DATA 514 Section 8 Worksheet - Jonathan Jacobs

Question 1: Key-Value Data Store

Implementation Using a Key-Value Data Store

A key-value data store is a type of NoSQL database where each item is stored as a key-value pair. For the Payroll/Regist application, we need to design keys and values to support the required methods efficiently. Below is a detailed implementation:

Key Structures

1. **User Data (Payroll and Regist)**
   - **Key**: `user:{UserID}`
   - **Value**: A JSON object containing user information and their registered cars.

```
{
    "name": "Leslie",
    "job": "TA",
    "parkingPermit": "C15",
    "cars": [
        {"car": "Charger", "licensePlate": "123 AAA"}
    ]
}
```

2. **Car Data (ParkingTickets)**
   - **Key**: `car:{LicensePlate}`
   - **Value**: A JSON array of parking tickets for the car.

```
[
    {"parkingLot": "C15", "date": "2022-11-20", "amount": "$10"},
    {"parkingLot": "E01", "date": "2022-11-21", "amount": "$15"}
]
```

3. **Parking Lot Permissions**
   - **Key**: `permit:{ParkingPermit}:{LicensePlate}`
   - **Value**: `true`
   - **Explanation**: This structure allows quick checking of whether a specific license plate is permitted to park in a specific lot. Non-existent keys will imply `false`.

Detailed Breakdown of Keys and Values

1. **User Data Key-Value Pairs**

- **Key**: `user:{UserID}`
- **Value**: A JSON object with fields `name`, `job`, `parkingPermit`, and `cars`. The `cars` field is an array of car objects, each containing `car` and `licensePlate`. Example:

```
"user:123": {
    "name": "Leslie",
    "job": "TA",
    "parkingPermit": "C15",
    "cars": [
        {"car": "Charger", "licensePlate": "123 AAA"}
    ]
}
```

2. **Car Data Key-Value Pairs**

   - **Key**: `car:{LicensePlate}`
   - **Value**: A JSON array where each element is a parking ticket object with fields `parkingLot`, `date`, and `amount`.

   Example:

```
"car:MMM 1234": [
    {"parkingLot": "C15", "date": "2022-11-20", "amount": "$10"},
    {"parkingLot": "E01", "date": "2022-11-21", "amount": "$15"},
    {"parkingLot": "E01", "date": "2022-11-22", "amount": "$20"}
]
```

3. **Parking Lot Permissions Key-Value Pairs**

   - **Key**: `permit:{ParkingPermit}:{LicensePlate}`
   - **Value**: `true`

   Example:

```
"permit:C15:123 AAA": true
```

Method Implementations

1. **Listing the Permitted Parking Lot and Per-Car Tickets Incurred by Each User**

   - **Method Signature**: `uid -> [ {car1, [{tix1}, {tix2}]}, {car2, []} ]`
   - **Implementation**:
     - Fetch user data using `user:{UserID}`.
     - For each car in the user's `cars` array, fetch ticket data using `car:{LicensePlate}`.

2. **Counting How Many Tickets a License Plate Has Ever Had**

- **Method Signature**: `plate -> int` or `plate -> [ {tix1}, {tix2} ]`
- **Implementation**:
  - Fetch car data using `car:{LicensePlate}`.
  - Return the length of the array for the count or return the array itself for detailed tickets.

3. **Determining Whether a Plate is Allowed to Be in a Specific Lot**

- **Method Signature**: `(plate, lot) -> true/false`
- **Implementation**:
  - Check the existence of the key `permit:{ParkingPermit}:{LicensePlate}`.
  - If the key exists, return `true`; otherwise, return `false`.

Summary

This key-value data store implementation efficiently supports the application's methods by structuring keys and values to allow quick lookups and minimal data duplication. Each key represents a specific piece of data, and values are structured to provide all necessary details, enabling the application to perform the required operations with minimal overhead.

Question 2: Document Store

Implementation Using a Document Store Database

A document store database is a type of NoSQL database designed to store, retrieve, and manage document-oriented information. Documents are typically represented in JSON or BSON format. For the Payroll/Regist application, we will design the documents to efficiently support the required methods.

Document Structures

1. **User Document**

- Contains user information, including registered cars and parking permit details.
- Each user document includes an array of car documents.

```
{
    "userID": 123,
    "name": "Leslie",
    "job": "TA",
    "parkingPermit": "C15",
    "cars": [
        {
            "car": "Charger",
            "licensePlate": "123 AAA",
            "tickets": [
                {"parkingLot": "C15", "date": "2022-11-20", "amount":
"$10"}
            ]
        }
    ]
}
```

2. **Car Document (Separate Collection)**
    - Contains details about a car, including its license plate and any associated parking tickets.
    - This document is primarily used for querying ticket counts and validating parking lot permissions.

```
{
    "licensePlate": "MMM 1234",
    "ownerUserID": 567,
    "tickets": [
        {"parkingLot": "C15", "date": "2022-11-20", "amount": "$10"},
        {"parkingLot": "E01", "date": "2022-11-21", "amount": "$15"},
        {"parkingLot": "E01", "date": "2022-11-22", "amount": "$20"}
    ]
}
```

Collections

1. **Users Collection**
    - Each document in this collection represents a user and includes their details along with their cars and any tickets associated with those cars. Example Document:

```
{
    "_id": 123,
    "name": "Leslie",
    "job": "TA",
    "parkingPermit": "C15",
    "cars": [
        {
            "car": "Charger",
            "licensePlate": "123 AAA",
            "tickets": [
                {"parkingLot": "C15", "date": "2022-11-20", "amount":
"$10"}
            ]
        }
    ]
}
```

2. **Cars Collection**
    - Each document in this collection represents a car and includes its license plate, owner (userID), and any tickets associated with the car. Example Document:

```
{
    "_id": "MMM 1234",
    "ownerUserID": 567,
    "tickets": [
        {"parkingLot": "C15", "date": "2022-11-20", "amount": "$10"},
        {"parkingLot": "E01", "date": "2022-11-21", "amount": "$15"},
        {"parkingLot": "E01", "date": "2022-11-22", "amount": "$20"}
    ]
}
```

Method Implementations

1. **Listing the Permitted Parking Lot and Per-Car Tickets Incurred by Each User**
    - **Method Signature**: `uid -> [ {car1, [{tix1}, {tix2}]}, {car2, []} ]`
    - **Implementation**:
        - Query the `Users` collection for the document with `userID`.
        - Extract the `cars` array from the user document.
        - Each car in the `cars` array includes a `tickets` array with the incurred tickets.
2. **Counting How Many Tickets a License Plate Has Ever Had**
    - **Method Signature**: `plate -> int` or `plate -> [ {tix1}, {tix2} ]`
    - **Implementation**:
        - Query the `Cars` collection for the document with `_id` as `LicensePlate`.
        - Return the length of the `tickets` array for the count or return the `tickets` array itself for detailed tickets.
3. **Determining Whether a Plate is Allowed to Be in a Specific Lot**
    - **Method Signature**: `(plate, lot) -> true/false`
    - **Implementation**:
        - Query the `Cars` collection for the document with `_id` as `LicensePlate`.
        - Retrieve the `ownerUserID` from the car document.
        - Query the `Users` collection for the document with `userID` as `ownerUserID`.
        - Check if the `parkingPermit` matches the given lot.

Summary

Using a document store database like MongoDB, the implementation of the Payroll/Regist application involves creating two main collections: `Users` and `Cars`. Each user document contains personal details, parking permit information, and an array of car documents, which include ticket details. Each car document in the `Cars` collection contains the license plate, owner information, and tickets. This structure ensures efficient querying for all use cases, leveraging the strengths of document-oriented databases.