URL: <https://github.com/YeZhang125/cs6650-assignment3.git>

**Deployment**:

Configuration: Upload WAR file to EC2 instance, and modify the URL path in the MultiThreadedLiftRideClient class: private static final String SERVER\_URL.

Lauch new EC2 instance running Ubuntu and install Redis. Change Line 23 private static final String DBHost to EC2 instance IP address.

Change variable HOST in SkierConsumer and ServerAPI accordinging to EC2 instance ip address that host Rabbitmq.

To run the project, start the ServerAPI project, the Client project and Consumer project separately, as they are built independently using Maven. To run Consumer, upload jar file to EC2 instance and run java -jar Consumer-1.0-SNAPSHOT.jar.

**Database Design:**

1.Database choice: **Redis (for In-Memory Caching and Analytics)**

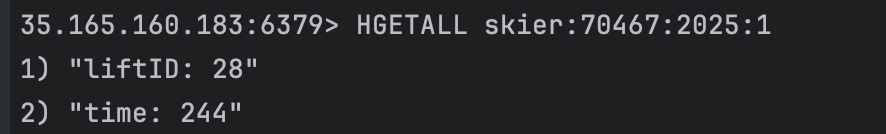
2.Deployment on AWS: **Redis** is deployed on **EC2 instances** running **Ubuntu** in the **AWS Free Tier** environment, specifically chosen to optimize cost-efficiency during development and testing. The EC2 instances are configured for high performance by **t2.micro** instance, which is eligible for the Free Tier.

3. Database Design Description:

The system utilizes **Redis** as the backend data store to manage skier information, lift rides, resort visits, and vertical totals. Here's a breakdown of the database schema and how it is structured:

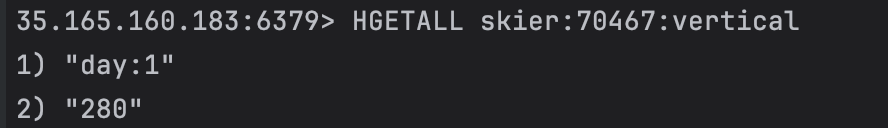
**Skier Data (Hash)**

* **Key:** skier:<skierID>:<seasonID>:<dayID>
* **Purpose:** Stores the lifts a skier rode on a specific day during a specific season.
* **Fields:**
  + liftID:<liftID>: Each lift ID is stored as a field with its corresponding value being the time (in seconds) the skier spent on the lift.
  + **Example:** skier:123:2025:10 could have fields like liftID:10 with a value of 150 (150 seconds spent on lift 10).



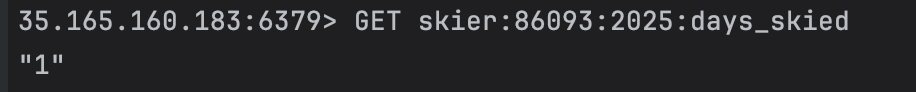
**Vertical Totals (Hash)**

* **Key:** skier:<skierID>:vertical
* **Purpose:** Tracks the vertical totals (in feet or meters) for each skier across multiple ski days. This is done by accumulating the lift rides (liftID multiplied by 10) for a given day.
* **Fields:**
  + day:<dayID>: For each day a skier skied, their vertical total for that day is incremented.
  + **Example:** skier:123:vertical could have fields like day:10 with a value of 500 (total vertical achieved on day 10).



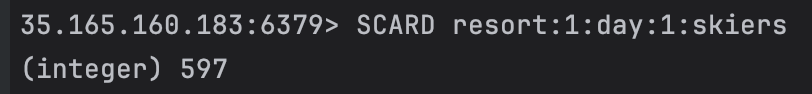
**Skier Days Skied (Integer)**

* **Key:** skier:<skierID>:<seasonID>:days\_skied
* **Purpose:** Counts how many days a skier has skied in a given season. Each time a skier skis on a new day, this counter is incremented.
* **Example:** skier:123:2025:days\_skied could have a value like 5 (indicating the skier has skied 5 days in the 2025 season).



**Resort Data (Set)**

* **Key:** resort:<resortID>:day:<dayID>:skiers
* **Purpose:** Tracks the unique skiers who visited a specific resort on a specific day. A Redis set is used here to ensure uniqueness.
* **Members:** The set stores skier IDs who visited the resort on the given day.
* **Example:** resort:1:day:10:skiers might contain member 123 (skier 123 visited resort 1 on day 10).



**Async Task with Redis and CompletableFuture**

* The system uses **CompletableFuture.runAsync** to perform non-blocking asynchronous tasks of storing messages in Redis.
* **Multithreading:** The executorService is used to run multiple consumers that process messages in parallel, improving scalability and performance.

### **Redis Operations:**

* **Incrementing Counter:** jedis.incrBy(key, 1) is used to increment the number of days a skier has skied.
* **Storing Lift Data:** jedis.hset(key, field, value) stores data in the Redis hash, where each lift ride is stored with the time spent.
* **Storing Vertical Totals:** jedis.hincrBy(key, field, increment) is used to accumulate vertical totals for each skier.
* **Adding to Sets:** jedis.sadd(key, member) adds a skier's ID to the set representing skiers who visited a resort on a given day.

Client screenshots and RMQ console clips:

