

| Business Template  **AWS Cloud Data Serives** |
| --- |
|  |

**Notice:** I uploaded this file to my S3 bucket, along with a ZIP archive named **AWS Data Services Scripts**, which contains MySQL scripts for the first 2 Tasks and then my json files that I used in the CLI to create my table, select and delete.

# TASK 1. RDS MYSQL

In this task let’s work with MySQL database to see that it is quite similar with what you have already done in PostresSQL and Oracle previously.

Mentors started one or two RDS MySQL databases. Please, first of all, using admin user (see below) create your own user in this MySQL RDS database. All further database manipulation should be from your own users.

Usually, to initialize or upgrade any database special initial script is prepared, in which you combine necessary DDL and DML operations to establish database environment.

Prepare the initial script in which you:

Initial script should be provided to Mentors and screens of the task steps should be in the Report.

User:admin

Password: admin\_mysql

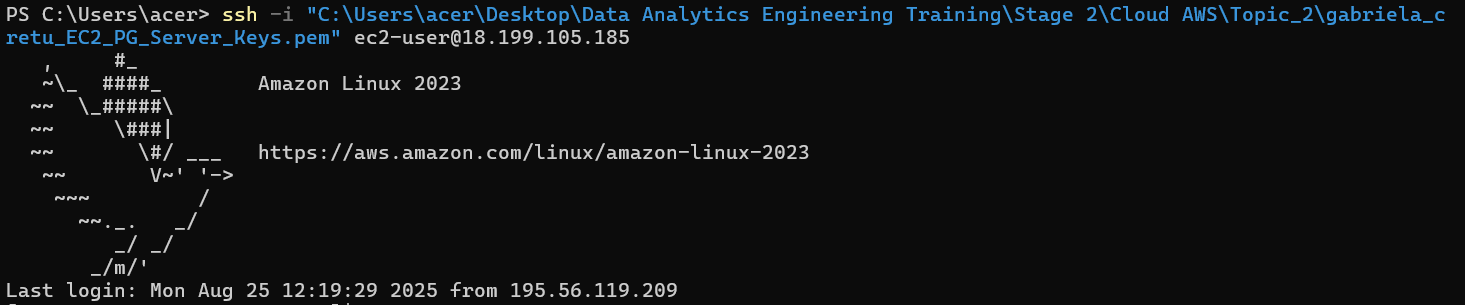
Db: dilab\_dev

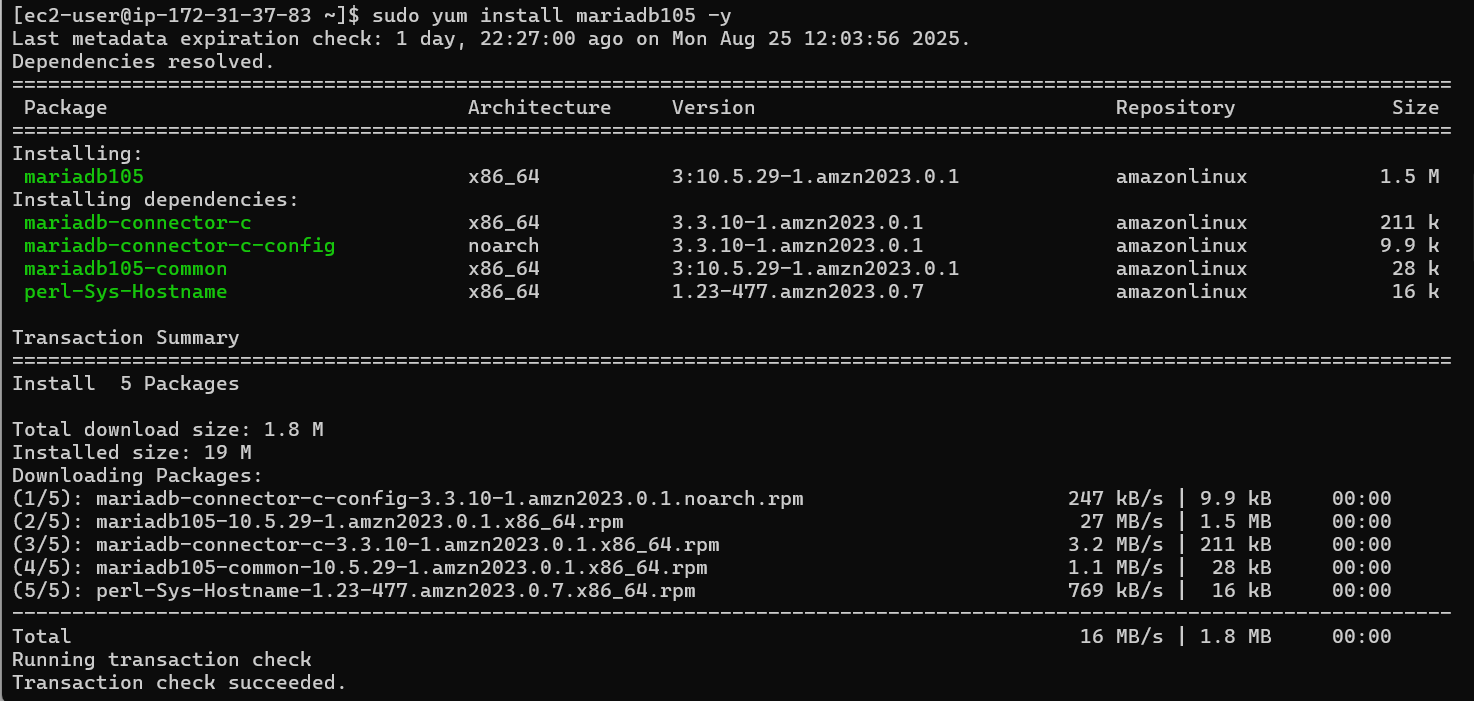
User:gabriela\_cretu

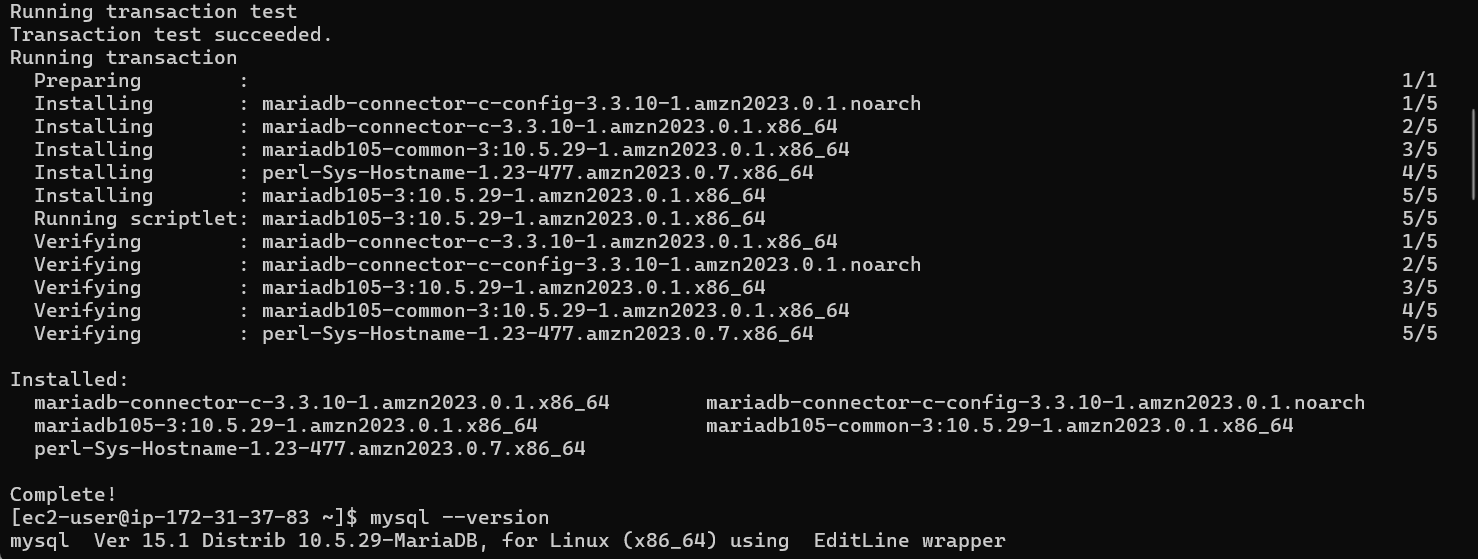
Password:CretuPaul1990.

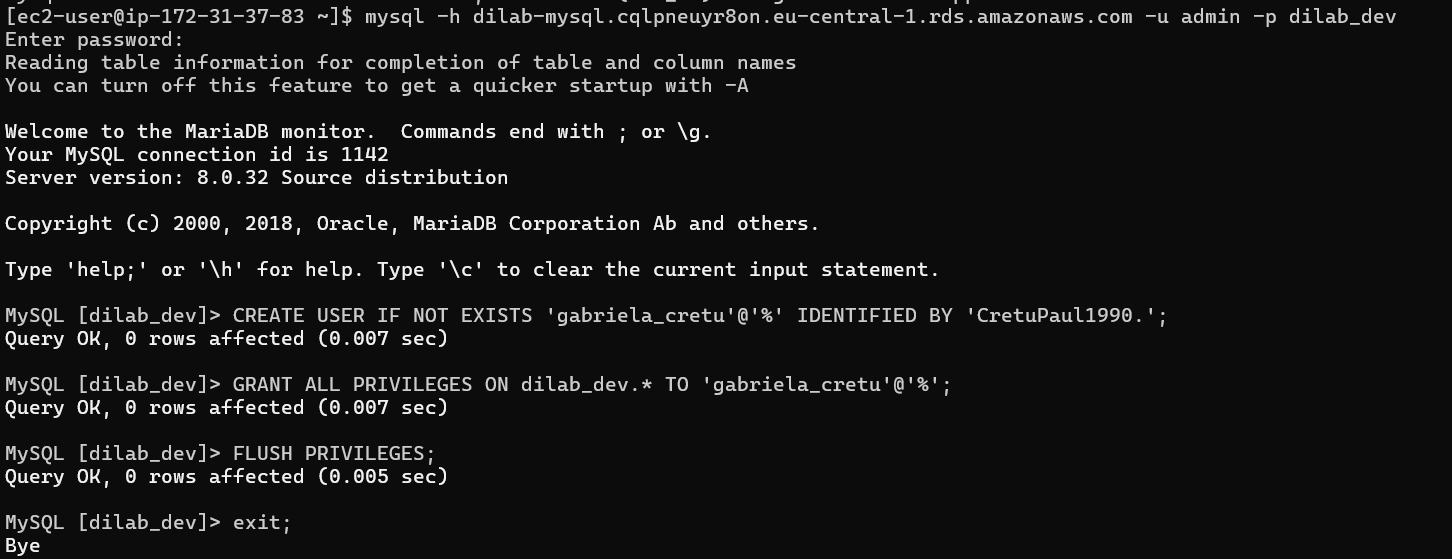
-establish the connection to any of 1-2 provided mysql databases using your own user

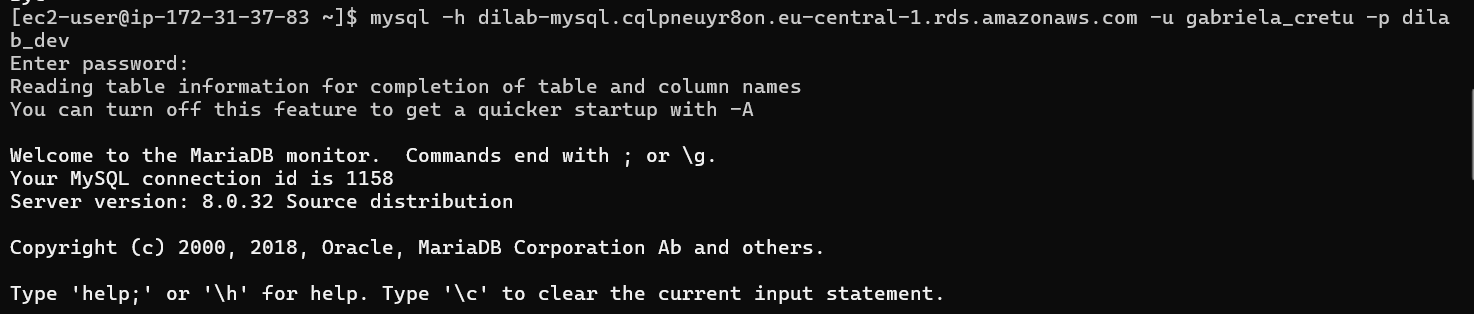


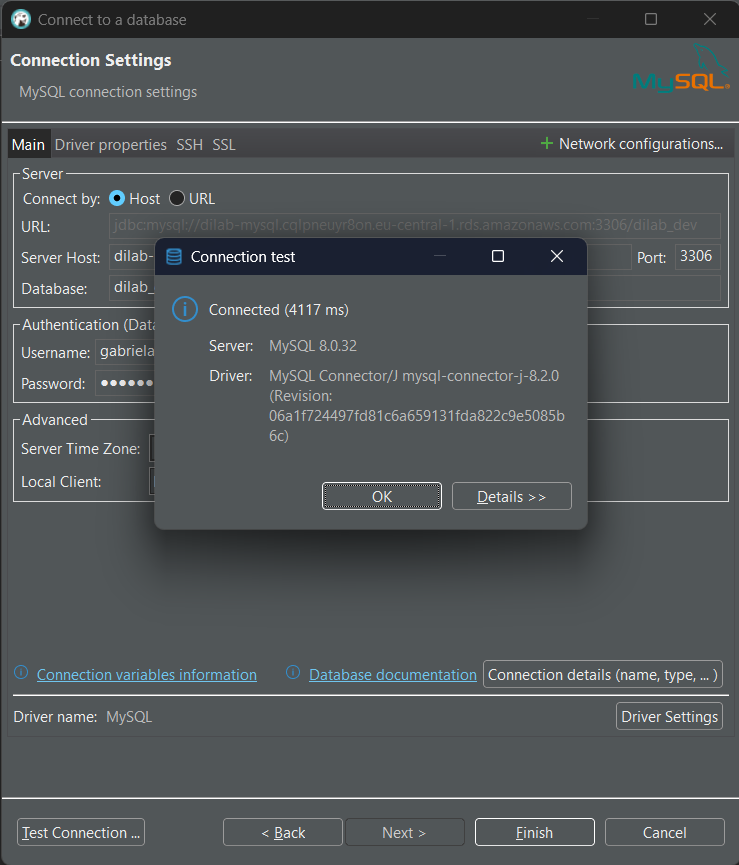
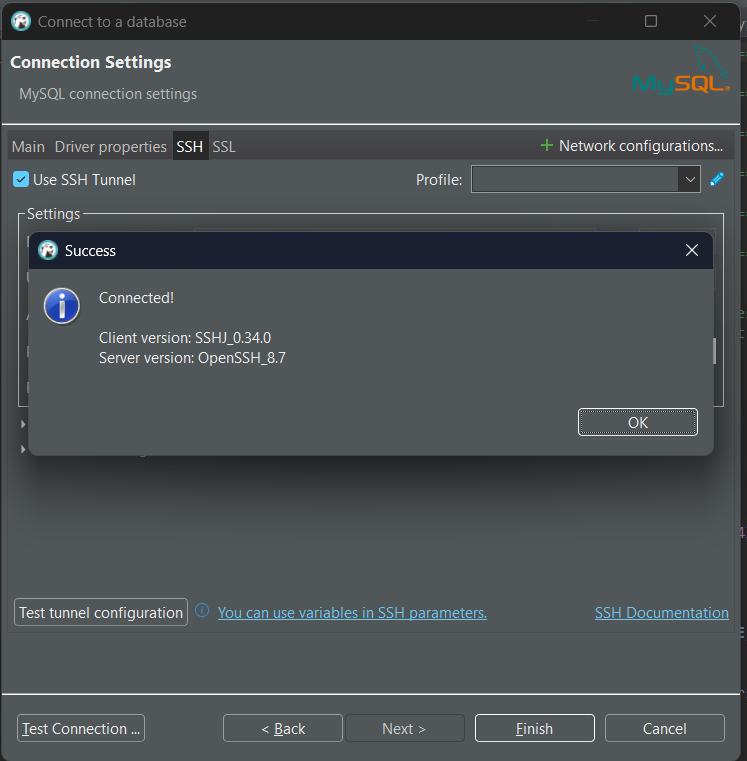


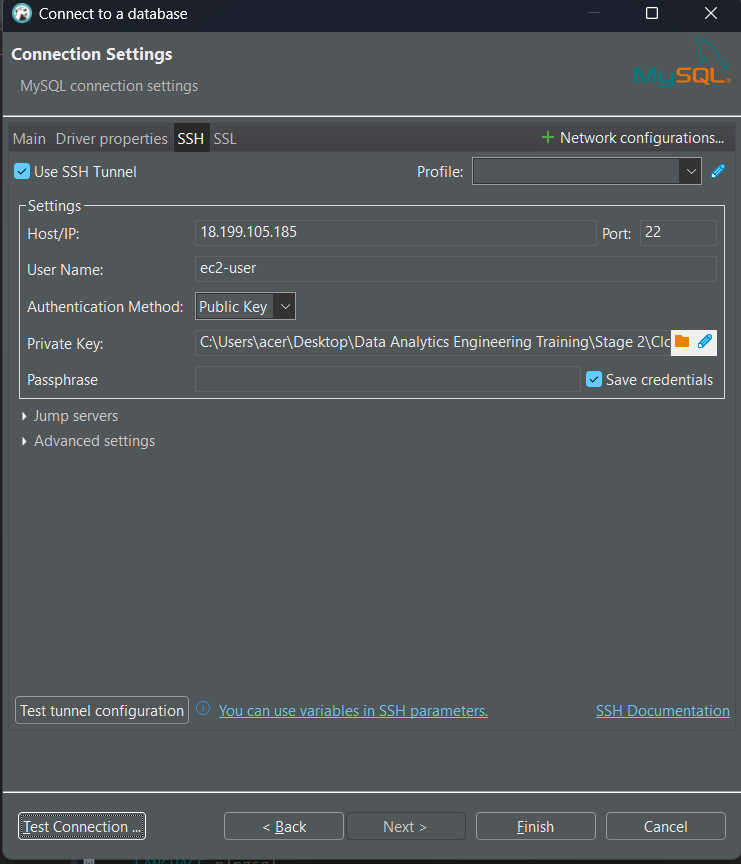
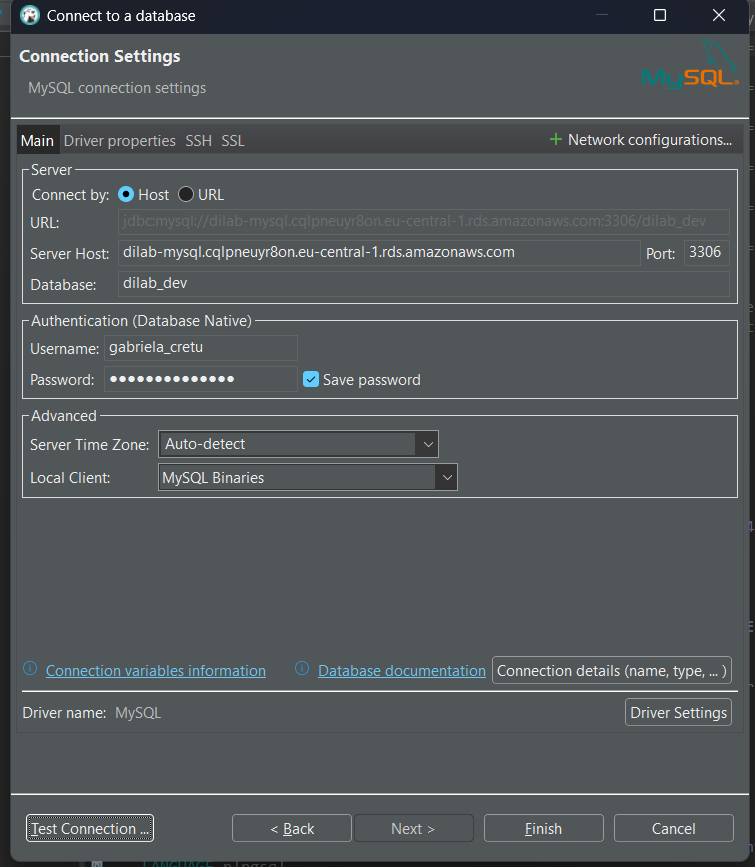




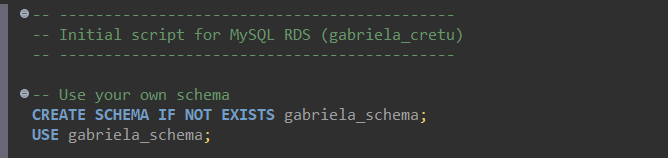
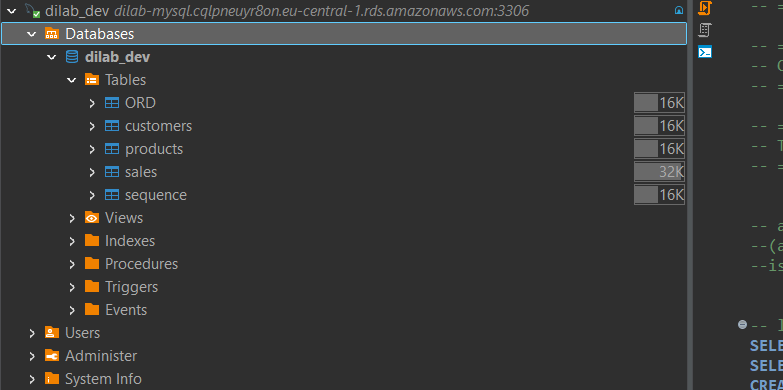




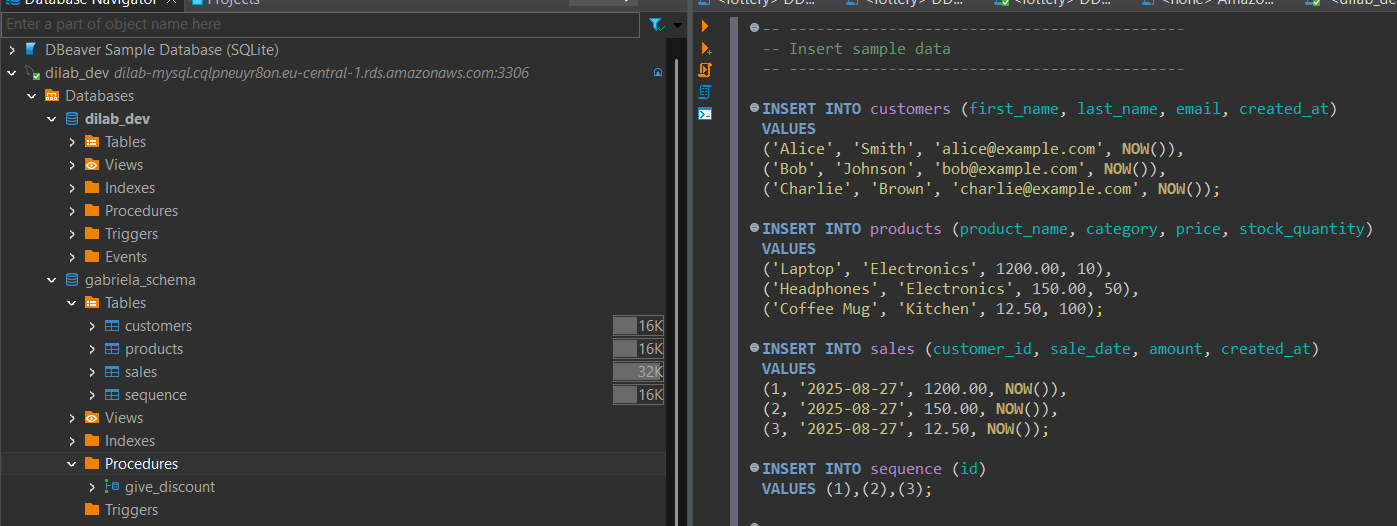
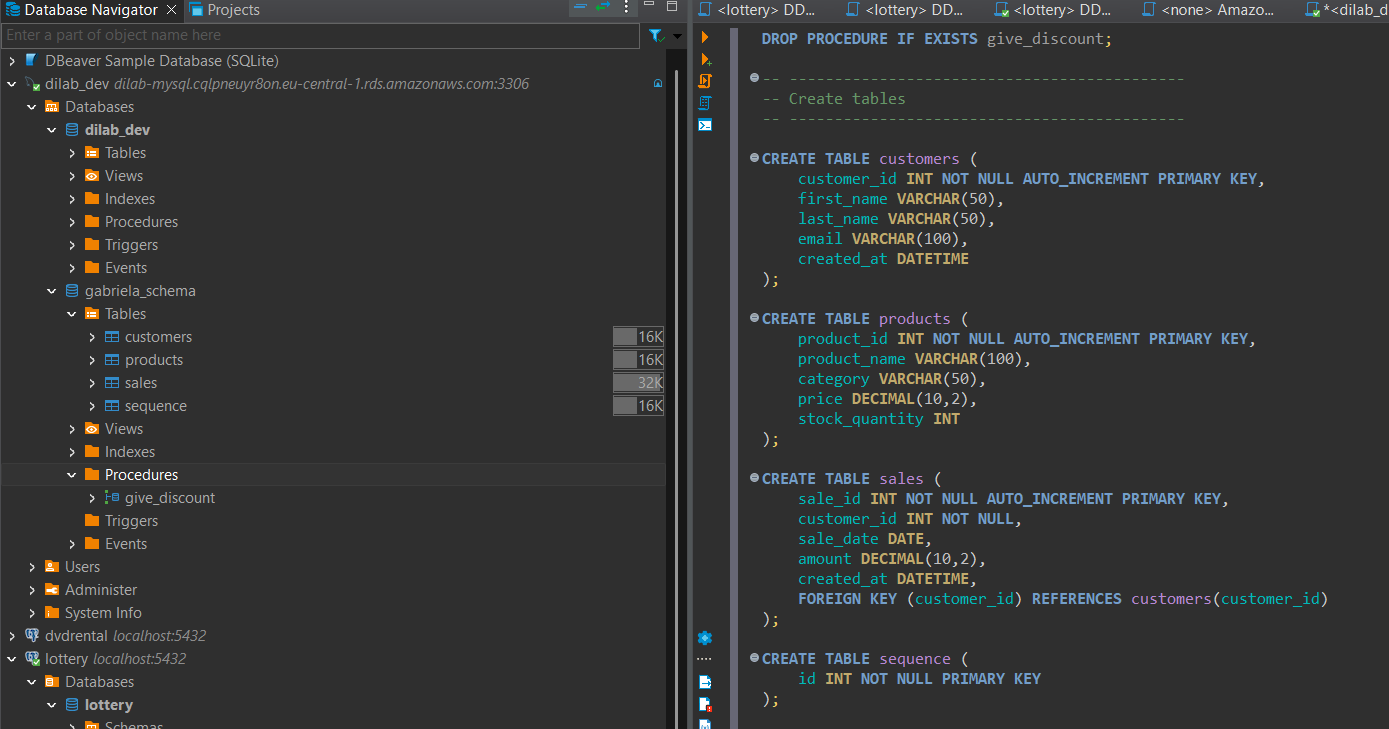


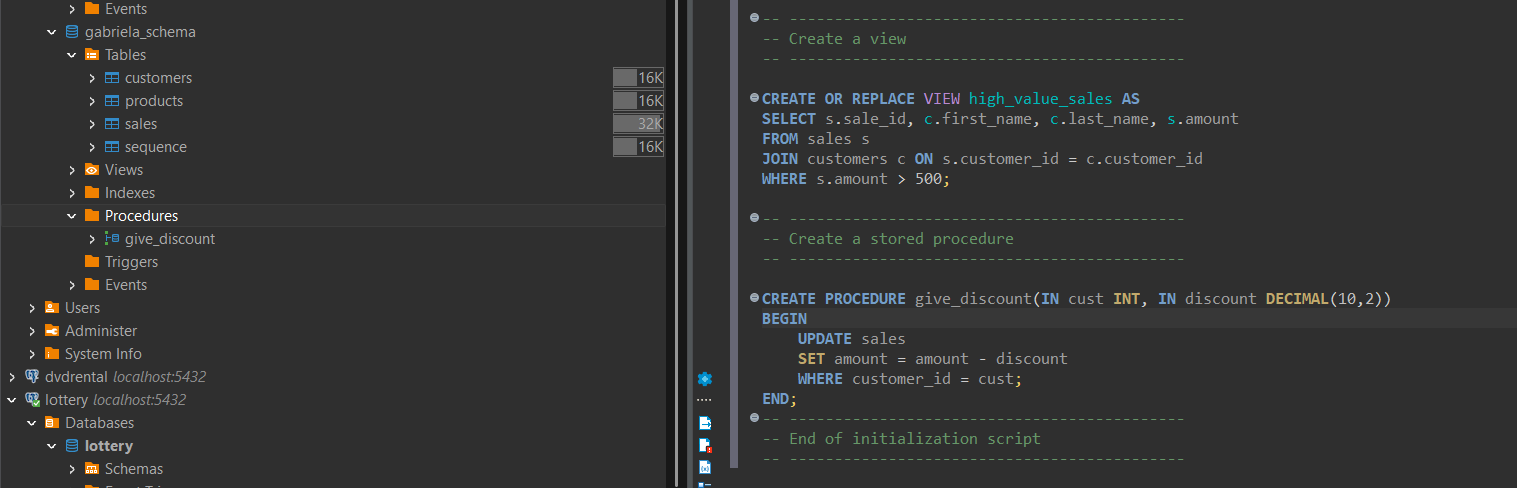


-create a separate schema

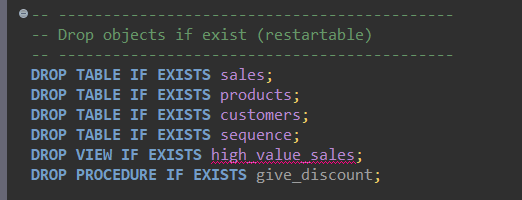


-create several objects in your schema -> at least table, view, procedure





-this script should be restartable

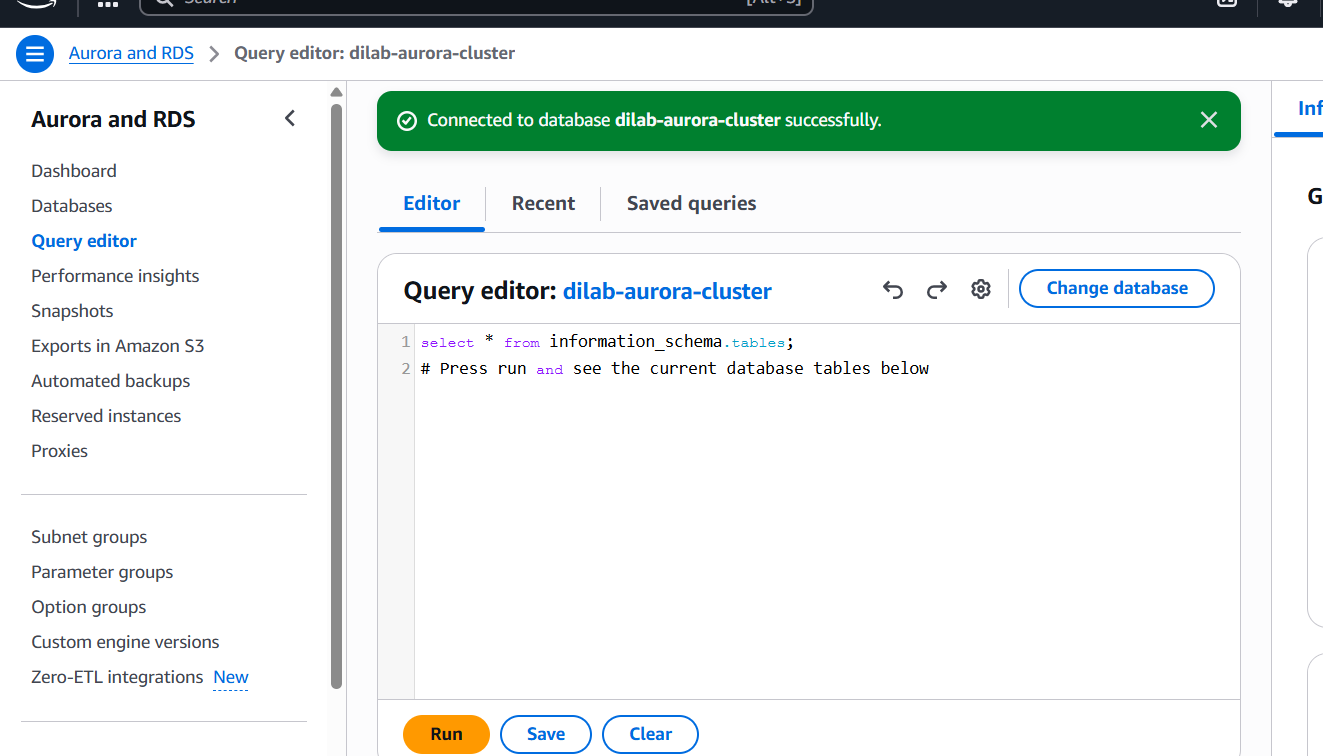
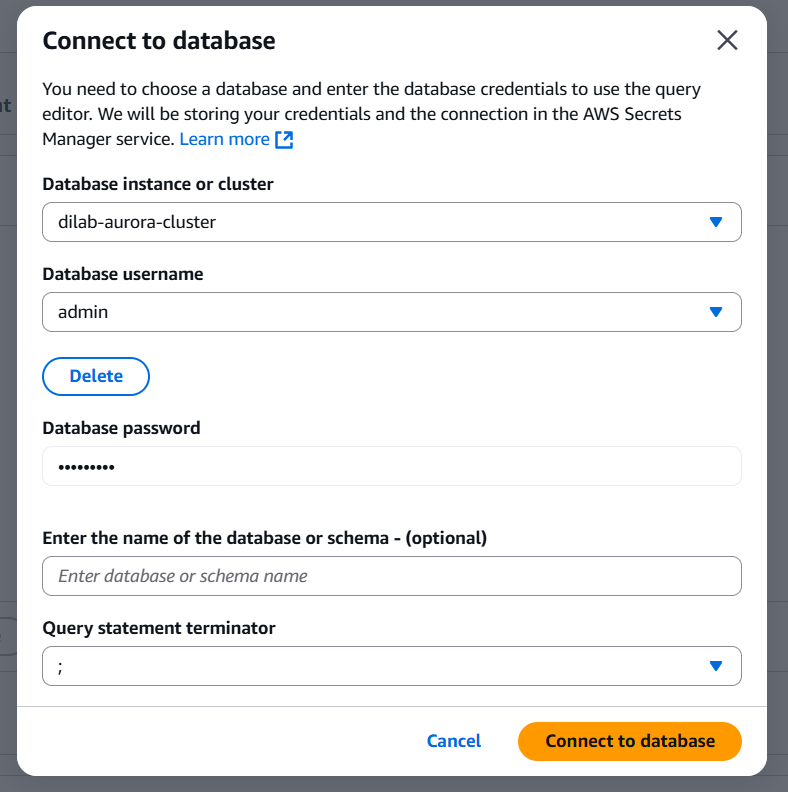


# TASK 2. RDS AURORA

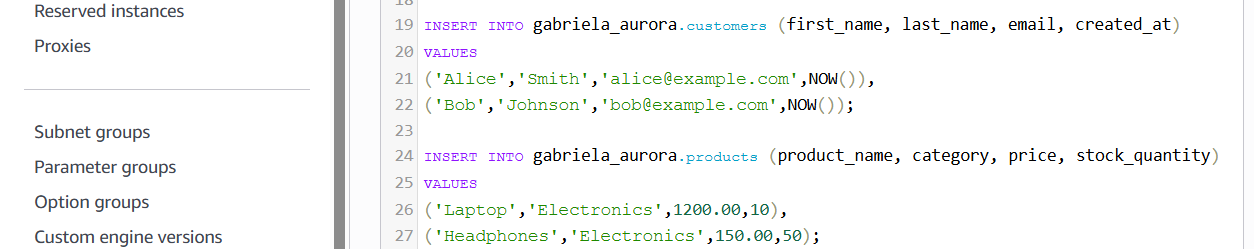
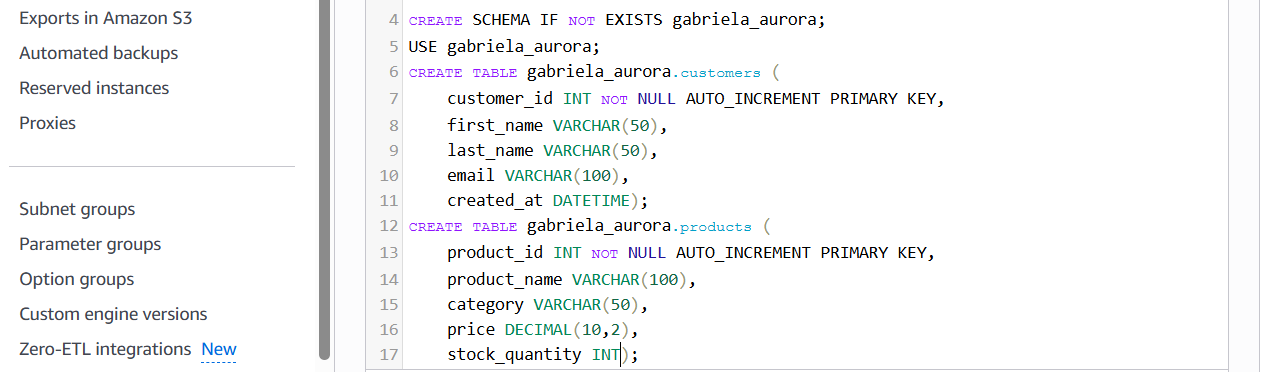
User: admin

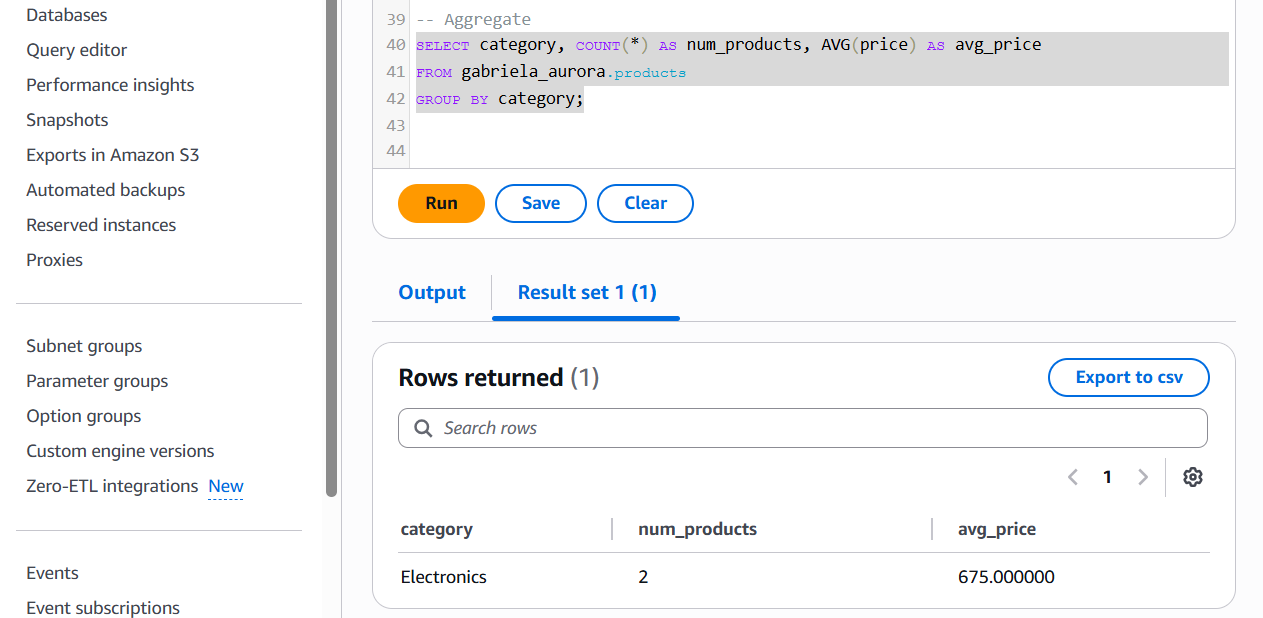
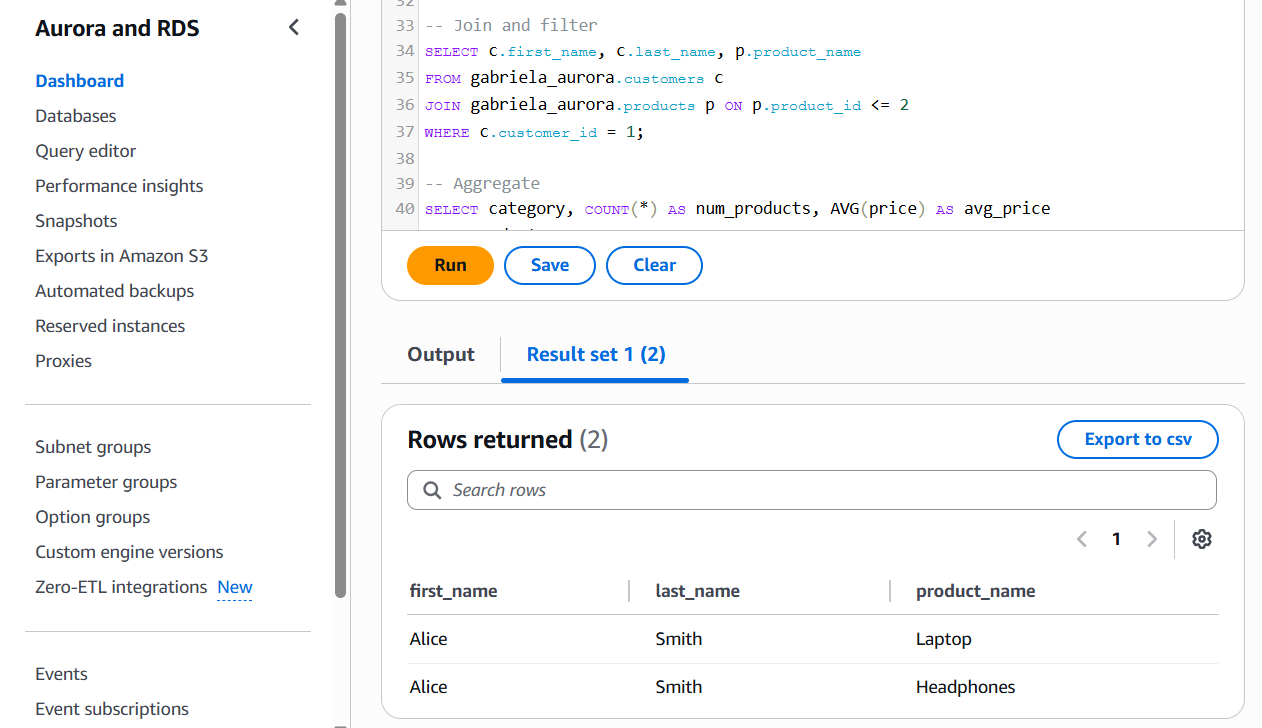
Password: admin\_aurora Db: dilab\_dev / sakila

-Let’s try to manipulate Aurora RDS database directly from AWS Console.

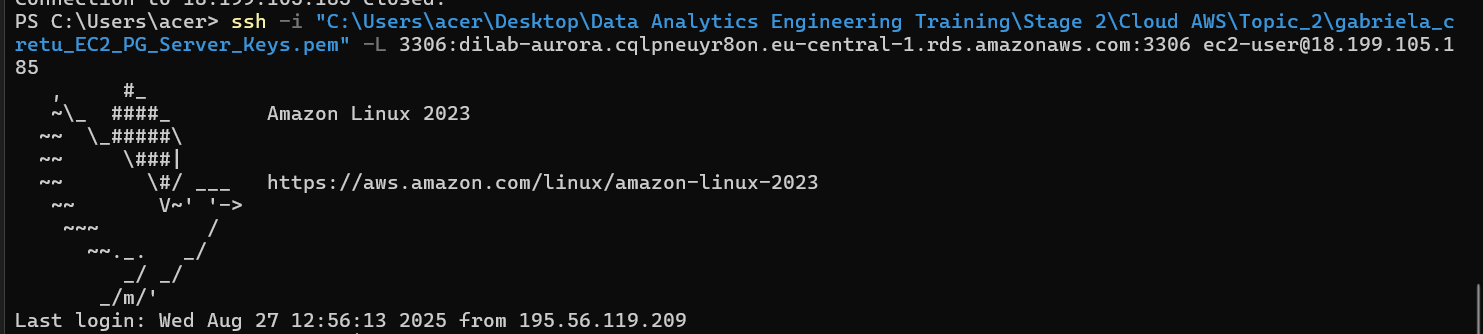


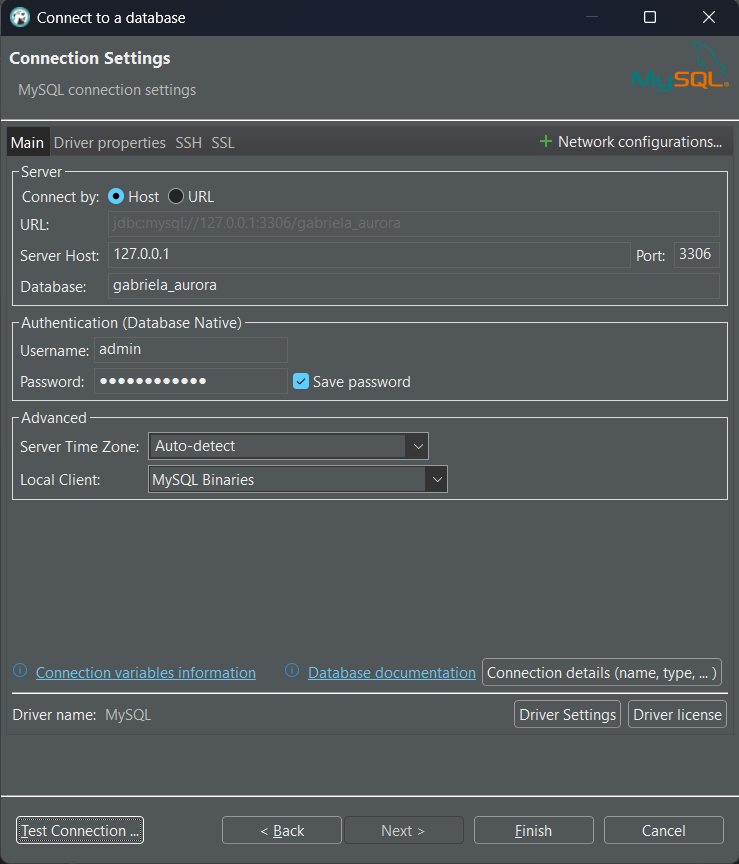
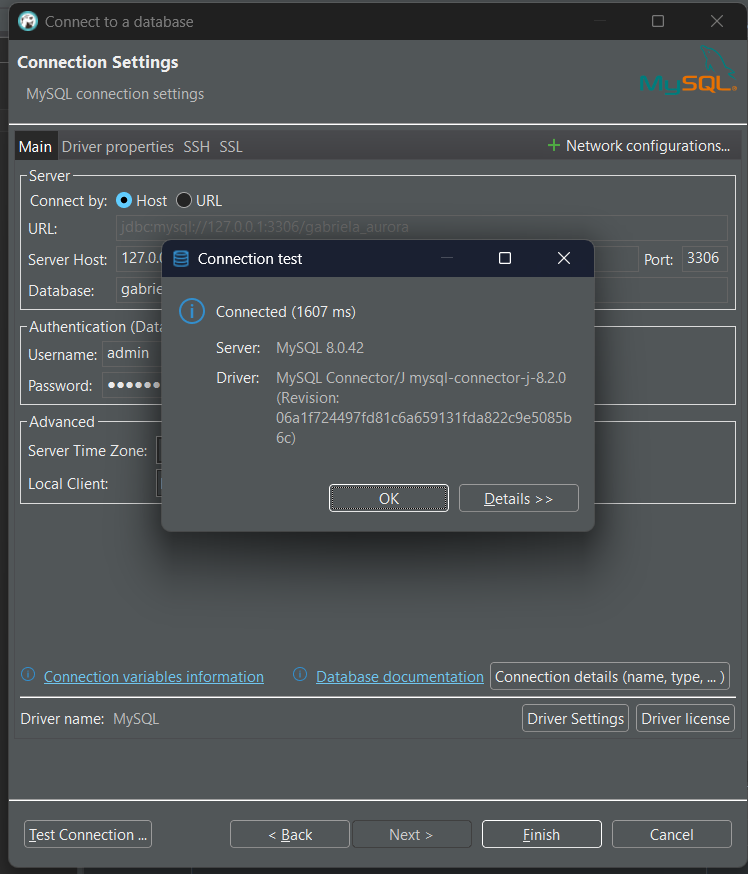
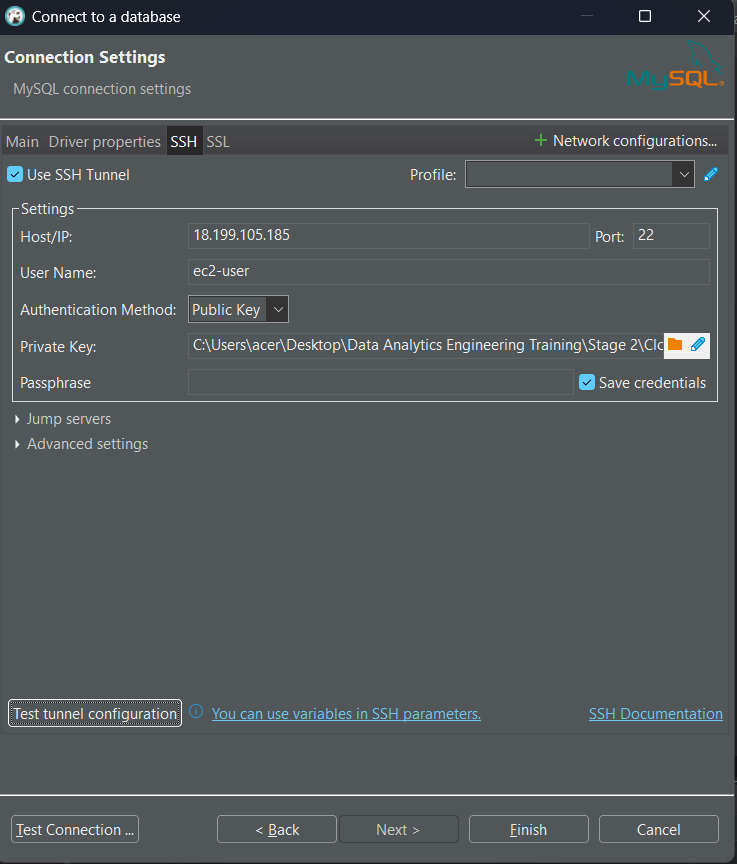
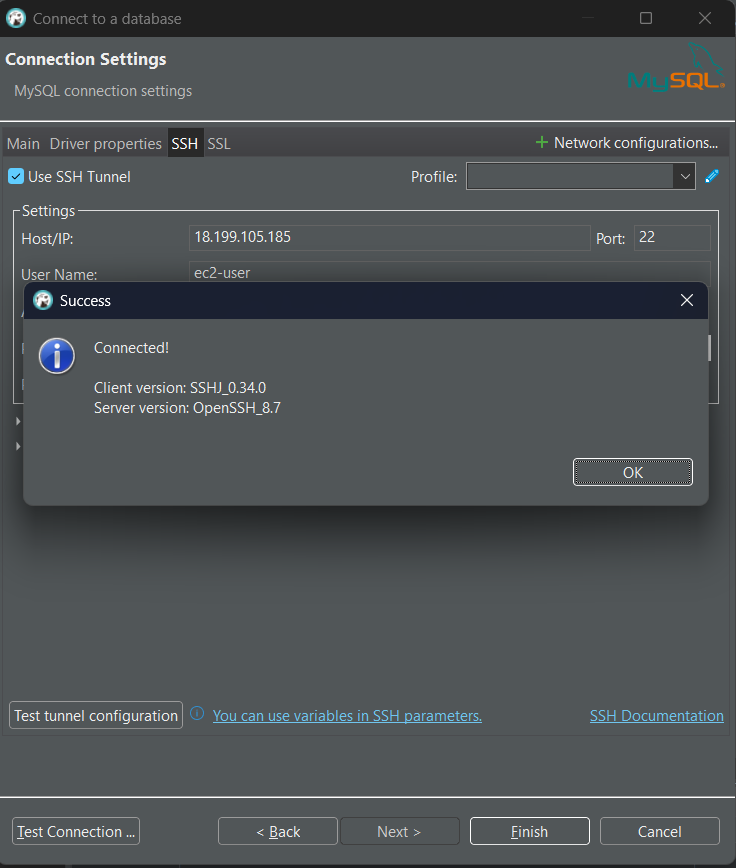
-Use Query editor to create several meaningful (not simple) queries to the data in the Aurora db. For this you could create your own schema in the database and populate it or use pre- created sakila schema in this db.

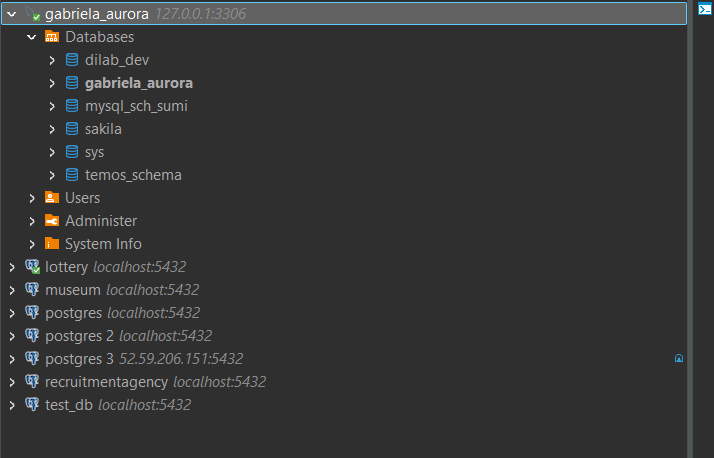


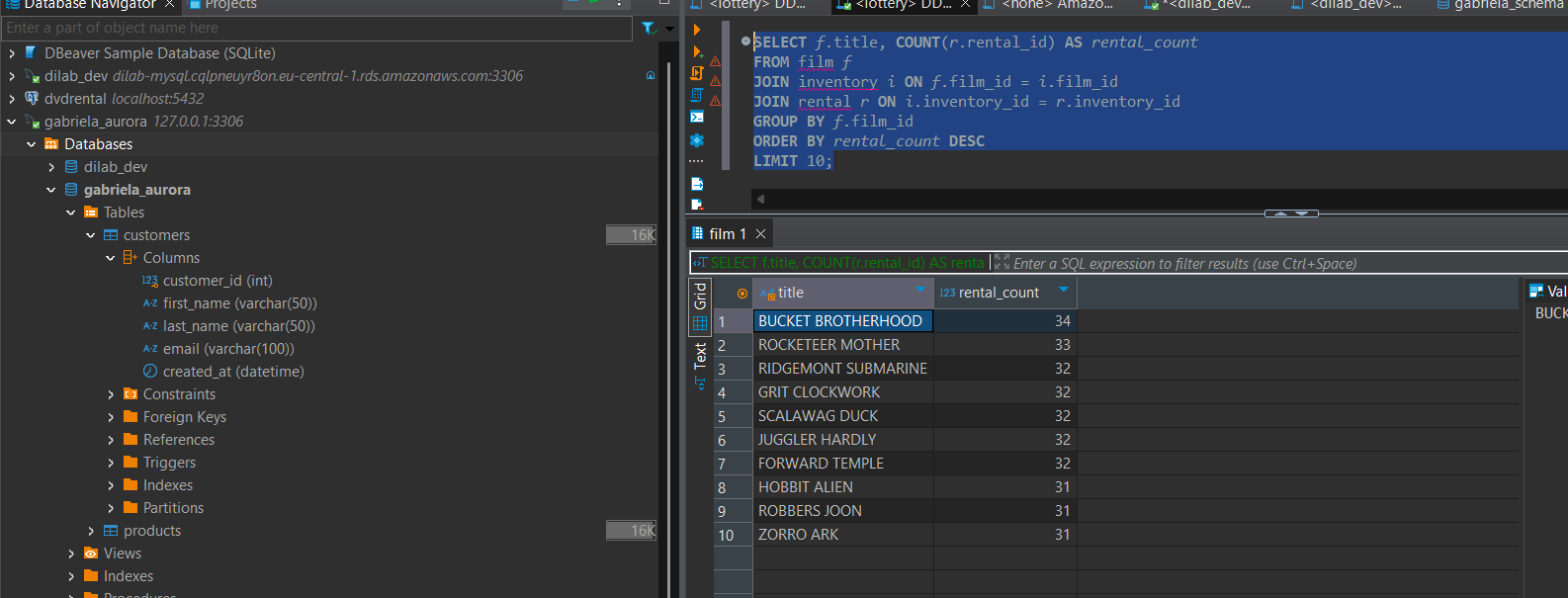
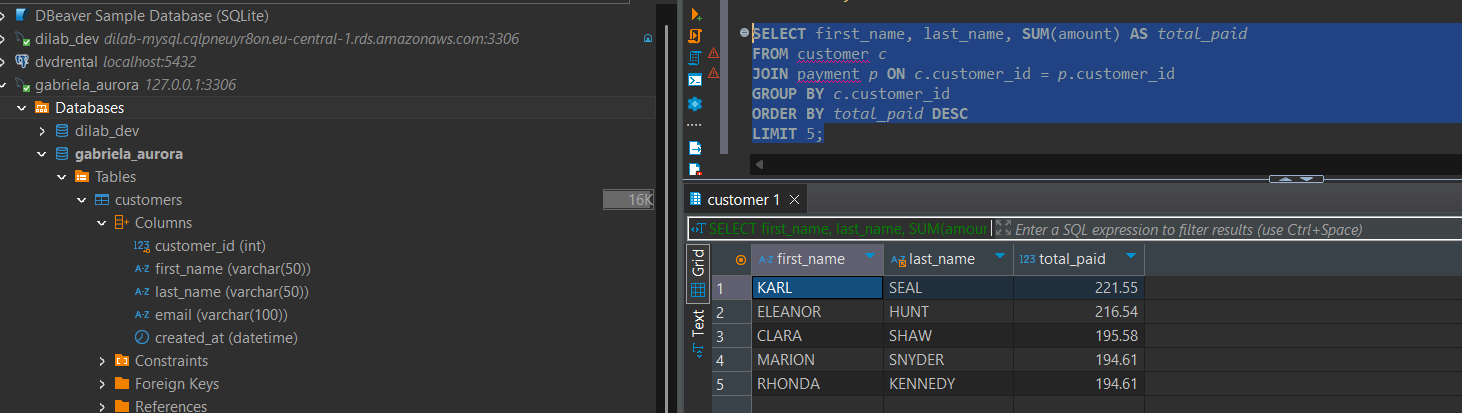


-Use any EC2 created by you in the HW02 as a jump-server (aka bastion server), pay attention to the keys management (e.g. https://aws.amazon.com/premiumsupport/knowledge- center/systems-manager-ssh-vpc-resources/) and connect to the Aurora RDS from your PC using any SQL client.

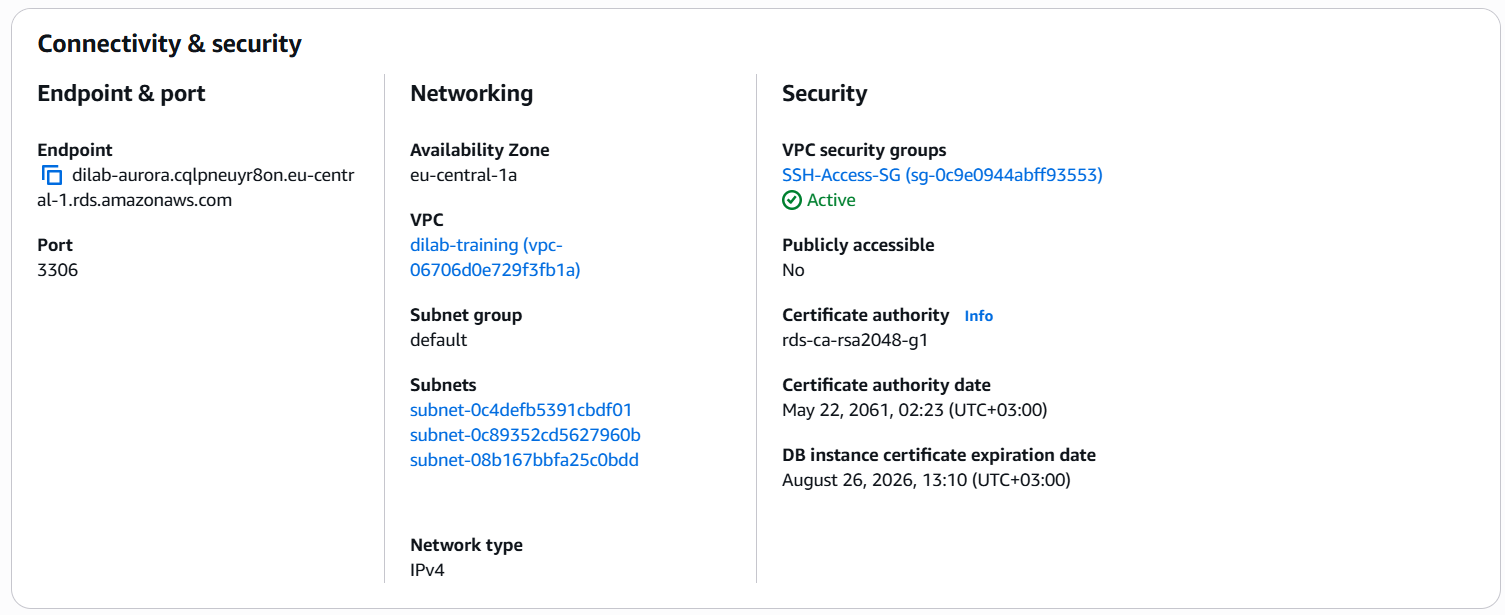
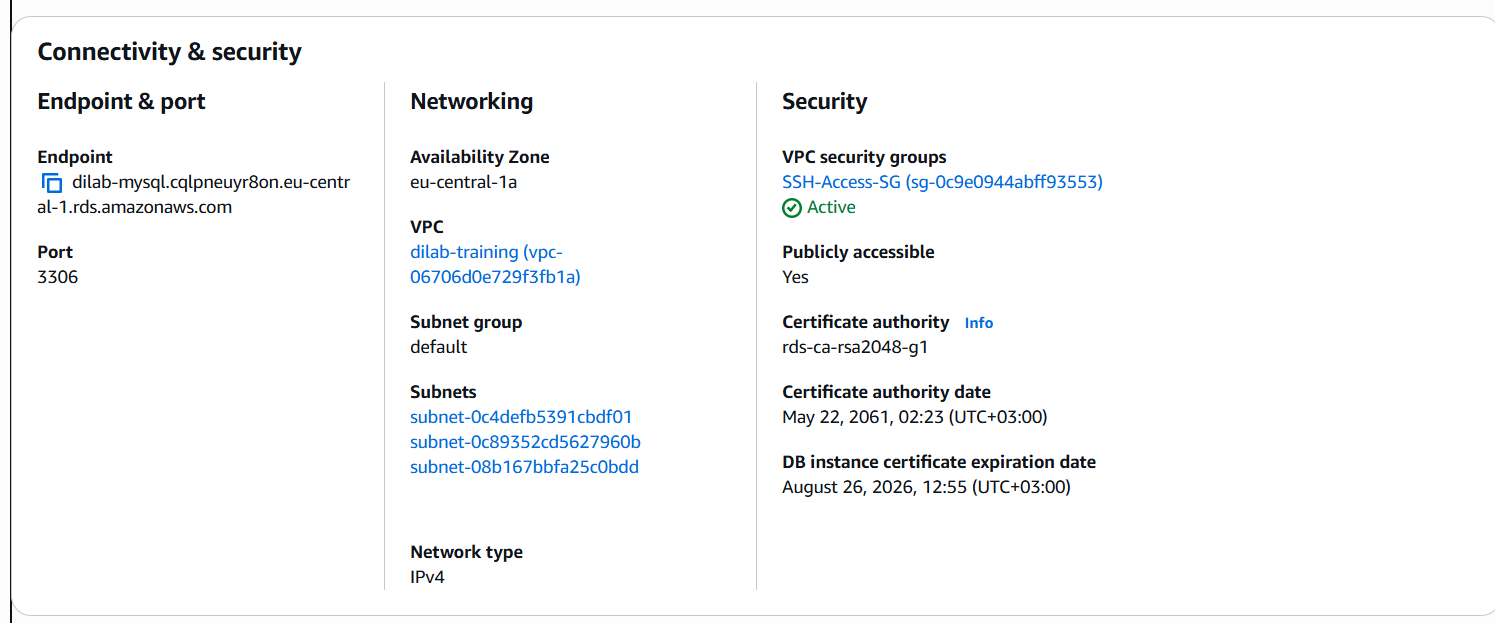








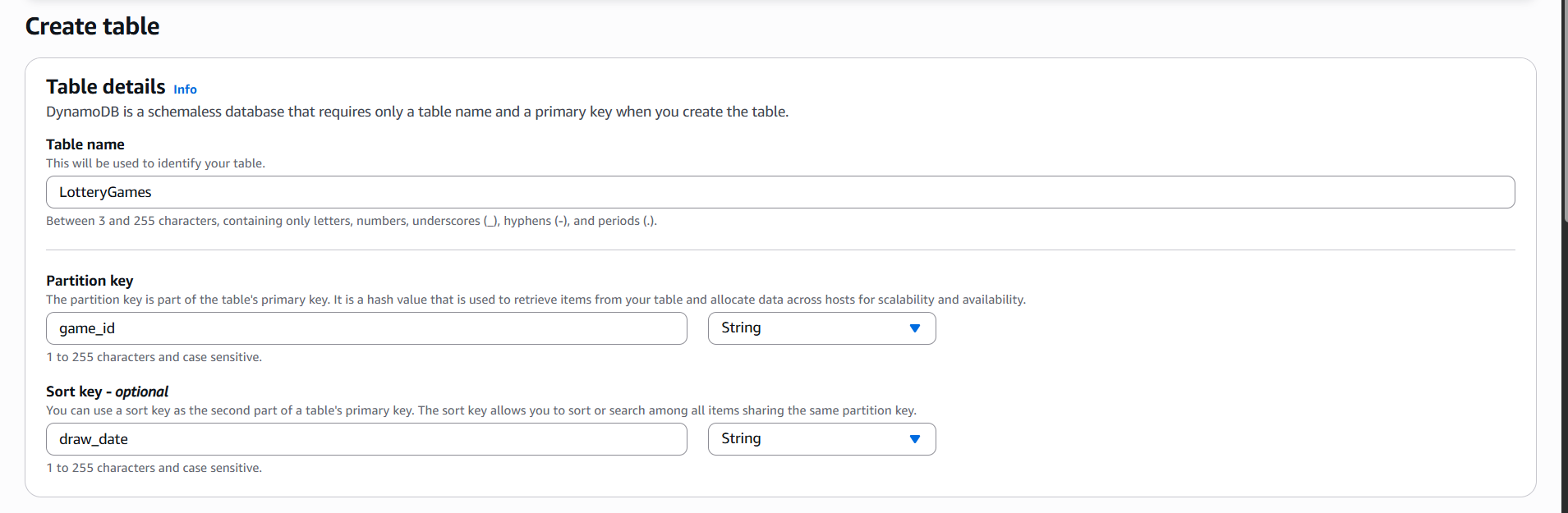
-Describe why it is the only way to connect to Aurora from local PC oppose to Amazon RDS MySQL.

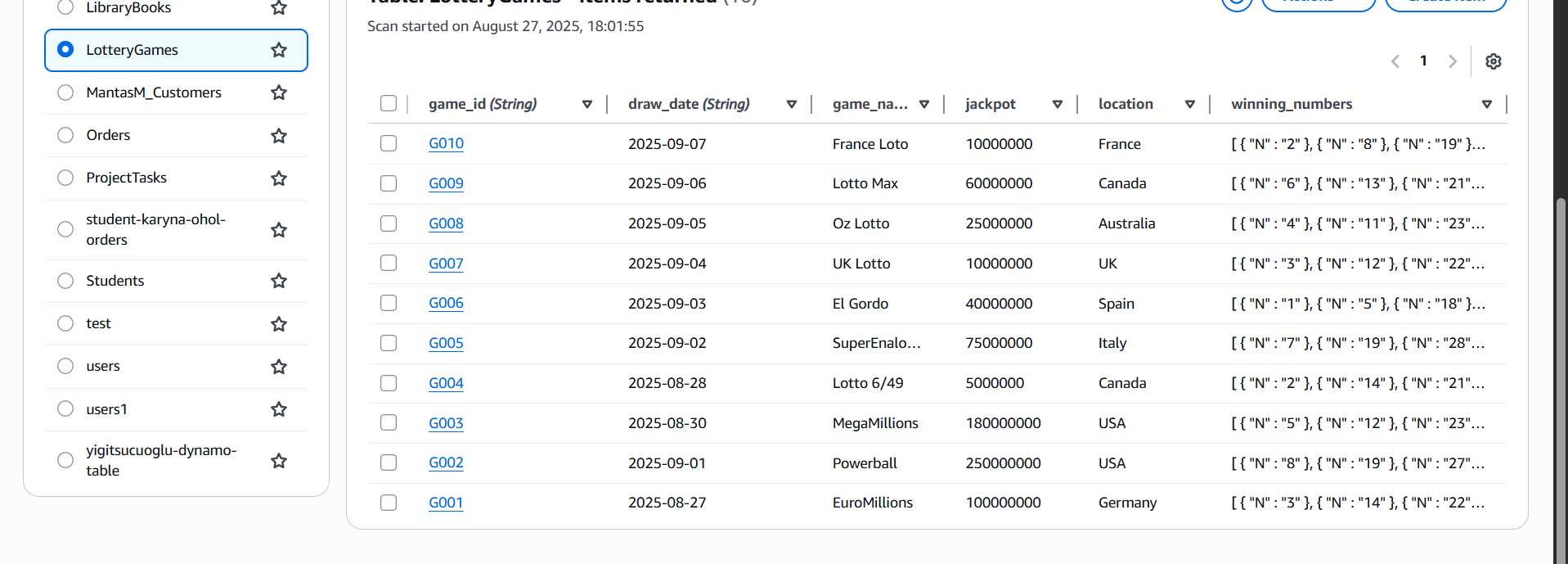


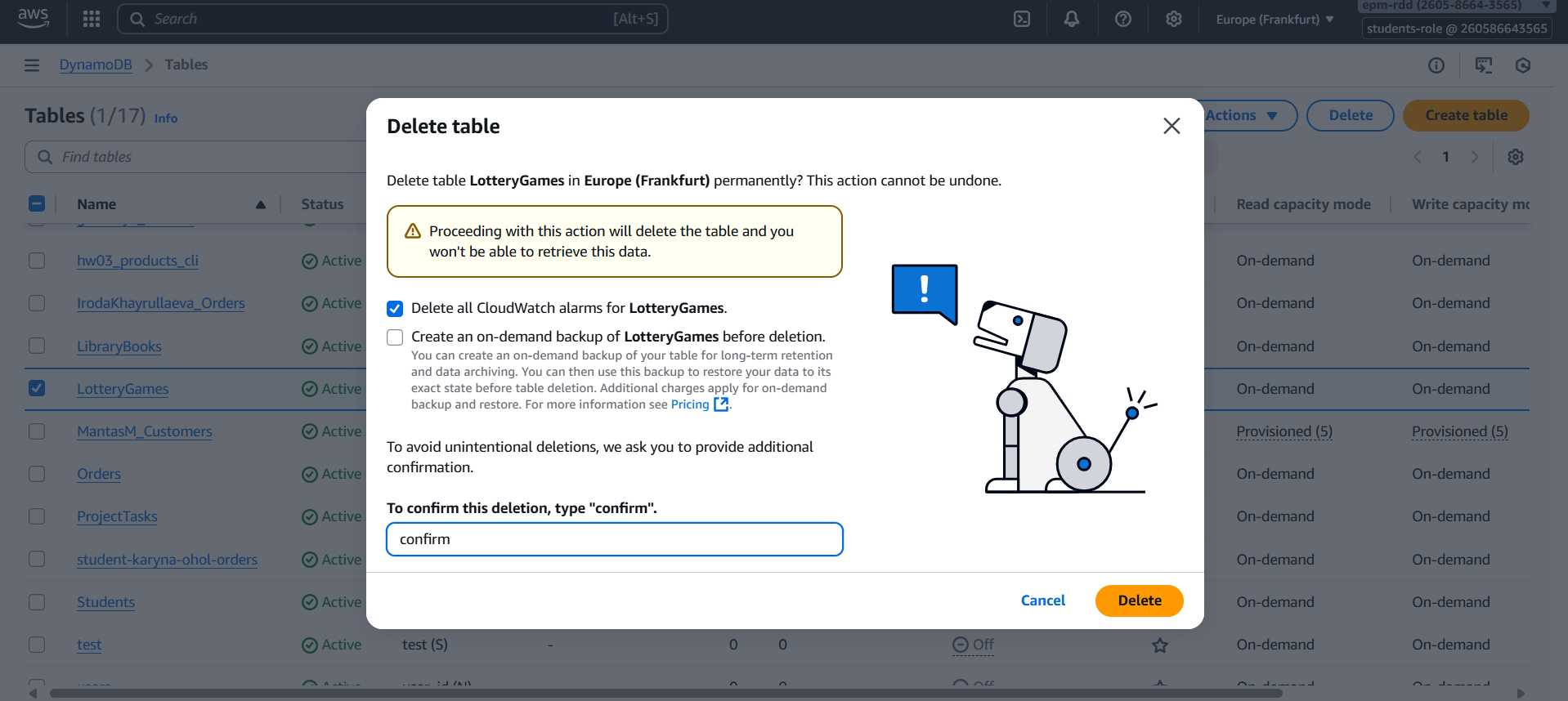
Unlike a standard Amazon RDS MySQL instance, which can be made publicly accessible and connected to directly from a local PC, Amazon Aurora does not allow this type of connection. Aurora clusters are designed to run entirely inside a VPC for security and high availability, and their endpoints are not exposed to the public internet. This means that to reach an Aurora database from outside AWS, you must first connect into the VPC using a bastion (jump) server, VPN, or AWS tools like Query Editor or Cloud9. In our case, we used an EC2 instance as a bastion server and created an SSH tunnel, which allowed us to securely forward traffic from our local SQL client to the private Aurora endpoint. This setup ensures that Aurora remains protected inside the AWS network, unlike RDS MySQL, which can be made directly accessible if needed.

# TASK 3

- Add table in dynamodb using AWS Console and populate it manually. Make screenshots and delete it. All further AWS Dynamodb manipulations should be using api calls from aws cli (follow the link):







- Create table in AWS DynamoDb. Primary key in this table is mandatory. Please explain why you choose this primary key for your table (help: <https://aws.amazon.com/ru/blogs/database/choosing-the-right-dynamodb-partition-key/>)

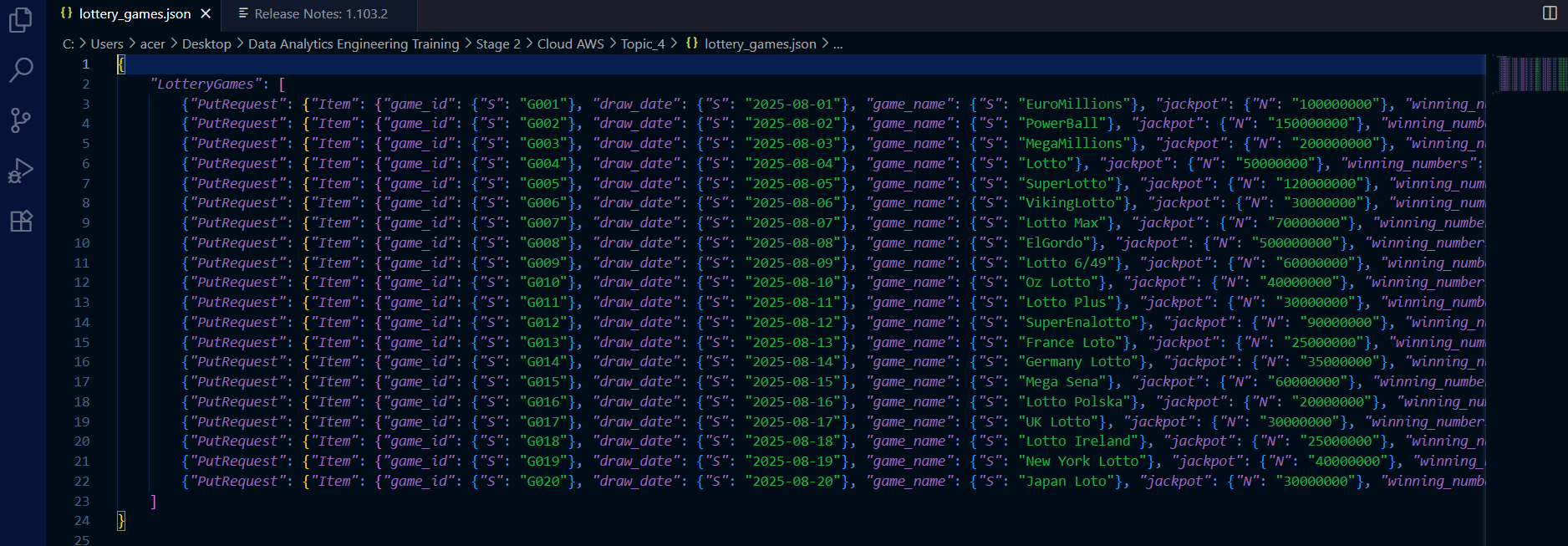


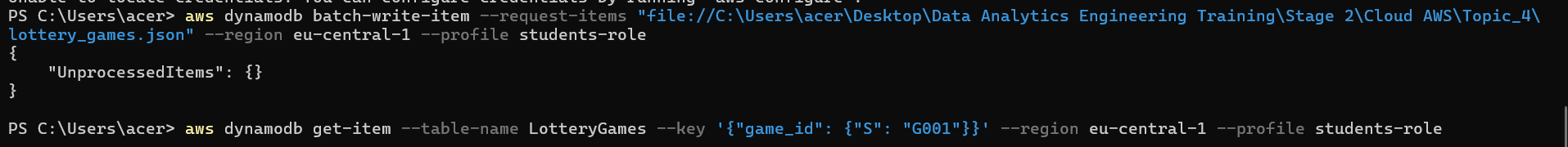
For our LotteryGames DynamoDB table, we chose **game\_id as the primary key** because every lottery game needs a unique identifier. In DynamoDB, the partition key serves as this unique identifier and determines how data is distributed across storage nodes. By using game\_id, we ensure that each entry is distinct and can be retrieved directly without scanning the entire table, making lookups extremely fast.

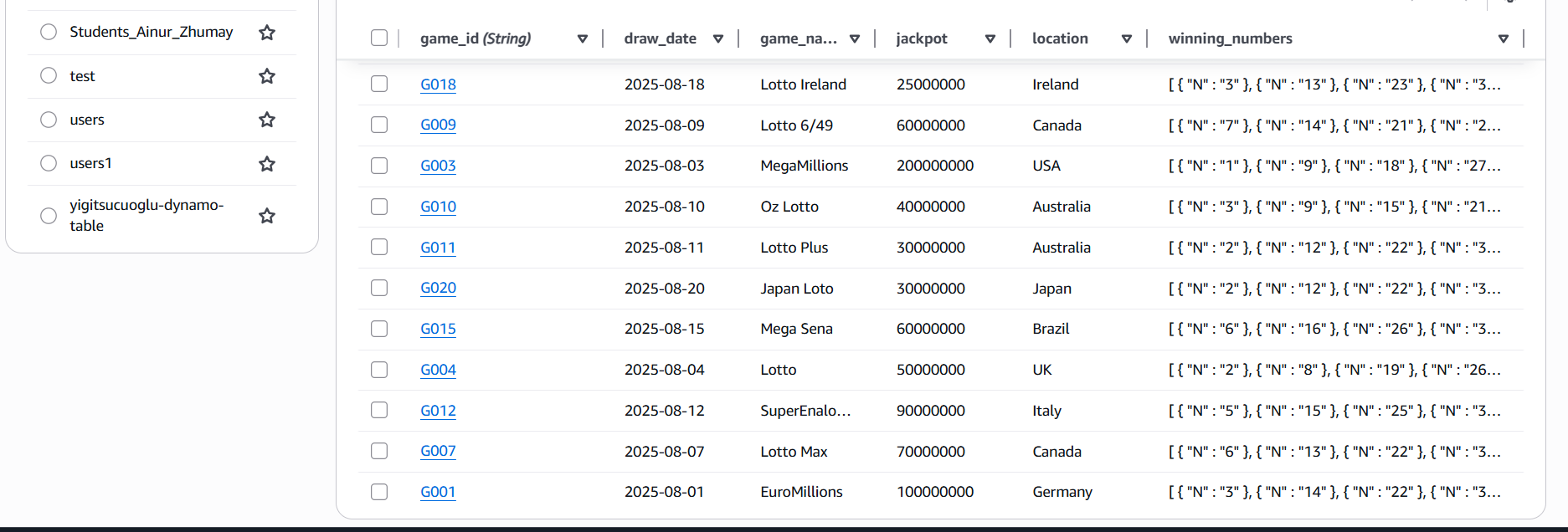
Additionally, game\_id provides **even data distribution**, which is crucial for scalability in DynamoDB. Each game is stored based on the hash of its game\_id, preventing hotspots and allowing the table to handle a large number of entries efficiently. While we could have added a sort key like draw\_date to query multiple draws for the same game, for our purposes a single partition key is sufficient to uniquely identify each lottery game.

In short, game\_id was chosen because it uniquely identifies each record, enables fast and efficient retrieval, and supports DynamoDB’s distributed architecture.

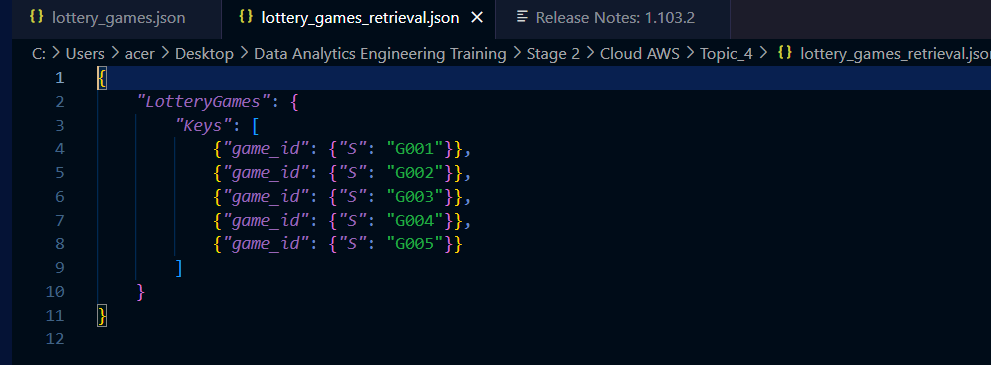
-Add 20+ rows to the table.

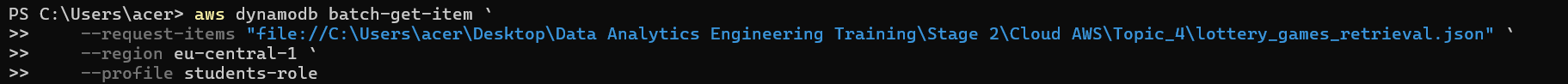


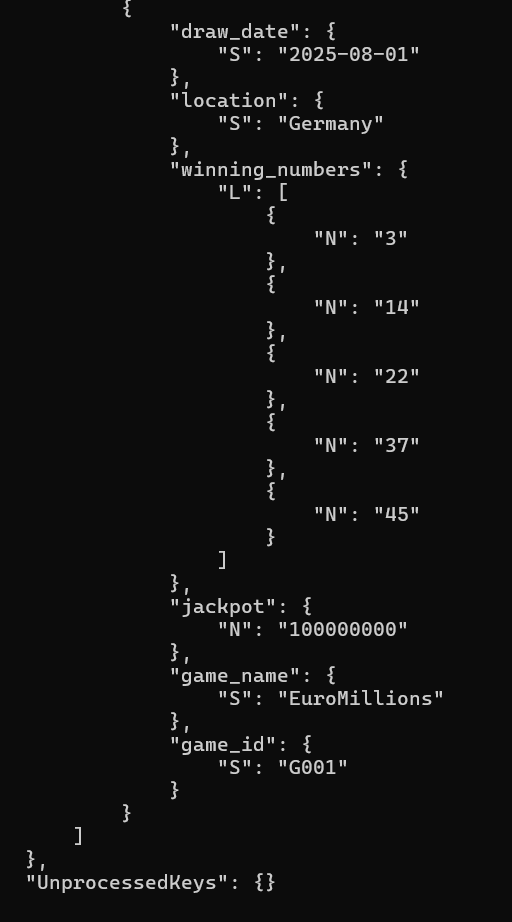
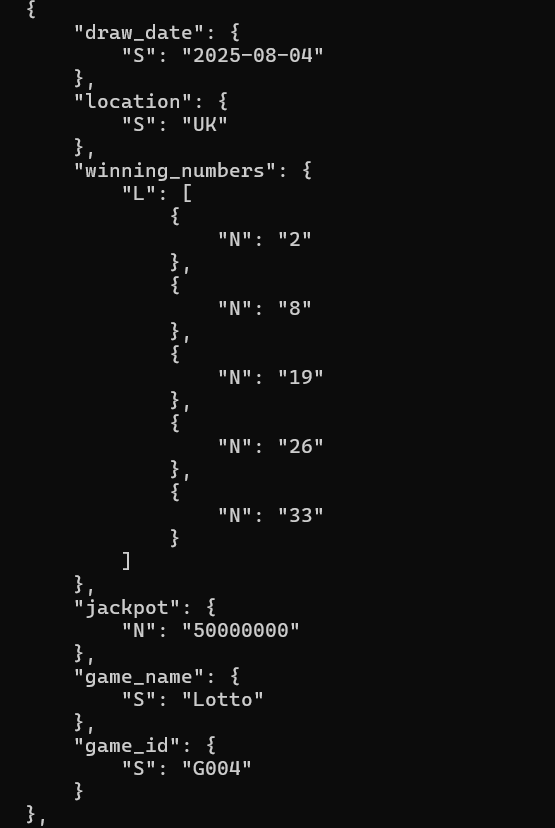
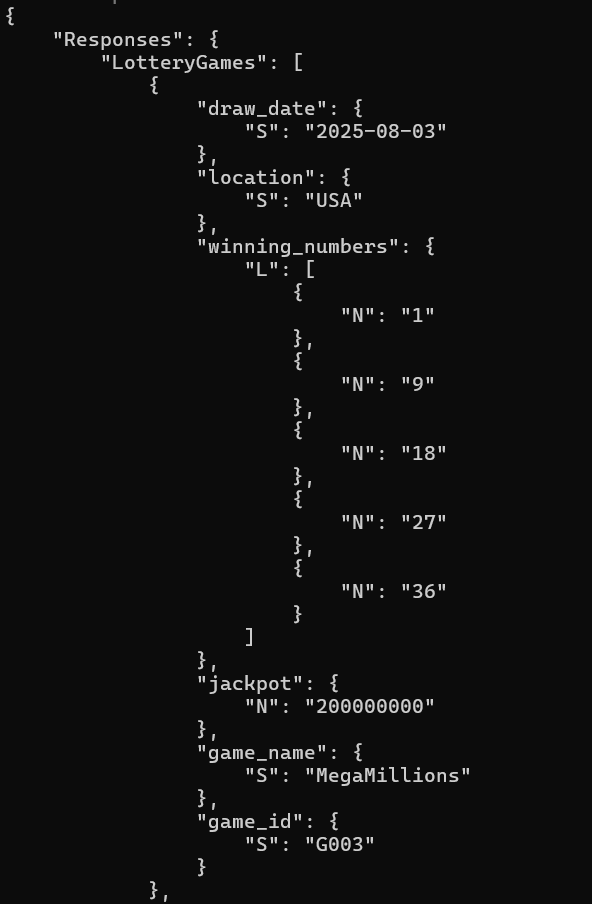




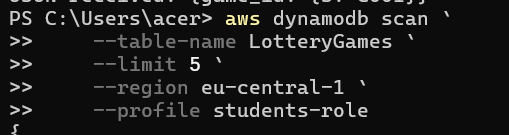
-Retrieve 5+ rows from the table using keys.

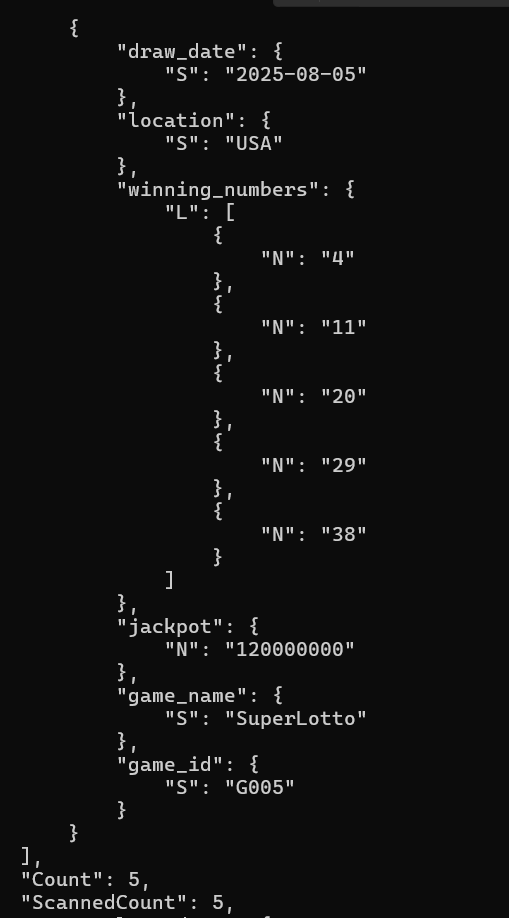
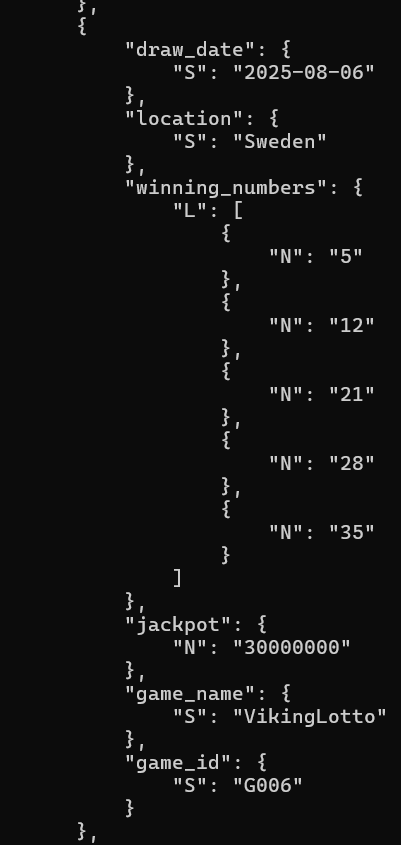
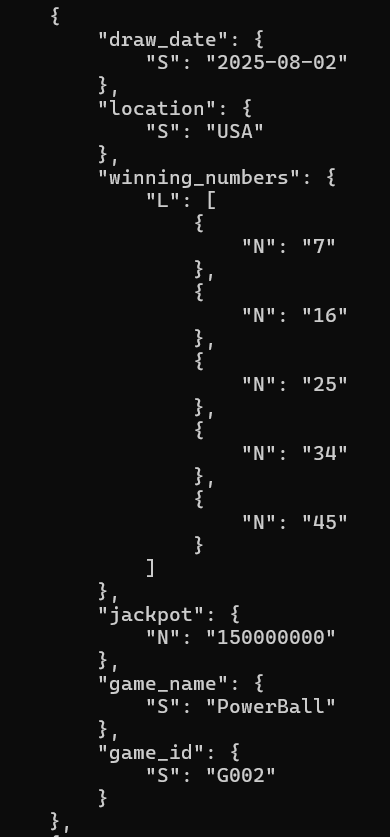
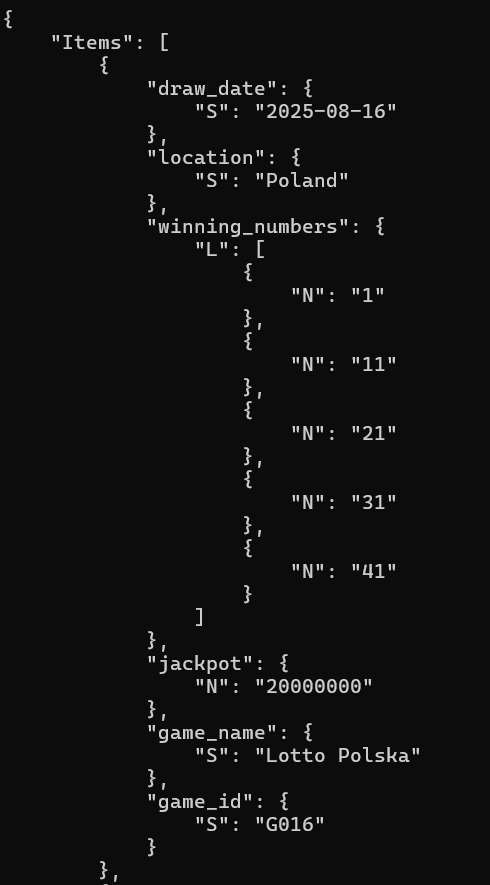


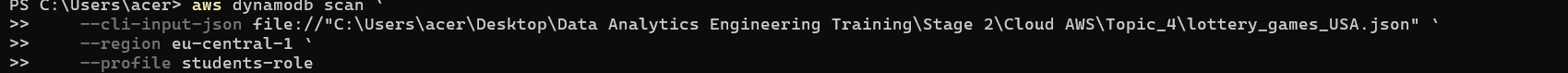




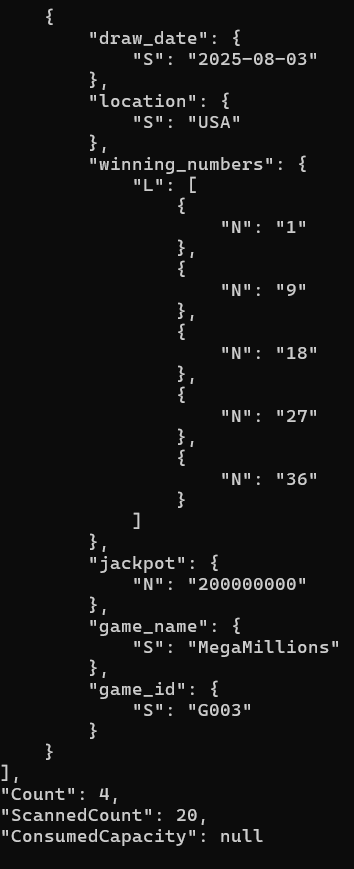
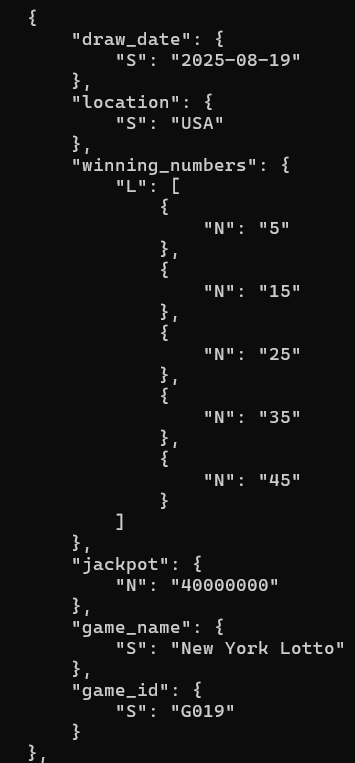
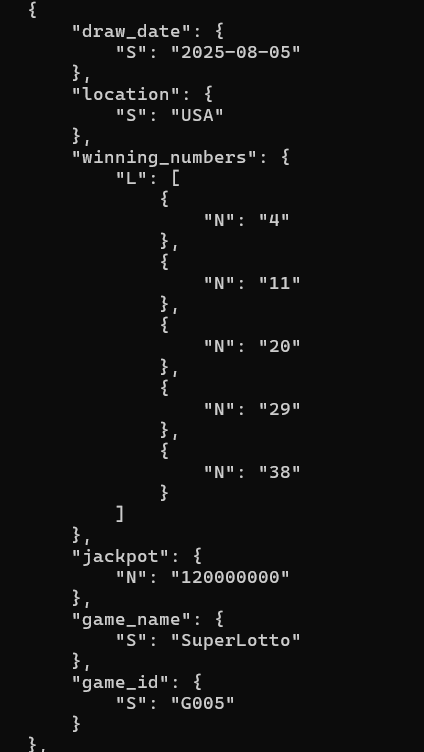
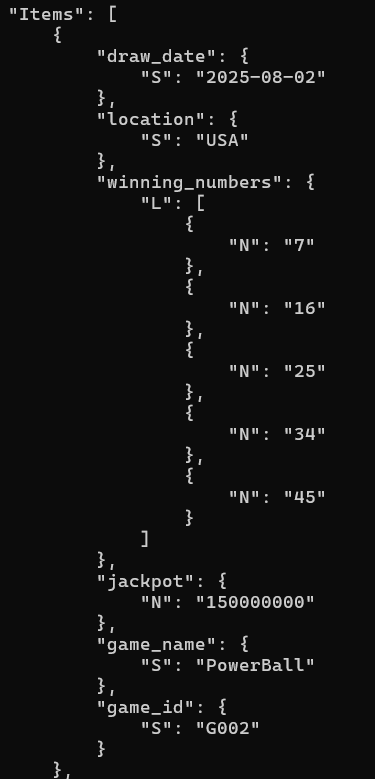
-Make several selects using syntax showed you on the lecture.











-Delete 2 rows from the table.

