Rubber Duck Cult's Final Project Data PDF

Members:

Ahnaf Tausif
Devin Curry
Liam Stine
Morgan Rupert

Introduction

We are the Rubber Duck Cult and we decided to create a food ordering app called Grub & Go. The intention of this app was to create a way for consumers to order food from various restaurants and have it delivered to their doorstep. The theory behind this design choice is that convenience can be a great selling point, and with Grub & Go, we plan to capitalize on that. With the idea of people wanting fast and easy food, Grub & Go makes that plan happen. Customers may browse from various different restaurants and place orders with just a few taps. Restaurants can create their own menus, manage said menus, track incoming orders, and update their availability quickly and accurately. With clear order statuses, accurate pricing, and strong auditing for every important interaction, Grub & Go ensures convenience and trust for all involved parties of the transaction.

Data Generation

The data we used for our design and database was handcrafted to help express our program in its entirety.

Account		
BIGINT	account_id	PK
VARCHAR	email	
VARCHAR	password_hash	
ENUM	role	
DATETIME	created_at	
DATETIME	updated_at	

For the accounts, we followed a table that allowed for the customers and restaurant owners to follow. Each account has an account ID, email, password (which will be stored into a hash for encryption), a role, and a date/time for when it was created or updated. This allows for the user of the account to have substantial access to their account while still keeping it secure and private. This also ensures that every account is unique by its ID number. A role may be selected for an account that is either a Customer role or Restaurant role. With this role, Customers and Restaurants will be able to do different things. Customers will be able to order food and have it delivered, whereas Restaurants will be able to create menus and update their availability accordingly. Lastly, logins to any account are rate-limited and are validated through a password hash to ensure security.

Customers, when they create accounts, are able to enter their own name to tie to their account, and then it is linked to its account. Customer ID is linked to the Account ID to tie this process together.

Customer		
BIGINT	customer_id	PK,FK
VARCHAR	customer_name	

Restaurant		
BIGINT	restaurant_id	PK,FK
VARCHAR	restaurant_name	
TINYINT	is_open	

Restaurants, on the other hand, are able to enter their Restaurant name and if they are currently open. Restaurant ID is linked to their Account ID, similarly to the customer accounts. This process creates an effective way to track accounts across multiple tables.

Next is the Menu table. This table contains data that relates to the restaurant it is tied to via the restaurant ID. This creates a push that menus must only exist because of a restaurant, and cannot exist alone. Menus contain their name, if they're active, and then a backend log of when it was created or updated. With this setup, menus can have names to specify details of itself, such as "Vegan", "Vegetarian", and "Dinner", to help further specify itself to a target audience. The active tag can be used to create menu cycles, so that certain menus, such

Menu		
BIGINT	menu_id	PK
BIGINT	restaurant_id	FK
VARCHAR	name	
TINYINT	is_active	
DATETIME	created_at	
DATETIME	updated_at	

as “Dinner”, are only active during specific hours. If the menu is not active, it will not be able to be seen.

MenuItem		
BIGINT	menu_item_id	PK
BIGINT	menu_id	FK
VARCHAR	name	
TEXT	description	
DECIMAL	price	
TINYINT	is_available	
DATETIME	created_at	
DATETIME	updated_at	

MenuItem is the next table, which is tied into the Menu table. This table contains the name of the item, the description of the item, the price, if it's available, and then another audit log of when the item was created or updated. The name of the item is simple, it allows the item to be named when placed into the menu. The description allows for the restaurant to give an item description for their item, such as a flavorful description, a list of potential allergens in their food, and other various bits of information. The price is straightforward, it is the cost of the menu item, which is placed near the side of the item when on the menu. When items are ordered, a snapshot is created of the item at the time it was ordered and logs the information so that future changes may not affect past orders. If a menu item is changed or removed, it supports referential integrity so that a cascade will not occur within the system.

Order is the next table, which is tied to customers this time. Orders can be placed by customers alone, and an order cannot exist without a customer. Within these orders, a status is provided, the submission time of the order, when it was completed, and when it was archived for proper logging/auditing. The order also calculates the subtotal, the tax, and the total of the order cost for the customer. It then keeps a log of when the order was created or updated for auditing purposes as well. This process allows for customers to create an order that takes items from a restaurant and compile them together to finalize and pay for their own food. Each order must contain at least one order item as well, there should never be an order created that does not contain any items. Statuses for the order (Ordered, Delivered, Cancelled) are validated between flows through each state, and once orders are either completed or cancelled, the items are unchangeable. Any and all critical state changes of an order are audited to make sure any logs can reveal any issues, should they occur during this entire process.

Order		
BIGINT	order_id	PK
BIGINT	customer_id	FK
BIGINT	restaurant_id	FK
ENUM	status	
DATETIME	submitted_at	
DATETIME	completed_at	
DATETIME	archived_at	
DECIMAL	subtotal	
DECIMAL	tax	
DECIMAL	total	
DATETIME	created_at	
DATETIME	updated_at	

OrderItem		
BIGINT	order_item_id	PK
BIGINT	order_id	FK
BIGINT	menu_item_id	FK
INT	quantity	
DECIMAL	unit_price	
TEXT	notes	

Lastly is the OrderItem table, this table exists as an extension of the Order table, and contains the item quantity, the item price, and any additional notes by the customer. This process is to help link each step together smoothly within the order process and help make sure everything works fine. The OrderItem table also connects to the MenuItem to make sure the item being ordered is a valid item. The quantity and price are self-explanatory, and that they represent the number of that specific item ordered, and the price of the item ordered at the time it was ordered. The quantity cannot be a negative integer, and must be of equal value to or greater than 1. A backend process runs to make sure that no order item quantity is less than at least 1. The notes are to allow for specific order instructions, such as leaving onions off a burger, or to substitute something from the item with something else. Once this order is completed or cancelled, this section cannot be changed.

Business Rules

FN: Functional
NF: Non-Functional

Account & Identity

BR-001 (FN)	Single Account Per Party	Every Account belongs to exactly one party, either a Customer or a Restaurant. Each customer or Restaurant is backed by exactly one Account.
BR-002 (FN)	Account Identity	Account and Email must be unique across all accounts.
BR-003 (NF)	Authentication Data	Passwords are stored in a hash and login attempts are rate-limited.
BR-004 (NF)	Account Creation Audit	Creating, deactivating, and deleting accounts must be auditable with time and actor recorded.

Customers

BR-005 (FN)	Customer Contact	A Customer must have at least one method of contact (email or phone number).
BR-006 (NF)	Payment Encryption	Payment tokens are stored securely, and are not left as raw card numbers.
BR-007 (FN)	Permissions	Customers can perform actions only for their own accounts and orders.

Restaurants

BR-008 (FN)	Menu Ownership	A Restaurant may own one or more menus at any given time. A Menu cannot exist without a Restaurant.
BR-009 (FN)	Status & Hours	Restaurants will have an operating status {Open, Temporarily Closed, Permanently Closed} and will display operating hours for each day. Menu availability and accepting orders depend on the status of the restaurant.

Menus & MenuItems

BR-010 (FN)	Menu → Restaurant Cardinality	Each Menu belongs to exactly one Restaurant, and a Restaurant may have multiple labeled menus (i.e. Lunch, Dinner, Specials).
BR-011 (FN)	Menu Lifecycle	Menus have an operating status (Active/Inactive). Inactive menus will not be shown to customers and cannot be ordered from.
BR-012 (FN)	MenuItem → Menu Cardinality	Each MenuItem belongs to exactly one Menu. Menus may have zero to many MenuItems.
BR-013 (FN)	Item-Active Flag	Menu items have availability flags and optional time-based availability (i.e. Only available between 11:00 - 14:00).
BR-014 (FN)	Item Snapshots	When an order is placed, the order stores a log of the MenuItem alongside all of its details, such as name, description, and price to preserve historical accuracy. Subsequent changes to the MenuItem will not affect past orders.
BR-015 (FN)	Item Uniqueness	Within a single Menu, MenuItem.name must be unique or use the MenuItemSKU for uniqueness.
BR-016 (FN)	Modifiers	MenuItems may have zero to many modifiers. Each modifier option has a price delta. Constraints on modifiers (min/max selections) must be stored.

Orders & OrderItems

BR-017 (FN)	Order Ownership	Each Order is placed by exactly one Customer.
BR-018 (FN)	Restaurant Target	Each Order targets exactly one Restaurant.
BR-019 (NF)	Order Items Required	An Order must contain at least one OrderItem.
BR-020 (FN)	OrderItem → MenuItem Reference	Each OrderItem references exactly one MenuItem (by ID) and stores the snapshot of the item (i.e. name, unit, price, modifiers, etc.)
BR-021 (NF)	Quantity	Each OrderItem has a positive integer quantity (i.e. a quantity greater than or equal to 1).
BR-022 (FN)	Order Totals	Order totals are computed from [sum(orderitem.unit_price * quantity) + fees + taxes + tip - discount]. Totals are stored on the order once finalized.
BR-023 (FN)	Immutable Finalization	Once an order has been finalized (Completed or Cancelled if a refund is processed), its line items and totals are immutable.
BR-024 (NF)	Order Timestamps	Orders record key timestamps, such as created_at, confirmed_at, prepared_at, picked_up_at, delivered_at, cancelled_at.

Order Lifecycle & Status

BR-025 (FN)	Status Values	Orders have statuses such as {Created, Confirmed, Preparing, Ready, OutForDelivery, Delivered, Cancelled, Failed}.
BR-026 (FN)	Allowed Transitions	Only certain transitions are allowed. Implementation must enforce allowed transitions. An example canonical flow would be {Created - Confirmed - Preparing - Ready - OutForDelivery - Delivered}. Created or Confirmed may transition to Cancelled. Any final state {Delivered, Cancelled, Failed} is terminal.

BR-027 (FN)	Refunds	Refunds are triggered based on cancellation timing, payment method constraints, or failure from the restaurant. Refunds are recorded as financial transactions and reference the original Order.

Payment Criteria

BR-028 (FN)	Price Source & History	MenuItem prices are the current price seen on the menu, but orders store the price snapshot at the time of the order for historical accuracy. Price history may be auditable.
BR-029 (FN)	Taxes	Taxes are calculated per order based on restaurant jurisdiction and stored on the order. Tax rates used must be recorded on the order for auditing purposes.
BR-030 (NF)	Rounding	Currency rounding rules must be defined (i.e. round to cents using bankers/round half up) and applied consistently.

Explanation of Business Rules:

Functional:

I placed these business rules into the Functional category since they mainly deal with the core operations and features that the system must perform for Grub 'N' Go to operate effectively. These rules define the required behavior of the system in response to user actions, such as creating accounts, placing orders, managing menus, and calculating totals. They ensure that all essential relationships between entities (such as customers, restaurants, menus, menu items, orders, and order items) are properly enforced and maintained. Functional rules also handle validations, such as ensuring orders contain at least one item, quantities are correct, and only valid state transitions occur for order processing. Overall, these rules guarantee that the system performs its intended tasks reliably and consistently.

Non-Functional:

I placed these business rules into the Non-functional category since they mainly deal with back-end security and performance for the system. These business rules also regulate how the program/system will run as time progresses, showing different areas for breakpoints in the process of ordering food through Grub 'N' Go. It also shows fallback plans in the event of an error or mistake in the code that can be identified. Lastly, the menu is created in a way that allows ease of editing for Restaurants for creation and deletion due to their independence with the items. This idea helps the backend maintain its integrity should any errors occur in the display of menu items.