Amazon Gaming Memberships

Introduction

Amazon Gaming - an internal site exclusively available to employees - provides the ability to play games online with other users by forming gaming groups. Users can also create discussion groups to have private conversations about games.

The code provided in this activity is responsible for maintaining the memberships to these groups.

The three key entities in our service right now are:

- Groups: The gaming or discussion groups that a user can join
  - Partition key: id
- GroupMemberships: The user memberships in groups
  - Partition key: userId
  - Sort key: groupId
- GroupMembershipAudits: A record of a member being added or removed from a group
  - Partition key: userId_groupId - a string made up of both a userId and a groupId
  - Sort key: editedAt - the time the add or remove took place

Our service will support a number of requests regarding these entity types. The relevant cases at this point are:

- Groups
  - GetGroup
  - CreateGroup
- GroupMemberships
  - AddUserToGroup
  - CheckUserInGroup
  - GetGroupsForUser
  - GetUsersInGroup
  - RemoveUserFromGroup

The code base follows the Activity-DAO-DynamoDBMapper pattern that we've come to know and love. Each Activity class implements one service operation.

This service follows a data deletion pattern that you haven't seen before, followed by teams at Amazon with a requirement to actually delete data (often for privacy reasons). The `GroupMembershipDao` is responsible for interacting with both the `GroupMembership` model and the `GroupMembershipAudit` model. When the service receives a request to remove a user from the group, it actually deletes the corresponding `GroupMembership` item from the data store; to provide accountability, it *tracks* the deletion in a `GroupMembershipAudit` item.

Disclaimer: These Activity classes don't yet accept or return Response/Result objects. They accept and return primitives and POJOs. Teams sometimes do this for prototyping. It's fairly simple to define the the objects once the activity requirements have stabilized, but we won't be doing that today.

Phase 0: Preliminaries

1. Create the tables we'll be using for this activity by running these aws CLI commands:

```
aws cloudformation create-stack --region us-west-2 --stack-name
caching-groups-table --template-body
file://cloudformation/groups_table.yaml --capabilities CAPABILITY_IAM
aws cloudformation create-stack --region us-west-2 --stack-name
caching-groupmemberships-table --template-body
file://cloudformation/group_memberships_table.yaml --capabilities
CAPABILITY_IAM
aws cloudformation create-stack --region us-west-2 --stack-name
caching-groupmembershipaudits --template-body
file://cloudformation/group_membership_audits_table.yaml --
capabilities CAPABILITY_IAM
```

2. Log into your AWS account and verify that the tables exist and have sample data.
3. Discuss the different attributes with your team to make sure you all understand what they represent.
4. As a final verification, run `Phase0Test` make sure it passes.

GOAL: `Groups` and `GroupMemberships` tables are all created in your AWS Account, and the attributes make sense

Phase 0 is complete when:

- You understand the 3 data types and their relationships
- Groups, GroupMemberships, and GroupMembershipAudits tables exist with some sample data
- `Phase0Test` passes

Phase 1: Implement your Cache

It's time to implement the cache! There are three steps:

1. Implement your cache design in `GroupMembershipCachingDao`.
2. Write unit tests.
3. Use you caching DAO in `CheckUserInGroupActivity`.

Implement your cache design

Update the class `GroupMembershipCachingDao` to implement your group's design from the previous phase.

Write unit tests

Implement the two unit tests in `GroupMembershipCachingDaoTest`. We've added comments to the GIVEN, WHEN, AND THEN sections describing what to do. You'll need to implement them with your own models.

To populate the cache, make a request to your caching DAO. This should call the delegate DAO. To verify that a mock was called a specified number of times, use code like this:

```
verify(mockDao, times(1)).callToGetSourceData(param1, param2);
```

Use your caching DAO

Update the `CheckUserInGroupActivity` class to utilize your newly created cache.

- I'm not sure what method to call in the dao

Phase 1 is complete when:

- You've implemented your cache design in `GroupMembershipCachingDao`.
- You've written two unit tests in `GroupMembershipCachingDaoTest` covering the hit and miss case
- You've updated `CheckUserInGroupActivity` to use your cache
- `Phase1Test` passes