**ISS4014 – Database Systems and Web Integration**

**Chapter 14 – Activities and Homework**

**Chapter 14 Review Questions (10 points - 2 points each)**

* Respond to the following questions (Some responses to review questions are not in the book).

1. What are the traditional 3 Vs of Big Data? Briefly define each.

2. Explain the difference between scaling up and scaling out.

3. What are the key assumptions made by the Hadoop Distributed File System approach?

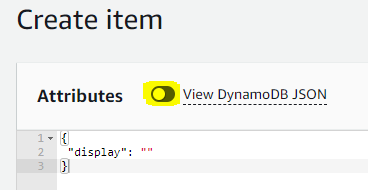
4. How does data mining work? Discuss the different phases in the data mining process.

5. Explain the difference between the traditional way of storing data in a relational database and storing data in a NoSQL database such as MongoDB or DynamoDB.

**AWS DynamoDB Lab (20 points)**

Part 1 – Creating a DynamoDB table, loading items via a JSON string, and exploring a Schemaless structure

1. Log into the AWS Academy Canvas system, go to the AWS Academy Learner Lab course, go to Modules, select “Launch AWS Academy Learner Lab,” click on “Start Lab,” and wait for the AWS status icon dot to turn green. Once the status dot is green, click on AWS to open the AWS console home page.
2. In the search bar, type “DynamoDB” and select the DynamoDB service
3. From the DynamoDB service screen, click on “Create table".
4. Name the table “Fact” (no quotes) and set the Partition key to “display” – type String (all lowercase – no quotes). Leave everything else as default and click on the “Create table” button at the bottom of the screen.
5. Wait for the table to be created (about 30 seconds), and then click on the “fact” table name.
6. Click on the “Explore table items” in the top right hand of the screen.
7. Click on the “Create item” button near the bottom of the screen.
8. Click on the “JSON view” button near the top right hand of the screen and de-select the “View DynamoDB JSON switch near the top left hand of the screen.



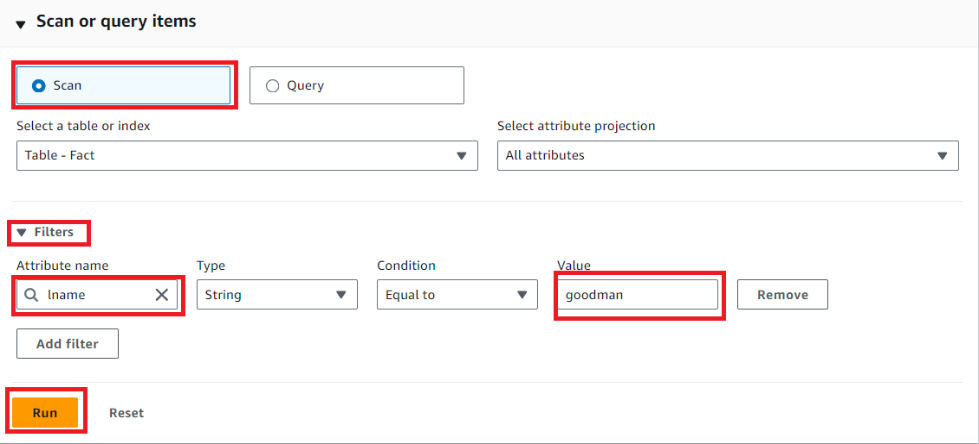
1. In the create item window, delete the current content and copy and paste the first JSON Fact record found below. Toggle the “View DynamoDB JSON” switch on and off again to format the text as an indented JSON. Note the key “display” and the value “Jeff Owens”. You should see how this matches up to the key defined in the DynamoDB table. Click on the “Create item” button.

JSON Fact Records:

{"display": "Jeff Owens", "fname": "jeff", "lname": "owens", "type": "student", "age": 23, "checkouts" : [{"id": "91043", "year": 2017, "month": 4, "day": 29, "book": "5244", "title": "Cloud-based Mobile Applications", "pubyear": 2015, "subject": "cloud"}]}

{"display": "Homer Goodman", "fname": "homer", "lname": "goodman", "type": "student", "age": 23, "checkouts" : [{"id": "91040", "year": 2017, "month": 4, "day": 28, "book": "5237", "title": "Mastering the database environment", "pubyear": 2015, "subject": "database"}, {"id": "91066", "year": 2017, "month": 5, "day": 19, "book": "5242", "title": "C# in Middleware Deployment", "pubyear": 2015, "subject": "middleware"}]}

1. Click the “Create item” button and add the second JSON record (found above) to the database.
2. You should now see two items in the “Items returned” section of the Fact table.
3. Using the Scan/Query tool, Scan for all items where the lname = goodman
   1. Select the “Scan” radio button.
   2. Expand the filter and add lname for an attribute, String for a type, = for the operator and goodman for the value.
   3. Click “Run”. You should have one item returned from the scan.



Explore other Scan options to see how the search tools work. You may notice that if you switch to Query, you must specify the display attribute first. The Query feature in DynamoDB will only search for a specific table item key value, while Scan will search for all items.

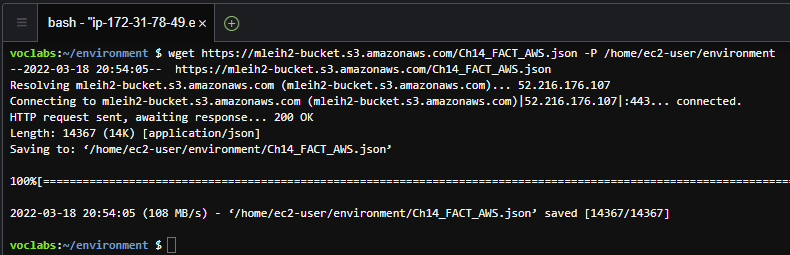
1. **(3pts) Reset the Scan properties and rerun the scan so both items appear in the “Items returned” section. Take a screenshot of the Items tab showing the two items that were loaded into the Fact table and paste the image into the Chapter 14 Response document under Image 1 – two items.**

Part 2 – Mass loading a DynamoDB table from Cloud9 using a file uploaded to an S3 Bucket.

1. Download the Ch14\_FACT\_AWS.json file from this assignment in Canvas to your local drive.
   1. Open the file in a text editor (do not change anything) and explore the JSON string in the file. It contains the name of the Fact table to load items and a series of PutRequest commands to complete a mass load of the Fact table with the remaining display items.
   2. Close the file without changing it.
2. In your AWS Learner lab environment, go to the S3 services and create a new Bucket.
   1. Click on the “Create bucket” button on the top right-hand corner of the screen.
   2. Give the S3 Bucket a unique name.
   3. Select “ACLs enabled” in the Object Ownership section
   4. Uncheck the “Block all public access” in the Block Public Access settings for this bucket section.
   5. Check the security-warning acknowledgement.
   6. Click Create bucket.
3. Click on the bucket name just created and click on the “Upload” button.
4. Click the “Add files” button and select the Ch14\_FACT\_AWS.json file from your local drive.
5. Expand the “Permissions” section and select “Grant public-read access”. Check the security warning acknowledgment.
6. Click the “Upload” button.
7. Go to the Cloud9 service and click the “Create environment” button. Give the Environment a name; under the Network setting section, select “Secure Shell (SSH),” leave everything else to default, and click on the “Create” button.
8. Once the environment is fully started, click on the “Open” link of the new Cloud9 environment.
9. In the Cloud9 environment, close the Welcome window.
10. From the Windows menu, open a new Terminal session.
11. Here, we will use the “wget” command to copy the data from our S3 bucket to a file in our Cloud9 environment. On the Bash command line, enter the command to get the Ch14\_FACT\_AWS.json file from the S3 bucket and copy it to the /home/ec2-user/environment directory. Use the command below and replace the <bucket name> with the name of your S3 bucket.

**wget https://<bucket name>.s3.amazonaws.com/Ch14\_FACT\_AWS.json -P /home/ec2-user/environment**

A successful look will look like the image below:



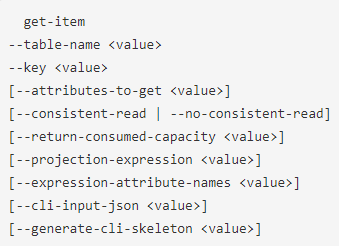
1. On the Bash command line, enter the command to call the batch-write-item DynamoDB API to mass load the Fact table with the remaining items in the Ch14\_FACT\_AWS.json file. If the load runs correctly, you should receive a response showing no unprocessed items.

**aws dynamodb batch-write-item --request-items file://Ch14\_FACT\_AWS.json**

1. Click on the Cloud9 icon in the top left-hand of the screen (just left of the File menu) and select “Go To Your Dashboard.” This will leave your Cloud9 environment open on one browser tab and create a second tab to open other AWS services.
2. Return to the DynamoDB service, select “Tables” from the left-hand menu, click on the Fact Table, and click on the “Explore table items” button to view the new items that were loaded into the Fact Table. There should be 27 items listed.
3. **(3pts) Take a screenshot of Items returned section showing the first 10 or more items and paste the image into the Chapter 14 Response document under Image 2 – 27 items.**

Part 3 – Using the DynamoDB get-item and scan API command calls to query the Fact table.

1. Return to your Clound9 environment.
2. The get-item API command will return all the items in the database table that match the key passed into the call. For the Fact database, it will return just one item as each item in the Fact database has a unique “display” value. The get-item command has the following parameters, where the --table-name and --key are required and the remaining parameters are optional.



Enter the following get-item API command to return the Cedric Baldwin item and projecting just the display, fname and lname attributes.

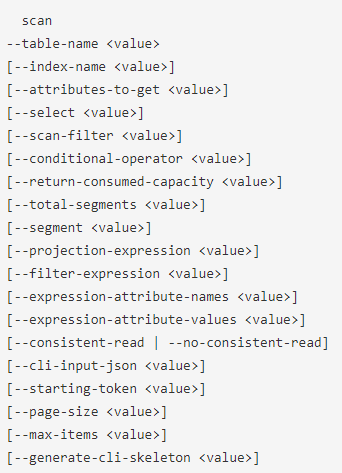
**aws dynamodb get-item --table-name Fact --key '{ "display": { "S": "Cedric Baldwin" } }' --projection-expression "display, fname, lname"**

The get-item API command is very fast as it uses the hash index to find and return the information, only accessing the specific item key. If you remove the --project-expression parameter, the call will return all the attributes.

1. **(1pt) Copy and paste an image to the response document under “Image 3 – get-item” that includes both the get-item API command used above and the correct JSON result string.**
2. Using what you learned so far about the get-item API command, create a get-item API command that will return the Marina King item from the Fact DynamoDB and project the display and age attributes.
3. **(3pts) Copy and paste an image to the response document under “Image 4 – Marina King” of the command along with an image of the result.**
4. The checkouts attribute is an array of sub-attributes containing information about the specific books a person has checked out. To access that information, we must use the array index and sub-attribute name via dot notation. For example, to return the display attribute and the book title of the first book checked out by Thomas Duran, we would use the following get-item API command. Note the checkouts attribute as the 0 index (the first item in the array), followed by the dot notation and sub-attribute name of title.

**aws dynamodb get-item --table-name Fact --key '{ "display": { "S": "Thomas Duran" } }' --projection-expression "display, checkouts[0].title"**

1. **(1pt) Copy and paste an image to the response document under “Image 5 – Thomas Duran” that includes both the get-item API command used above and the correct JSON result string.**
2. Using what you learned so far about the get-item API command and sub-attributes, create a get-item API command that will return the two books checked out by Betsy Malone (including the display attribute) from the Fact DynamoDB.
3. **(3pts) Copy and paste an image to the response document under “Image 6 – Betsy Malone” of the command, along with an image of the result.**
4. The scan API command will scan all items in a table and return any matching items via a JSON string. The advantage of using the scan command is that it can be used to search all non-key attributes. However, because the scan searches all items, this can be a lengthy command (and costly command) to execute for large tables. It is preferable to use the query or get-item command if they item key attribute is known. The scan API command uses to the following parameters.



Note: Only the --table-name parameter is required, so to return all items and all attributes from a table, the command should just include the --table-name. Execute the following command to test this. You should have 27 items returned. (Press the space bar to advance a page and “q” to exit)

**aws dynamodb scan --table-name Fact**

1. **(1pt) Copy and paste an image of the last several records of the end of the return result showing that 27 records were returned in the ScannedCount to the response document number “Image 7 – scan”.**
2. To return all the items from the table who are students and over the age of 30, projecting only the display attribute, the scan API command would be like the following.

**aws dynamodb scan --table-name Fact --projection-expression "display" --filter-expression '#t = :type and age > :age' --expression-attribute-values '{":type":{"S":"student"}, ":age":{"N":"30"}}' --expression-attribute-names '{"#t":"type"}'**

The --filter-expression parameter is used to build an expression with variables (names prefaced with a colon). The --expression-attribute-values assigns values to those variables. And the --expression-attribute-names parameter is needed to give the attribute “type” an alias because “type” is also a keyword and cannot be used in the API call.

1. **(1pt) Copy and paste an image of API command to the response document under Image 8 – over 30.**
2. To return the fname and lname attributes of all the Items in the table where the person checked out a book with the subject of database, the scan API command would look like the following:

**aws dynamodb scan --table-name Fact --projection-expression "fname, lname" --filter-expression 'contains(checkouts[0].subject, :sub) or contains(checkouts[1].subject, :sub) or contains(checkouts[2].subject, :sub)' --expression-attribute-values '{":sub":{"S":"database"}}'**

Note that the --filter-expression parameter needed to include a contains clause, which is used to search a sub-attribute. There are three contains clauses to search all three array elements of the checkout attribute (the maximum number of any item in the Fact table).

1. **(1pt) Run the command, scroll back up to where the command was entered, and paste an image of the command and the first part of the result to the response document under Image 9 - database.**
2. Now that you have worked a little with DynamoDB API commands, write a command that will return the display attribute and age attribute of all items in the Fact table where the type attribute is “student” and the first booked they checked out had the subject of “programming” (only consider the checkout[0] element of the item.
3. **(3pts) Run the command, scroll back up to where the command was entered, and paste an image of the command and the first part of the result to the response document under Image 10 - programming.**

Explore AWS and AWS commands as much as you like. Once complete, sign out of the AWS environment and end the lab.