Tasks

Contents

Task 1

Provide the problem requirements and the conceptual model in UML for your project. You can reuse the one made on previous projects, but describe the functionalities that you selected to be used as an in-memory key-value storage, (e.g. most viewed products, a shopping cart, current logged-in users, etc).

Problem Requirement

Background

As a database system management application, my Airbnb Host Management System may experience some special cases. Our database will be asked to serve frequently requested items at sub-millisecond response times, enables us to easily scale for higher loads without growing the costlier backend. For example, my Airbnb Host Management System can show the most popular listings for hosts to update and shows the latest added host for Airbnb staff to check and approve their application. In this circummundane, it is better to create a real-time ranked list or sorted set is as easy as updating a user's score each time it changes.

Redis is a great choice for implementing such a highly available in-memory cache to decrease data access latency and increase throughput. Therefore, in my project 3, I decided to use Redis as my backend database to development my Airbnb Host Management System.

User

Airbnb Hosts

Airbnb Host can use this database and management application to manage their profile and check their ratings. They can also update the information of their listings.

Airbnb Staff

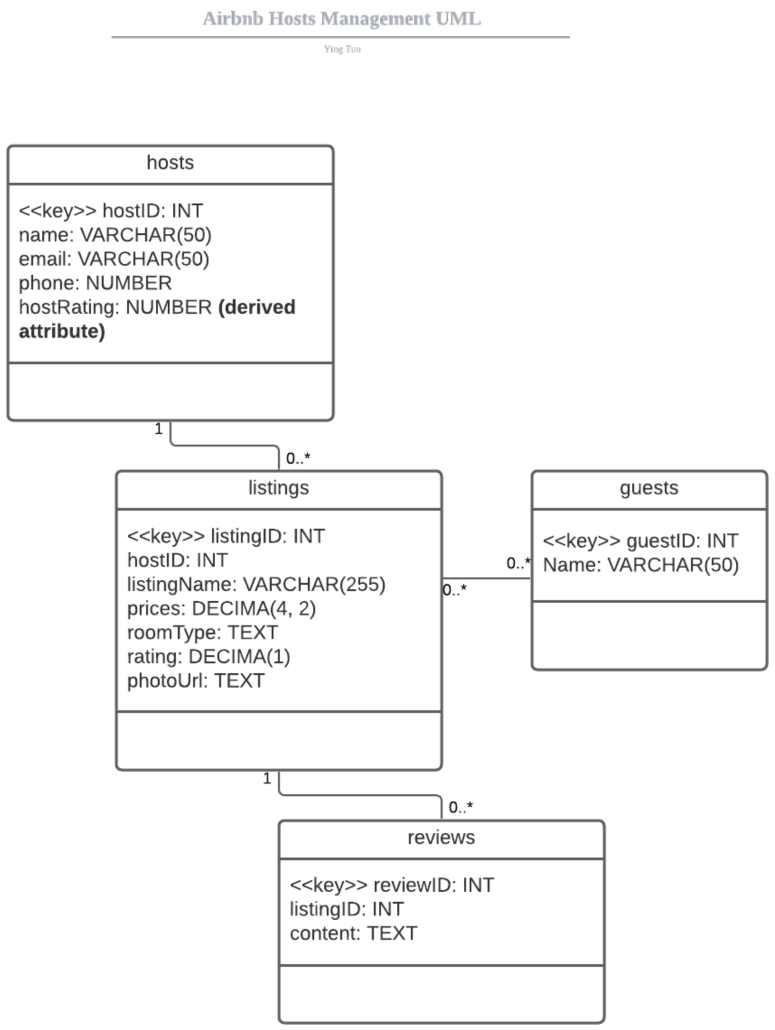
Airbnb staff can use this database and application check the status, profile and rating of Hosts. They can also update the rating of Hosts. Airbnb staff will mainly focus on the latest join hosts, and they will check the new hosts' information and then decide to approve or deny their applications.

Rules

- This database aim to be an internal tool to manage Airbnb Host.
- As for the Data schema, I used the data from Airbnb's website: http://insideairbnb.com/get-the-data.html.
- This internal host management tool is used for Airbnb hosts and Airbnb staff to manage the hosts' profile, listing information and listing reviews.
- New airbnb hosts can create their account in this application. In the right side of the website, there is a form to submit their name and email to create a new account. Bresides, they will be given a unique id, which is hidden in this page but will be recorded in the Redis database.
- A new Airbnb host has the default rating as 5 starts.
- Airbnb hosts can update their information in this application, for example, update their names, email addresses and phone numbers.
- In order to implement the most popular hosts routes, we need the get the count of hosts'/listings' views; and thus, to sorted the hosts/listings based on their views.
- For the Airbnb staff, we need to build another collection to record the timestamp that a new host is created; and therefore, they can check hosts' information and approve their application based on a sorted collection according to the creation time.
- Airbnb staff do not have the access to update the rating of hosts based on their listings rating. Because the hostRating which is a derived attribute, is calculated by their listings' s ratings.
- The hostRating attribute does not physical exist in our database, and it is calculated base the rating attribute in listings collection.
- The guests collection stores the guests' data of Airbnb.
- Once guests have stayed in one listing, their record will be paired.

Tasks

The above rules require us to create four collections and four routes. I used LucidChart to create UML for these collections,
- The first collection is ***hosts*** collection.
- The second collection is ***listings*** collection.
- The third collection is ***reviews*** collection.
- The fourth collection is ***guests*** collection.



https://lucid.app/lucidchart/invitations/accept/1f2a8f5d-4a25-44cf-9aed-88ba31dcc08d

- As the description in the above question, the relationship between hosts to listings is **zero to many**, while the relationship between listings to hosts is **one to many**; The relationship between listings to reviews is **one to many**, while the relationship between reviews to listings is **one to one**; the relationship between listings to guests is **many to many**

- Our database should build appropriate schema for this database, so that each collection can be associate with others and build relationship.
- In the listings collection, hostID is the foreign key for this listing.
- In order to develop this application, helping Airbnb and Airbnb Host to manage their data, we should have the following APIs to operate our database.
- `/hosts` The home page will redirect to ("/hosts"), which is the main routes. In this page, it shows the Airbnb host database with their name, email, the date they joined Airbnb and most importantly, their hostRating.
- `/hosts/create` The CREATE interface let users to create a new Airbnb user, and post data to database.
- `/hosts/delete` This route let us to delete the records in the Hosts collection. Besides, because tht hostid in the Hosts collection is foreign key in Listings collection, and listingid in Listing collection is the foreign key in Reviews Table, all of the related records will be deleted.
- `/hosts/update` This route let us to update the information and records in the Hosts collection.

Task 2

Describe the Redis data structures that you are going to use to implement the functionalities you described in the previous point. (e.g. To implement the most viewed products I will use a Redis sorted set with key "mostViewed:userId", product ids as the values and a score of the number of views of the product.). You can use/describe more than one data structure, you will need to implement at least one.

There are five basic data structure in Redis,

String	The simplest type of value you can associate with a Redis key.
List	Collections of string elements sorted according to the order of insertion. They are basically linked lists.
Set	Collections of unique, unsorted string elements.
Sorted Set	Similar to Sets but where every string element is associated to a floating number value, called *score*.
Hashes	Maps composed of fields associated with values. Both the field and the value are strings.

In my Database System Design, I used three data structures of them, **List**, **Set** and **Sorted Set,** and implement five collections.

No.	**Data Structure**	**Objectives**	**Summary**
1	**List**	To store reviews	• To implement the latest reviews, I will use a Redis **list** to record the reviews. • When every new review is submitted, it will be added to the last of the review List.
2	**Set**	To store the current logged-in user's(host's) listings	• To implement the current logged-in user's(host's) listings, I will use a Redis **set** to store the listings that the host have or will have. • The operation time complexity of a set, like find(), delete() and insert() is **O(1)**, and thus to use set as a Memcached is more fast than other data structures. • We do not need to care about the order of the current user, and we just need to demonstrate them on the home page of the user. Therefore, as an unsorted data structure, set can be used to implement this functionality. • There are no duplicated items in the listings of a host. Set is a great data structure to granulate this feature.
3	**Sorted Set**	Sort the hosts information based on added timestamp	• I will use sorted set to store hosts information, because the expansibility of sorted set is better than other data structure. For example, we can sort the host information based on their created timestamp, count of views and ratings.

			• To implement the latest-add airbnbHost, I will use a Redis **sorted set** with key "newAdded: createdTime"; host ids as the values that hosts were added.
			`pzadd("hosts", +new Date(), host.id);`
			• In order to get the latest added airbnbHost, we need to define the order of this sorted set as descending order, and therefore the above commond should be written as,
			`pzadd("hosts", –new Date(), host.id);`
			• I do not plan to use list to get the latest added aribnbHost, because there maybe many Redis clusters, and their "clock" maybe different. Therefore, I will use the **timestamp** as the key to get the sorted latest added hosts.
			• After the set is sorted based on the timestamp that hosts are been created, Airbnb staff can easily check the latest join hosts' application and check their information.

4	**Sorted Set**	Sort the hosts information based on their views	• To implement the "most popular hosts" page showing the hosts with high views. I will use the **sorted set**. • In this database design, if the host's homepage has been viewed, the count of views will be added by one. Therefore, we can easily get the top 10 popular hosts by using such a sorted set.
5	**Sorted Set**	To store the guest information	• To store the guests information and their corresponding Airbnb listings, I will use a Redis **sorted set** with key "listings: listingId", and guestId ids as the values to denote the gusest have stay in this listing.

Task3

Create a basic Node + Express application that let's you create, display, modify and delete at least one Redis data structure from the ones describe in the previous point. No need to have a polished interface, and you can use the code created in class as a starting point, and/or the code you created for previous projects

My repo for this project is: https://github.com/tuoying96/airbnbHostRedis