locks (FOR SHARE) to indicate MySQL that we are selecting the rows from the table only for reading and not for modification before the end of the transaction. This is to ensure that there would not be any exclusive locks acquired (using FOR UPDATE) until the shared locks are deactivated. Meanwhile, the FOR-UPDATE clause informs MySQL that the rows we are selecting can be modified in the next phase within the current transaction, this will put other transactions on that wait list until the current transaction either commits or rollbacks.

3. Database Consistency

a. Transactions:

Consisting of numerous database duties bundled into a single unit of work, transactions ensure that either the task is completed successfully, or the database reaches a stable state in case of an issue (rwestMSFT, 2023). Our business_rules.sql file transactions began with the START TRANSACTION statement followed by COMMIT or ROLLBACK. Procedures like sp_move_product, sp_delete_warehouse, and sp_fulfill_inbound_order rely on transactions to achieve atomicity. Procedure execution integrity is preserved by rolling back the transaction in case of error occurrence.

b. Triggers:

Special procedures known as triggers are carried out automatically in response to certain database changes, such as INSERT, UPDATE, or DELETE operations on tables (Markingmyname, 2022). With the fact that the trig_reject_buyer_order trigger is activated upon updating the buyer_order table, and order status analysis typically results in relevant actions being taken upon detecting invalid changes like cancelled orders being reactivated. Thus, our code relies on trigger implementation to enforce business rules and uphold data quality consistency.

c. Error Handling:

For handling exceptions effectively, error handling mechanisms are incorporated within the triggers and stored procedures. For example, the DECLARE CONTINUE HANDLER FOR SQL EXCEPTION statement is used to control how a specific stored procedure or a trigger should behave whenever an exception is caught. Thus, appropriate actions are taken, such as indicating a failed transaction by setting a rollback flag or providing feedback through custom error messages.

Summary: In this section, we have implemented the combination of transactions, triggers, and error handling to ensure data changes attach to business rules and preserve consistency in the database. By choosing this approach, it helped us prevent such scenarios where partial changes are made, or data becomes inconsistent due to errors or unexpected events.

4. Data Security

a. The usage of database permissions

The Warehouse Administrator shall be able to:

- CRUD *their own* account (username and password): wh_admin table.
- CRUD warehouses: warehouse table.
- CRUD categories and their attributes: product_category, product_attribute, and product_category_attribute_association table.
- CRUD products in case a product needs to be manually edited: product table.
- CRUD stockpiles, to move products between warehouses.
- CRUD inbound orders, to fulfill those orders, or in case some inbound order needs to be manually edited: inbound_order table.
- CRUD buyer orders, to fulfill buyer orders, in case some buyer order needs to be manually edited: buyer_order table.
- View seller username: SELECT privilege on seller table, restricted to username column.
- View buyer username: SELECT privilege on buyer table, restricted to username column.

The Warehouse Administrator shall not be able to:

- Modify seller information.
- View seller password hash.
- Modify buyer information.
- View buyer password hash.

Summary: The Warehouse Administrator has unlimited CRUD privileges on all tables within the database, except for the seller, and buyer tables, where they only have the SELECT privilege on. The Warehouse Administrator also does not have any access to the lazada_user, which contains sensitive password hash for all Lazada users.

The Seller shall be able to:

- CRUD *their own* account (username and password): seller and lazada_user table.
- CRUD *their own* products: product table.
- View categories and their attributes to support the above operation: product_category, product_attribute, and product_category_attribute_association table.
- CRUD inbound orders: inbound_order table.
- CRU stockpile information to support the above operation.
- View warehouse information, to support the above operation: SELECT privilege on warehouse table.

- View buyer orders for *their own* products.
- View buyer username to support the above operation: SELECT privilege on buyer and lazada_user table, restricted to username column.

The Seller shall not be able to:

- Have access to Warehouse Admin information.
- Modify warehouses.
- Delete stockpile information.
- View buyer password hash.
- Modify buyer information.
- Modify buyer orders.
- Modify any product category and their attributes.

Summary: The Seller has unlimited CRUD privileges only on the tables required for their main operations: account management, product management, and inbound order management, plus some other SELECT privileges on relevant information to support these operations.

The Buyer shall be able to:

- CRUD *their own* account (username and password): seller and lazada_user table.
- View products: product table, excluding the id and warehouse_id columns.
- CRUD stockpile information for when we simulate buyer_oder fulfillment on the buyer's end: stockpile table.
- View warehouse id, to support the above operation: warehouse table, restricted to only the id column.
- View categories and their attributes to support the above operation: product_category, product_attribute, and product_category_attribute_association table.
- CRUD buyer orders: buyer_order table.
- View buyer orders for **their own** products.
- View buyer username to support the above operation: SELECT privilege on buyer and lazada_user table, restricted to username column.

The Buyer shall not be able to:

- Have access to Warehouse Admin information.
- Have any access to warehouse information, except the warehouse id.
- Have any access to inbound order information.
- Modify product information.
- Modify any product category and their attributes.
- View seller password hash.
- Modify seller information.

Summary: The Buyer has unlimited CRUD privileges only on the tables required for their main operations: account management, viewing products, and buyer order management, plus some other SELECT privileges on relevant information to support these operations.

b. SQL injection prevention

In this part, our group has tried to utilize the least privilege principle; as mentioned above, we limited the permissions of database users. Users (buyer, seller, and warehouse admin) were granted the necessary permissions to perform their required tasks. This reduced the possible damage of a successful breach that could occur to our database. As such we applied stored procedure due to its advantage in restricting access to the data by allowing users to manipulate the data only through stored procedures that executed with their definer's privileges (GeeksforGeeks, 2020).

Another essential point to consider is that we have implemented HTML escape characters in the front-end section. This serves as a robust mechanism for inputting data into the front end and subsequently sending it to the back-end database. The reason behind this approach is that whenever a user adds information using a form, the information is automatically encoded into HTML escape characters. This encoding ensures that special characters like semicolons, brackets, and others are not misunderstood as queries within the database.

c. Password hashing

In our application, we implemented the node.bcrypt.js, a bcrypt library for NodeJS, which was installed via NPM. We believe that bcrypt is the best implementation to fulfill this requirement. Since it is known as a cryptographic hash function originated for password hashing and safe storing in the backend of applications in a way that is less susceptible to dictionary-based cyberattacks (Grigutytė, 2023).

The compareSync function from bcrypt library works by taking the plain text password and the hashed password as inputs. The hashed password actually contains the salt used to hash the original password. This salt is extracted and used to hash the plain text password again. The resulting hash is then compared with the original hashed password. If they match, it means the plain text password is correct.

The reason why we did not store the salt separately is because bcrypt can automatically include it as part of the hashed password. For further explanation, the moment the system hashes password, it actually produces a string that contains all information about the version of bcrypt used, including the cost factor (which determines how computationally intensive the hashing process is), the salt, and finally the hash itself, all of which are linked together into a single string, and eventually, is stored in the database accordingly.

d. Additional security mechanisms

Our project benefits from employing a `.env` file to conceal important details. Sharing our code requires the .env file to keep sensitive data like passwords and API keys separate from the actual code (Schimelpfening, n.d.).

Protection of sensitive information and user data takes front seat thanks to JWT integration. This aspect plays a crucial role in safeguarding authentications/authorizations procedures. Decreasing database queries helps safeguard sensitive data via JWTs, per Frontegg (2023). Data integrity protection follows login verification by means of JWTs comprising individualized claim sets or ID Tokens (Auth n.d.).

A further point to discuss is that our JWTs are not stored in a Local Storage but in Cookies to mostly avoid the Cross-Site Scripting attacks. As such, if your application stores tokens (which might be used as credentials for API calls) in Local Storage, an XSS attack could potentially lead to these tokens being stolen (Cure, 2015). As a result, there is a high likelihood of account hijacking, credential theft, data leakage, and other security breaches that could compromise users' sensitive data.

III. Conclusion

Throughout this report, we have so far discussed the four leading points regarding database design, performance analysis, data consistency, and data security. The combination of them is a great catalyst to complete a basic streamlined Fulfill-By-Lazada (FBL) model database.

Overall, the database design helped us form our foundation of the database models as well as its necessary business rules while the performance analysis covered our importance query optimizations which explained the implementation of indices, and the decisions regarding concurrency access for optimal performance. As a major part and also a great contributor to our database performance, the efficient combination of transactions, triggers, and error handling enforced business rules, and preserve data consistency in every part of various database operations. Regarding data security, we used stored procedures to restrict user permissions to their intended roles, fortified our SQL injection prevention with least privilege principle and HTML escaped characters, securely hashed passwords with bcrypt, and added extra safeguards like .env file and JWTs in cookies to effectively enhance the protection of sensitive data.

In conclusion, this project has been a great, yet simplified example of the FBL model database, addressing key objectives in database design, performance, data consistency, and security. This has given us a chance to build a robust foundation to ensure efficiency, reliability, and safety of our future database applications in the dynamic landscape of e-commerce systems.

IV. Reference

Concurrent Database access. (n.d.). https://docs.raima.comaccess..4_1/ug/sql/ConcurrentDbAccess.htm

MySQL :: MySQL 8.0 Reference Manual :: 24.6 Restrictions and Limitations on Partitioning. (n.d.). https://dev.mysql.com/doc/refman/8.0/en/partitioning-limitations.html

rwestMSFT. (2023, February 28). *Transactions (Transact-SQL) - SQL server*. Microsoft Learn. https://learn.microsoft.com/en-us/sql/t-sql/language-elements/transactions-transact-sql?view=sql-server-ver16

Markingmyname. (2022, December 29). *CREATE TRIGGER (Transact-SQL) - SQL Server*. Microsoft Learn. https://learn.microsoft.com/en-us/sql/t-sql/statements/create-trigger-transact-sql?view=sql-server-ver16

GeeksforGeeks. (2020). Advantages and Disadvantages of using Stored Procedures SQL. *GeeksforGeeks*. https://www.geeksforgeeks.org/advantages-and-disadvantages-of-using-stored-procedures-sql/

Grigutytė, M. (2023). What is Bcrypt and how it works? | NordVPN. *NordVPN*. https://nordvpn.com/blog/what-is-bcrypt/#:~:text=Bcrypt%20is%20a%20cryptographic%20hash,susceptible%20to%20dictionary%2Dbased%20cyberattacks.

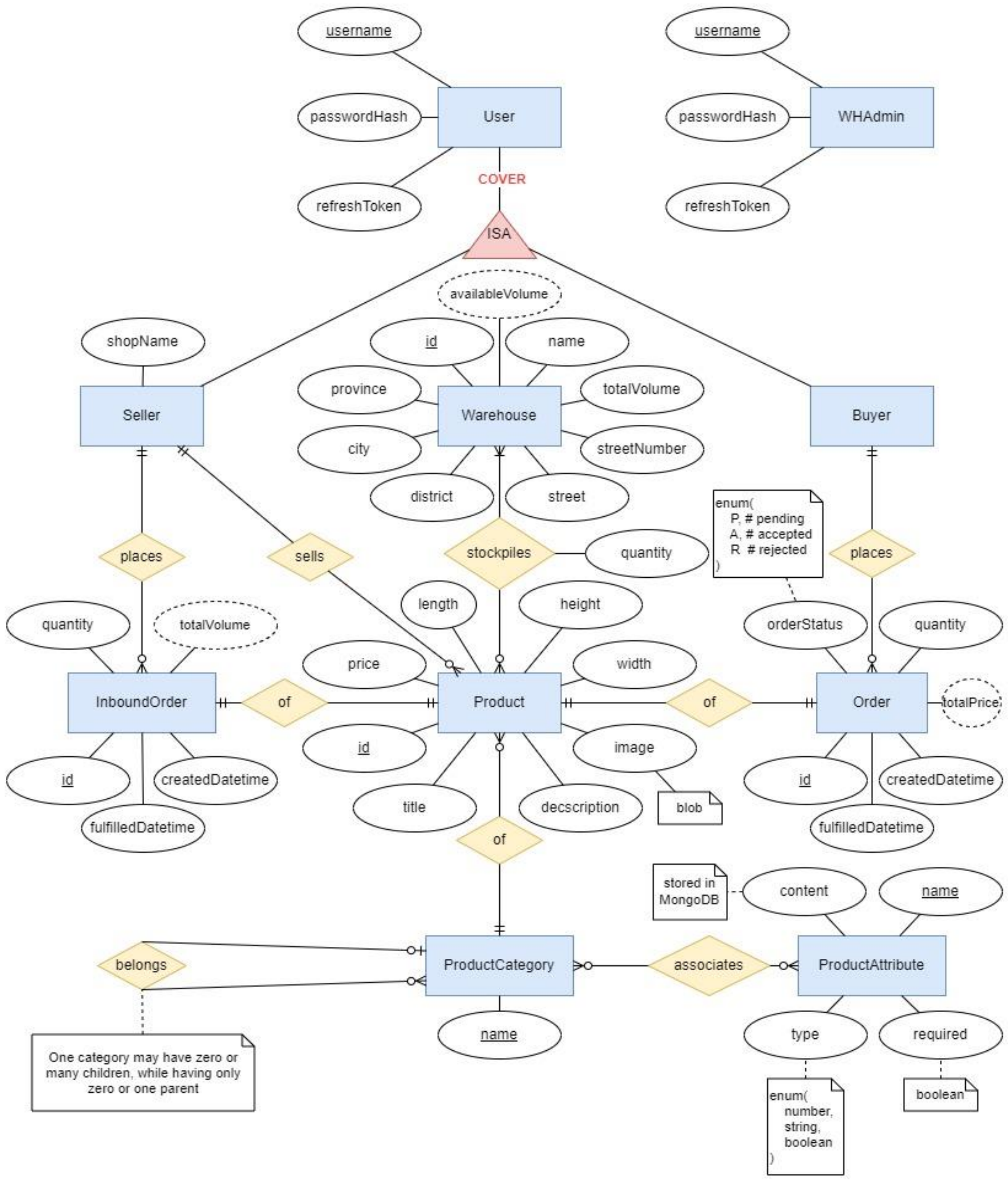
Schimelpfening, A. (n.d.). *How to use dotenv in your CSE341 projects*. Gist. https://gist.github.com/727021/5e17e739bd0a286cd5e30b2c8f69ed2d#using-env

Frontegg. (2023). OAuth vs. JWT: What Is the Difference? Can You Use Them Together? *Frontegg*. https://frontegg.com/blog/oauth-vs-jwt#:~:text=JWT%20Advantages&text=This%20eliminates%20the%20need%20to,sends%20it%20to%20the%20client.

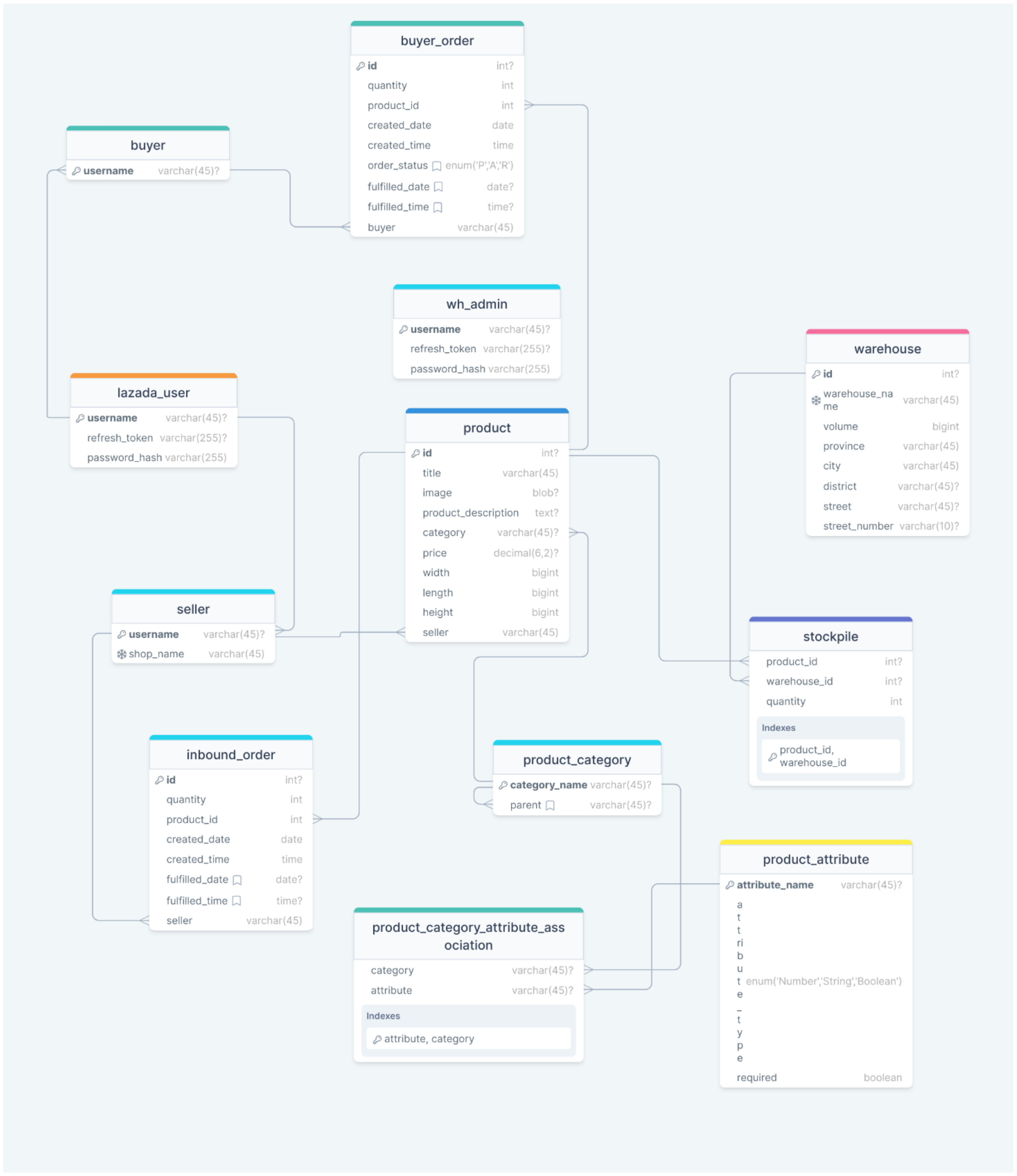
Auth. (n.d.). *JSON Web Tokens*. Auth0 Docs. https://auth0.com/docs/secure/tokens/json-web-tokens

Cure, A. (2015). C#/.NET/Core training in Denver, CO – May 2019. *cypressdatadefense.com*. https://cypressdatadefense.com/blog/cross-site-scripting-vulnerability/

V. Appendix



Appendix 1. ERD



Appendix 2. Relational Schema