## Redis Data Structures Implementation

1. **Storing Appointment Details as Serialized JSON Objects**
    - **Data Structure**: String
    - **My Approach**: In my application, I store the complete details of each appointment as a serialized JSON object in Redis. Whenever I retrieve an appointment from MongoDB, I serialize it into a JSON string and store it in Redis with a unique key. This is handled in the `setData` function of my Redis utilities.
    - **Key-Value Example**:
        - Key: `appointment:<appointmentId>`
        - Value: A serialized JSON string representing all the appointment details.

    I find this method efficient for storing and retrieving entire objects, which is very useful for accessing the full details of an appointment quickly.

2. **Deleting Appointment Details from Cache**
    - **My Usage**: Whenever I delete an appointment in the database, I also remove the corresponding entry from the Redis cache. This ensures that my cache remains consistent with the database.
    - **Operation**:
        - I use `DEL appointment:<appointmentId>` to remove the cached data.

    This step is important to prevent serving outdated information from the cache.

3. **Updating Appointment Details in Cache**
    - **My Usage**: Although my current implementation doesn't directly update the cached appointment data when an appointment is modified, it's something I plan to handle. I should update the cache with new data or invalidate the existing cache entry whenever an appointment is updated in the database.
    - **Suggested Operation**:
        - I should either update the Redis entry with the new details of the appointment or delete the existing cache entry. This way, it can be refreshed from the database the next time it's accessed.

## Summary

In my implementation, I use Redis effectively as an in-memory cache to store serialized JSON objects that represent the details of appointments. This approach allows quick access to appointment details and efficiently manages the cached data, enhancing the performance of my application by reducing the frequency of database queries. My method, while straightforward and not utilizing Redis hashes, is aptly suited to meet the requirements of my healthcare management system.