HU Extension Assignment 06 E-90 Cloud Computing

Handed out: 10/10/2015 Due by 11:59 PM EST on Friday, 10/16/2015

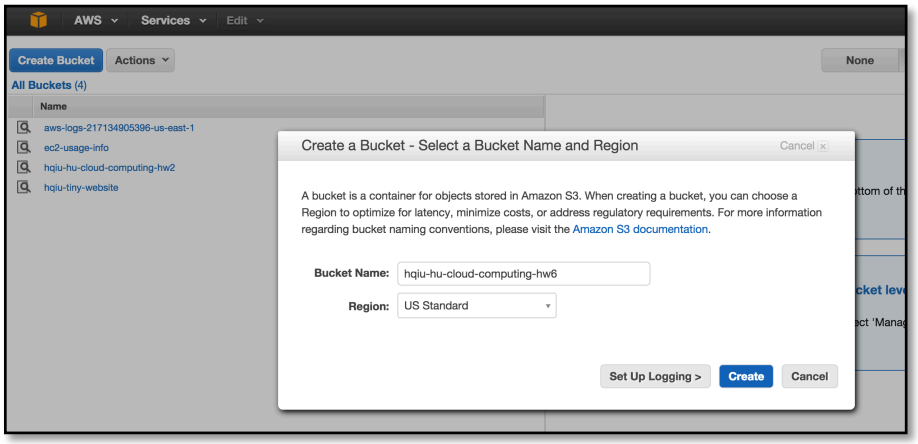
Place all of your narratives and illustrations in a single Word or PDF document named E90_LastNameFirstNameHW06.docx [.pdf]. Use this assignment as the initial template. Add your steps and your code below problem statements used for that problem. Upload your homework file and your working code (e.g., filename.java) into your Assignment 6 folder. Do not include executables.

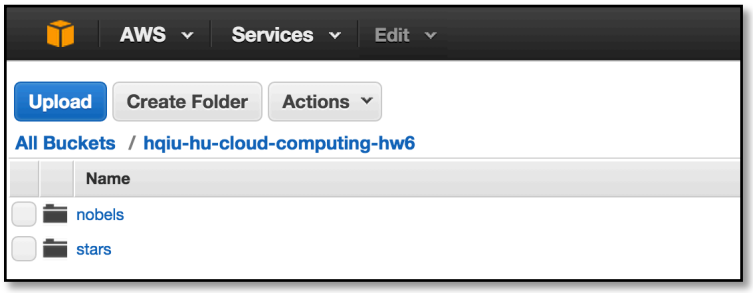
Problem 1.

I. Manually populate an S3 bucket with Images and Resumes of movie stars and Nobel laureates:

a. Create folder structures with two folders: **stars** and **nobels**. Inside either folder you should have subfolders **images** and **resumes**.

Create a S3 bucket from the AWS console. Set the Region to "US Standard". Create the hierarchical structure in the folder.





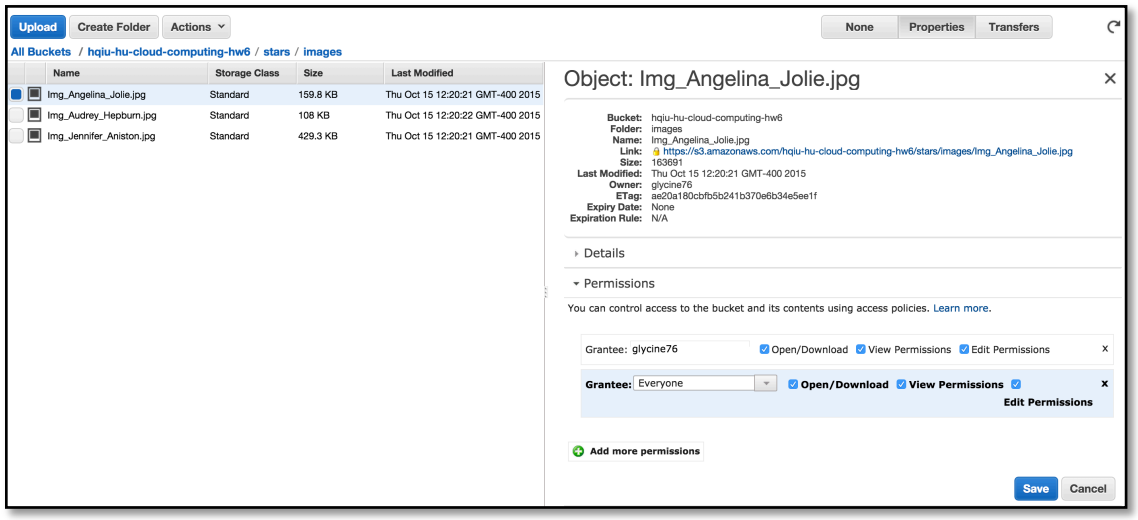
b. From Google Images fetch images (pictures) of three movie stars and three Nobel winners. Produce 3+3 MS Word documents with bogus resumes for selected movie stars and Nobel winners on your own. The resumes should be very short; three sentences each.

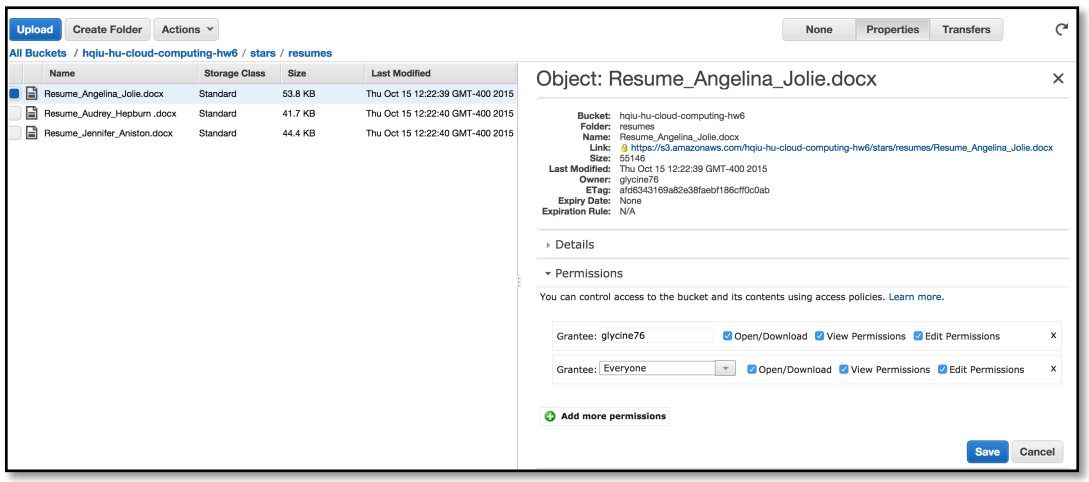
Create some pictures and fake resumes.

c. Upload those images and resume into your S3 bucket manually.

d. Manually grant general public access to those images and resumes.

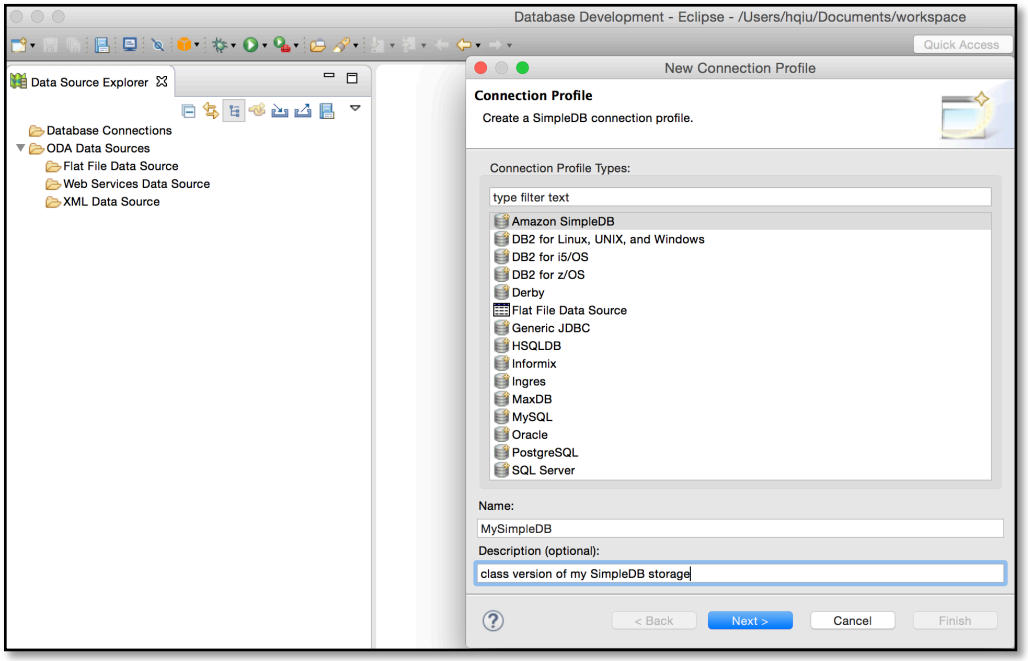
Upload corresponding images and resumes to “stars” and “nobels” folder. In “Properties”->”Permissions”, grant “open”, “view” and “edit” permission to “everyone”.

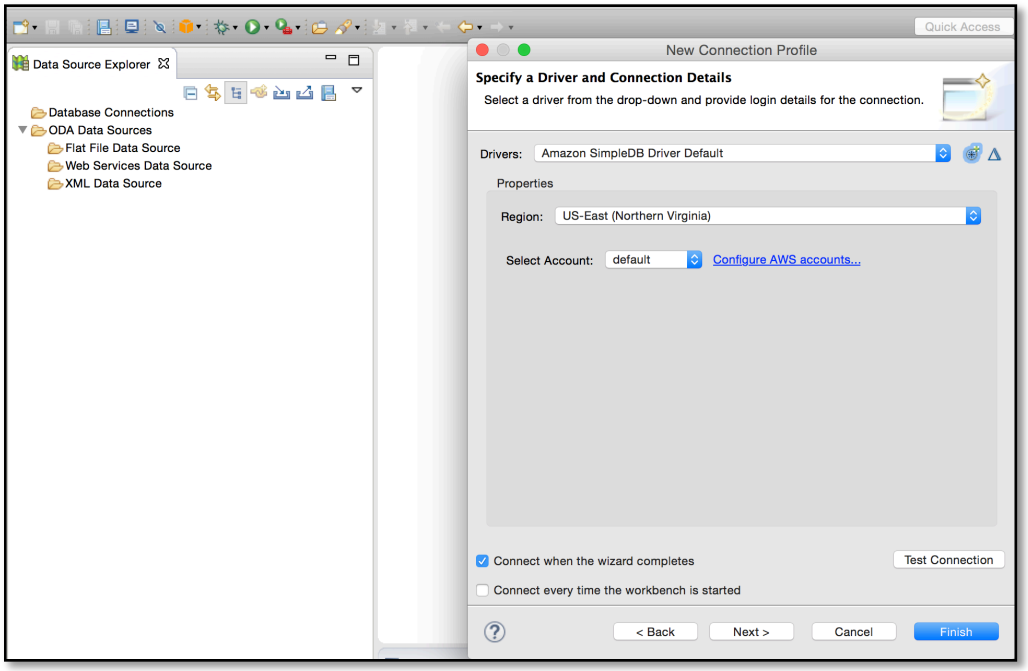




II. Create one SimpleDB domain and programmatically insert one row for each movie star and each Nobel laureates - write your program either in Java or .Net or your favorite language supported by AWS:

Create a new SimpleDB project in Eclipse.





Take the example code as an example. First do some initialization. Set up the credentials for connecting the SimpleDB. I use the “default” credentials here.

```
public class SimpleDB {

    public static void main(String[] args) throws Exception {

        AWSCredentials credentials = null;
        try {
            credentials = new ProfileCredentialsProvider("default").getCredentials();
        } catch (Exception e) {
            throw new AmazonClientException(
                    "Cannot load the credentials from the credential profiles file. " +
                    "Please make sure that your credentials file is at the correct " +
                    "location (/Users/hqiu/.aws/credentials), and is in valid format.",
                    e);
        }
        AmazonSimpleDB sdb = new AmazonSimpleDBClient(credentials);
        Region usEast1 = Region.getRegion(Regions.US_EAST_1);
        sdb.setRegion(usEast1);

        System.out.println("===========================================");
        System.out.println("Getting Started with Amazon SimpleDB");
        System.out.println("===========================================\n");
```

a. Use their full names with hyphens between the first and the last name as the item keys.
b. In rows for movie stars record full name, most popular movie, S3 URL of the picture of the star and S3 URL of his or her resume.
c. Use those same attributes for the Nobel laureates and add the year they won the prize and the field of science in which they got the prize as two additional attributes.

First create a domain in the Simple DB.

```
try {
    // Create a domain
    String myDomain = "Celebrities";
    System.out.println("Creating domain called " + myDomain + ".\n");
    sdb.createDomain(new CreateDomainRequest(myDomain));

    // List domains
    System.out.println("Listing all domains in your account:\n");
    for (String domainName : sdb.listDomains().getDomainNames()) {
        System.out.println("  " + domainName);
    }
    System.out.println();
```

Insert three rows for movie stars and another three rows for Nobel laureates using the API “BatchPutAttributesRequest()”. The primary key of the domain is the full names of the celebrities with hyphens between the first and last name. In function “createSampleData()”, the parameters in the “ReplaceableItem ()” is the primary key. The attributes are set by “ReplaceableAttribute()”.

```
// Put data into a domain
System.out.println("Putting data into " + myDomain + " domain.\n");
sdb.batchPutAttributes(new BatchPutAttributesRequest(myDomain, createSampleData()));
```

```
/* Creates an array of SimpleDB ReplaceableItems populated with sample data. */
private static List<ReplaceableItem> createSampleData() {
    List<ReplaceableItem> sampleData = new ArrayList<ReplaceableItem>();

    sampleData.add(new ReplaceableItem("Angelina-Jolie").withAttributes(
            new ReplaceableAttribute("Name", "Angelina Jolie Pitt", true),
            new ReplaceableAttribute("Movie", "Lara Croft: Tomb Raider", true),
            new ReplaceableAttribute("PictureURL", "https://s3.amazonaws.com/hqiu-hu-cloud-computing-hw6/stars/images/Img_Angelin
            new ReplaceableAttribute("ResumeURL", "https://s3.amazonaws.com/hqiu-hu-cloud-computing-hw6/stars/resumes/Resume_Ange

    sampleData.add(new ReplaceableItem("Audrey-Hepburn").withAttributes(
            new ReplaceableAttribute("Name", "Audrey Hepburn", true),
            new ReplaceableAttribute("Movie", "Roman Holiday", true),
            new ReplaceableAttribute("PictureURL", "https://s3.amazonaws.com/hqiu-hu-cloud-computing-hw6/stars/images/Img_Audrey_
            new ReplaceableAttribute("ResumeURL", "https://s3.amazonaws.com/hqiu-hu-cloud-computing-hw6/stars/resumes/Resume_Audr

    sampleData.add(new ReplaceableItem("Jennifer-Aniston").withAttributes(
            new ReplaceableAttribute("Name", "Jennifer Joanna Aniston", true),
            new ReplaceableAttribute("Movie", "The Good Girl", true),
            new ReplaceableAttribute("PictureURL", "https://s3.amazonaws.com/hqiu-hu-cloud-computing-hw6/stars/images/Img_Jennife
            new ReplaceableAttribute("ResumeURL", "https://s3.amazonaws.com/hqiu-hu-cloud-computing-hw6/stars/resumes/Resume_Jenn

    sampleData.add(new ReplaceableItem("Albert-Einstein").withAttributes(
            new ReplaceableAttribute("Name", "Albert Einstein", true),
            new ReplaceableAttribute("Movie", "Any Movie", true),
            new ReplaceableAttribute("PictureURL", "https://s3.amazonaws.com/hqiu-hu-cloud-computing-hw6/nobels/images/Img_Albert
            new ReplaceableAttribute("ResumeURL", "https://s3.amazonaws.com/hqiu-hu-cloud-computing-hw6/nobels/resumes/Resume_Alb
            new ReplaceableAttribute("YearOfNobel", "1921", true),
            new ReplaceableAttribute("FieldOfScience", "Physics", true)));
```

Check that we can select some data that we just inserted.

```
// Select data from a domain
// Notice the use of backticks around the domain name in our select expression.
String selectExpression = "select * from `" + myDomain + "` where Movie = 'Roman Holiday'";
System.out.println("Selecting: " + selectExpression + "\n");
SelectRequest selectRequest = new SelectRequest(selectExpression);
for (Item item : sdb.select(selectRequest).getItems()) {
    System.out.println("  Item");
    System.out.println("    Name: " + item.getName());
    for (Attribute attribute : item.getAttributes()) {
        System.out.println("      Attribute");
        System.out.println("        Name:  " + attribute.getName());
        System.out.println("        Value: " + attribute.getValue());
    }
}
System.out.println();
```

d. Demonstrate that you can change (correct) the year of one Nobel prize award, programmatically.

Update the “YearOfNobel” of “Albert-Einstein”. First setup a “replaceableAttributes” and then invoke the “putAttributes()” request.

```
// Replace/Change the year of Nobel prize award of a Nobel laureate.
System.out.println("Replacing YearOfNobel of Albert-Einstein with 1922.\n");
List<ReplaceableAttribute> replaceableAttributes = new ArrayList<ReplaceableAttribute>();
replaceableAttributes.add(new ReplaceableAttribute("YearOfNobel", "1922", true));
sdb.putAttributes(new PutAttributesRequest(myDomain, "Albert-Einstein", replaceableAttributes));
```

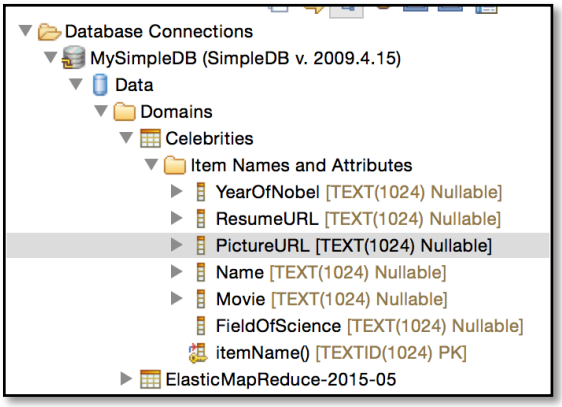
e. Demonstrate that you can delete one movie star from SimpleDB domain programmatically.

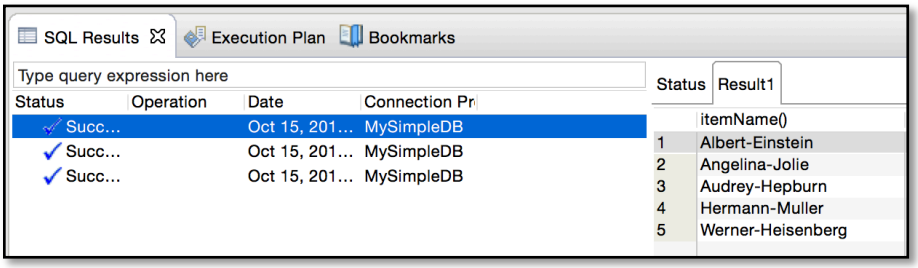
Delete one movie star using “deleteAttributes()” with the primary key.

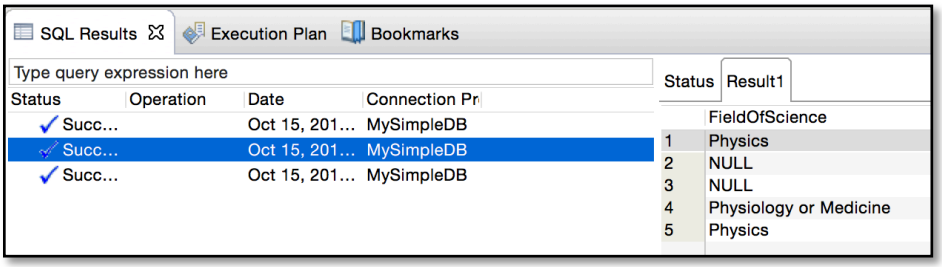
```
// Delete a movie star and all of its attributes
System.out.println("Deleting star Jennifer Aniston.\n");
sdb.deleteAttributes(new DeleteAttributesRequest(myDomain, "Jennifer-Aniston"));
```

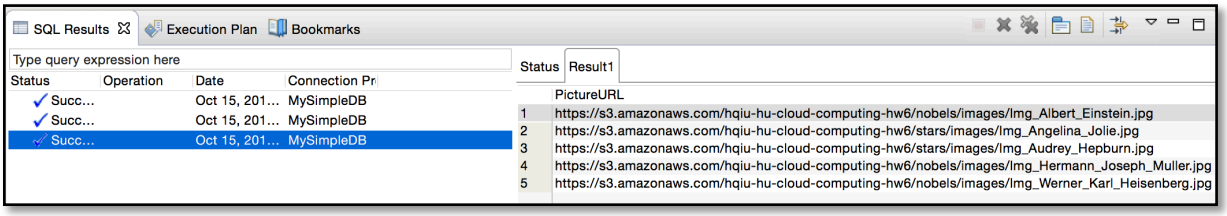
f. Capture the content of your database as displayed in the Database Development perspective.

Check from the “database perspective” in Eclipse. We have deleted one actor before, so we only have five celebrities here.



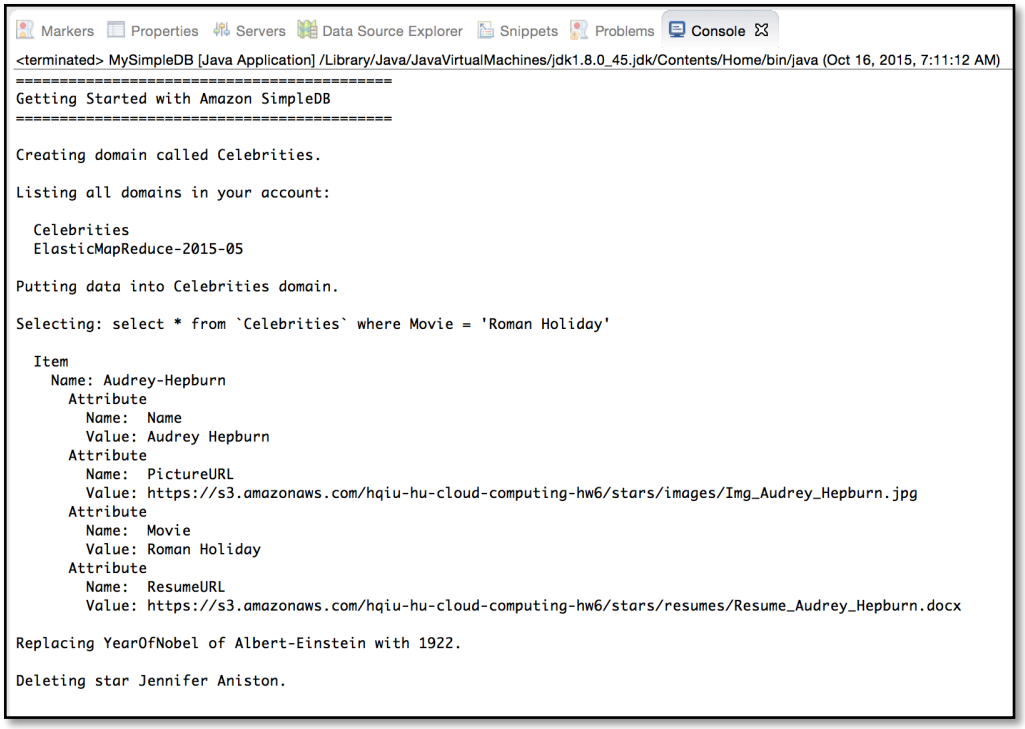






Provide working code and capture all stages of testing. Write your program either in Java or .Net or your favorite language supported by AWS. **[30 Points]**

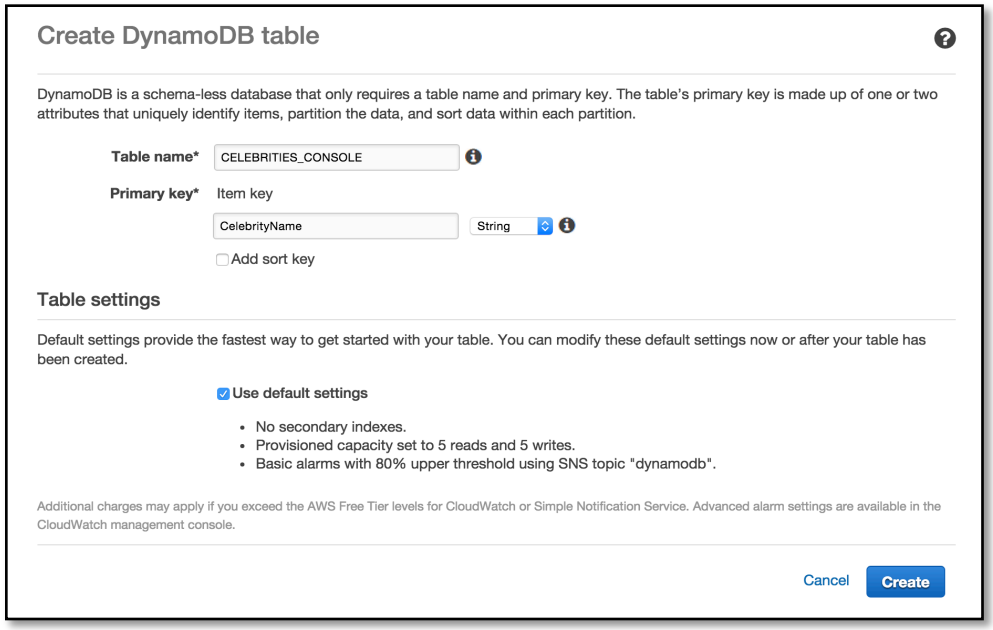
Results from the Eclipse console.



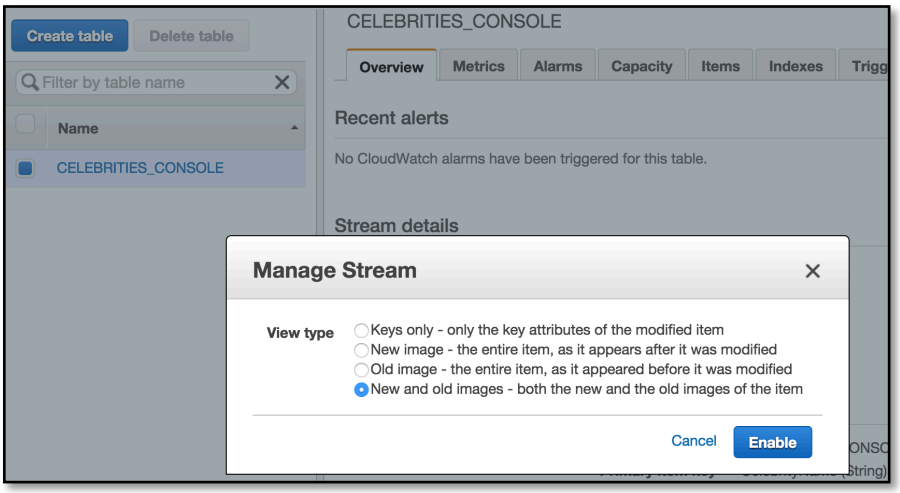
Problem 2.

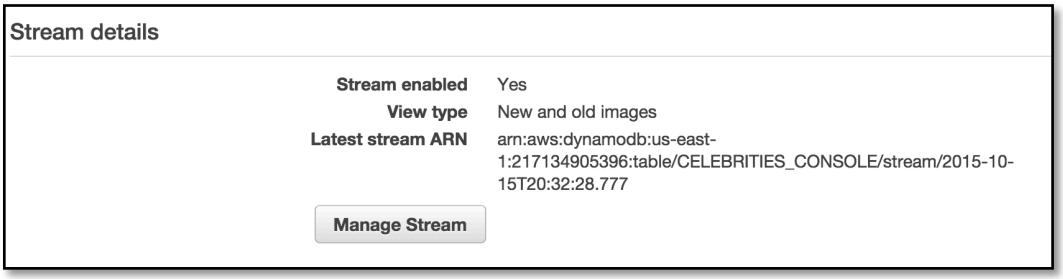
DynamoDB version. Use the same S3 bucket, images and resumes created in Problem 1. Use AWS DynamoDB Console for all the work

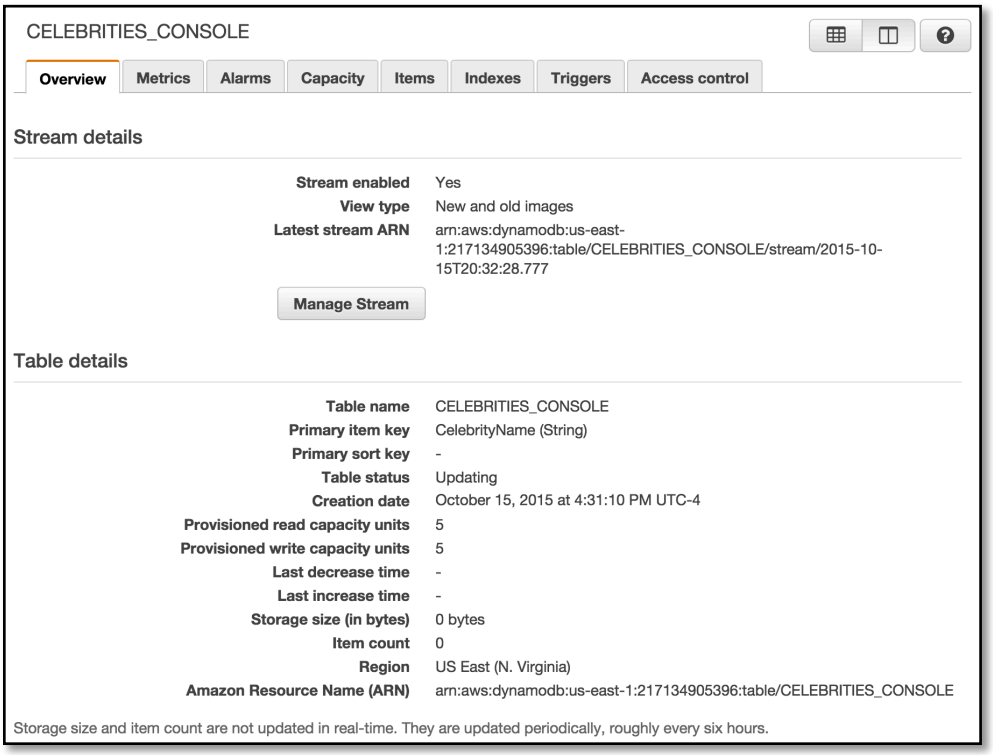
a) Create a new table, "CELEBRITIES_CONSOLE", where primary key is of a Hash type, and is constructed as "firstName" + "-" + "lastName". (for example, "Julia-Roberts")



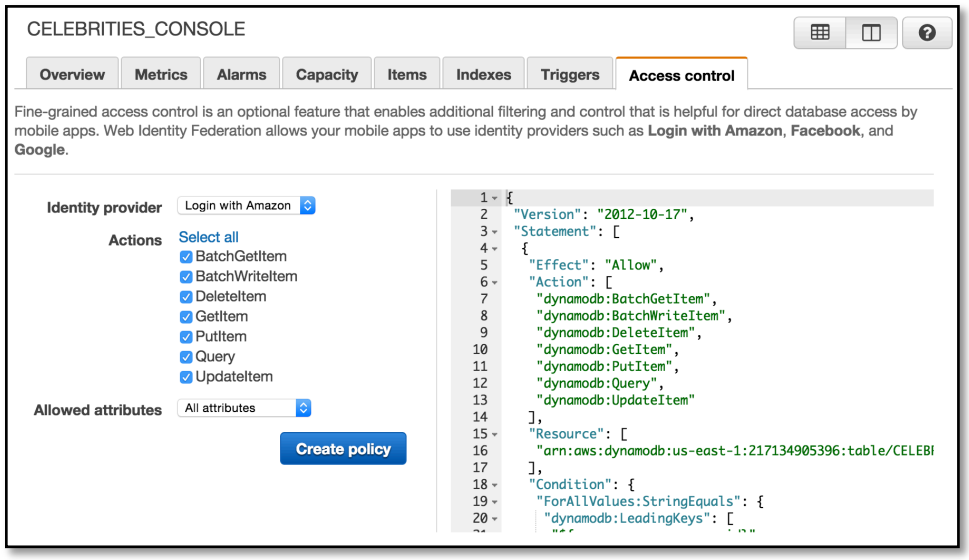
Create the Dynamo DB table, table name is “CELEBRITIES_CONSOLE”, primary key is “CelebrityName”. The table setting (provisioned throughput, etc.) uses the default settings here. Once the table is created, enable the “Manage Streams”.







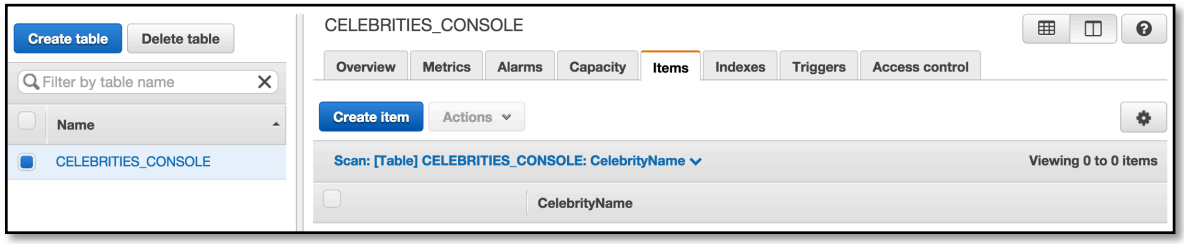
Set the Access Control. Enable all the APIs when “Login with Amazon”.

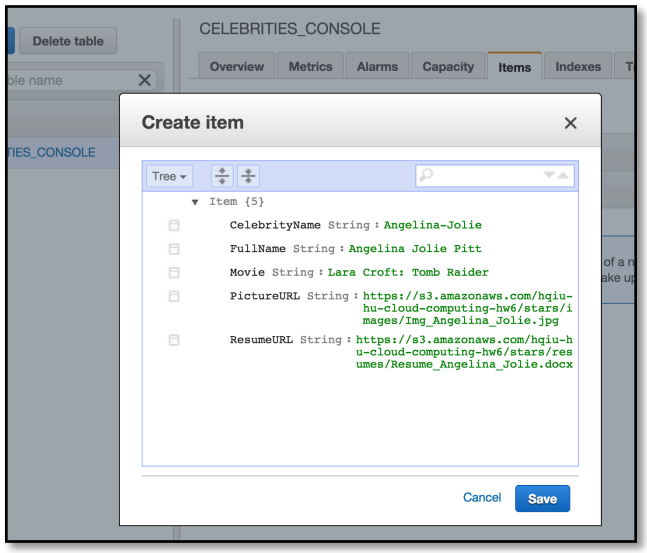


b) Insert three rows for movie stars, with attributes: full name (= firstName + " " + lastName), most popular movie, S3 URL of the picture of the star and S3 URL of his or her resume

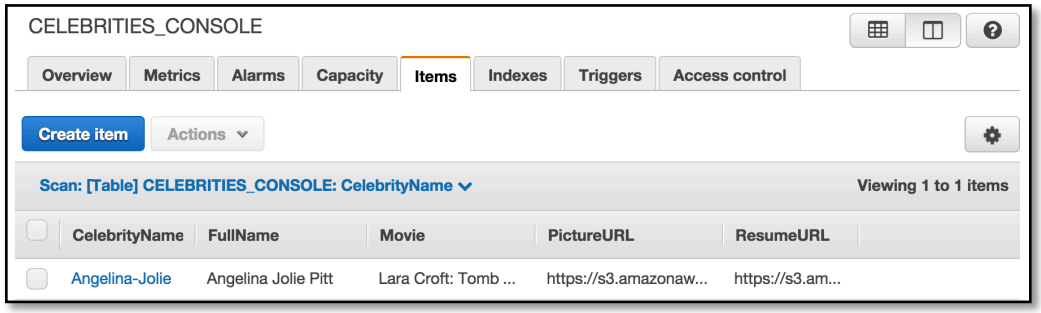
c) Insert three rows for Noble laureates, with attributes: full name (same format as above), S3 URL of the picture of the lauret, S3 URL of his or her resume, award year, science field in which they got the prize

In the dashboard, select “Tables”, we will see the table we just created.

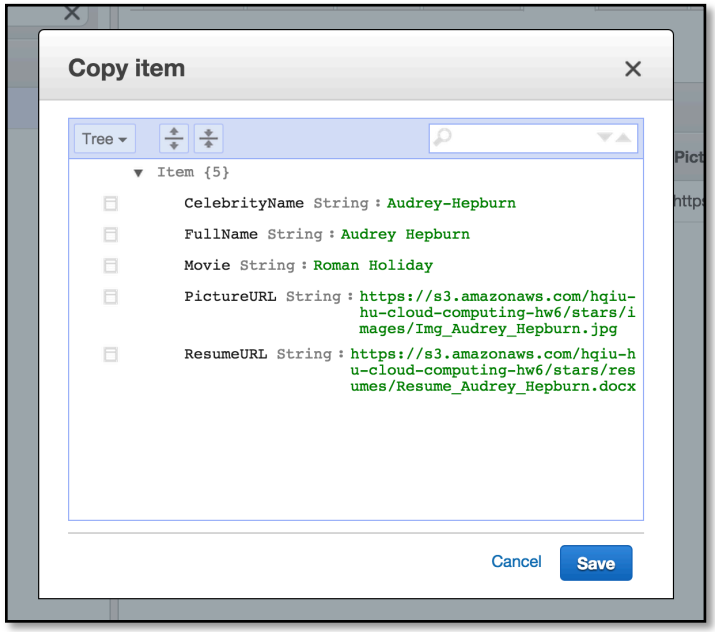




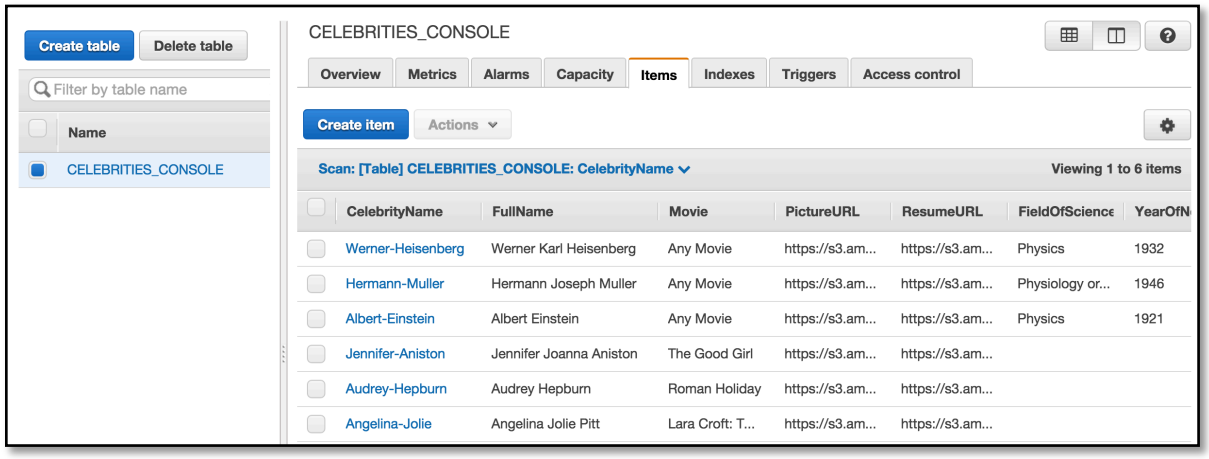
Select “Create Item” and create new items into the table. Click “Save”, we will see the item we just created immediately.



We can copy the item and edit it to another new item.

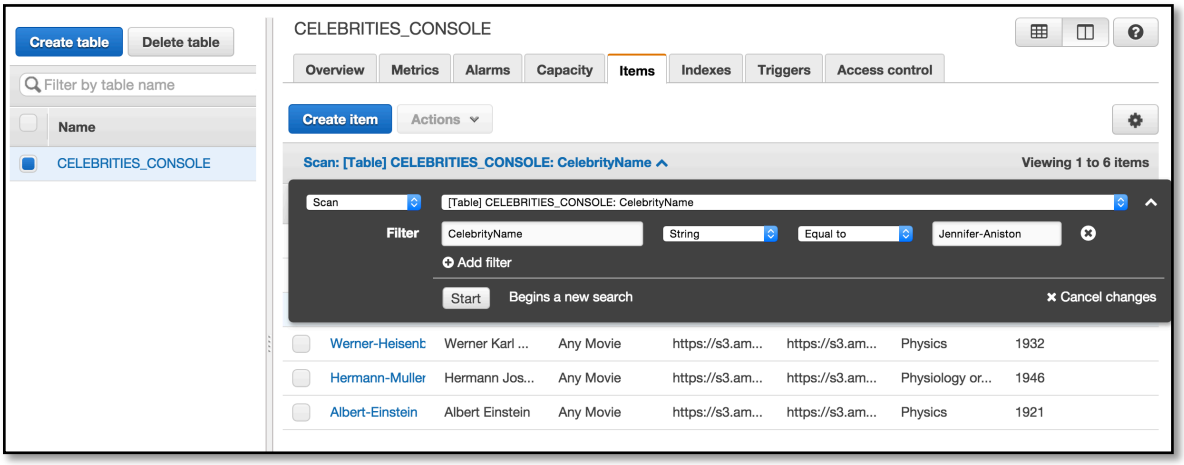


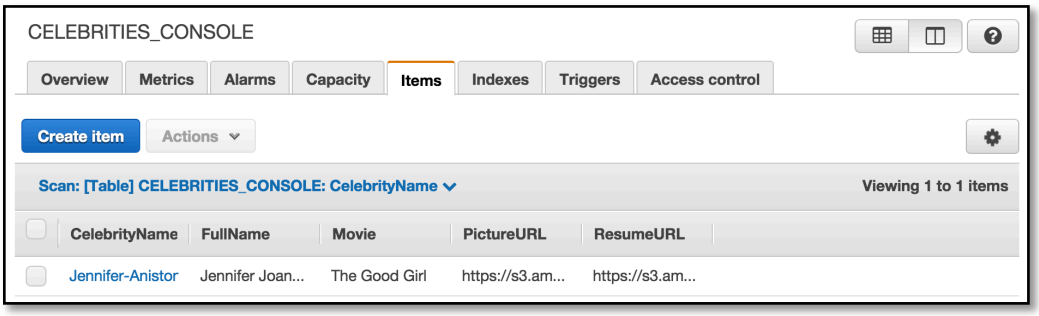
Add three stars and three Nobel laureates. The Nobel laureates have two more attributes “YearOfNobel” and “FieldOfScience”.



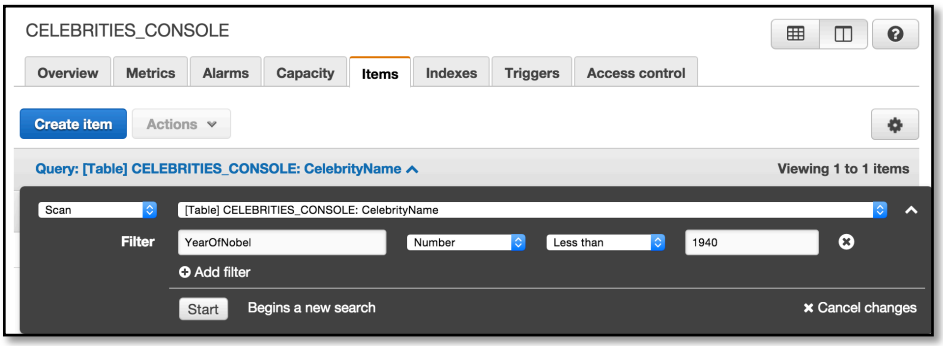
d) Explore the table in the Console and search by providing filters on ID - both scans and get by ID types of queries

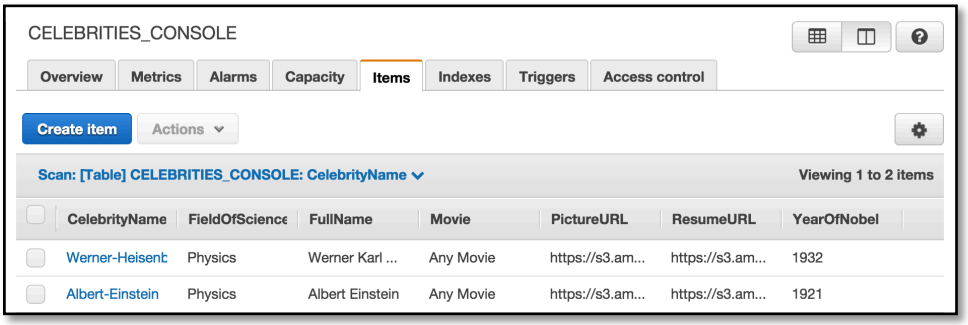
First try to scan the table and give some searching criteria. We want to filter the item whose "CelebrityName" equals to "Jennifer-Aniston". Hit "Start", only one item will be filtered out.



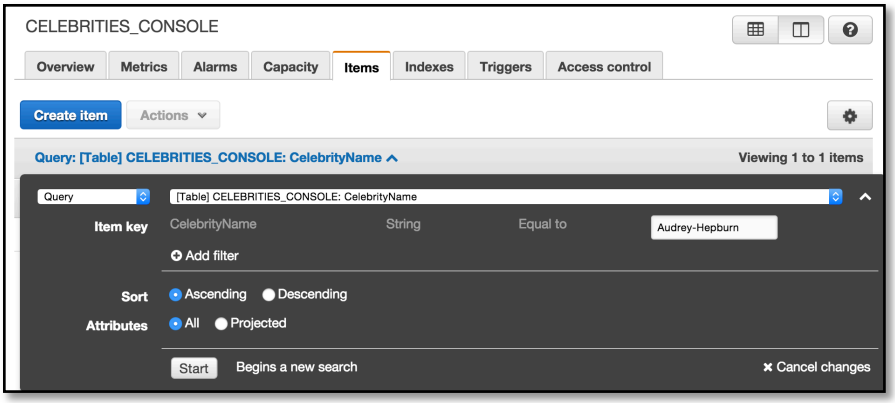


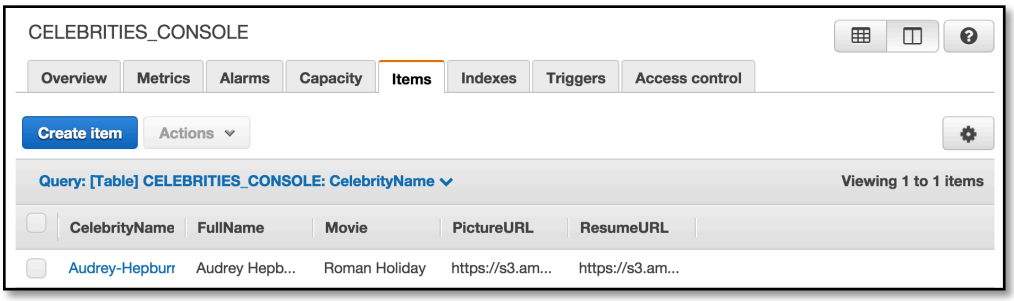
Try to search the items whose "YearOfNobel" is less than 1940.



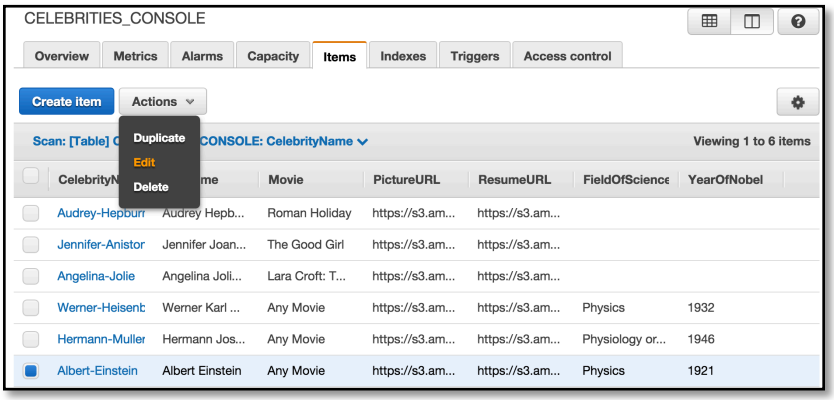


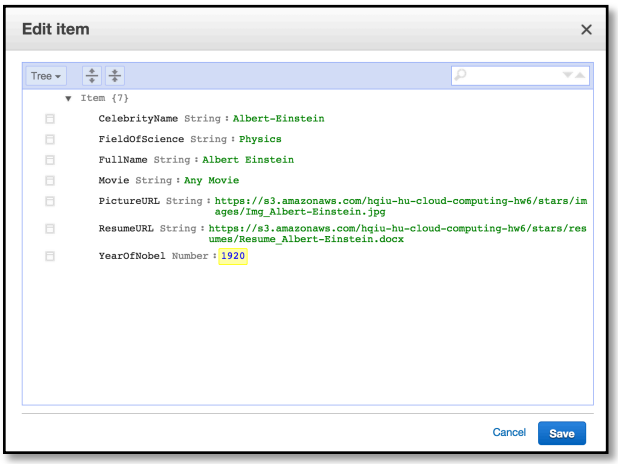
For the Query, we should give a primary key while doing query. The functionality of filter is identical to the previous one.





e) Update an award year of one of the Nobles rows - demo that the data is updated. Capture all stages of testing. **[15 Points]**





We have two ways to edit the table. We can either select “Actions”->”Edit” to modify the value, or click on the value to change it. Either way can change “YearOfNobel” from “1921” to “1920”.

Problem 3.

Use AWS SDK (Java or any other) to do the same work as in Problem 2, with minor changes:

First create a Dynamo DB project in Eclipse. We can choose the Dynamo DB template. It will generate the template code automatically. The code is the same with the Dynamo DB Sample code. Set up the credential information in the init() step.

```
private static void init() throws Exception {
    /*
     * The ProfileCredentialsProvider will return your [default]
     * credential profile by reading from the credentials file located at
     * (/Users/hqiu/.aws/credentials).
     */
    AWSCredentials credentials = null;
    try {
        credentials = new ProfileCredentialsProvider("default").getCredentials();
    } catch (Exception e) {
        throw new AmazonClientException(
                "Cannot load the credentials from the credential profiles file. " +
                "Please make sure that your credentials file is at the correct " +
                "location (/Users/hqiu/.aws/credentials), and is in valid format.",
                e);
    }
    dynamoDB = new AmazonDynamoDBClient(credentials);
    Region usEast1 = Region.getRegion(Regions.US_EAST_1);
    dynamoDB.setRegion(usEast1);
}
```

a) Same - except name the table "CELEBRITIES_SDK"

Create a table named “CELEBRITIES_SDK” using “CreateTableRequest()”.

```
try {
    tableName = "CELEBRITIES_SDK";

    // Create table if it does not exist yet
    if (Tables.doesTableExist(dynamoDB, tableName)) {
        System.out.println("\n\nTable " + tableName + " is already ACTIVE");
    } else {
        // Create a table with a primary hash key named 'name', which holds a string
        CreateTableRequest createTableRequest = new CreateTableRequest().withTableName(tableName)
            .withKeySchema(new KeySchemaElement().withAttributeName("CelebrityName").withKeyType(KeyType.HASH))
            .withAttributeDefinitions(new AttributeDefinition().withAttributeName("CelebrityName").withAttributeType(ScalarAttrib
            .withProvisionedThroughput(new ProvisionedThroughput().withReadCapacityUnits(1L).withWriteCapacityUnits(1L));
            TableDescription createdTableDescription = dynamoDB.createTable(createTableRequest).getTableDescription();
        System.out.println("Created Table: " + createdTableDescription);

        // Wait for it to become active
        System.out.println("Waiting for " + tableName + " to become ACTIVE...");
        Tables.awaitTableToBecomeActive(dynamoDB, tableName);
    }
```

The “KeySchema” is the primary key “CelebrityName”. It’s a hash type. We can also set the provisioned throughput through this step.

b) Same

c) Same

I used two different ways to insert rows into the table. The first method “addStart()” uses Dynamo DB class to get the table, and put the item directly into the table with the primary key. The second method “addNobel()” uses the template code, it invokes the method “PutItemRequest()” of the Dynamo DB client and insert the row with table name.

```
private static void addStar(String keyName, String fullName, String movie, String picURL, String resumeURL) {

    System.out.println("Adding stars: " + keyName + "/" + fullName + "/" + movie +
            "/" + picURL + "/" + resumeURL);

    /*
    Map<String, AttributeValue> item = new HashMap<String, AttributeValue>();
    item.put("CelebrityName", new AttributeValue(keyName));
    item.put("Name", new AttributeValue(fullName));
    item.put("Movie", new AttributeValue(movie));
    item.put("PictureURL", new AttributeValue(picURL));
    item.put("ResumeURL", new AttributeValue(resumeURL));

    PutItemRequest putItemRequest = new PutItemRequest(tableName, item);
    dynamoDB.putItem(putItemRequest);
    */

    Item item = new Item().withPrimaryKey("CelebrityName", keyName)
            .withString("Name", fullName)
            .withString("Movie", movie)
            .withString("PictureURL", picURL)
            .withString("ResumeURL", resumeURL);
    table.putItem(item);
}

private static void addNobel(String keyName, String fullName, String movie, String picURL, String resumeURL,
        Integer year, String field) {

    System.out.println("Adding nobel laureates: " + keyName + "/" + fullName + "/" + movie +
            "/" + picURL + "/" + resumeURL + "/" + year + "/" + field);

    Map<String, AttributeValue> item = new HashMap<String, AttributeValue>();
    item.put("CelebrityName", new AttributeValue(keyName));
    item.put("Name", new AttributeValue(fullName));
    item.put("Movie", new AttributeValue(movie));
    item.put("PictureURL", new AttributeValue(picURL));
    item.put("ResumeURL", new AttributeValue(resumeURL));
    item.put("YearOfNobel", new AttributeValue().withN(Integer.toString(year)));
    item.put("FieldOfScience", new AttributeValue(field));

    PutItemRequest putItemRequest = new PutItemRequest(tableName, item);
    dynamoDB.putItem(putItemRequest);

    // PutItemResult putItemResult = dynamoDB.putItem(putItemRequest);
    // System.out.println("Result: " + putItemResult.toString());
}
```

```
addStar("Jennifer-Aniston", "Jennifer Joanna Aniston", "The Good Girl",
        "https://s3.amazonaws.com/hqiu-hu-cloud-computing-hw6/stars/images/Img_Jennifer_Aniston.jpg",
        "https://s3.amazonaws.com/hqiu-hu-cloud-computing-hw6/stars/resumes/Resume_Jennifer_Aniston.docx");

addNobel("Albert-Einstein", "Albert Einstein", "Any Movie",
        "https://s3.amazonaws.com/hqiu-hu-cloud-computing-hw6/nobels/images/Img_Albert_Einstein.jpg",
        "https://s3.amazonaws.com/hqiu-hu-cloud-computing-hw6/nobels/resumes/Resume_Albert_Einstein.docx",
        1921, "Physics");
```

d) Query both CELEBRITIES_CONSOLE and CELEBRITIES_SDK tables via

SDK APIs - demonstrate that you see the inserted data

First query the data using the “query()” method. Similar to the AWS console, query through the “Query” should give the primary key/schema key, which is the “CelebrityName”here. We can also add some filter criteria here. Since the “YearOfNobel” of “Albert-Einstein” is 1921, it will meet the requirements and be filtered out.

```
// Query items for Nobel laureates with a primary key Albert-Einstein from CELEBRITIES_SDK
QuerySpec spec = new QuerySpec()
        .withHashKey("CelebrityName", "Albert-Einstein")
        .withFilterExpression("YearOfNobel between :v_start_yn and :v_end_yn")
        .withValueMap(new ValueMap()
                .withInt(":v_start_yn", 1920)
                .withInt(":v_end_yn", 1922));
ItemCollection<QueryOutcome> items = table.query(spec);

System.out.println("\nQuery items for Nobel laureates with a primary key Albert-Einstein from table "+ tableName + ":");
for (Item item: items) {
    System.out.println(item);
}
```

Query the data using the “Scan” facility. The “ScanRequest” class needs to be created with a table name. The first “scan” will query the data from table “CELEBRITES_SDK”. It will scan for items whose “YearOfNobel” is less than1940. The second “scan” will query the data from table “CELEBRITIES_CONSOLE”. It will search for items whose “movie” name is “Roman Holiday”.

```
// Scan items for Nobel laureates with a year attribute smaller than 1940 from CELEBRITIES_SDK
HashMap<String, Condition> scanFilter = new HashMap<String, Condition>();
Condition condition = new Condition()
    .withComparisonOperator(ComparisonOperator.LT)
    .withAttributeValueList(new AttributeValue().withN("1940"));
scanFilter.put("YearOfNobel", condition);
ScanRequest scanRequest = new ScanRequest(tableName).withScanFilter(scanFilter);
ScanResult scanResult = dynamoDB.scan(scanRequest);
System.out.println("\nScan items for Nobel laureates with a YearOfNobel attribute smaller than 1940 from table "+ tableName + ":");
System.out.println("Result: " + scanResult);

// Scan items for Stars with a specific movie name from CELEBRITIES_CONSOLE.
scanFilter = new HashMap<String, Condition>();
condition = new Condition()
    .withComparisonOperator(ComparisonOperator.EQ.toString())
    .withAttributeValueList(new AttributeValue().withS("Roman Holiday"));
scanFilter.put("Movie", condition);
scanRequest = new ScanRequest("CELEBRITIES_CONSOLE").withScanFilter(scanFilter);
scanResult = dynamoDB.scan(scanRequest);
System.out.println("\nScan items for Stars with a movie named \"Roman Holiday\" from table CELEBRITIES_CONSOLE:");
System.out.println("Result: " + scanResult);
```

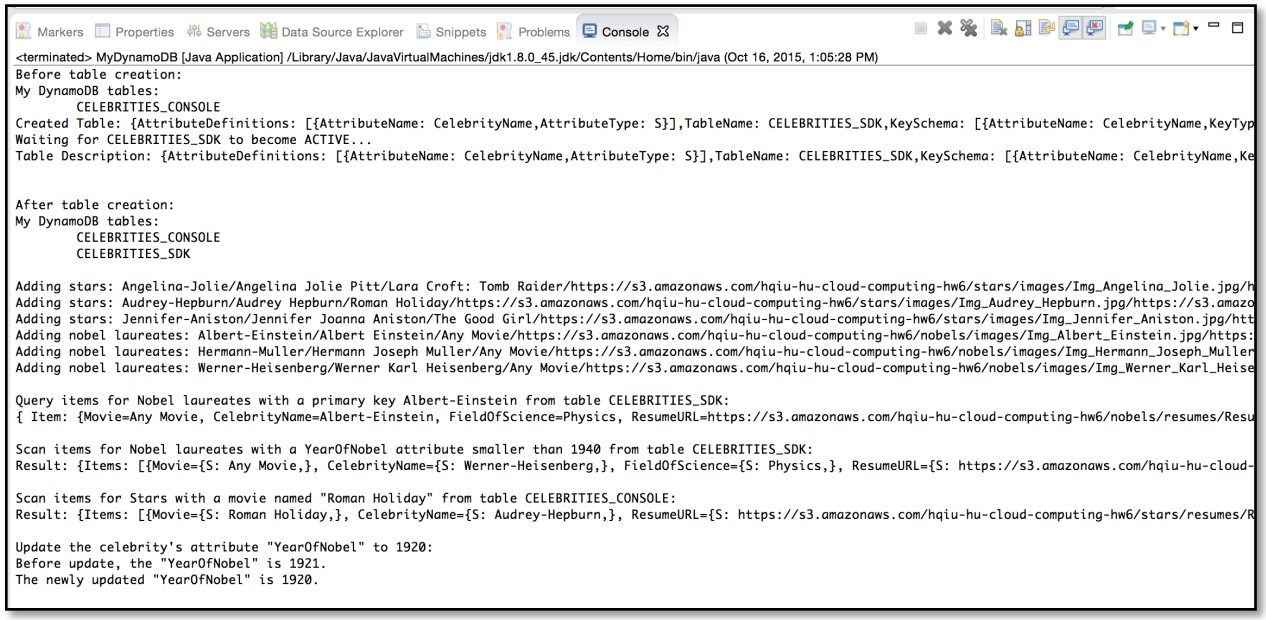
e) Same

Update the item by invoking the method “updateItem()” of the Dynamo DB “table” class.

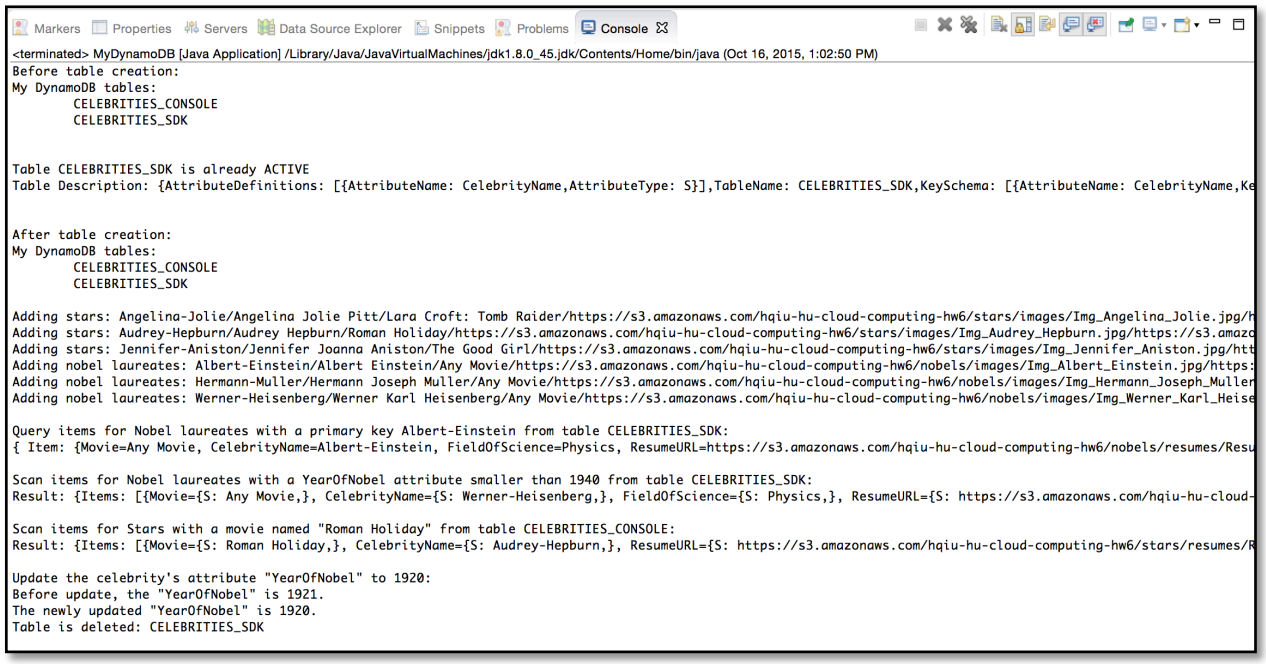
```
public static void updateYearOfNobel(String keyName, int year) {
    System.out.println("\nUpdate the celebrity's attribute \"YearOfNobel\" to " + year +":");

    System.out.println("Before update, the \"YearOfNobel\" is " + getYearOfNobel(keyName) + ".");

    table.updateItem("CelebrityName", keyName,
                new AttributeUpdate("YearOfNobel").put(year));

    System.out.println("The newly updated \"YearOfNobel\" is " + getYearOfNobel(keyName) + ".");
}

public static int getYearOfNobel(String keyName) {
    GetItemOutcome outcome = table.getItemOutcome(new GetItemSpec()
                .withPrimaryKey("CelebrityName", keyName)
                .withConsistentRead(true));

    Item item = outcome.getItem();
    return item.getInt("YearOfNobel");
}
```

Provide working code and capture all stages of testing. **[25 Points]**

Results from the Eclipse Console. The results below shows that we created a new table called "CELEBRITIES_SDK". We added a few rows into the table. The "Query" function queries out "Albert-Einstein", whose "YearOfNobel" is between 1920 and 1921. The two "Scan" function select out the items whose "YearOfNobel" is less than 1940 and whose "movie" is "Roman Holiday" respectively. The program also updates "YearOfNobel" to 1920 in the end.



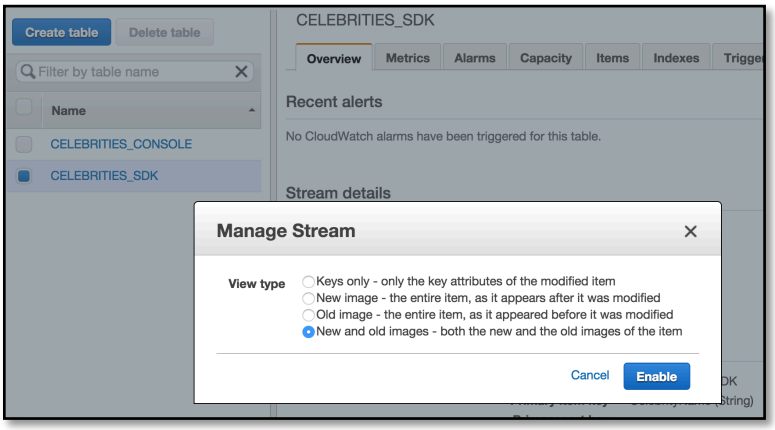
The results below are almost the same with the previous one. It just skipped to create the table since the table already exists. It also shows the table is deleted in the end. The code to delete the table will be shown in Problem 5.

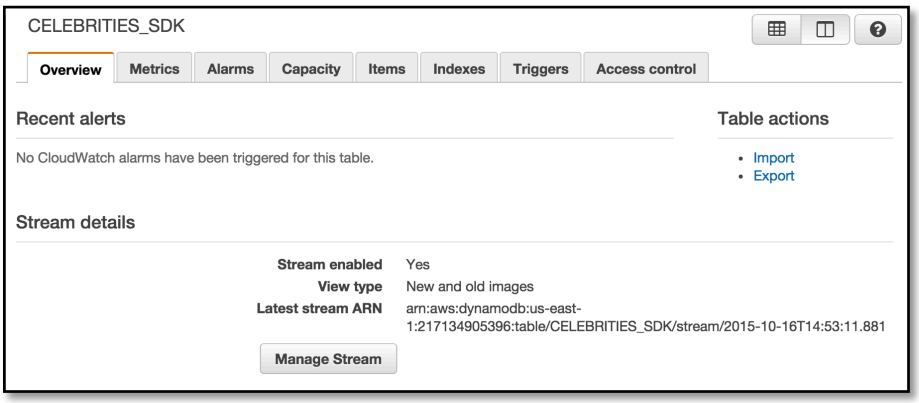


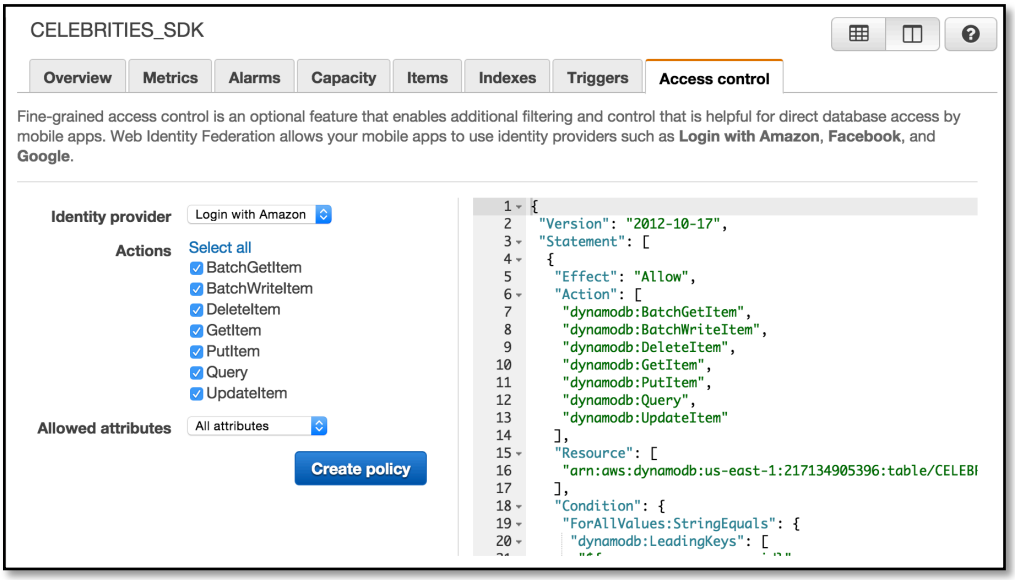
Problem 4.

Create a Lambda function that will monitor DynamoDB stream of table CELEBRITIES_SDK and write all operations performed on that table into CloudWatch logs. Use Java class developed in Problem 3 to generate load on table CELEBRITIES_SDK. Comment out the section of the class where you create the table itself. Capture all steps of your development and testing. [**20 Points]**

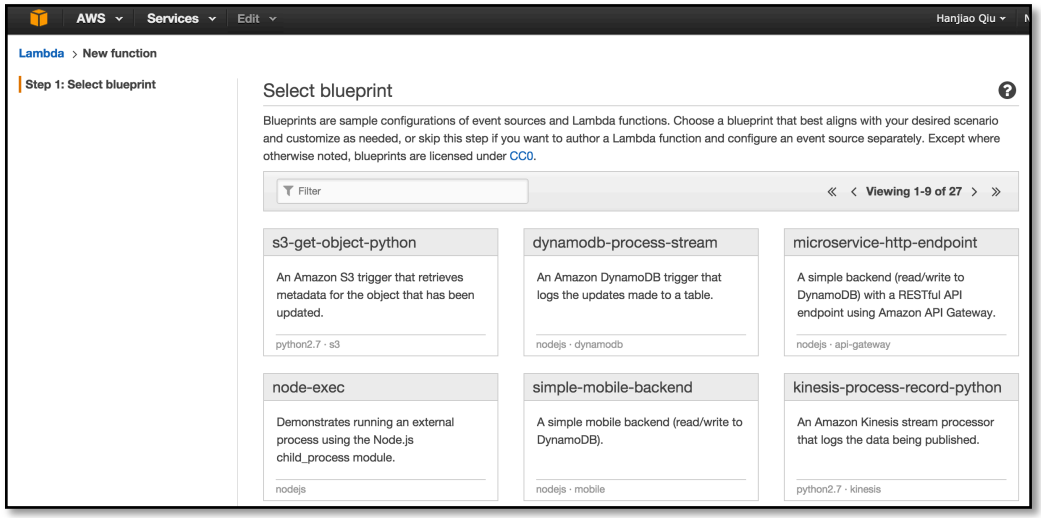
a) Enable the streams, choose the "new and old images". Set the "Access Control".



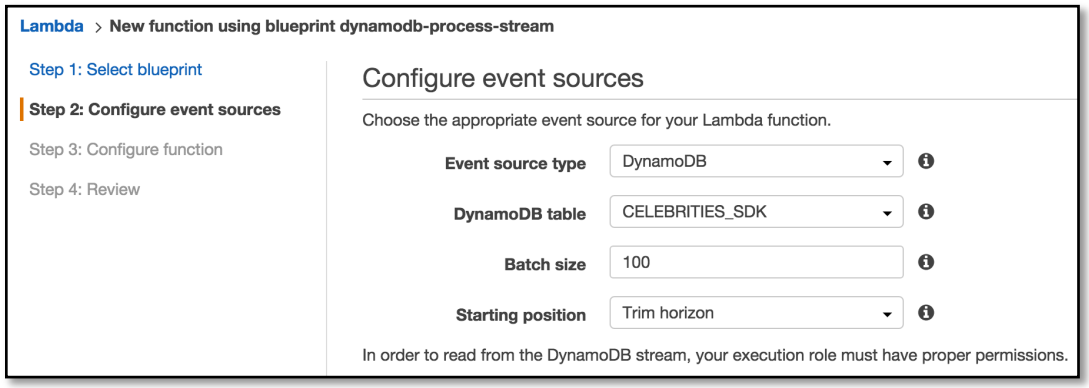




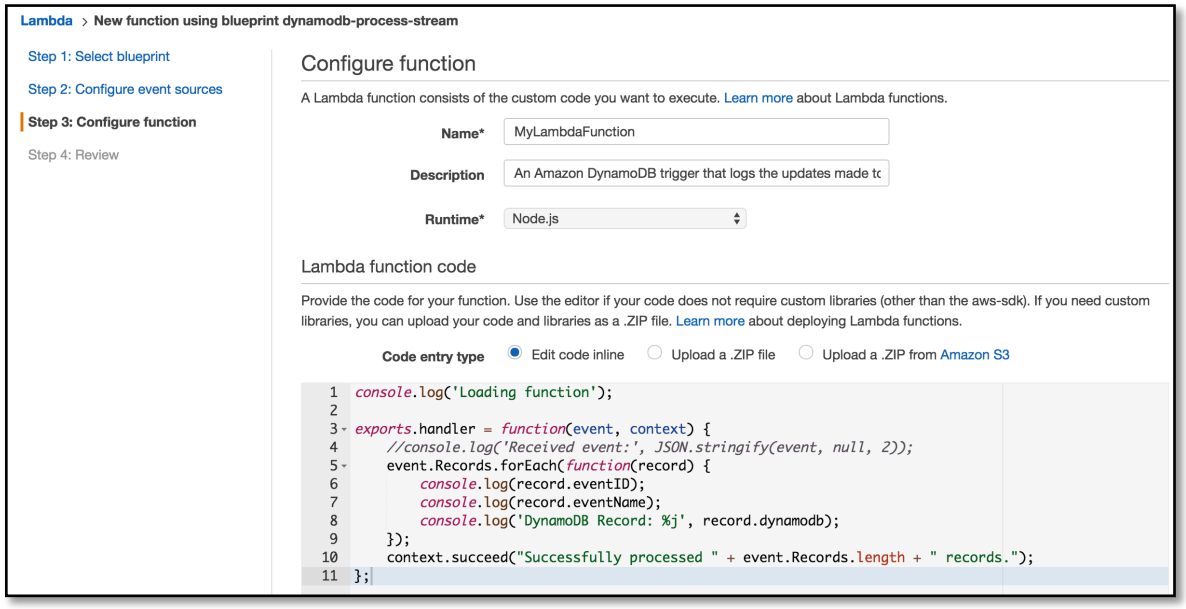
b) Go to AWS Lambda service and create a Lambda function. On the “Select blueprint” screen, choose the second one on the first row “dynamodb-process-stream”.



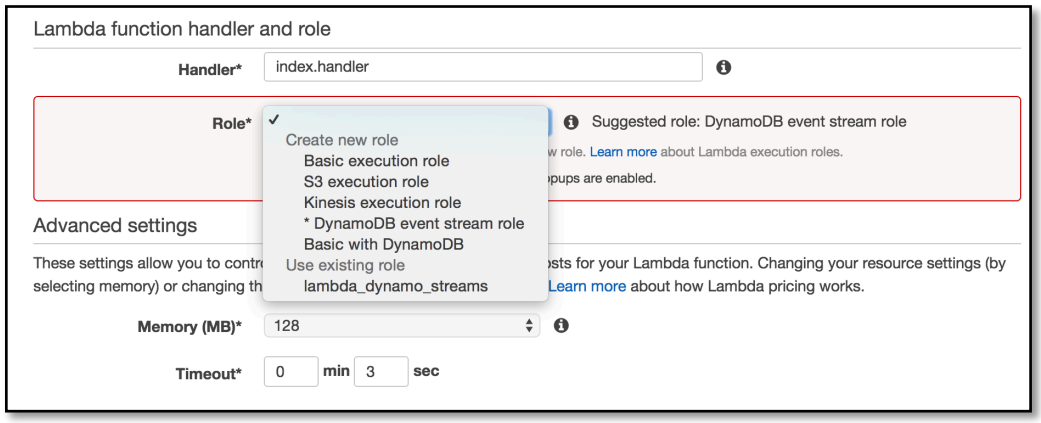
c) On the “Configure event sources” screen, select “CELEBRITIES_SDK” table.



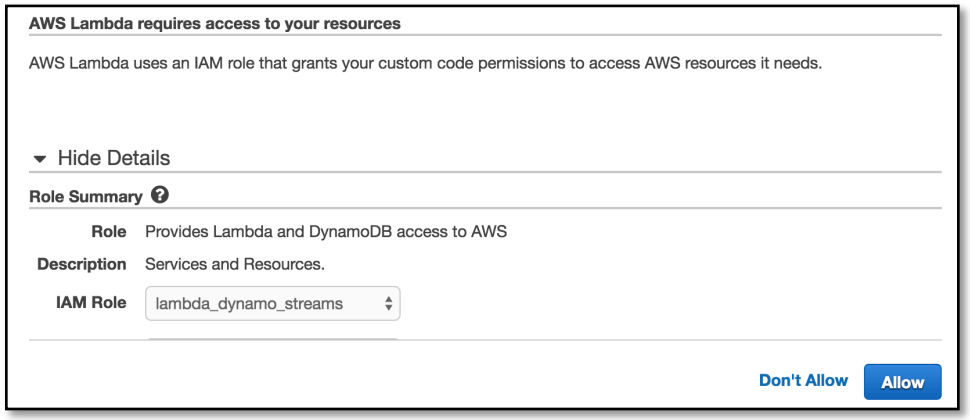
d) Configure the function, provide a function name and leave the function unmodified.



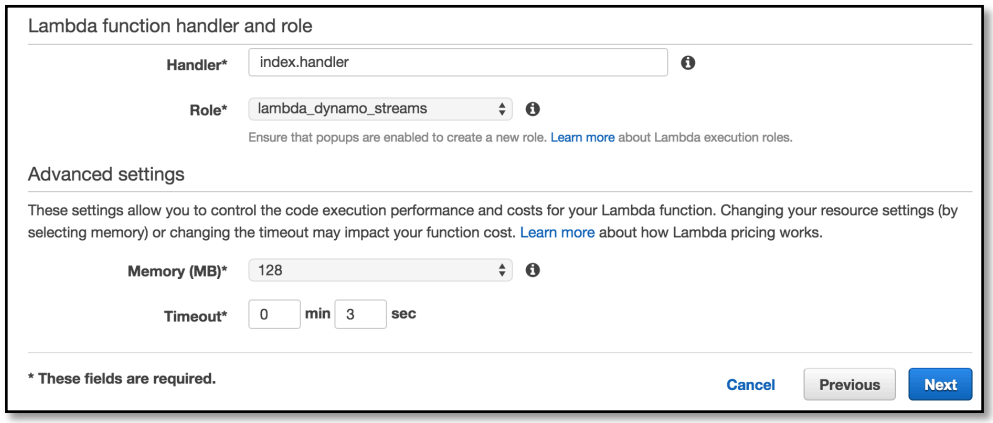
e) Adjust handler and role. On the bottom of “Configure function” screen leave “Handler” as is, as “Role” select suggested value “DynamoDB event stream role”. New screen opens asking for IAM role.



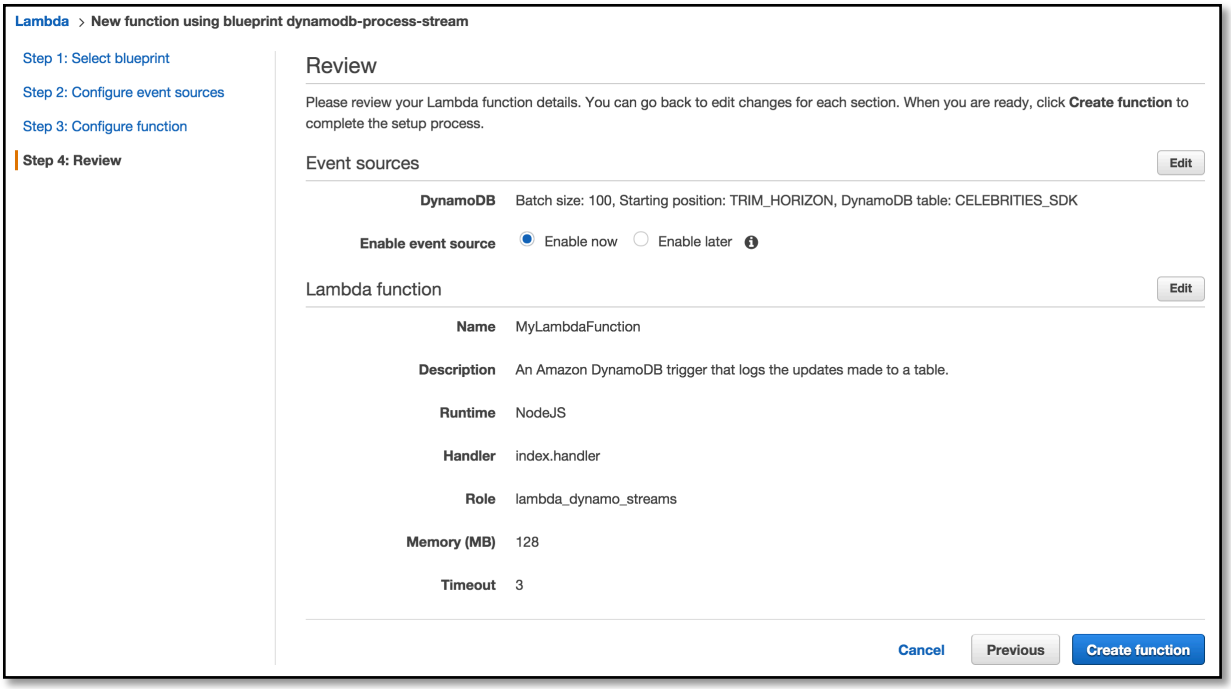
Accept, allow suggested IAM role.

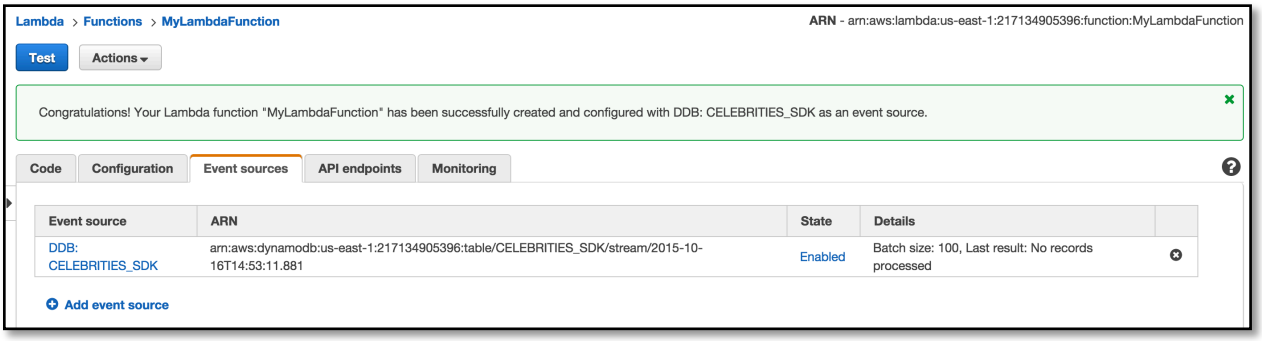


Leave advanced settings. Select “Next”.

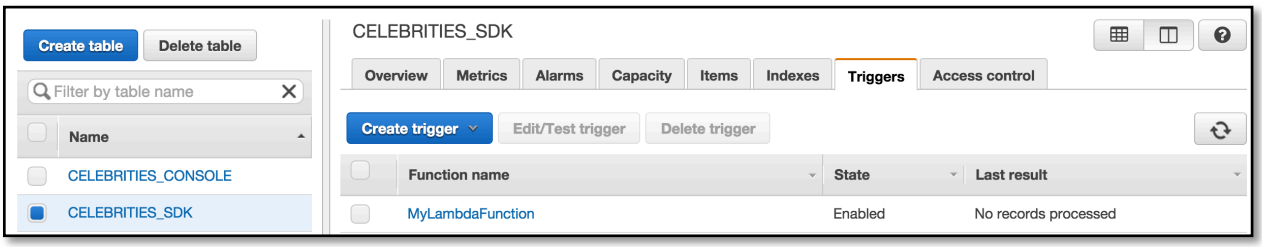


f) Review and select “Enable now”. Hit “Create Function”. The “Lambda Function” is ready to use.

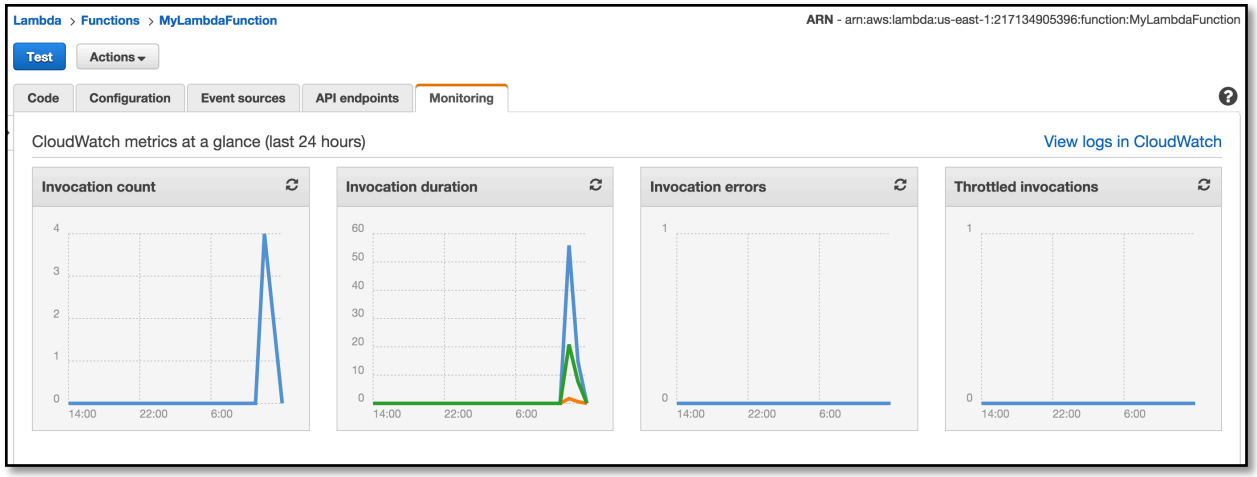




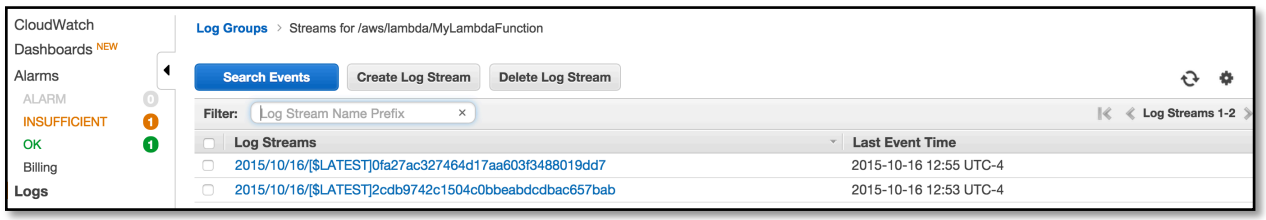
In DynamoDB Dashboard, select the table “CELEBRITIES_SDK”. We can also see that the lambda function was in the “Triggers” Tab of the DynamoDB table “CELEBRITIES_SDK”.



g) In AWS Lambda Dashboard, select the lambda function “MyLambdaFunction” we just created. Double click on it and select “Monitoring” Tab. CloudWatch graphs will show up.



h) On the top right of the page, select “View logs in CloudWatch”. We will see the logs below. Re-run the Java class developed in Problem 3 to generate load on table “CELEBRITIES_SDK”. Double click the log, we can see the comparison of “newImage” and “oldImage” here. The “YearOfNobel” is updated from 1921 to 1920.




```
    "NewImage": {
        "Movie": {
            "S": "Any Movie"
        },
        "CelebrityName": {
            "S": "Albert-Einstein"
        },
        "FieldOfScience": {
            "S": "Physics"
        },
        "ResumeURL": {
            "S": "https://s3.amazonaws.com/hqiu-hu-cloud-computing-hw6/nobels/resumes/Resume_Albert_Einstein.docx"
        },
        "PictureURL": {
            "S": "https://s3.amazonaws.com/hqiu-hu-cloud-computing-hw6/nobels/images/Img_Albert_Einstein.jpg"
        },
        "YearOfNobel": {
            "N": "1920"
        },
        "Name": {
            "S": "Albert Einstein"
        }
    },
    "OldImage": {
        "Movie": {
            "S": "Any Movie"
        },
        "CelebrityName": {
            "S": "Albert-Einstein"
        },
        "FieldOfScience": {
            "S": "Physics"
        },
        "ResumeURL": {
            "S": "https://s3.amazonaws.com/hqiu-hu-cloud-computing-hw6/nobels/resumes/Resume_Albert_Einstein.docx"
        },
        "PictureURL": {
            "S": "https://s3.amazonaws.com/hqiu-hu-cloud-computing-hw6/nobels/images/Img_Albert_Einstein.jpg"
        },
        "YearOfNobel": {
            "N": "1921"
        },
```

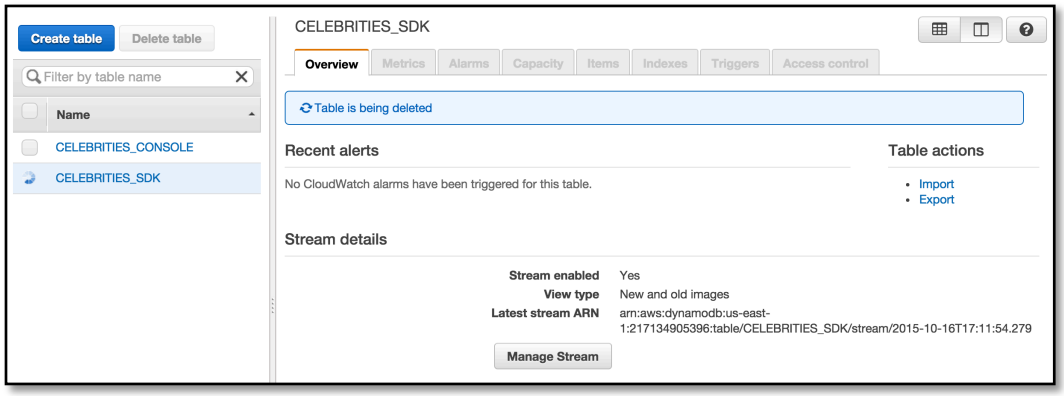
Since we didn't recreate the table, there's no log for creating tables here.

Problem 5.

When done, delete all DynamoDB tables and Lambda functions, programmatically or using AWS CLI. Provide working code and capture all stages of testing. **[10 Points]**

a) Delete the DynamoDB tables through Java SDK. It basically gets the "table" object and calls the "delete()" function.

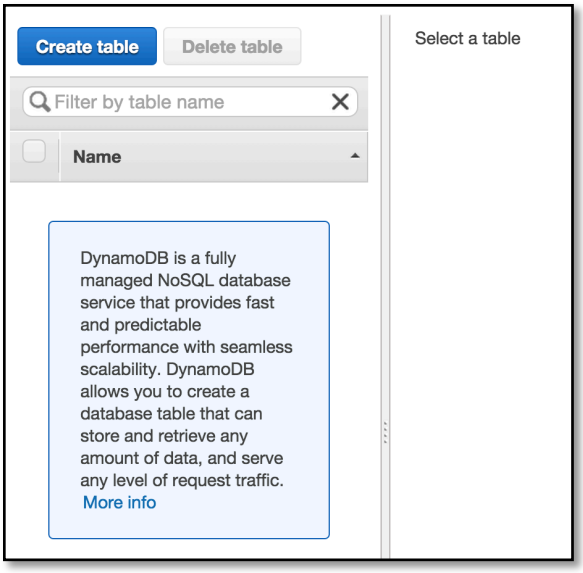
```
private static void deleteTable(String tableName) {
    Table table = new DynamoDB(dynamoDB).getTable(tableName);

    if (table == null)
        return;
    table.delete();

    try {
        table.waitForDelete();
    } catch (InterruptedException e) {
        System.out.println("Failed to delete table: " + tableName);
        e.printStackTrace();
    }
    System.out.println("Table is deleted: " + tableName);
}
```



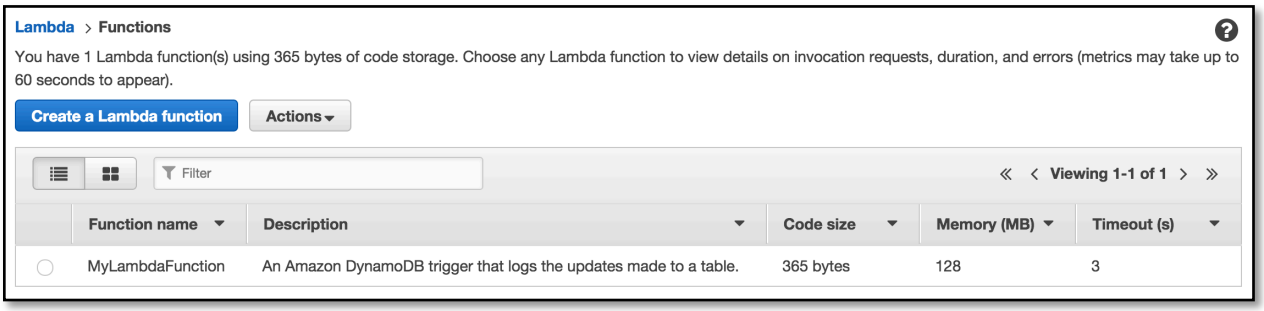
b) Delete the Dynamo DB tables through AWS CLI.

```
hqiu@bos-mpdei>> aws dynamodb delete-table --table-name CELEBRITIES_CONSOLE
{
    "TableDescription": {
        "TableArn": "arn:aws:dynamodb:us-east-1:217134905396:table/CELEBRITIES_CONSOLE",
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 1,
            "ReadCapacityUnits": 1
        },
        "TableSizeBytes": 0,
        "TableName": "CELEBRITIES_CONSOLE",
        "TableStatus": "DELETING",
        "StreamSpecification": {
            "StreamViewType": "NEW_AND_OLD_IMAGES",
            "StreamEnabled": true
        },
        "LatestStreamLabel": "2015-10-15T20:32:28.777",
        "ItemCount": 0,
        "LatestStreamArn": "arn:aws:dynamodb:us-east-1:217134905396:table/CELEBRITIES_CONSOLE/str
eam/2015-10-15T20:32:28.777"
    }
}
```

After delete:



c) Delete the lambda function through AWS CLI.




```
hqiu@bos-mpdei>> aws lambda delete-function --function-name MyLambdaFunction --region us-east-1
hqiu@bos-mpdei>>
```

After delete: