Micah Simmerman
Design and Analysis of Database Systems
CSPB 3287, Spring 2023

Project Proposal for a Peer-to-Peer Retail Application Database

Purpose

This proposal will attempt to outline the basic design choices and tools for creating a model P2P marketplace data management system. This essay will provide a roadmap for the completion of the project idea outlined in my initial ideation essay. I will begin by describing the various sections of the entity-relationship model that I plan on using and conclude with a brief discussion on methods and tools.

The Entity-Relationship Model

The good people of FabricBlog.inc have provided a solid general outline that I plan on using as a basis for my early design choices. These entity relationship diagrams describe many commonly used ecommerce design patterns.

As you read through this document, you will notice frequent use of artificial primary keys, a common design choice in many ecommerce platforms. To understand why this design choice is helpful, imagine a user who has recently lost access to their account and needs to generate a new one as quickly as possible. Timestamping is another a common design choice in ecommerce retail applications, because of the flexibility it affords to conflict management systems. These timestamp attributes are typically generated by “on insert”, “on update”, and “on delete” triggers.

The User-Admin Management Tier

The user management section is a sub-tier of the overarching database design where the user records are kept. Here, a “user” is defined as anyone who can engage in buying or selling activities online.

The core user management schema (shown in the right half of figure 1) is composed of the **user**, **user_address**, and **user_payment** tables. These tables are well normalized and reference the primary key of the user table through the **user_id** foreign keys.

Administrator (AKA “Seller”) Management

The **adminuser** and **admin_type** relations (shown in the left half of figure 1) are associated with the core user management schema. Administrators are special user subtypes who possess admin privileges. The adminuser table is a weak entity supported by the user relation. As such, the adminuser table adopts its primary key (which is also a foreign key) from the user table. This common key can be used to perform fast record searches in the user table, as well as the relations that the user table associates with. Each adminuser can possess at most one admin_type, ensuring an (optional) one-to-one cardinality is held between the user and adminuser tables (see figure 1 for more detail).

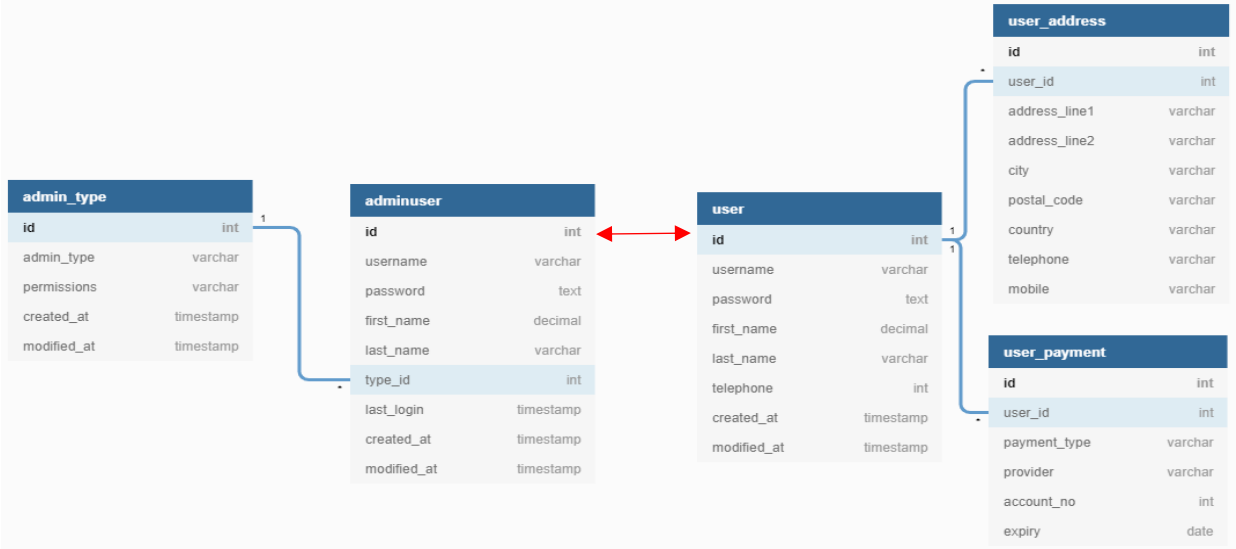

Figure 1.) The user-admin management tier. The adminuser table is a weak entity supported by the user table. Numeric attributes of user_address, such as *telephone* and *postal_code* are defined as varchars to facilitate common transactions, including the generation of sales receipts. The permissions attribute of admin_type will be replaced with a new comma-delimited and **fixed-field text string** (VARCHAR) a list of the product ids associated with the seller.

The Product Management Tier

The product management tier is composed of the **product**, **product_category**, **product_inventory**, and **discount** relations. The product table contains basic information about a product being sold, including its **id**, **name**, description (**desc**), **SKU**, **category_id**, **inventory_id**, **price**, **discount_id**, and a set of **record timestamps**.

Users can interact with the product management tier in one of two ways, depending on whether the user is an admin with permissions to modify a particular product. Users who possess admin privileges are free to modify any attribute in the product management tier, except for time stamp attributes, which are modified and controlled by triggers.

For users without admin privileges, records in the product management tier are **read-only**. The only attribute that a base user can modify at this level is **product_inventory.quantity**. Namely, by invoking a trigger during the shopping process that alters the inventory of the selected product (i.e., by making a purchase).

The product table attributes **category_id**, **inventory_id**, and **discount_id** are foreign keys that provide direct access to records located in the **product_category**, **product_inventory**, and **discount** tables, respectively. Each of these tables is described in the list below.

- **product_category -** This table contains records with the attributes **id**, **name**, **desc** (“description”), and a collection of **record timestamps**. Each product belongs to a single category, while several products may exist under a category id.

- **product_inventory** - Each product is associated with a product_inventory record that contains a **quantity** attribute and an important set of **timestamps** (i.e., **created_at**, **modified_at**, and **deleted_at**). This table is vital to the core transactions of the shopping process and must be updated whenever a sale is made.
- **discount** – this table possesses the attributes **name**, **description**, **discount_percent**, **active *(T/F)***, and a set of **record timestamps**. Discount is an optional relationship, and many products can refer to the same discount. Only one discount may be applied to a single product at any time.

Figure 2.) The product management tier. Foreign keys of the product table provide direct access to information about the category, inventory, and discounts that apply to a given product. Users with appropriate admin privileges can modify these relations (except for timestamps) at any time. Base users can access a limited cross-section of these attributes on a read-only basis.

<u>The Shopping Process</u>

The shopping process is the next level of granularity in this proposed database system. The role of this tier is to connect the user and product management systems through a common set of weak entity relationships and shopping process triggers. Two additional logic tiers, **cart management** and **order management**, are introduced at this layer of the system. Each of these sections is composed of a set of weak entity relationships supported by the product and user tables. Each of these is described below.

Cart Management

Cart management (highlighted green in figure 3), is composed of the **cart_item** and **shopping_session** weak entity tables. The cart_item relation is supported by the product and shopping_session tables, while shopping_session is supported by user. A single shopping_session can support multiple cart_items, and each cart_item is associated with at most one shopping_session. Cart item records indicate the shopping_session record they belong to with the session_id foreign key.

Order Management

The order management tier (highlighted in red) is composed of the **order_items**, **order_details**, and **payment_details** tables. The cart management system (highlighted in green) is responsible for combining product and user data that describes a user's shopping session. When the user completes a purchase, the cart management system produces a new record in the order management tier. This insert is used as the basis of a trigger that populates the remaining fields of the order management system with the appropriate user data. This additional information is kept in the user management tier.

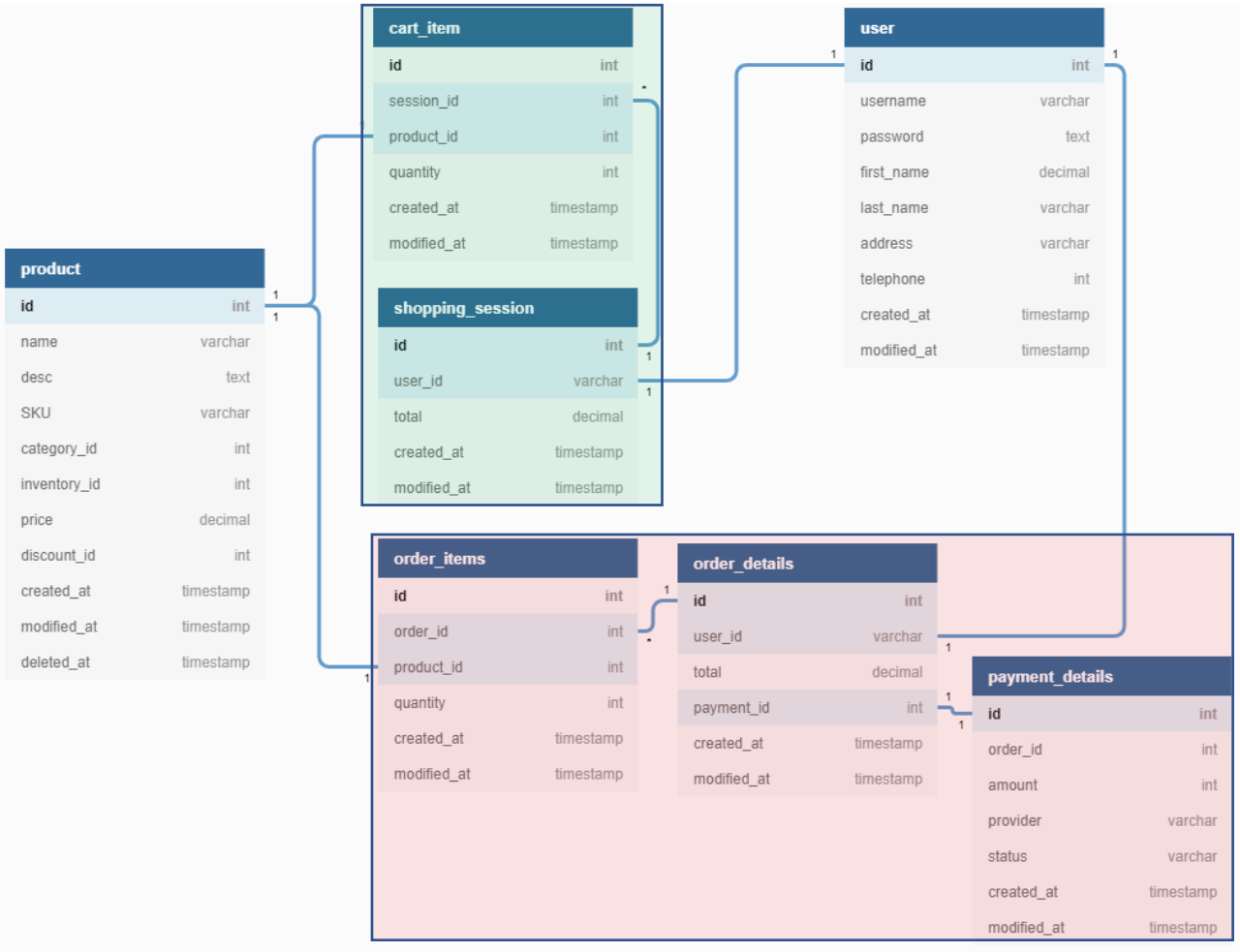

Figure 3.) The shopping process. The cart management system (green) is responsible for combining product and user data, as well as creating transient records that are used to populate the tables of the order management system (red). At least one trigger is required to connect these logical tiers.

Tools and Methods for Developing the Database

In my original project ideation, I mentioned an interest in learning about a NoSQL database technology. After a bit of research, Amazon DynamoDB appears to be a top choice among NoSQL vendors for this type of database. This opinion appears to be based on the factors of cost and overall reputation within the ecommerce community. According to a recent listing by the vendor (Amazon), DynamoDB currently supports up to 200 million reads/writes per month and is included in the free tier of Amazon Web Services (AWS).

However, I plan to reserve this NoSQL section until the end of my project. I will instead focus my efforts on designing the numerous schemas, relations, and triggers that will factor into the overall database. Once I am confident in the overall design and functionality of my relational database, I will experiment by migrating the more volatile sections of my database to NoSQL, if time permits.

In the initial stages of my project, I plan to use the MySQL server credentials that were provided with the course, SQLAlchemy, and a good old-fashioned python notebook. I will use these tools to create the relations and test the triggers that I will need to implement during the project. I have yet to locate any test records that I can load my database with. If I cannot locate these records soon, I will most likely fabricate some data myself by using a python script.