Justin Casteel

February 16, 2018

SDEV 400 6380 Secure Programming in the Cloud (2182)

Homework 3 – SQL and NoSQL

Step 1.

When using DynamoDB, unlike SQL, we do not need to provide all the attributes that we would like to use. In fact, when trying to add more attributes than you have keys, you will receive an error when running your batch file from the AWS CLI. So, what I did when creating the ‘sensors’ table in DynamoDB is I chose a HASH and RANGE key I thought would work appropriately for this schema. In this case I chose “Sensor” as the HASH key and “Filepath\_ID” as the RANGE key. I thought that these would work the best and since “Filepath\_ID” is an integer, so we will be able to query and find any data we insert easily and quickly.

Figure 1 below shows the create table command I used to create the sensors table using the AWS CLI.

Figure 1 - Create sensors Table

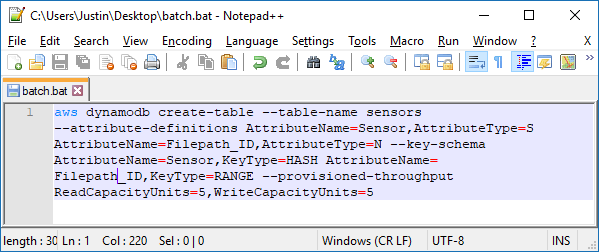
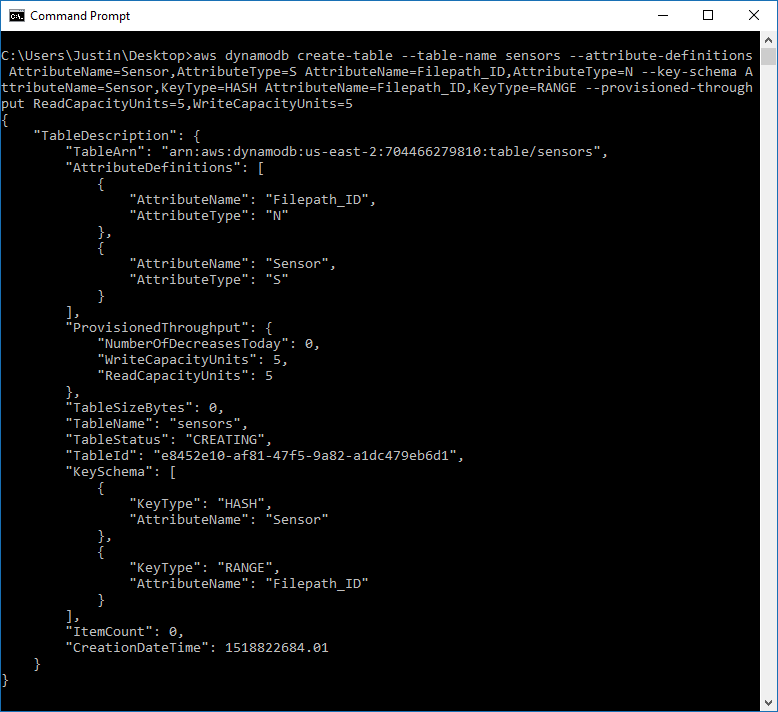
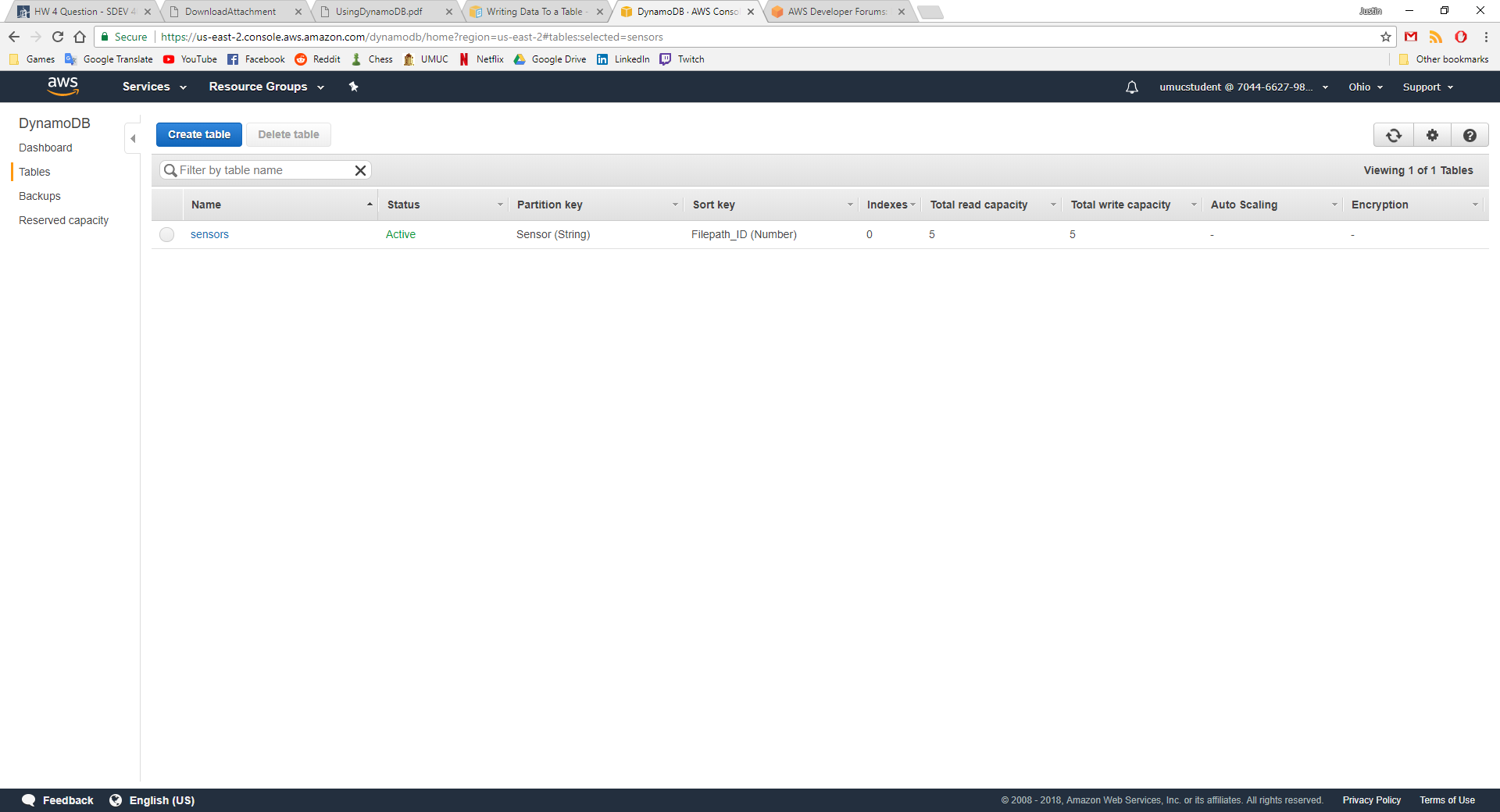


Figure 2 - Create sensor Table



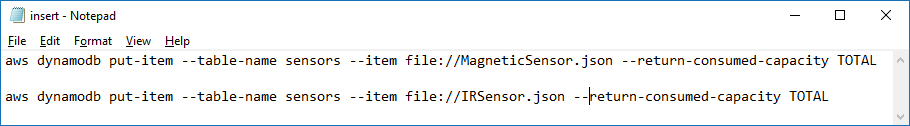
Now that I have created the sensors table through the AWS CLI, I will show that it has been created in the AWS DynamoDB interface as well.

Figure 3 - DynamoDB tables interface



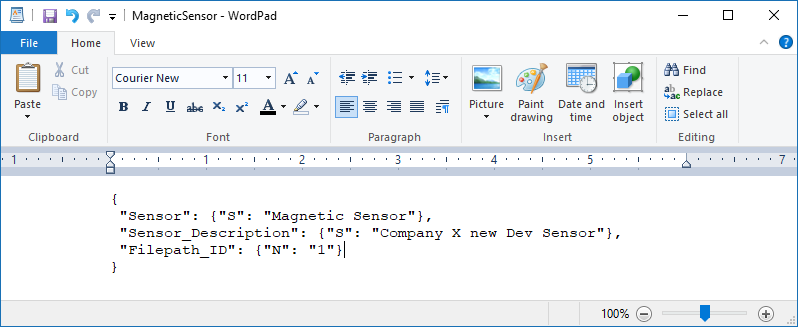
I then create two JSON files that will insert data into my newly created sensors table. The data that I will insert first is “Magnetic Sensor” using the first line of code from the following AWS CLI commands shown in figure 4.

Figure 4 – Insert data AWS CLI Commands



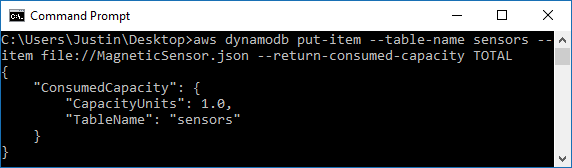
What these commands are showing is that it will insert data into the “sensors” table and the data that it will use will come from are the “MagneticSensor.json” and “IRSensor.json” files.

Figure 5 - MagneticSensor.json



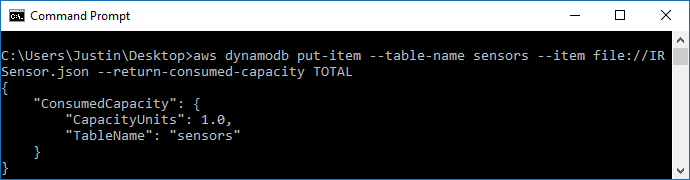
The first quotes specify the attribute the data will go in, the second quotes specify the type, in this case “S” for string and “N” for number since “Filepath\_ID” is an integer type. Figure 5 shows what the .json file contains.

Figure 6 - MagneticSensor.json Output



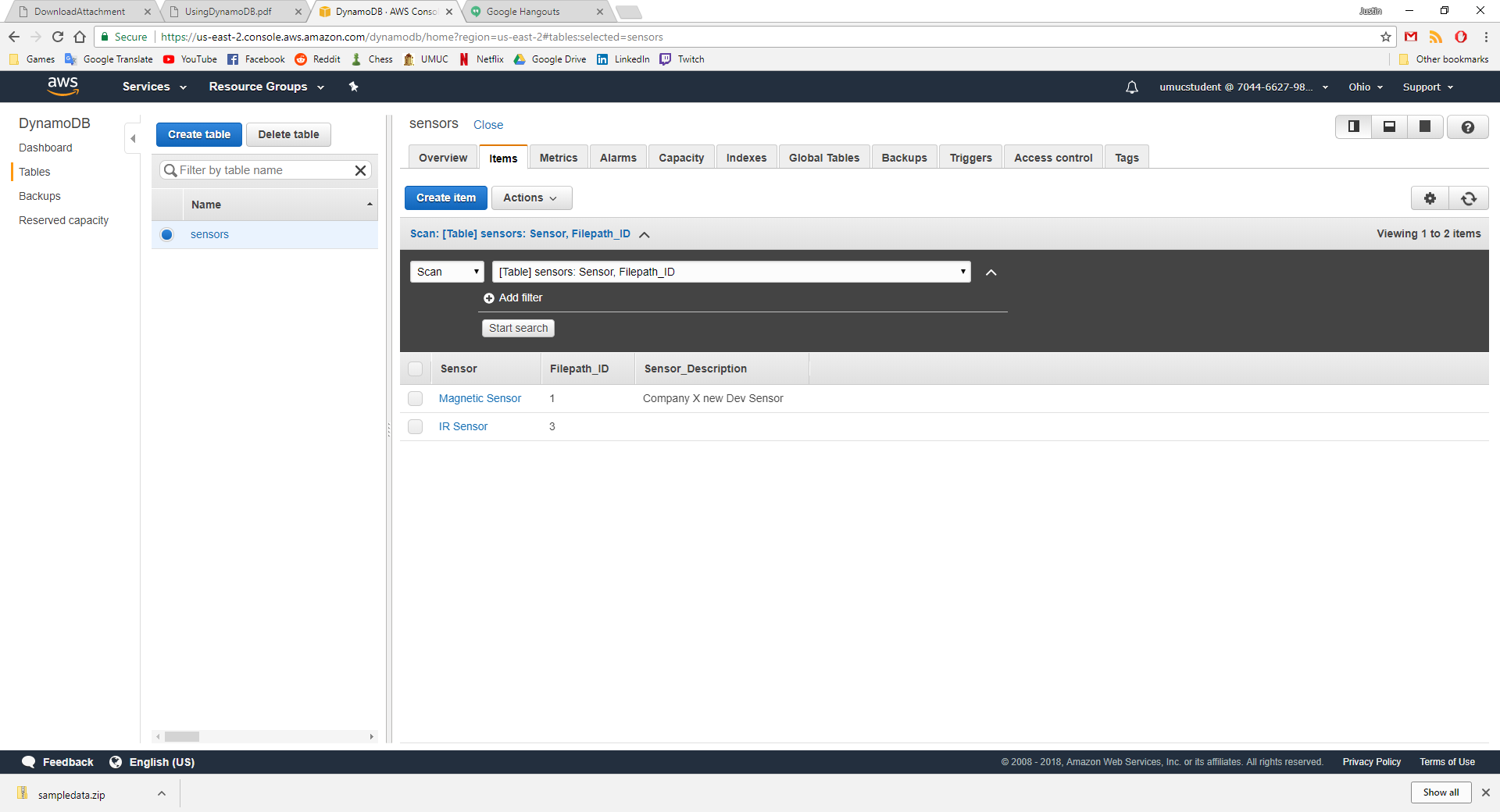
I then run the second command from the AWS CLI as well and receive the following output shown in figure 7.

Figure 7 - IRSensor.json Output



To check that these new data entries have been entered correctly into the table, we check in the AWS Management Console. We can see in figure 8 that under the “Items” tab it shows our newly created entries, “Magnetic Sensor” and “IR Sensor” with our provided attributes, “Filepath\_ID” and “Sensor\_Description”.

Figure - new entries

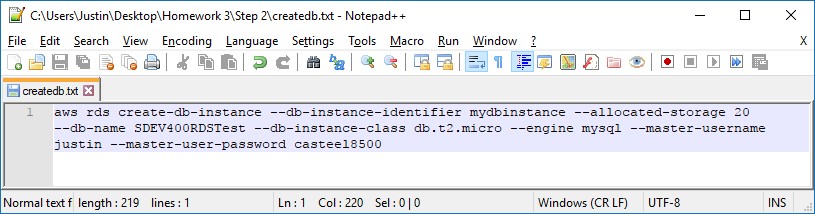


So, now I have successfully used the AWS CLI to create a table and I used JSON files along with CLI commands to insert new data into the table. The major difference in DynamoDB and a relational database such as SQL when it comes to creating tables is that in SQL you must define all the tables columns, data types, and the tables primary key. You can change this information after the fact, but they must still be provided when creating a table. In DynamoDB you only need to provide the attributes which are being used as keys, such as the HASH and RANGE keys for example. When inserting data into DynamoDB, you can create new attributes as you create items, and this saves a lot of time, so you don’t need to recreate tables to provide an attribute you might have missed or if you wanted to update and existing one. NoSQL databases such as DynamoDB use key-value pairs or document storage where an “SQL database requires a well-defined schema, where data is normalized into tables, rows, and columns. In an SQL database, all the relationships are defined among tables, columns, indexes, and other database elements” (Amazon, 2012). “DynamoDB is schemaless which means that every table must have a primary key to uniquely identify each data item, but there are no similar constraints on other non-key attributes. DynamoDB is also able to manage structures or semi-structured data, including JSON documents” (Amazon, 2012) as I have just shown above.

Step 2.

I first created a new database instance using the AWS CLI, shown in figure 9.

Figure 9 - CreateDB command



Figures 10 and 11 show the output when using the CLI command shown in figure 9.

Figure 10 - output1

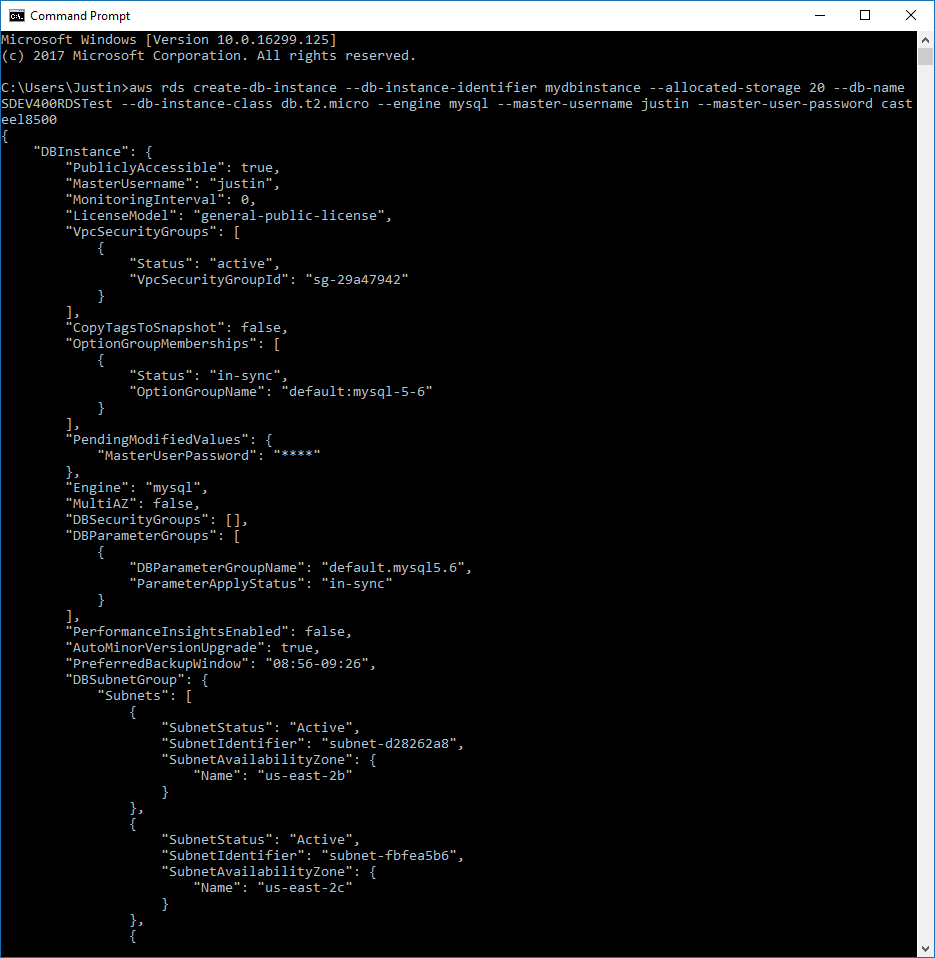
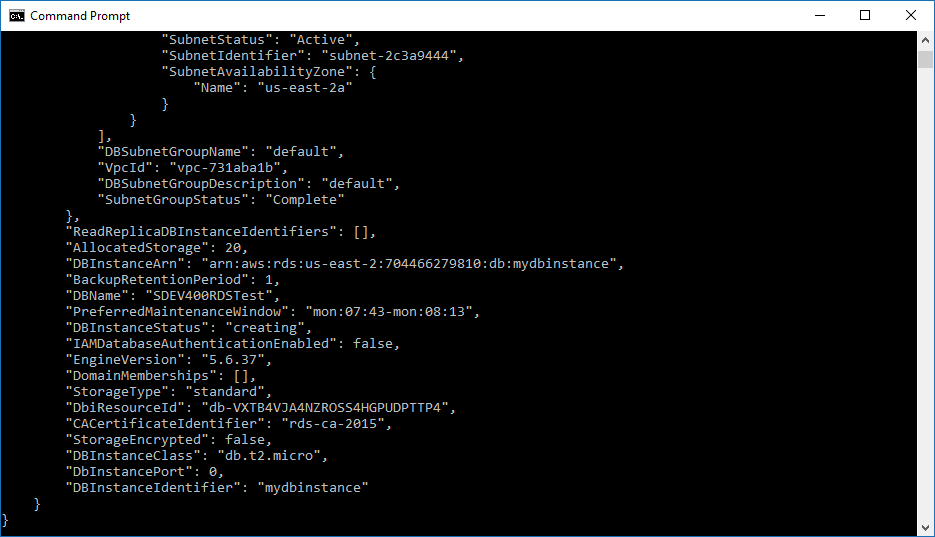


Figure 11 - output2



Now I use the describe command to show the db-instance that I just created.

Figure 12 - describedb.txt

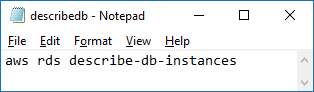


Figure 13 – JSON Response 1

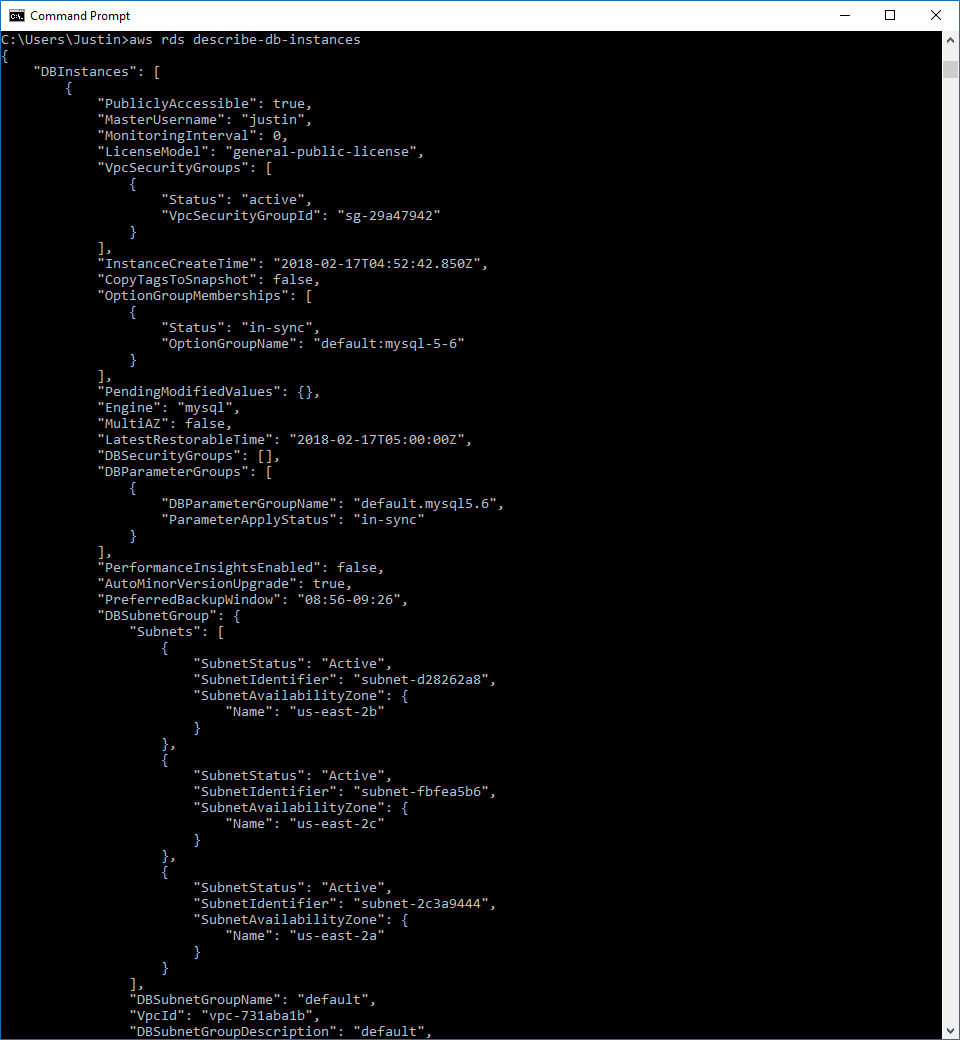
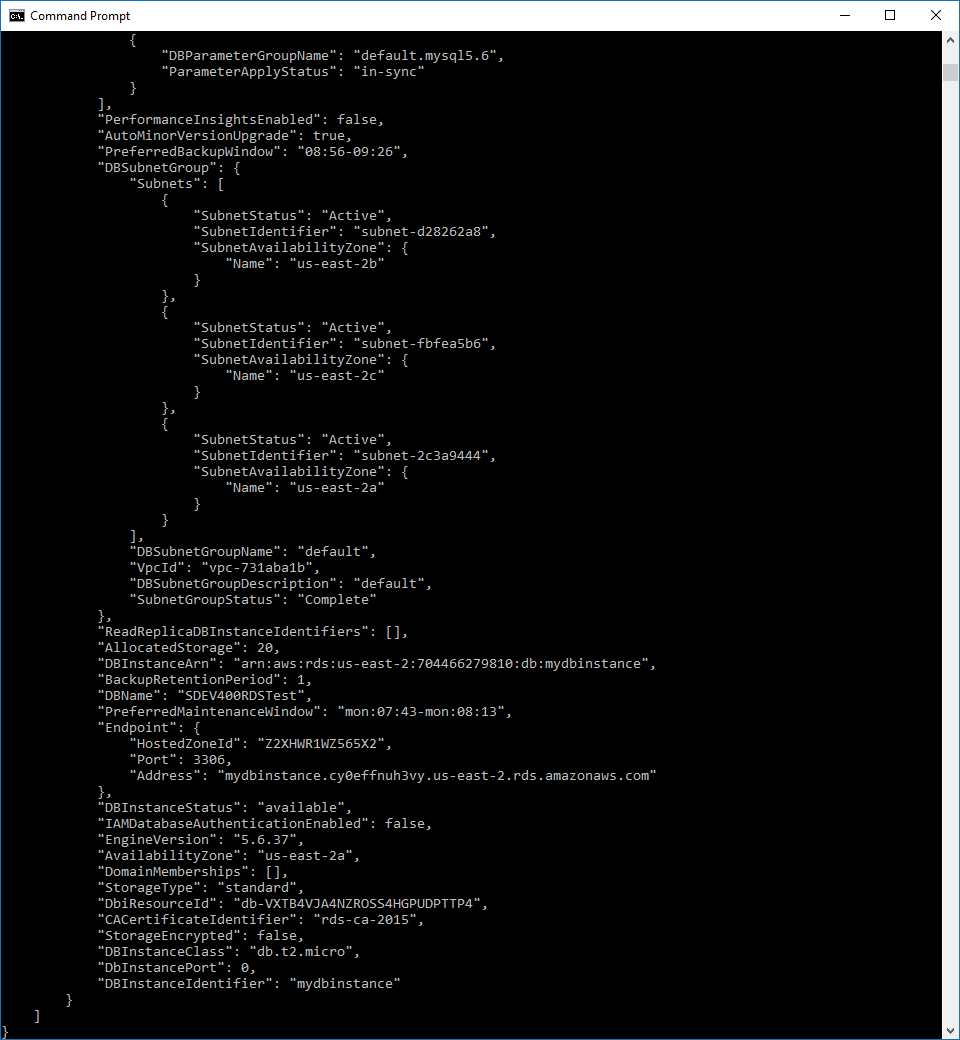
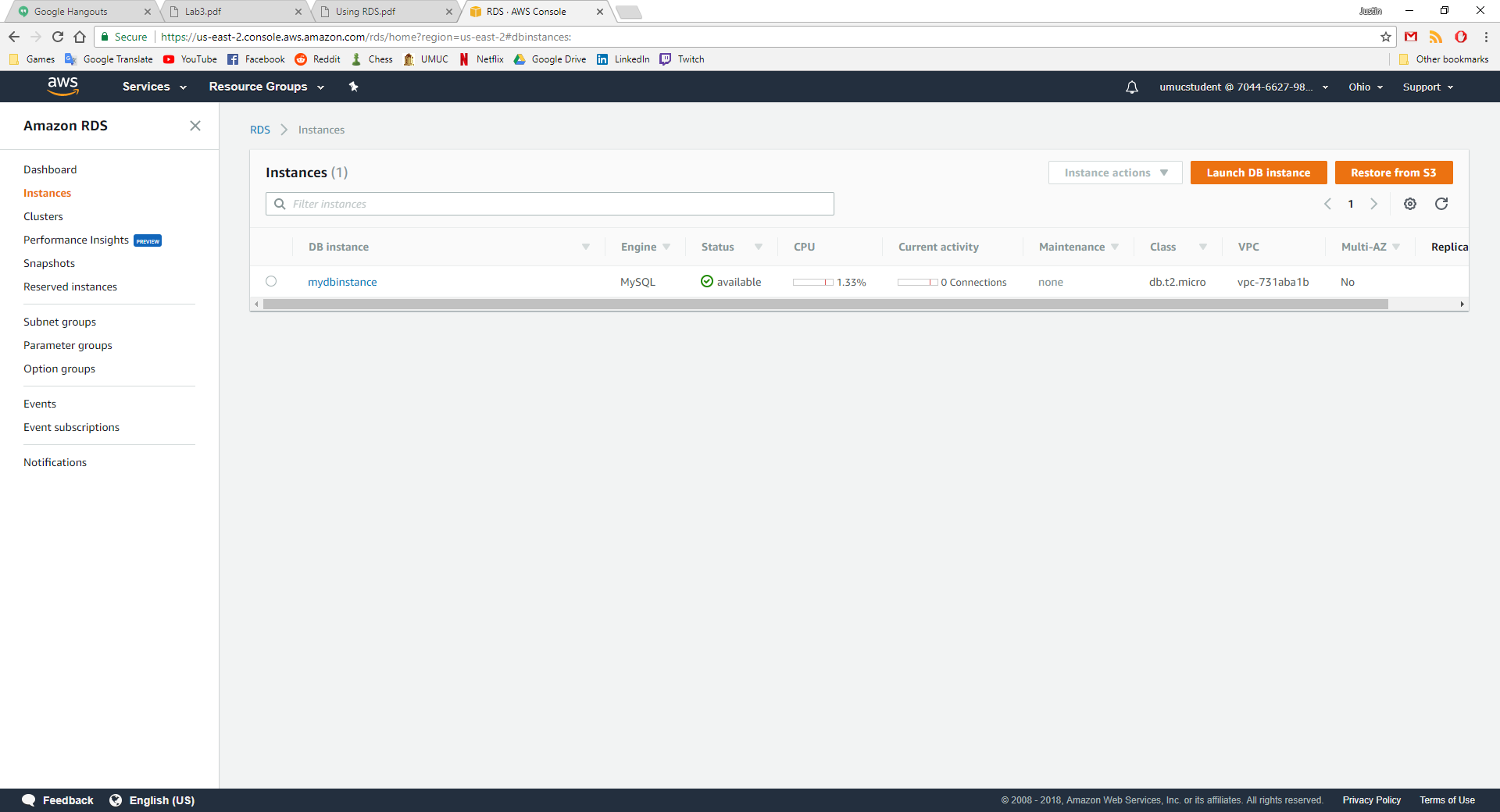


Figure 14 – JSON Response 2



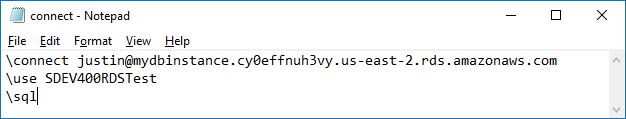
I then navigate to the Amazon RDS page to find that my instance is up and running with all the above parameters.

Figure 15 - AWS Management Console Instance Running



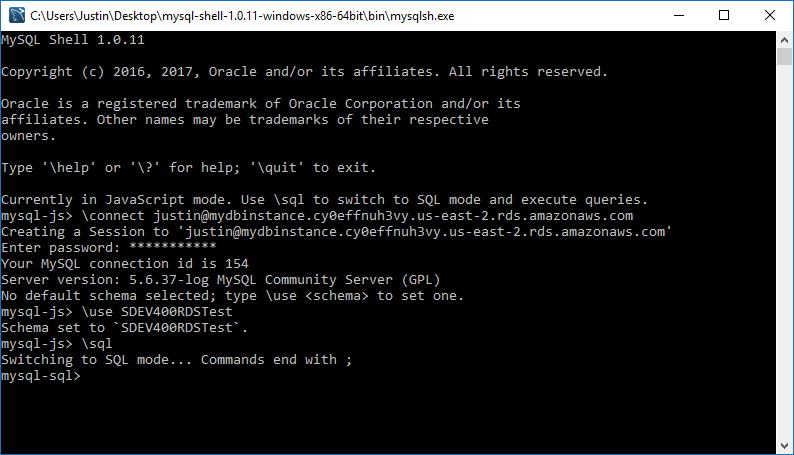
Now I connect to my newly created MySQL instance and change the default scheme and mode to sql by using the following commands:

Figure 16 - connect.txt



I connect to the instance by using MySQL Shell version 1.0.11 as shown in figure 17.

Figure 17 - MySQL Connection



I then begin to create my tables, using the following SQL code shown in figure 18.

Figure 18 - tables.txt

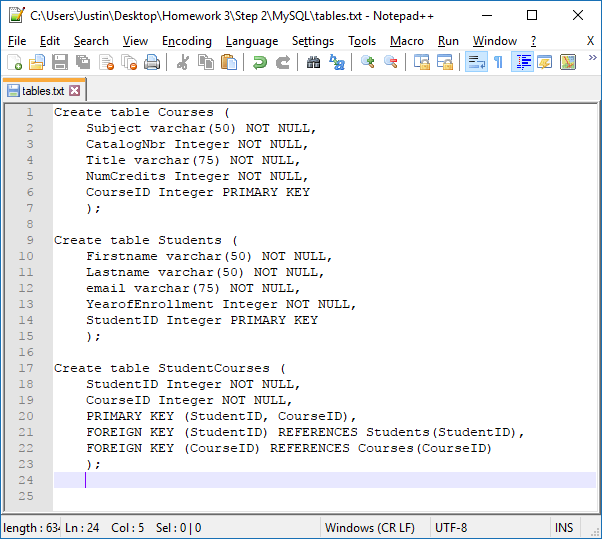
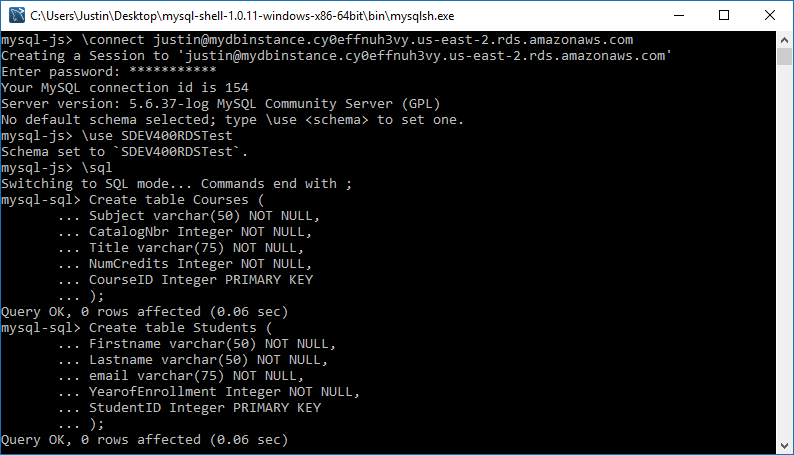


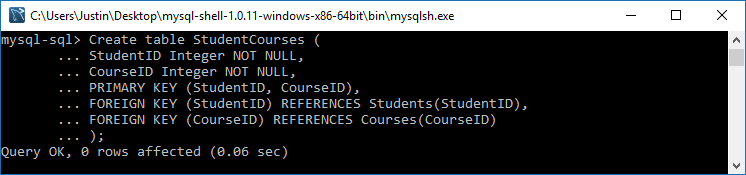
Figure 18 shows the successful creation of the Courses and Students tables.

Figure 19 - Courses and Students

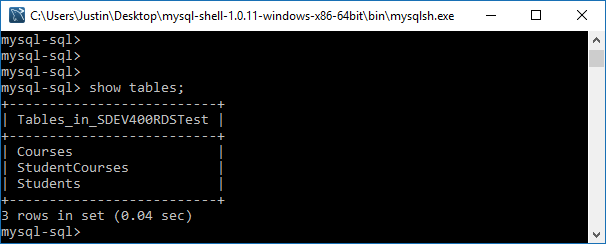


My final table, StudentCourses is created as well.

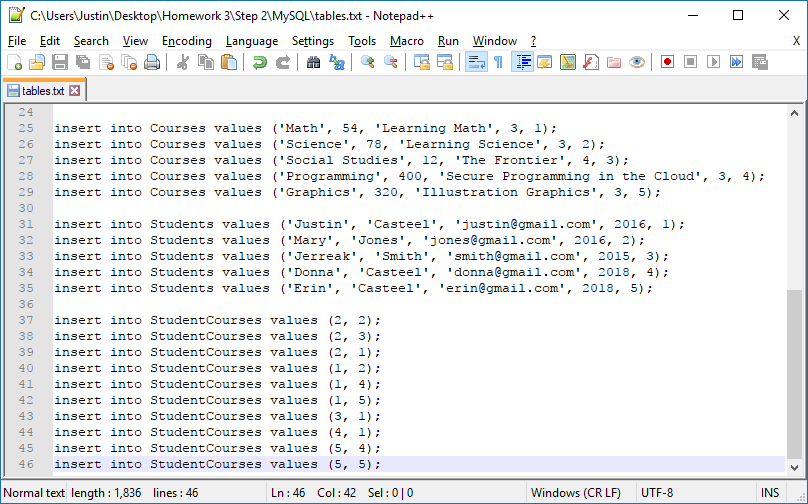
Figure 20 - StudentCourses table



Then I use the “show tables;” command to show the 3 tables that I have just created.

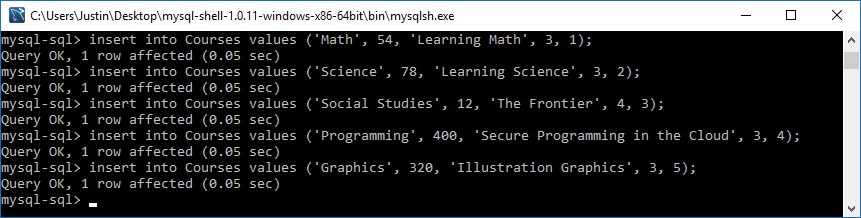


I will now insert data into each of the three tables I have just created by using the following commands:



First, I insert 5 rows of data into the Courses table.

Figure 21 - Courses table values



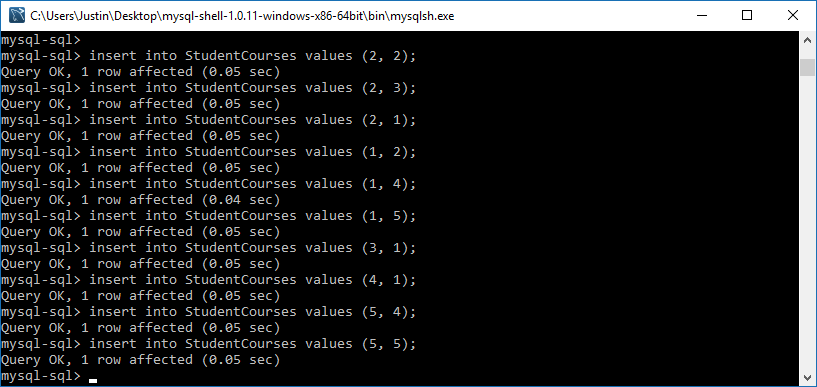
Then I insert 5 rows of data into the Students table.

Figure 22 - Students table values



And finally, I insert 10 rows of data into the StudentCourses table.

Figure 23 - StudentCourses table values



Now I check to see that the rows were inserted correct and into the proper tables. I use select statements to do this step.

select \* from Courses;

select \* from Students;

select \* from StudentCourses;

Figure 24 - Courses data using select statement

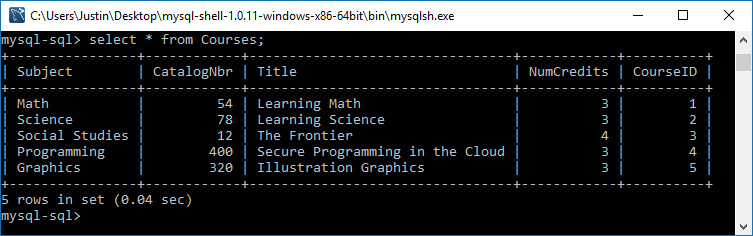


Figure 25 - Students data using select statement

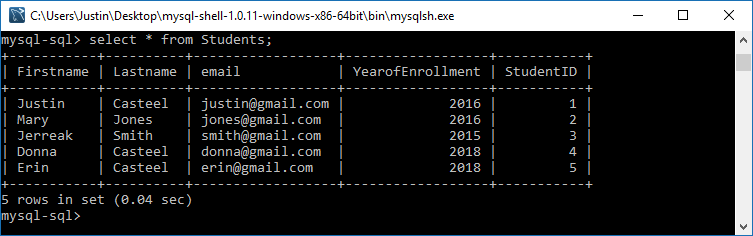
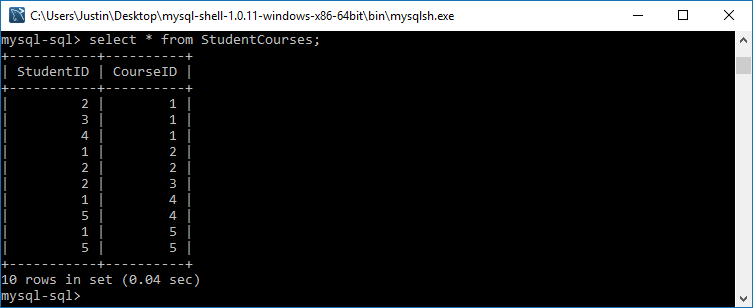
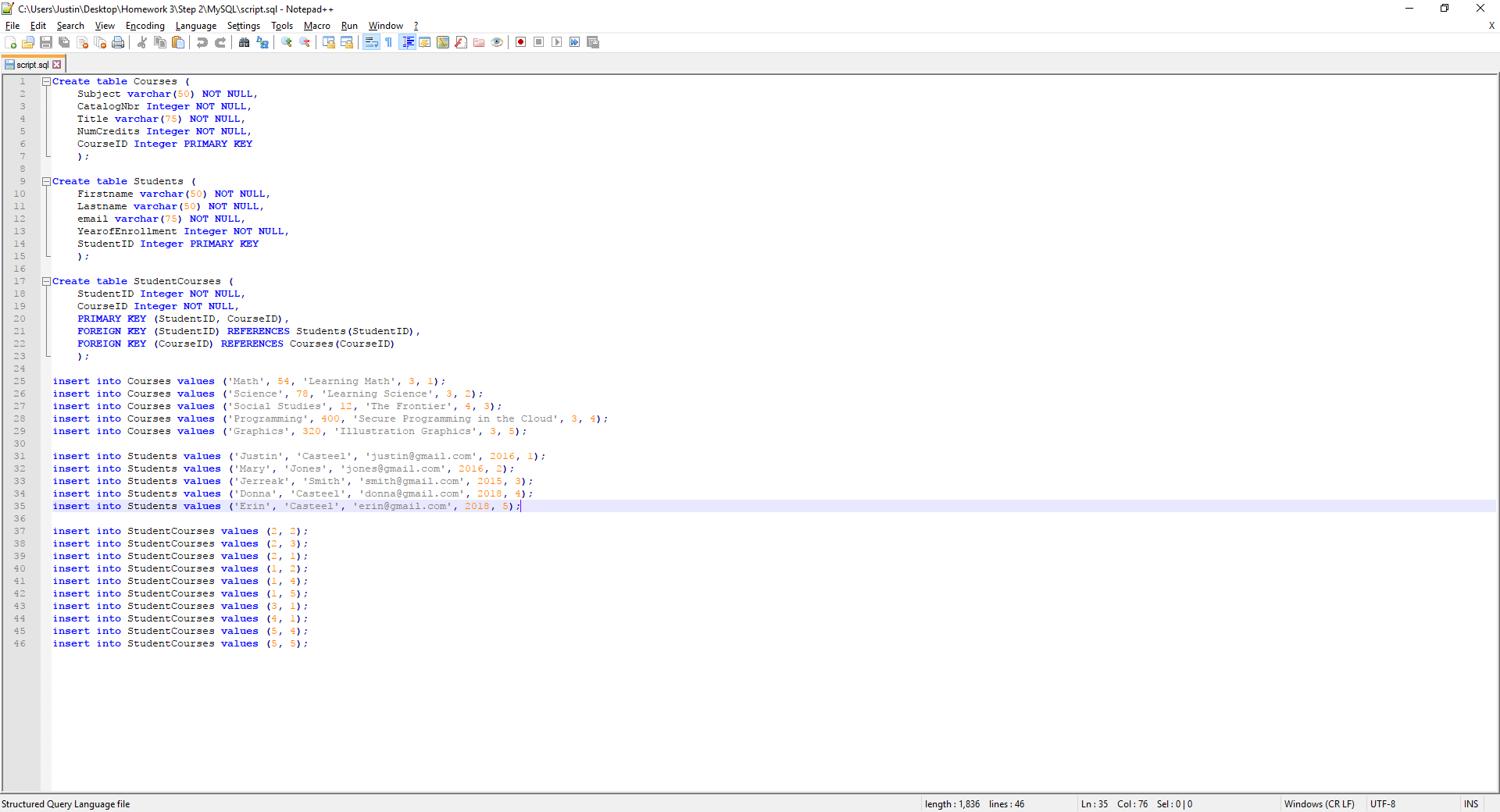


Figure 26 - StudentCourses data using select statement



Here is my script file with all the above commands.

Figure 27 - script.sql



To execute the SQL script file, I use the following command:

\source C:\Users\Justin\Desktop\script.sql

You simply change the directory to wherever your .sql file is located on your computer.

You can see in figure 28 that if I try to run this script, we will receive duplicate entry and already exists errors since all these tables are already created.

Figure 28 - script.sql execute

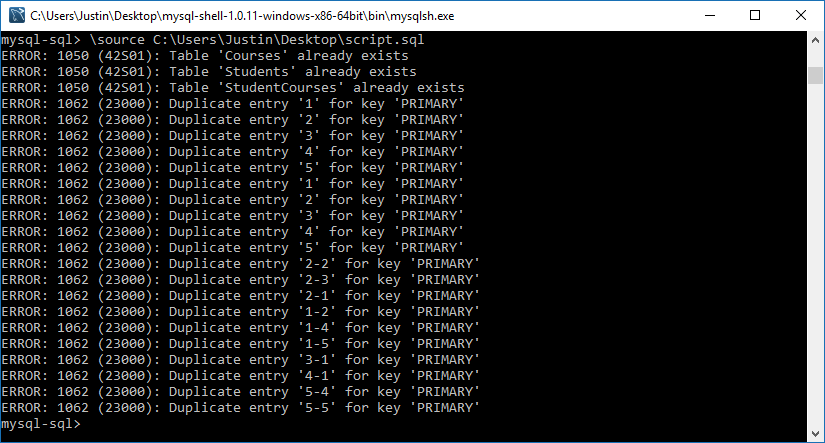
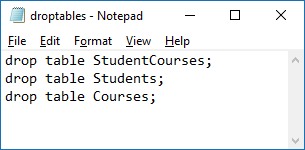


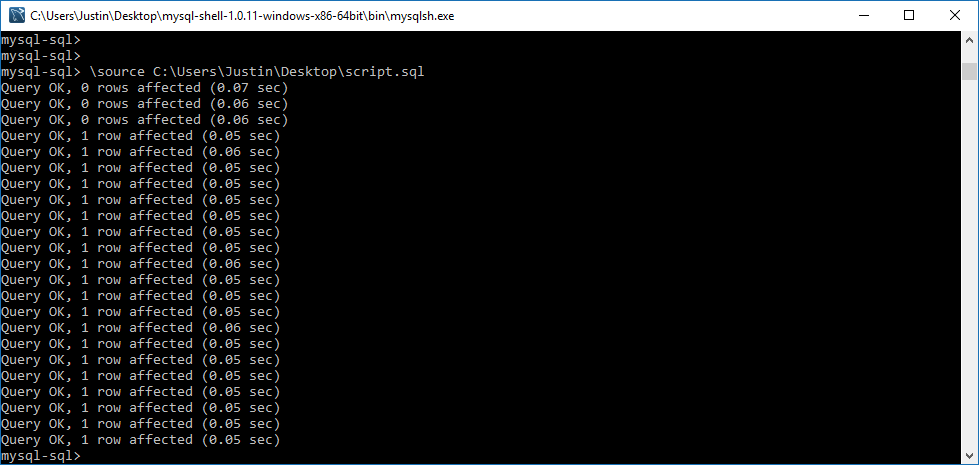
Figure 29 – deletetables.txt



I delete all the tables using the drop table command and then try to run the script using:

\source FILELOCATION.

Figure 30 - script



Step 3.

The first step is to re-create the tables using the AWS CLI interface. I use the following commands to do so. This will create the Courses, Students and StudentCourses tables with the appropriate HASH and RANGE keys. DynamoDB doesn’t support composite keys so for the StudentCourses table I used StudentID as the HASH key and CourseID as the RANGE key.

Figure 31 - AWS CLI table commands

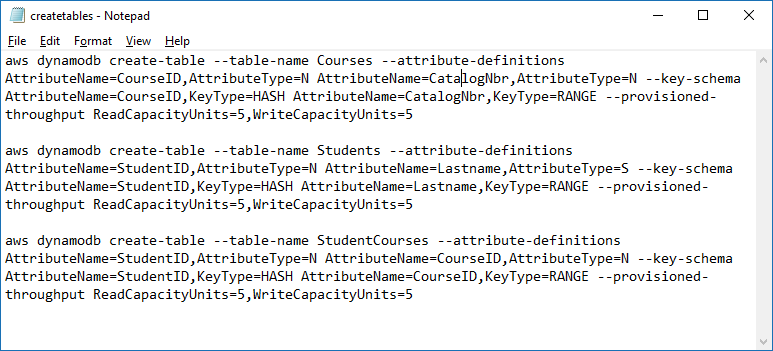


Figure 32 - Course table created

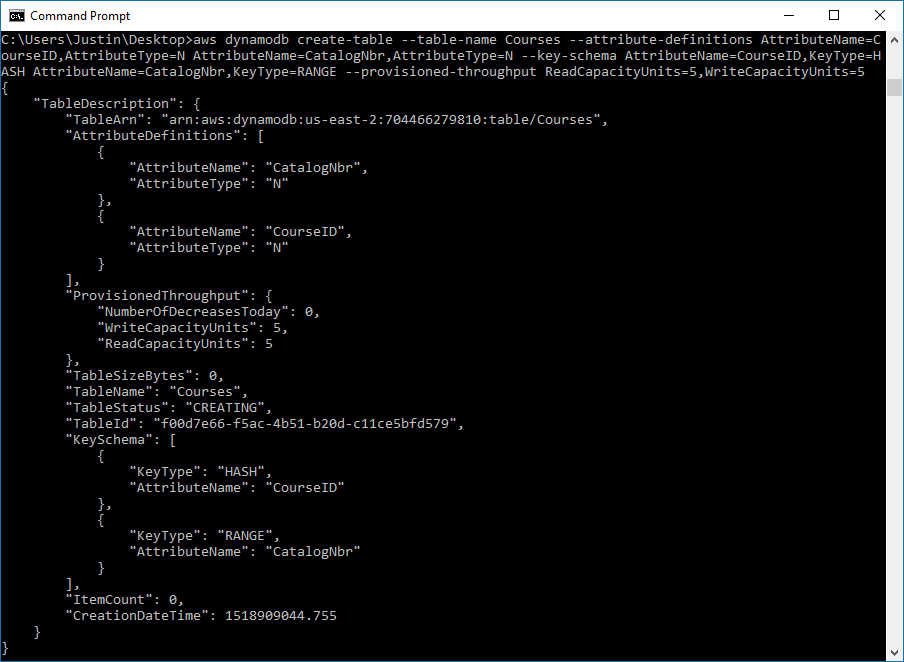


Figure 33 - Students table created

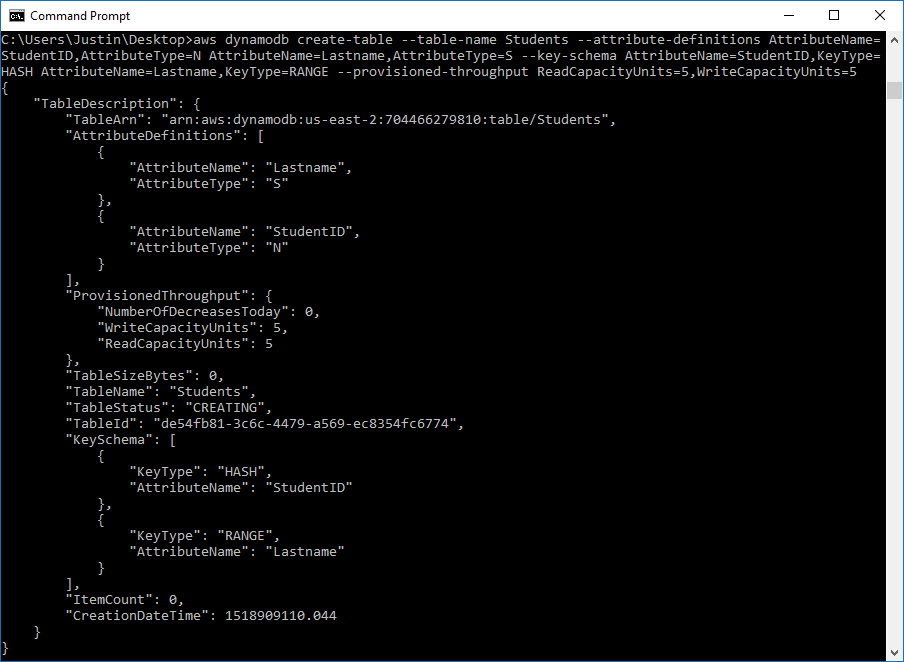
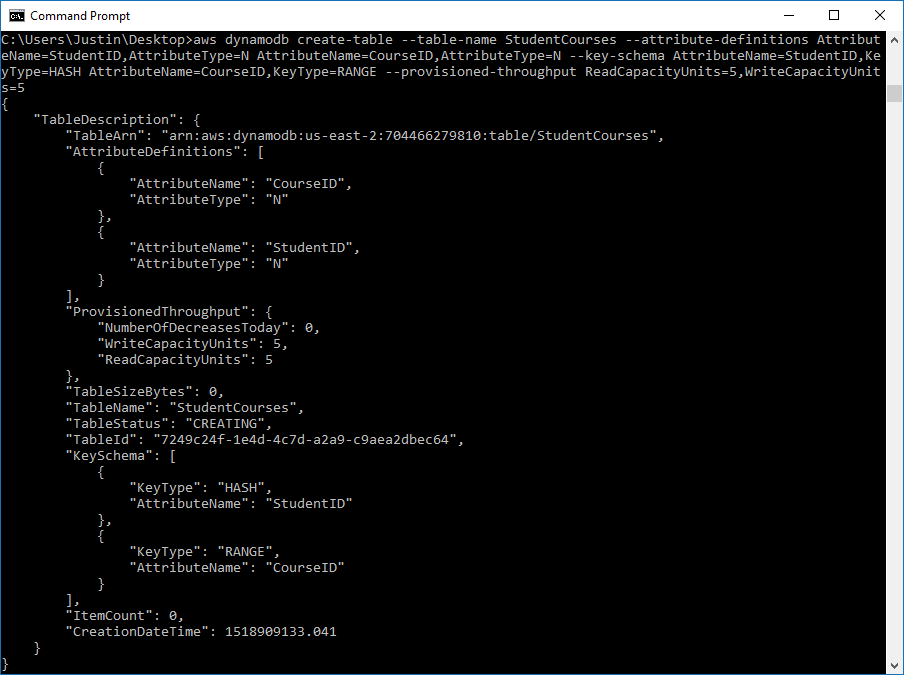
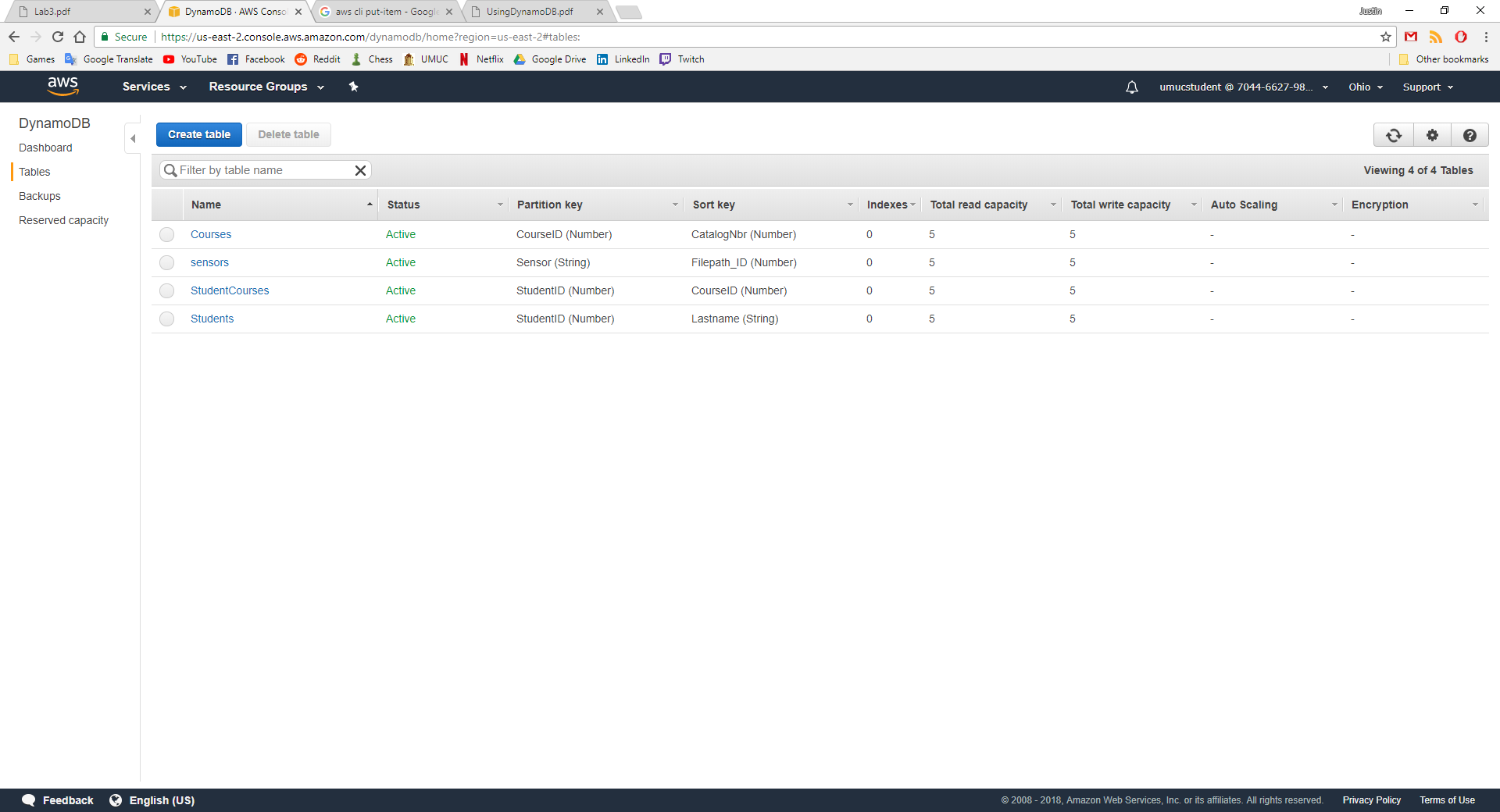


Figure 34 - StudentCourses table created



I check the AWS Interface to make sure that the new tables are created.

Figure 35 - New tables - Courses, StudentCourses, and Students



I was unable to find a way to insert of all the records from one single JSON file, so I did them all separately labeled 1-5, shown below.

Figure 36 - 1.json

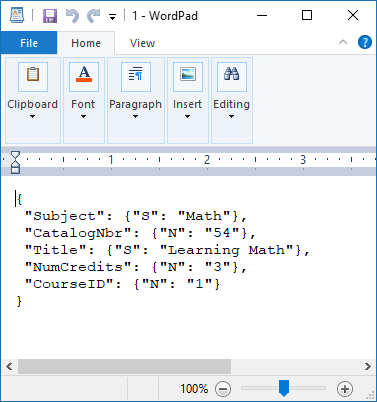


Figure 37 - 2.json

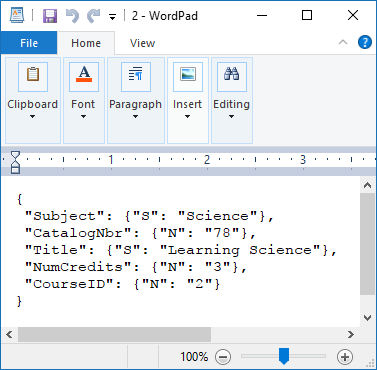


Figure 38 - 3.json

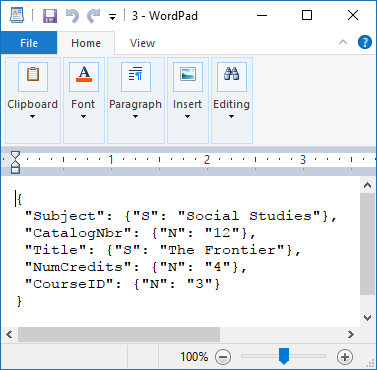


Figure 39 - 4.json

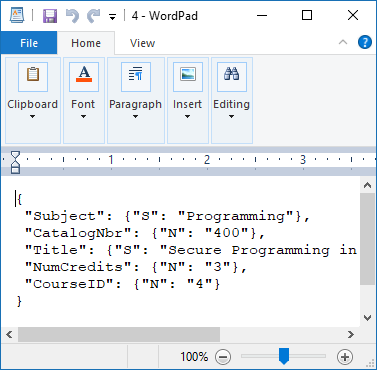
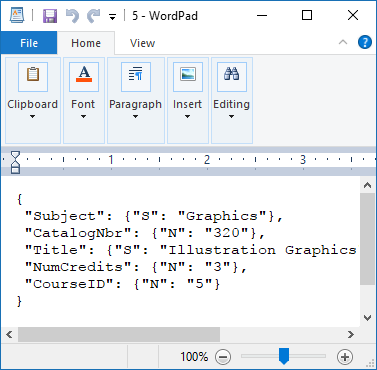


Figure 40 - 5.json



I then use the command to read the JSON contents from each individual file using the commands in figure 41.

Figure 41 - commands to read info from JSON files

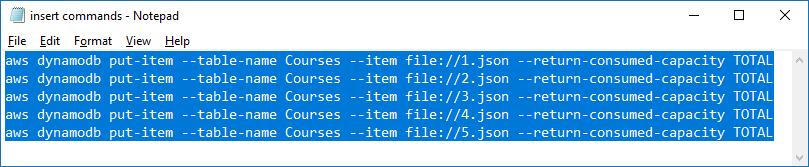
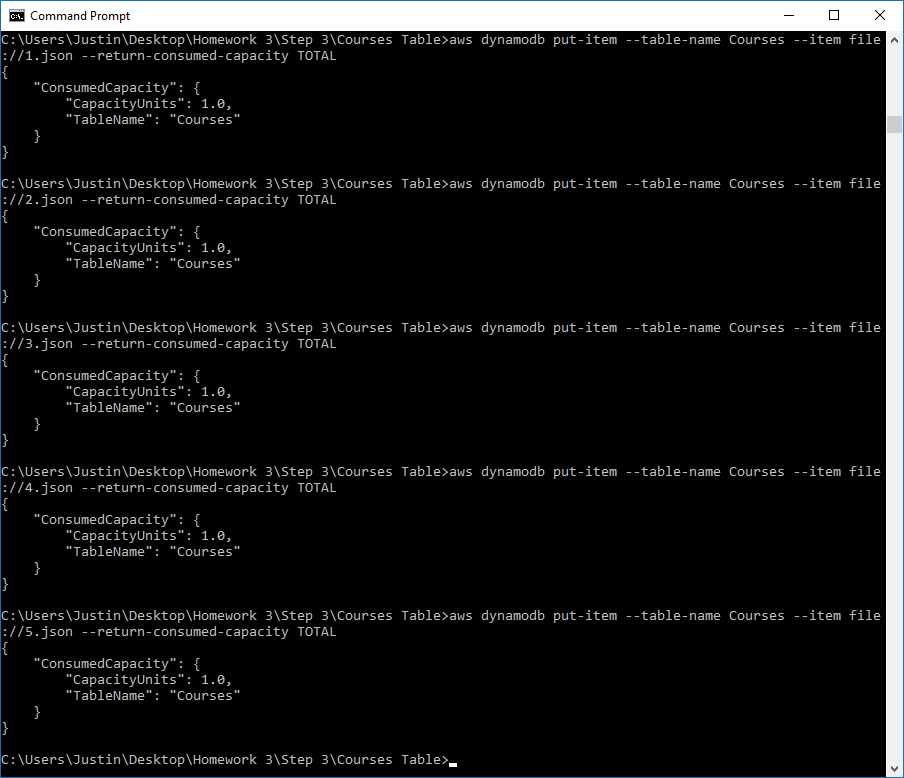
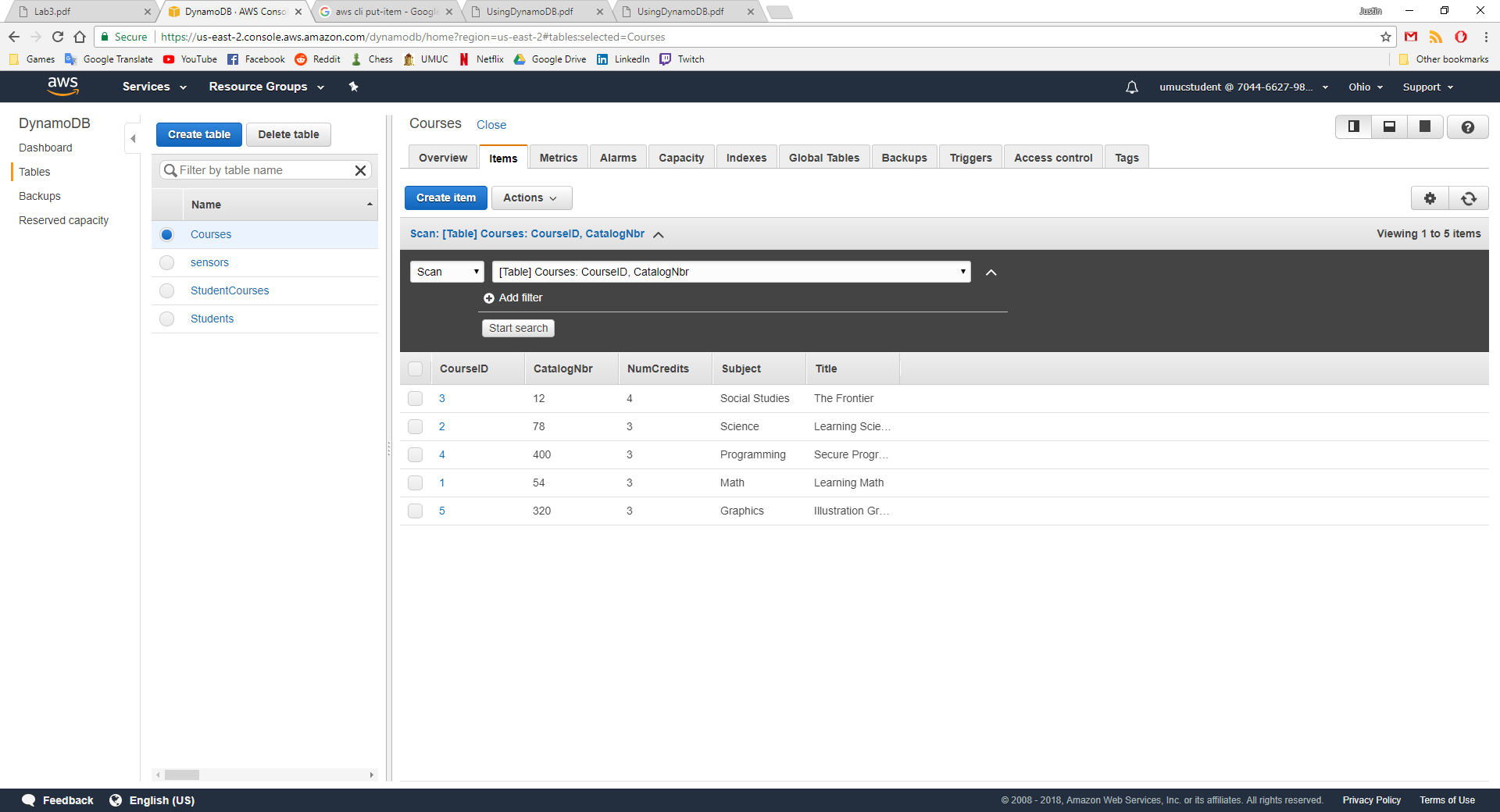


Figure 42 - Insert complete into Courses table



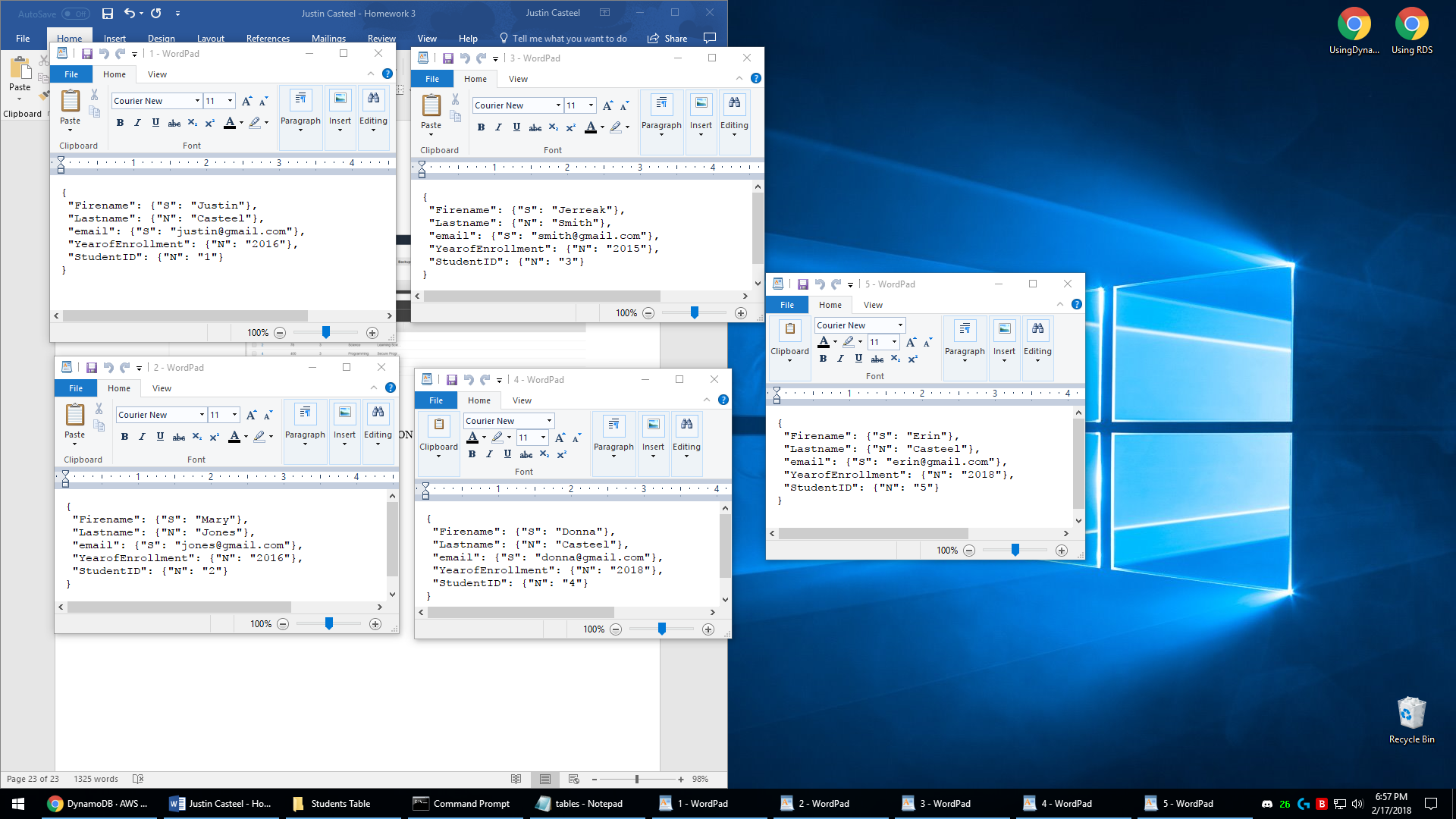
Below shows the newly added data into the Courses table.

Figure 43 - New data into Courses table



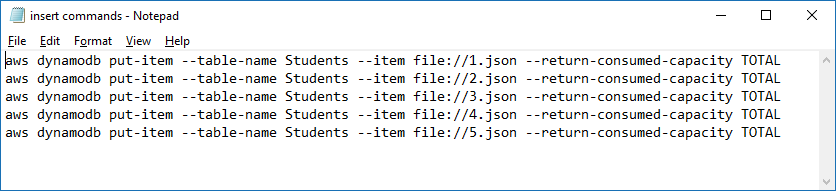
I repeat these steps for the Students table by creating 5 new JSON files and then inserting them using the AWS CLI.

Figure 44 - Student JSON files

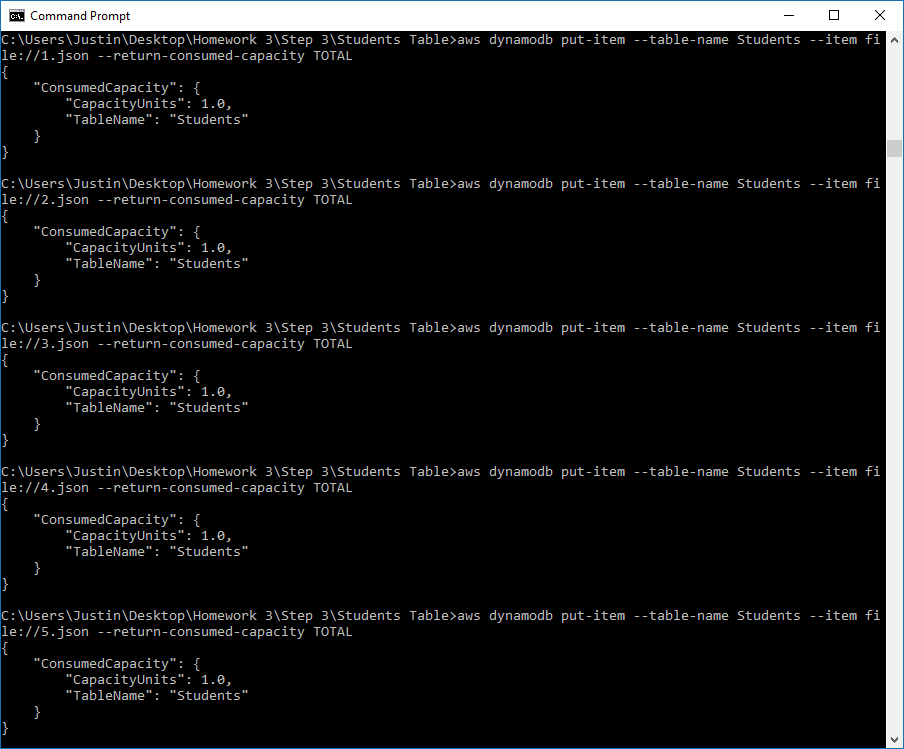


I use the following insert commands to tell the AWS CLI to read the JSON file and insert the data into the specified table.

Figure 45 - put-item commands for Students Table

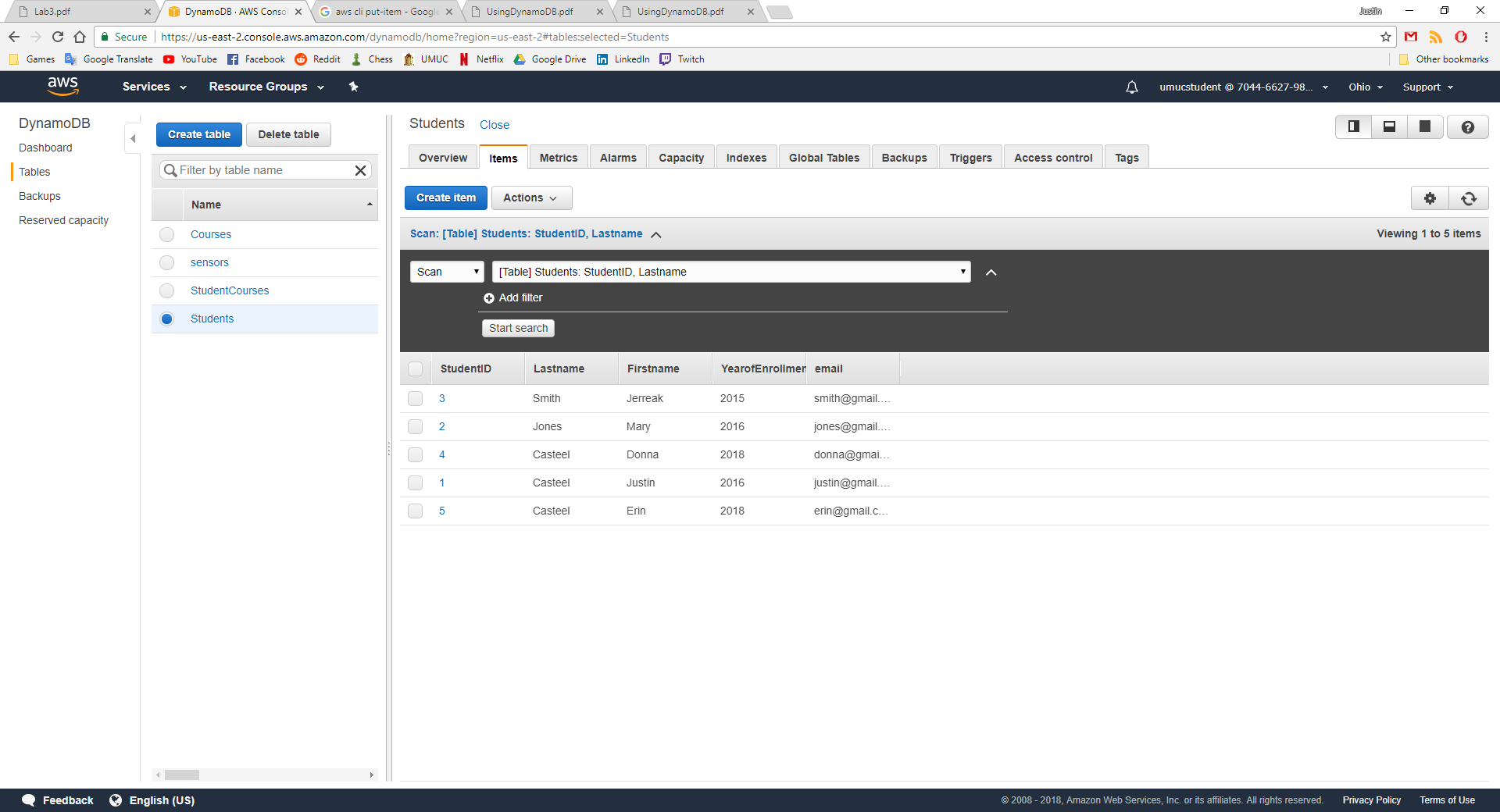


We receive the following output from the AWS CLI:



I check the AWS Interface to see that my data has been entered the Students table.

Figure 46 - Students table data



Next, I create 10 JSON files for the StudentCourses table for data to be inserted.

Figure 47 - StudentCourses 1.json

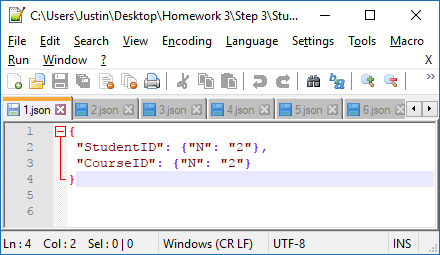


Figure 48 - StudentCourses 2.json

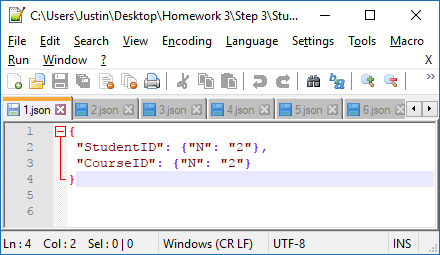


Figure 49 - StudentCourses 3.json

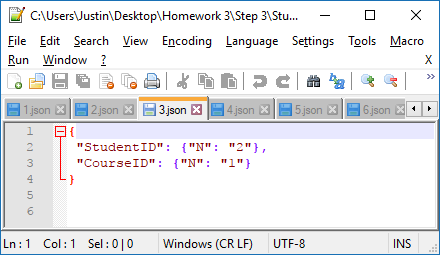


Figure 50 - StudentCourses 4.json

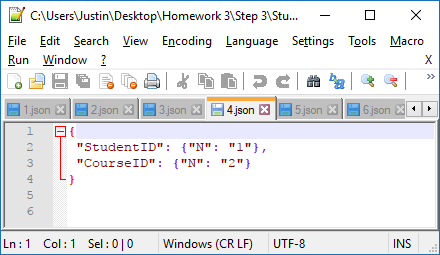


Figure 51 - StudentCourses 5.json

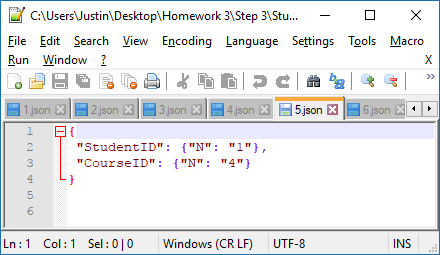


Figure 52 - StudentCourses 6.json

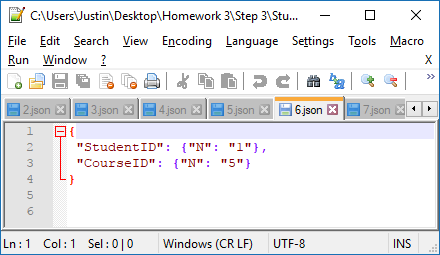


Figure 53 - StudentCourses 7.json

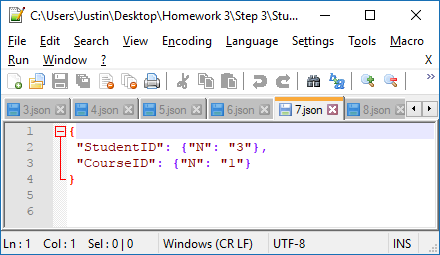


Figure 54 - StudentCourses 8.json

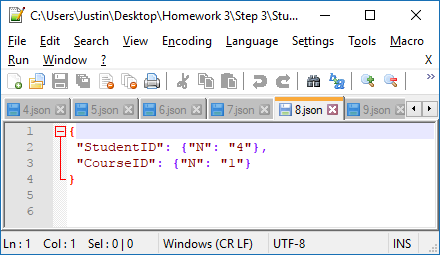


Figure 55 - StudentCourses 9.json

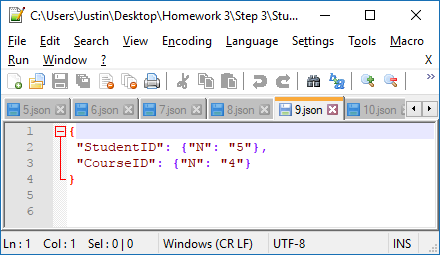
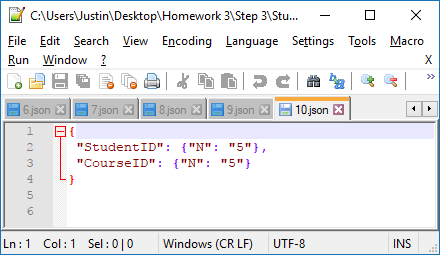


Figure 56 - StudentCourses 10.json



Then I use put-item commands to read from the JSON files I just created so I can insert data into the StudentCourses table.

Figure 57 - StudentCourses put-item commands

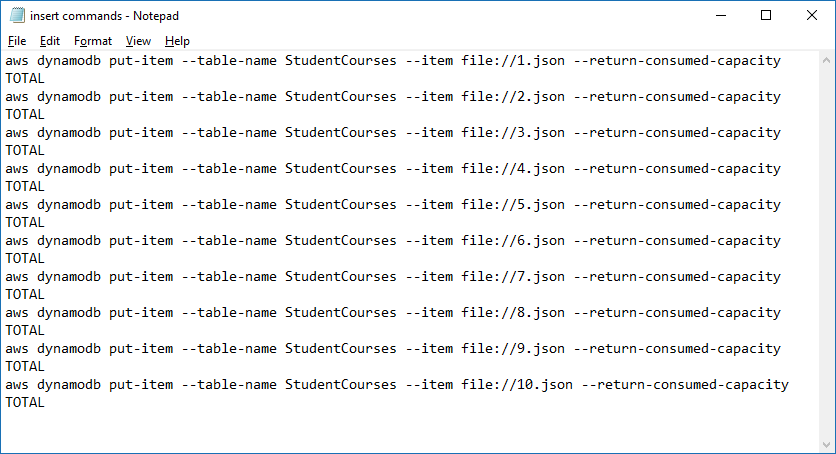


Figure 55 shows the first 5 inserts are successful from JSON files 1-5.

Figure 58 - StudentCourses JSON 1-5

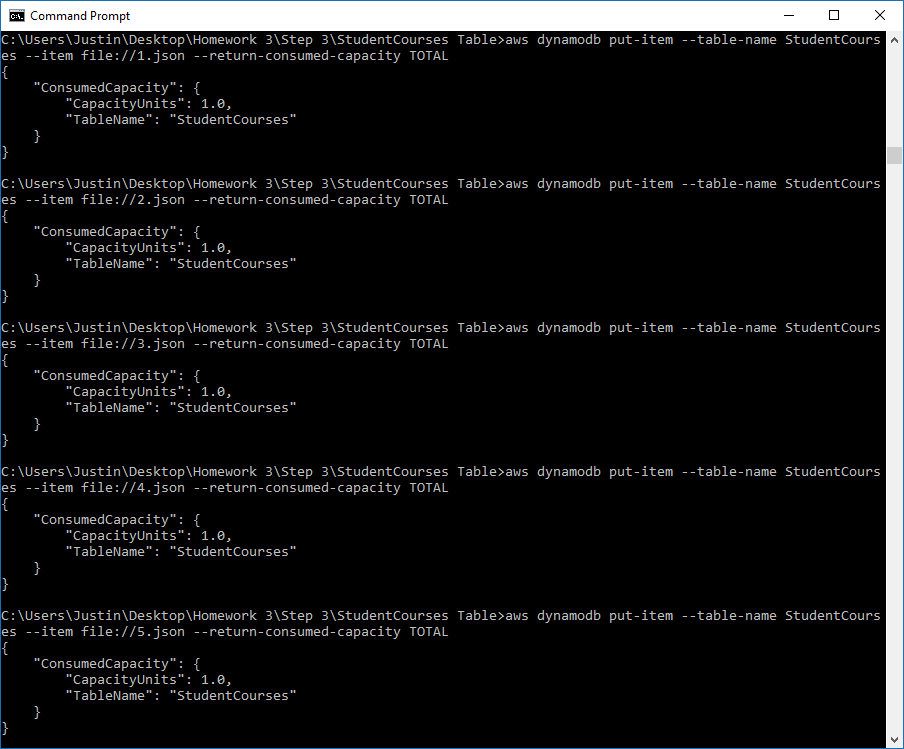
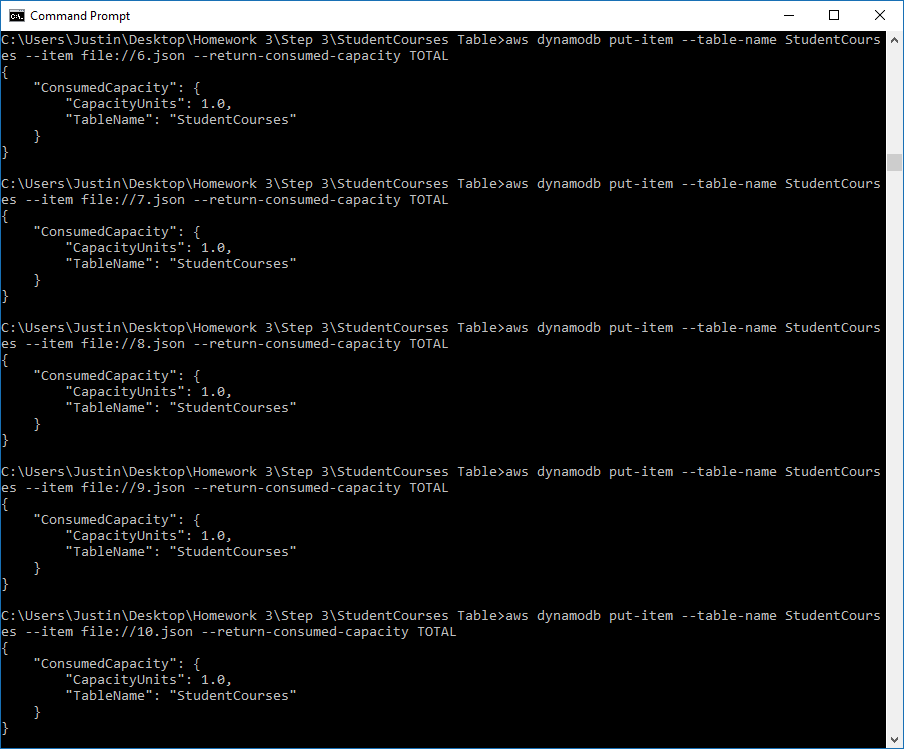


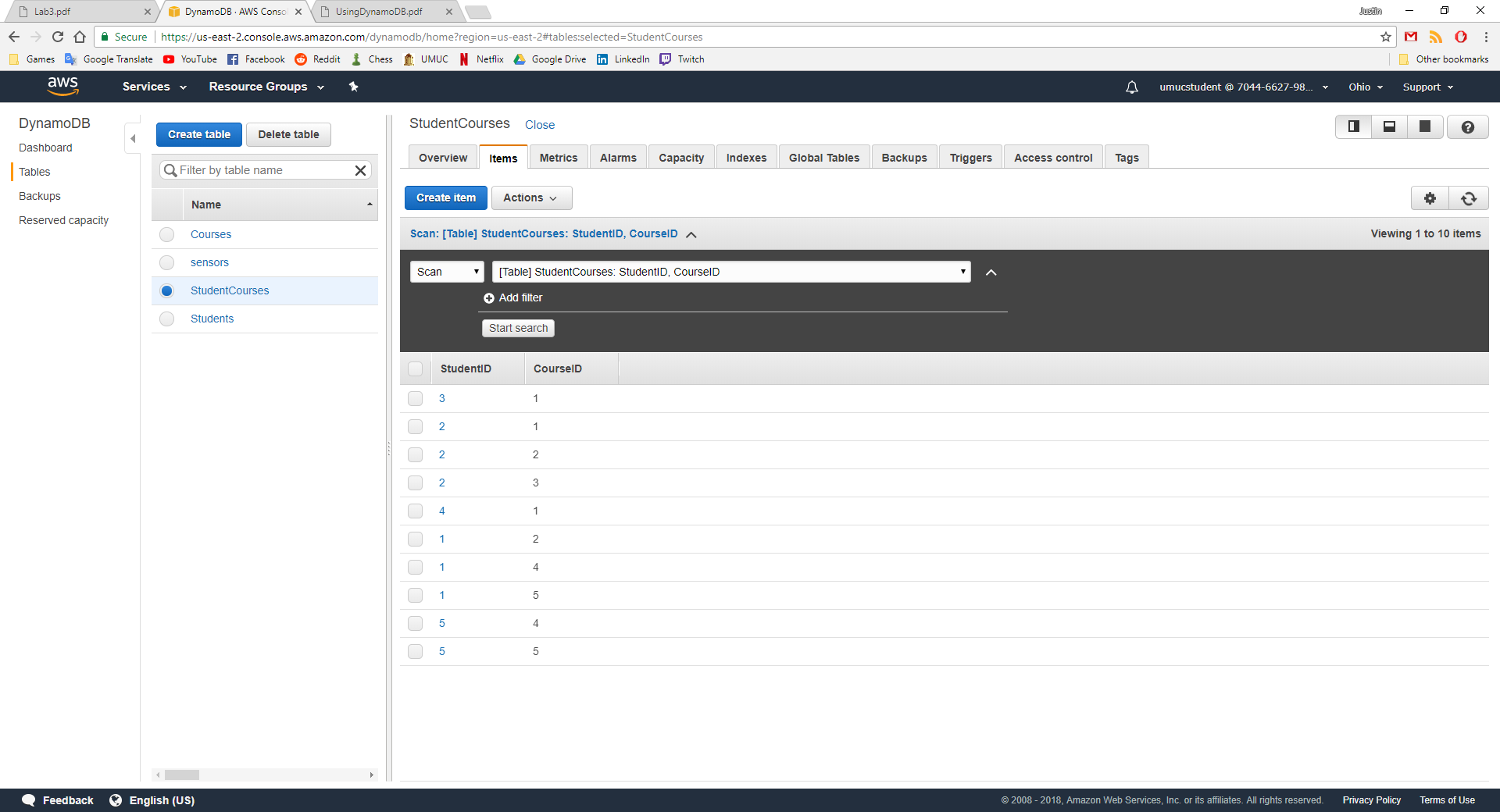
Figure 56 shows the last 5 inserts are successful from JSON files 6-10.

Figure 59 - StudentCourses JSON 6-10



I use the AWS interface to see that this data has been inserted into the StudentCourses table.

Figure 60 - StudentCourses data



I added two additional Student items to my DynamoDB Student table. For these students, I added additional attributes including GPA and ExpectedGraduationYear. I used the following commands and JSON files.

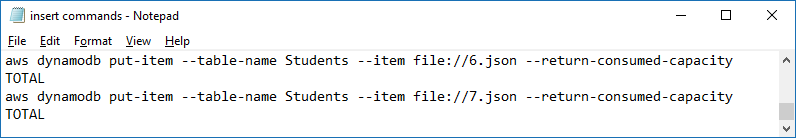


Figure 61 - StudentsTable 6.json

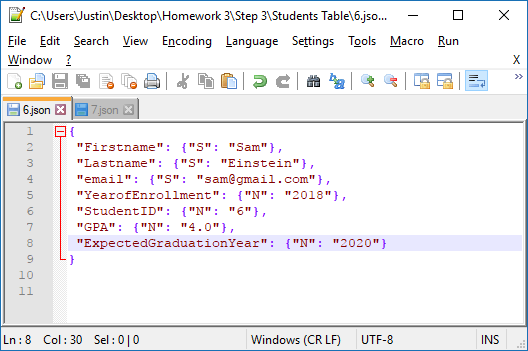


Figure 62 - StudentsTable 7.json

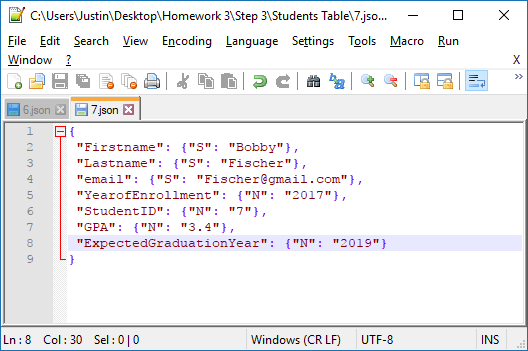
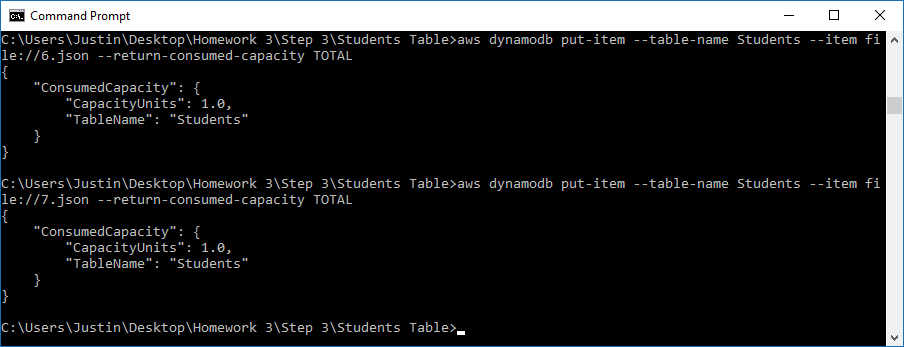
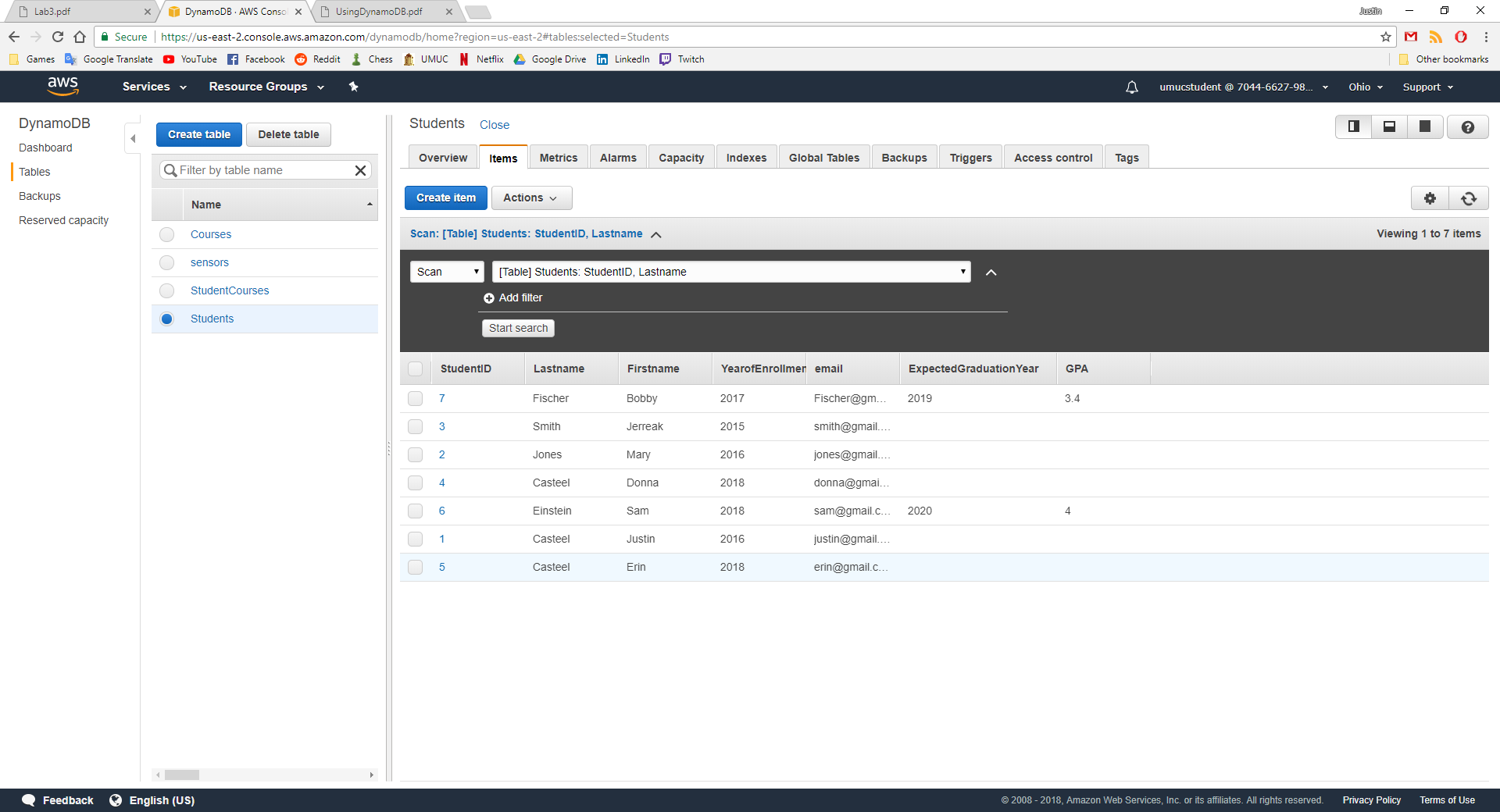


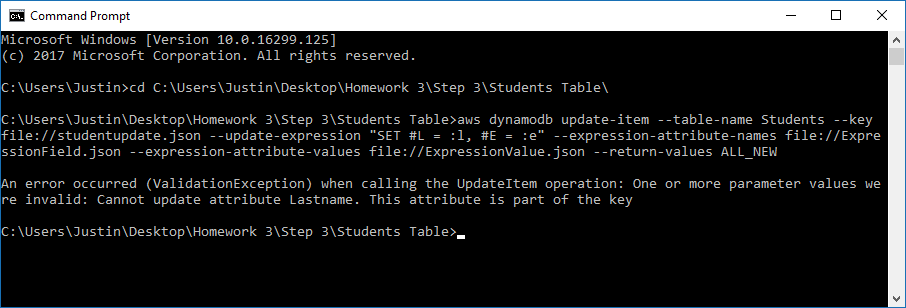
Figure 60 shows the successful insert of a new student with new attributes. This is one great feature of using a NoSQL database, you are able to add attributes on the fly without having to recreate the entire table.



I look at the Students table on the AWS interface to see the new Students added with the new attributes.



The one issue that I did run into on this next step when updating was that the StudentID and Lastname were my HASH and RANGE keys, so I was unable to update the Lastname attribute. I went ahead and chose another attribute that wasn’t part of the key to update. The error below is what I see.



Now if I wanted to update a user in my table using the AWS CLI I use the following code.

Figure 63 - update commands.txt

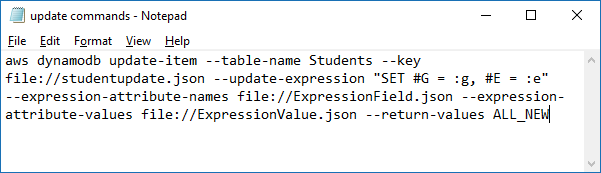
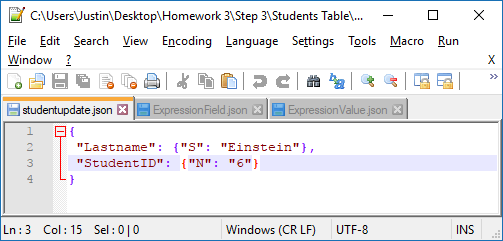


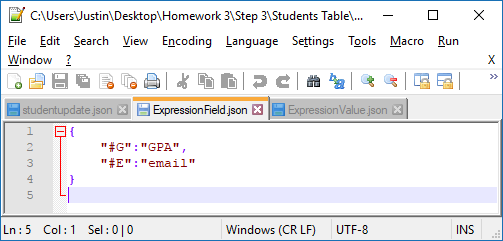
Figure 61 shows which user we would like to update. We provide only the keys in this file, in this case StudentID is the HASH key and Lastname is the RANGE key.

Figure 64 - studentupdate.json



In my ExpressionField.json file we simply associate #G and #E with GPA and email. So #G and #E are referencing our GPA and email attributes.

Figure 65 - ExpressionField.json



Finally, I provide new information to update this student with. “:g” and “:e” are providing the new values for GPA and email. This is defined in figure 63 above in the “update commands.txt” file.

Figure 66 - ExpressionValue.json

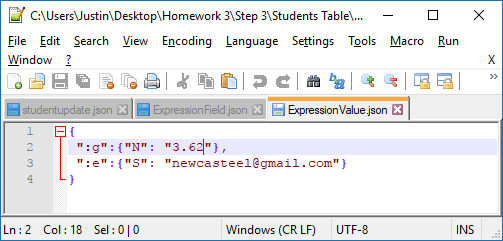
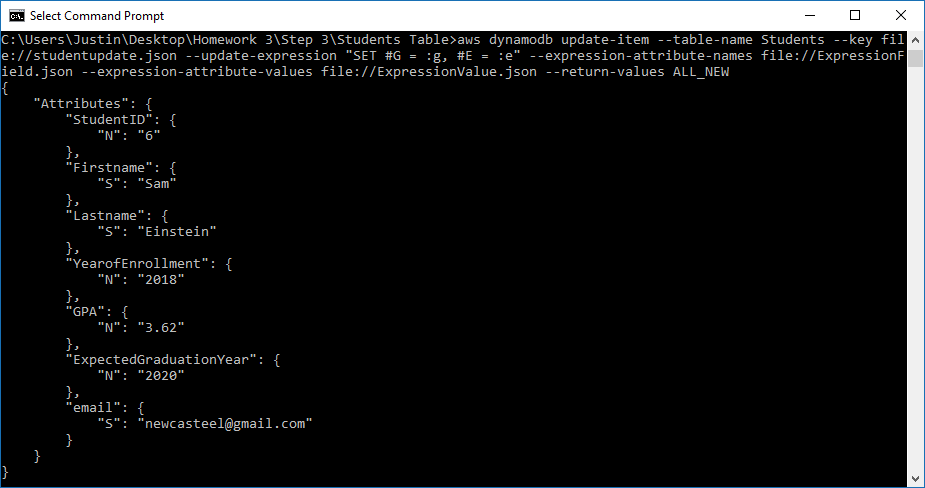
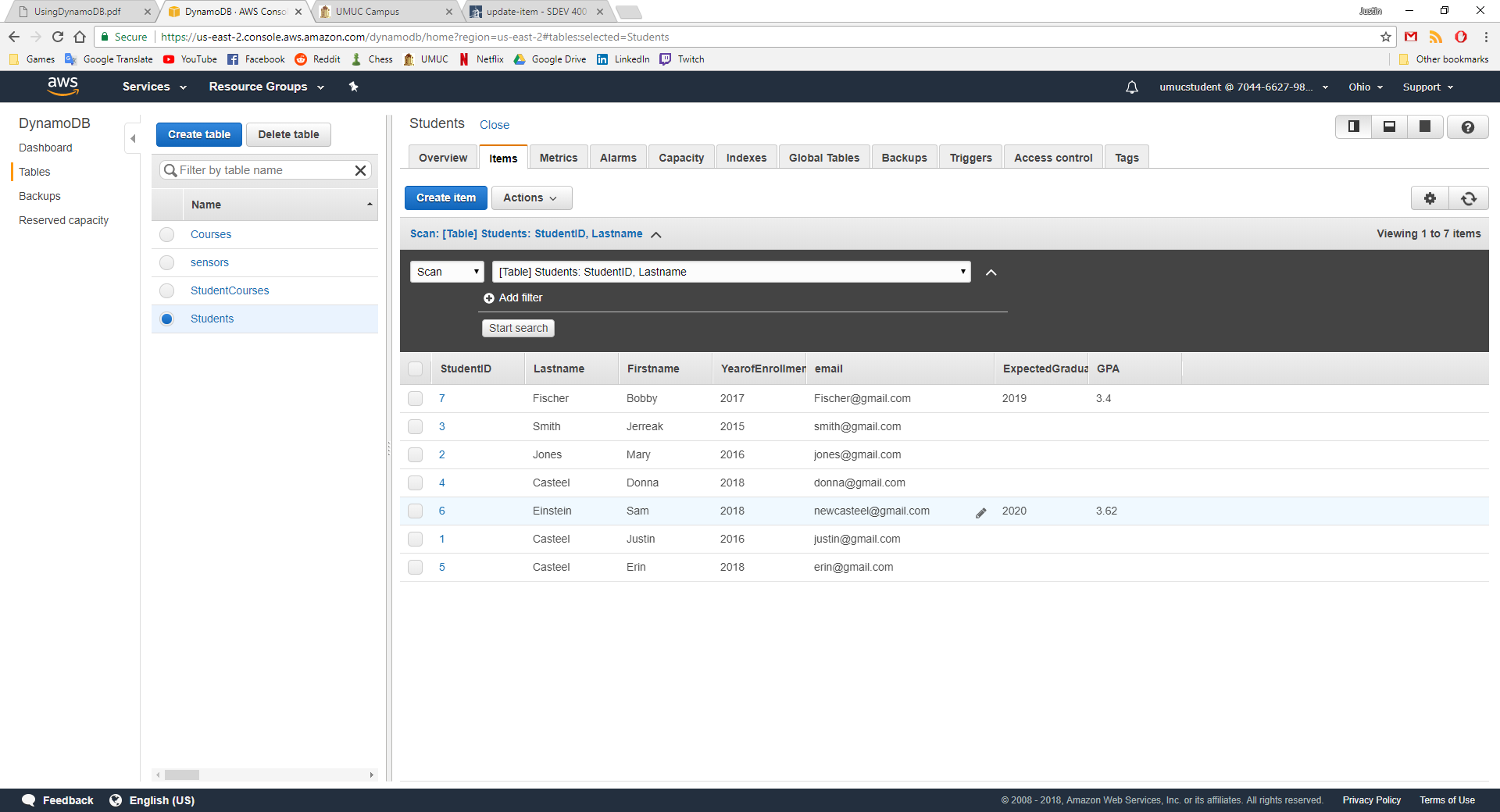


Figure 64 - Update user



Now I confirm that the DynamoDB table has updated this user with the new email and GPA that I provided above.

Figure - AWS interface update



The user with the ID of 6 has had its email and GPA updated as per our update query and files.

Finally, I delete all the tables that I have just created in both DynamoDB and SQL. I will first delete all of the DynamoDB tables by using the following commands.

Figure 68 - deletetables.txt

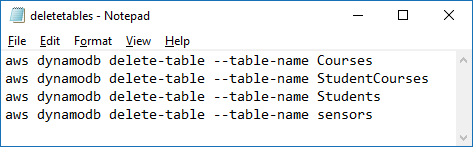


Figure 65 shows the deletion of all 3 tables. I will now check the AWS interface to see that they are deleted.

Figure 69 - delete tables

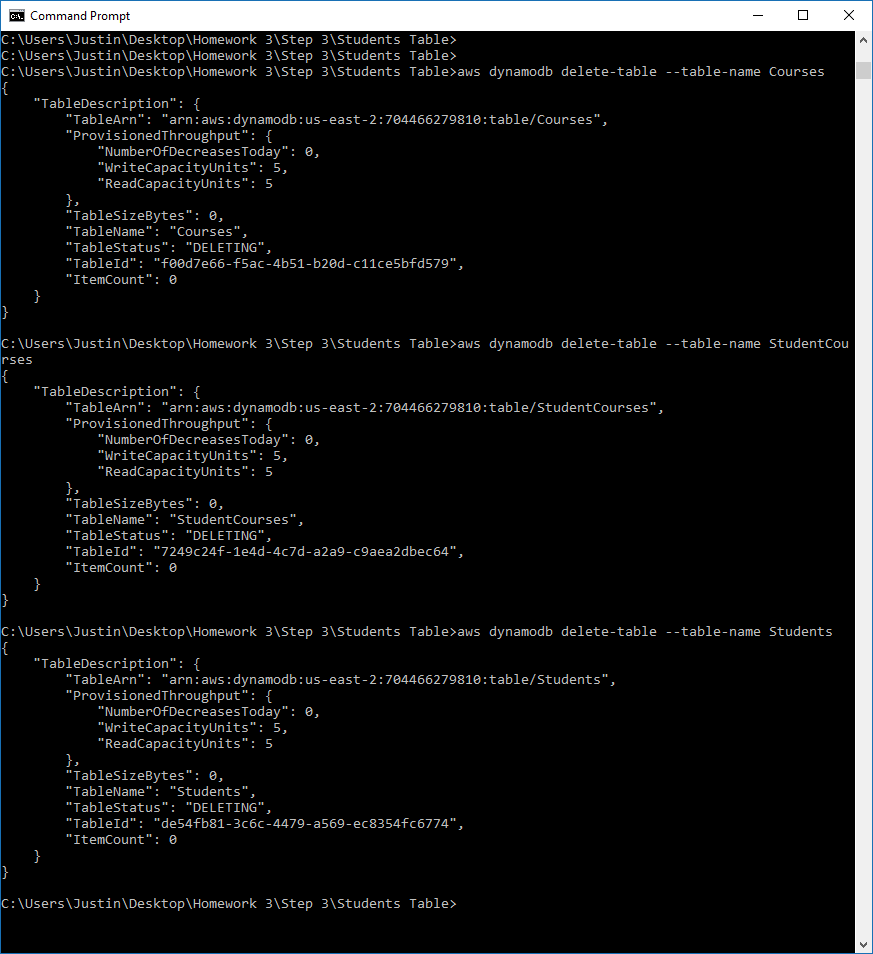
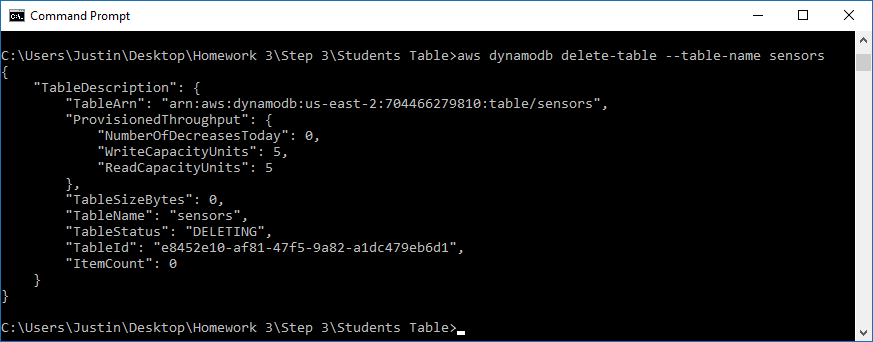


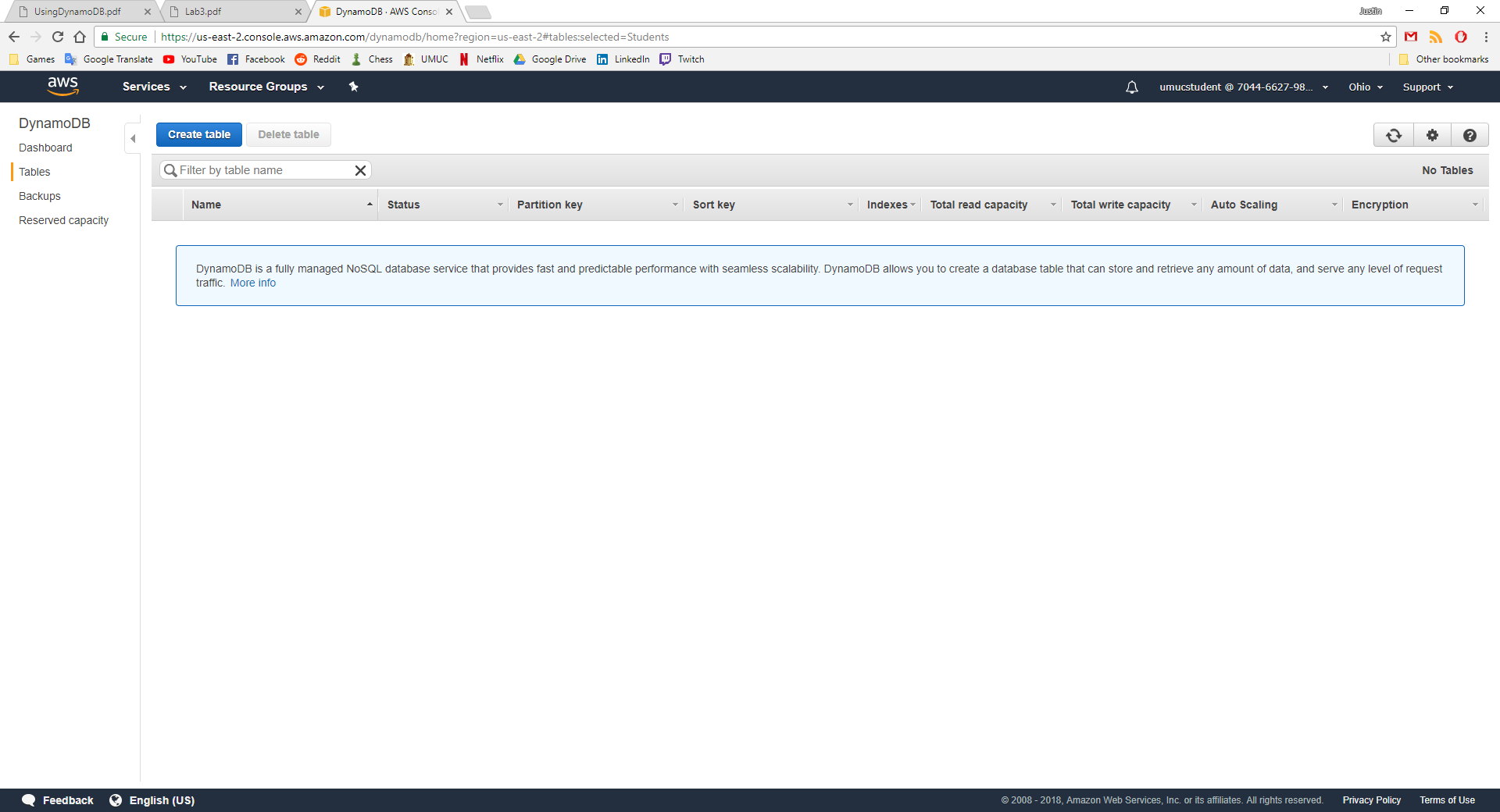
Figure 66 shows the deletion of the sensors table from step 1 of the lab.

Figure 70 - delete tables cntd.



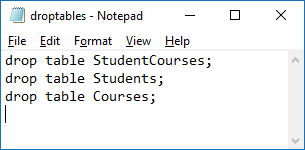
After checking the AWS interface, we can see that all our tables are deleted.

Figure 71 - No tables



To delete the tables in the RDS we simply use the “drop table” command.

Figure 72 - droptables.txt



I show that there are currently 3 tables, I use the drop command for each table and then use the “show tables;” command again to show that the tables have been dropped.

Figure 73 - tables dropped



# References

Amazon. (2010, May 15). *Amazon DynamoDB Table KeySchema*. Retrieved from AWS CloudFormation: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-dynamodb-keyschema.html

Amazon. (2012, August 10). *Creating a Table*. Retrieved from Amazon DynamoDB: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/SQLtoNoSQL.CreateTable.html

Amazon. (2012, August 10). *From SQL to NoSQL*. Retrieved from Amazon DynamoDB: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/SQLtoNoSQL.html#SQLtoNoSQL.WhyDynamoDB

Amazon. (2012, August 10). *Step 3: Create, Read, Update, and Delete an Item*. Retrieved from Amazon DynamoDB: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GettingStarted.PHP.03.html#GettingStarted.PHP.03.03

Amazon. (n.d.). *UpdateItem*. Retrieved from Amazon DynamoDB: https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API\_UpdateItem.html