Project 3- Design & Implement a Relational Database

Database & Design

Raneem Belbisi

Business Requirement:

Introduction:

This project focuses on developing a bike-sharing system for residents and tourists in the Bay Area. The system aims to provide convenient access to bicycles for short trips, promoting sustainable transportation and healthy lifestyles. Furthermore, it will generate valuable data that can assist in urban planning and traffic management.

Business Rules:

Unique Identification:

- Each bike is uniquely identified and linked to a specific station.

Station Specifications:

- Equipped with unique identifiers.
- Located at specific geographic coordinates.
- Accommodate a predetermined maximum number of bikes.

Rental Mechanics:

- Bikes can be rented based on their availability.
- Each rental records start and end times.

Operational Oversight:

- Administrators can access and generate usage analytics and reports.

Core Entities:

Users: Individuals registered to use the bike-sharing service.
Stations: Points where bikes are stored and rented.
Administrators: Users responsible for system oversight.
Trips: Recorded journeys undertaken by users.
Status: Current availability and condition of bikes and stations.
System: The underlying infrastructure supporting operational functionalities.

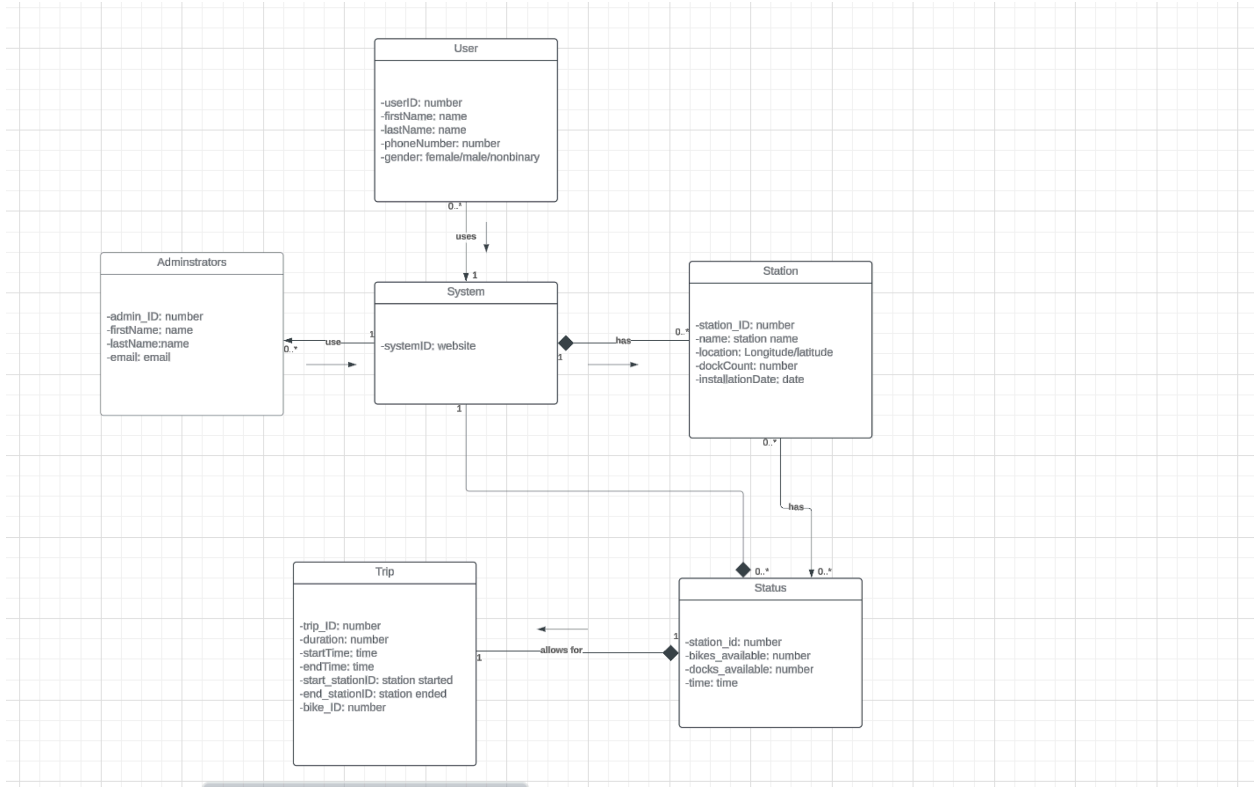
Key Activities:
Register: Users sign up for the service.
Record: Log trips and status updates of bikes and stations.
Reserve: Users book bikes for specific durations.
View: Users and administrators access various data points and reports.

How Redis Enhances the System

1) Real-Time Data Handling: Redis excels in managing real-time data. For a bike-sharing system, it can instantly update and retrieve the status of bikes and stations, which is crucial for users to know bike availability and for the system to adjust to demand dynamically.
2) Session Management: By using Redis to handle user sessions, the system can efficiently manage user logins and activity states, improving security and user experience.
3) Fast Data Retrieval: Redis's speed in data retrieval can be leveraged for quickly accessing trip histories and station statuses, making the system responsive and efficient.
4) Analytics and Reporting: Redis can quickly process large volumes of data, facilitating real-time analytics and timely reporting capabilities for administrators to monitor system usage and make informed decisions.

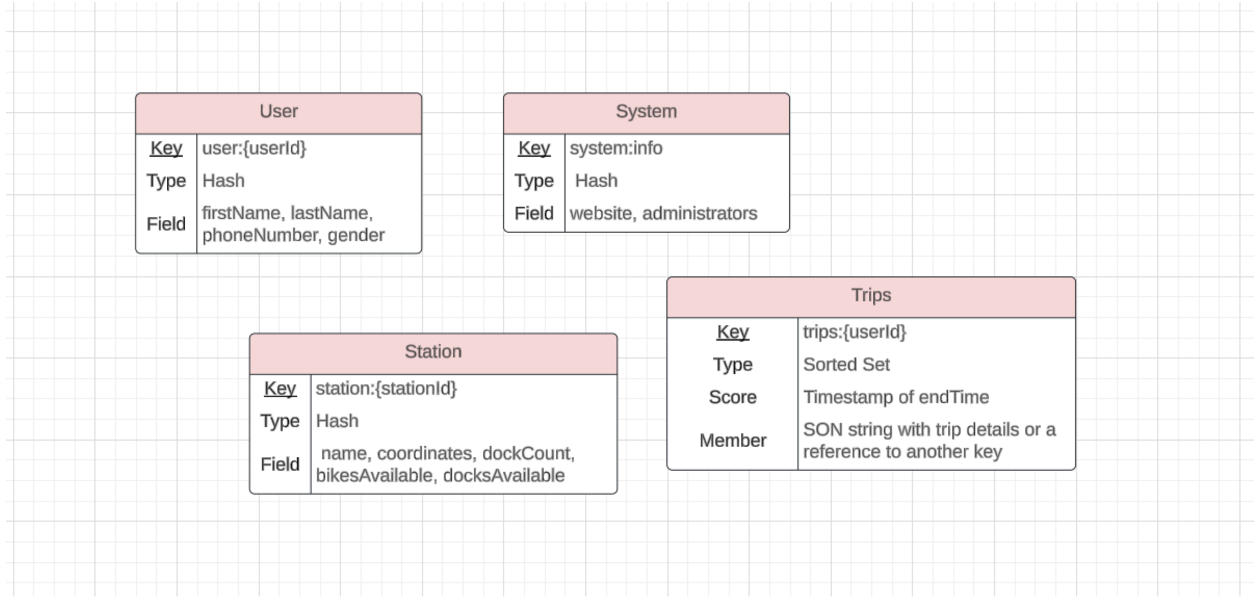
UML Class Diagram:

Link:https://lucid.app/lucidchart/invitations/accept/inv_8bb94a9e-7709-4259-889e-ade6e55ec3fb



Possible Redis Logical Model (extra, spoke with professor about it)

Link:https://lucid.app/lucidchart/invitations/accept/inv_8bb94a9e-7709-4259-889e-ade6e55ec3fb



User

Key	user:{userId}
Type	Hash
Field	firstName, lastName, phoneNumber, gender

System

Key	system:info
Type	Hash
Field	website, administrators

Station

Key	station:{stationId}
Type	Hash
Field	name, coordinates, dockCount, bikesAvailable, docksAvailable

Trips

Key	trips:{userId}
Type	Sorted Set
Score	Timestamp of endTime
Member	SON string with trip details or a reference to another key

Task 2: Redis Data Structures:
Real-Time Bike Availability:

Data Structure: Hash
Usage: Stores available bikes at each station, with the station ID as the key.
Key Example: stationAvailability
Hash Key-Value: {stationID: availableBikes}
Session Management

Data Structure: Strings
Usage: Manages session tokens with expiry for logged-in users.
Key Example: session:{userID}
Value: sessionToken
Ride History Cache

Data Structure: Sorted Set
Usage: Caches the most recent rides for a user, sorted by end time for quick access.
Key Example: recentRides:{userID}
Values: {rideID}
Score: EndTime (timestamp)

Task 3: Commands for CRUD Operations

- Create/update
    - Initialize/Reset Station Data:
        - HSET stationAvailability:station001 availableBikes 15
    - Adjust Bike Count (rent/return)
        - HINCRBY stationAvailability:station001 availableBikes -1 # Bike rented
        - HINCRBY stationAvailability:station001 availableBikes 1   # Bike returned
    - Read
        - HGET stationAvailability:station001 availableBikes
    - Delete
        - DEL stationAvailability:station001

Session Management

- Create (Set new session with expiration):
    - SETEX session:userID 3600 "sessionToken" # 3600 seconds = 1 hour
- Read (Get session token):
    - GET session:userID
- Update (Refresh or change session token):
    - SETEX session:userID 3600 "newSessionToken"
- Delete (End session):
    - DEL session:userID

Ride History Cache

- Create (Add a new ride entry)
    - ZADD recentRides:userID 1617983105 rideID123 # Score is the timestamp of ride end time
- Read (Retrieve top 10 most recent rides):
    - ZREVRANGE recentRides:userID 0 9 WITHSCORES
- Update (Update a ride's details or end time):
    - ZADD recentRides:userID 1617983150 rideID123 # Adjust score if necessary
- Delete:
    - Remove specific ride:
        - ZREM recentRides:userID rideID123
    - Clear all rides for a user:
        - DEL recentRides:userID

README:

- For the data collected, I used 4 tables from SF Bay Area Bikeshare from Kaggle and created two tables 'users' and 'admins' from Mackaroo. I joined them together on the db browser, created json files for each table then uploaded them into MongoDB Compass to complete my project.
- This Bike-Sharing System is designed to provide residents and tourists in the Bay Area with an accessible, affordable, and sustainable mode of transportation. It leverages real-time data to ensure availability and efficient management of bicycles across various stations.

System Requirements and Installations:

- Node.js (v14.0 or later)
- Redis server (v6.0 or later)

Data Import Script

This project includes a script that automates the process of importing data from JSON files into Redis. It is called script.js in the folder called project3. The script is designed to read JSON files containing data on users, stations, trips, and other entities relevant to the bike-sharing system and then populate the Redis database with this data using appropriate data structures. The script uses Node.js and the redis package to interact with the Redis server.

Steps for redis queries:

1) Install and connect to redis
2) Run the import script, navigate to the directory containing the script and execute it using Node.js in your terminal.
3) Connect to redis-cli by writing redis-cli into your terminal.
4) Execute the commands from task 3 above.

Basic Node + Express application:

This Node.js application uses Express to manage a Redis database for a bike-sharing system. It allows you to check and update the number of available bikes at various stations.

- Ensure you have Node.js and Redis installed on your machine. After cloning the project from the folder called bike-sharing app, install the required Node.js packages.
- Make sure your Redis server is running on its default port.
- Launch the application by running: node app.js