Take-Home Project 2

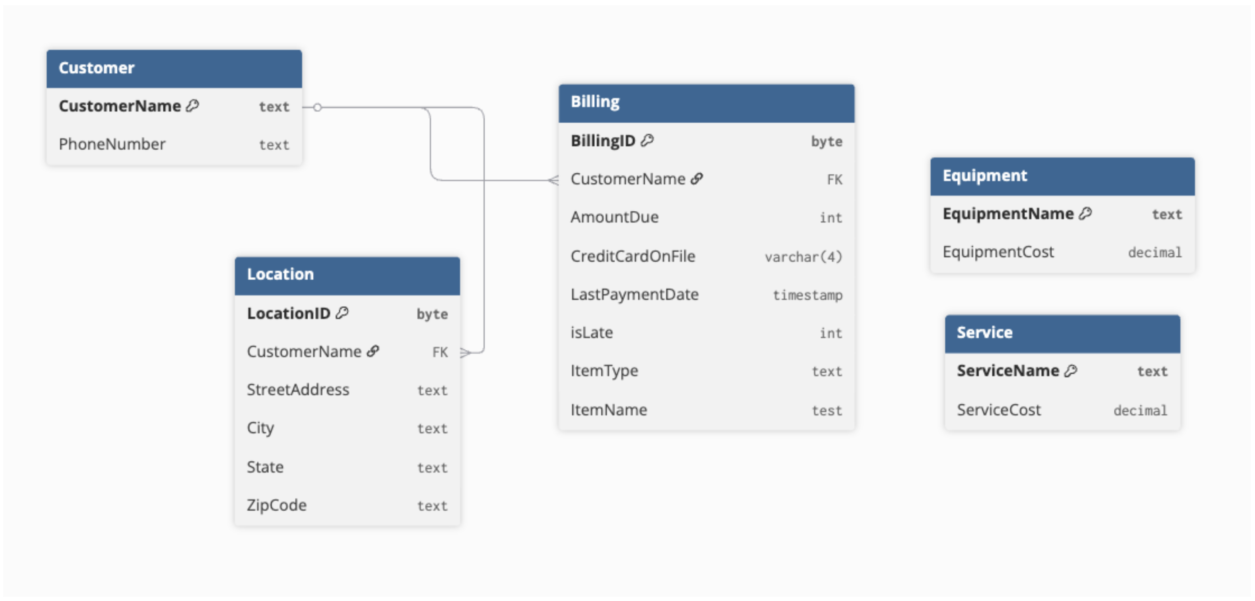
Overview

You will build an internet/cable company's customer tracking system. Your customer (the internet company) requires the following from your application:

- You must store the following information:
  - Customer information, this must include the customer's name, number, and the locations where they have services from the company. Note, a customer can own multiple locations.
  - Address of locations (1234 Street Rd. City, State)
  - Equipment at locations (router, modem, etc.).
  - Services at locations (internet, cable, etc.)
  - Cost of services and equipment. These are flat rates per service/equipment that can be used to create customer bills.
  - Billing information, including amount due each month, credit card on file, when the last payment was made, and if they are late on payment (You can assume they bill each month, so it can simply be the month (August) rather than the date (Jan 01 2023). If you want to store dates, that is fine too.)
- The user must be able to:
  - Insert new customers
  - Remove customers
  - Search for customers by name or location, retrieve the last payment date for customer.
  - View all customers who are late on payment
  - Update last payment made
  - Add and remove services and equipment for a customer
  - Change cost of service/equipment
  - Change bill amount due each month for a customer (or you can automatically due this when service/equipment costs are changed)

Note, a customer can have multiple locations. You can either assume they get billed per location or one bill combining the locations.

Part 1: Database Design



After reviewing the overview and the client’s needs (the internet and cable company), I recognized that I needed to design a database system capable of storing customer information while also linking each customer to their respective locations - specifically where their services and equipment are installed - and connecting that information to their billing records. The billing table needed to reflect which services or equipment each customer is paying for on a monthly basis.

Initially, I created connecting tables between the location, services, and equipment tables, but I later removed them, thinking they were redundant since customers were already linked to both their locations and billing information, which identified the services they subscribed to. However, in retrospect, I would have kept the connecting tables, as a customer can have multiple locations and may use different services or equipment at each one.

I also started with foreign keys in the billing table referencing both the equipment and service tables, but I realized that a customer might not always purchase equipment and services that are directly related. To address this, I chose to maintain separate tables for services and equipment and to record billing entries individually for each item - whether it be a piece of equipment or a specific service.

If I had more time to refine this project, I would further explore different database design strategies to determine which approach best fits the client’s operational needs and scalability requirements.

Part 2: Software Design

User Action Flow

Option 1: Add New Customer

1. User Journey:
2. User selects option "1" from the menu.
3. Program calls prompt_add_new_customer() in main.py.
4. User enters customer details (name, phone number, and address).
5. Function checks if the customer already exists using database.customer_name_exists().
6. If not found:
    a. Customer and location data are passed to database.add_new_customer().
    b. database.py executes INSERT statements into the Customer and Location tables.

Option 2: Remove Customer

User Journey:

1. User selects option "2".
2. System calls prompt_remove_customer(), prompting for the customer's name.
3. If the customer exists:
4. Calls database.delete_customer() which performs a DELETE from the Customer table.
5. Related Location and Billing records are deleted automatically due to foreign key cascading.

Option 3: Customer Payment Lookup

User Journey:

1. User selects option "3".
2. System calls view_customer_payments().
3. User chooses a search method (by name, address, city, state, or zip code).
4. Based on the choice, database.view_customer_payments() is called with the appropriate argument.
5. The function executes the SELECT_CUSTOMER_LAST_PAYMENT SQL query, joining Customer, Location, and Billing.

6. Returns the customer name and their last payment date.
7. Displays the results.

Option 4: View Late Payments

User Journey:

1. User selects option “4”.
2. System calls print_late_payments().
3. database.view_late_payments() executes an SQL query filtering customers where IsLate = 1.
4. Displays list of customers with overdue payments.

Option 5: Update Last Payment

User Journey:

1. User selects option “5”.
2. System calls prompt_update_last_payment().
3. User enters customer name and new payment date (YYYY-MM-DD).
4. Input is validated and converted into a timestamp.
5. Calls database.update_last_payment_date() which runs an UPDATE query on Billing.LastPaymentDate.

Option 6: Add Service or Equipment

User Journey:

1. User selects option “6”.
2. System calls prompt_add_service_equipment().
3. Displays all available services and equipment from the database.
4. User enters the customer name, item name, item type (service/equipment), cost, and credit card number.
5. Validation checks:
    a. Customer must exist.
    b. Item must exist in the correct table.
6. Calls database.add_service_equipment() to create a billing record.
7. Billing record includes item type, item name, cost, and payment info.

Option 7: Remove Service or Equipment

User Journey:

1. User selects option “7”.
2. System calls prompt_remove_service_equipment().
3. User enters customer name and item name.
4. Verifies both exist in billing via database.billing_exists().
5. Calls database.delete_service_equipment() which performs an SQL DELETE.

Option 8: Change Cost of Service or Equipment

User Journey:

1. User selects option “8”.
2. Calls prompt_update_cost_service_equipment().
3. Displays list of services and equipment.
4. User chooses item type and enters new cost.
5. Depending on the type:
   a. Updates the corresponding table using database.update_service_cost() or database.update_equipment_cost().
   b. Calls database.update_billing_cost() to reflect cost changes in billing.

Option 9: Exit

Ends the program loop and terminates the application.