Homework 4 Reports

DBS30 Team:
Binh Nguyen: Front end code, backend code, code general design, testing, report, debugging
Hoang Minh Bui: Database design, ER diagram, backend/backoffice code, video record, testing
Kevin Chau: testing
Max Pequeno: testing

Our project involves designing and developing a web application for managing a restaurant chain. The application includes features like employee management, scheduling, customer orders, bill generation, and transaction tracking. This report provides an overview of the database design, key functionalities, and a demo link for the app.

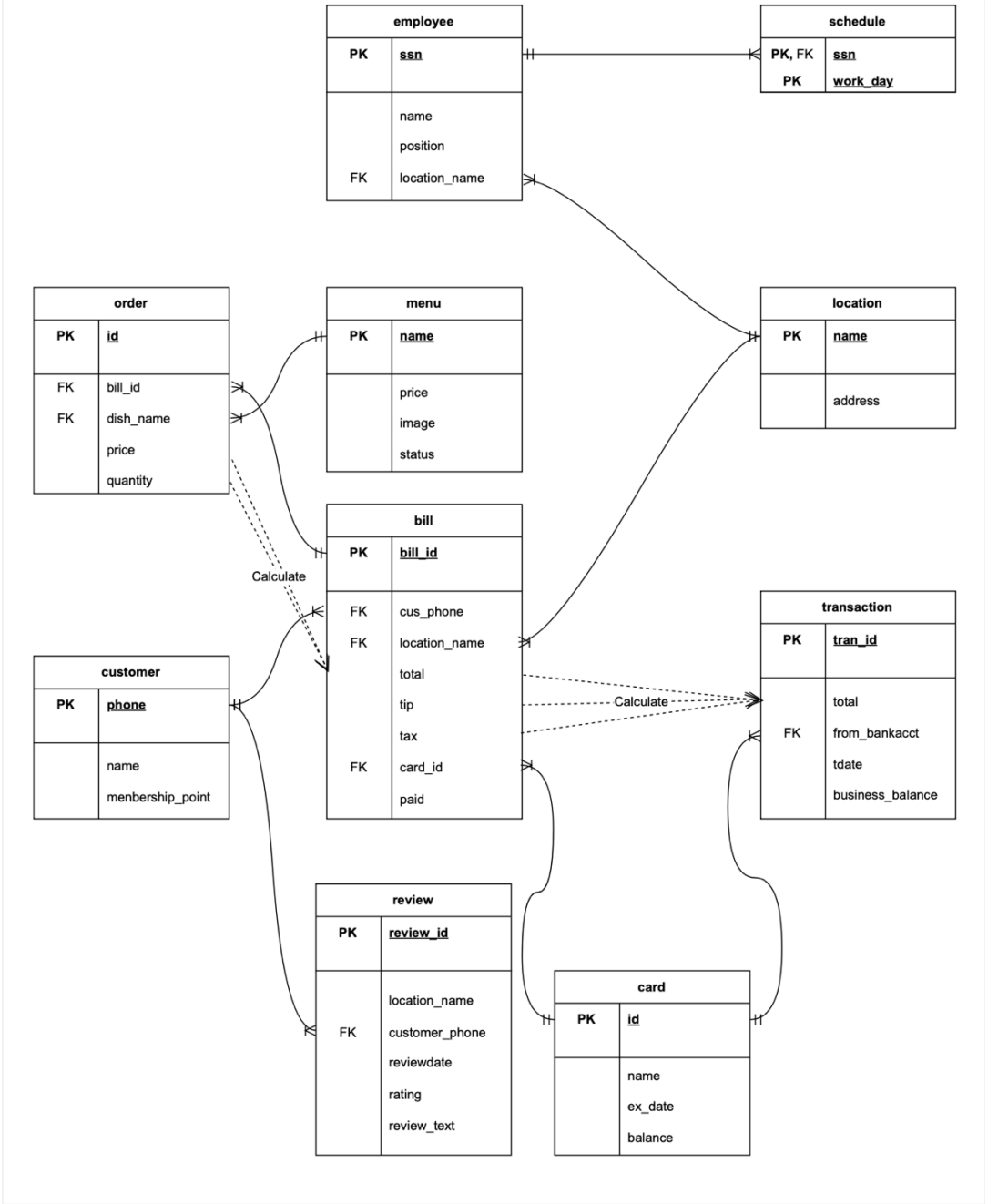
Video Link:
https://drive.google.com/file/d/1SBhDVYAAgTcX8QYfpjguGIEfurlmAKPZ/view?usp=sharing

ER diagram:
The ER diagram represents the core database schema for the web application. It consists of the following entities:

- **Employee**: Tracks employee details, including their name, position, and assigned location.
- **Schedule**: Maintains the working schedule for employees, with primary keys as the employee's SSN and work day.
- **Location**: Stores information about restaurant branches, such as name and address.
- **Menu**: Contains details of dishes offered, including price, image, and availability status.
- **Order**: Represents customer orders, linking to the menu items and associated bill.
- **Bill**: Summarizes order details, including customer info, total amount, tips, taxes, and payment methods.
- **Card**: Manages card payment details like balance and expiry date.
- **Transaction**: Records payment transactions, including total amount, date, and business balance.
- **Customer**: Tracks customer details and membership points for loyalty programs.
- **Review**: Allows customers to leave ratings and feedback for a specific location.

7. References
PostgreSQL Documentation: https://www.postgresql.org/docs/
W3Schools SQL Tutorials: https://www.w3schools.com/sql/
Stack Overflow: https://stackoverflow.com/
Bootstrap Framework: https://getbootstrap.com/

FINAL ER DIAGRAM



- The backend url is http://localhost:3000/ which can be opened by running “node server.js” in the terminal. The Back end is responsible for providing valuable information for the back office employee such as business overview, total transactions, daily revenue and bills, menu with items availability. The backend also provides GUI for the database that we use to run this restaurant.
- The Front end url is http://localhost:8000/ which can be opened by running “node app.js” in the terminal. The front end is a self-chekout service with an additional functionality such as simulating 100 customers ordering concurrently.

+ Self-checkout service: customers can add items to cart, edit quantities, and checkout like they normally would in the real world. Customers were given the options to pay cash or card

+ Simulation 100 customers is how we implement the concurrency part of this project. There will be 100 customers ordering at the same time. The website will process all customers concurrently and database will be able to process each request in a concurrent way

- Three important SQL

+ Query 1:

```
WITH bill_data AS (
  SELECT
    total, tax, paid
  FROM bill
  WHERE bill_id = $1
  FOR UPDATE
),
updated_bill AS (
  UPDATE bill
  SET
    cust_phone = $2,
    tip = $3,
    card_id = $4,
    paid = TRUE,
    location_name = $5
  WHERE bill_id = $1
  RETURNING (total + $3 + tax) AS total_amount
),
increment_points AS (
  UPDATE customers
  SET membership_point = membership_point + 1
  WHERE phone = $2
),
current_balance AS (
  SELECT
    COALESCE(
      (SELECT business_balance FROM transaction ORDER BY tran_id DESC LIMIT 1),
      5000.0
    ) AS current_balance
),
new_transaction AS (
  INSERT INTO transaction (total, from_bankacct, business_balance)
  SELECT
    ub.total_amount,
    $4,
    cb.current_balance + ub.total_amount
  FROM updated_bill ub, current_balance cb
```

```
  RETURNING business_balance
)
SELECT business_balance FROM new_transaction;
```

This SQL query:

1. Fetches the bill details for a specific bill ID.
2. Updates the bill to include customer info, tip, payment status, and location.
3. Rewards the customer with a membership point.
4. Checks the current business balance.
5. Inserts a new transaction, updating the business balance with the payment.
6. Returns the final business balance after the payment is completed.

It ensures all steps happen in the right order and are consistent within a single transaction.

- Query 2:

```
- WITH valid_dishes AS (
-   SELECT name, price, status
-   FROM menu
-   WHERE name IN (${orders.map(({ name }) => `'${name}'`).join(",
  ")})
- ),
- checked_dishes AS (
-   SELECT name, price
-   FROM valid_dishes
-   WHERE status = 'Available'
- )
- INSERT INTO orders (bill_id, name, price, quantity)
- SELECT $1, checked_dishes.name, checked_dishes.price,
  orders_data.quantity
- FROM checked_dishes
- JOIN (VALUES ${orderValues}) AS orders_data(name, quantity)
- ON checked_dishes.name = orders_data.name;
```

This SQL queries:

The query first checks if the dishes the customer wants are listed in the menu.

It ensures only available dishes are included in the order.

It inserts the valid orders into the orders table, including the bill_id, dish name, price, and quantity.

Dishes that are not available or invalid are ignored and not added to the order.

- Query 3:

```
-            "INSERT INTO cards (id, name, ex_date, balance) VALUES
   ($1, $2, $3, $4)";
```

- This query insert new customer cards into database

References
PostgreSQL Documentation: https://www.postgresql.org/docs/
W3Schools SQL Tutorials: https://www.w3schools.com/sql/
Stack Overflow: https://stackoverflow.com/
Bootstrap Framework: https://getbootstrap.com/