Assignment 3 Write-Up
Peter Tsanev
CS66500, Fall 2023
12/1/23

Github Url: https://github.com/tsanevp/CS6650-Assignment3

Server Design

The server architecture I devised for this assignment is elegantly simple. Leveraging Java's servlet application functionality, I crafted two servlets adhering to the Model-View-Controller design pattern. One servlet manages incoming POST requests for creating new albums, while the other addresses POST requests for album reviews. Further details about each servlet and the corresponding classes they employ are elaborated below.

Major Classes

AlbumsServlet:

The AlbumsServlet class is a crucial component in the server architecture, designed for handling incoming POST requests related to album creation. Leveraging Java servlet functionality and adhering to a Model-View-Controller design pattern, the servlet manages the reception and processing of requests. It validates and parses multipart form data, including album profile details and image information. The servlet then generates a UUID for the album, constructs JSON representations of image metadata and album profile, and posts this information to my MySQL database. Additionally, the class includes methods for parsing album profile data, extracting values using regular expressions, checking image content type, and validating the request endpoint. The structured approach and integration with database operations make the AlbumsServlet a key component in the overall functionality of the web application.

ReviewsServlet:

The ReviewServlet class plays a pivotal role in the server architecture, specifically handling incoming POST requests related to album reviews. The servlet establishes a connection to a RabbitMQ message broker using the RabbitMQService class. Upon initialization, a channel pool is created to manage communication with the RabbitMQ Exchange named "REVIEW_EXCHANGE efficiently." In the doPost method, the servlet processes the incoming request by extracting relevant information from the URL path, such as the review type ("like" or "dislike") and the album ID. It then constructs a JSON-formatted message containing this data and publishes it to the RabbitMQ Exchange using the appropriate channel from the pool. If successful, a "Review sent" response is returned to the client with a status code of 201 (Created). In case of errors, appropriate status codes and messages are returned adhering to the SwaggerAPI specs. The isUrlValid method validates the URI arguments to ensure they match the expected pattern for review types ("like" or "dislike") and that the length of the URL parts is as expected. Overall, the ReviewServlet class showcases a streamlined and effective design for handling album review requests, leveraging RabbitMQ for asynchronous communication, and maintaining a pool of channels for efficiency. This servlet is a crucial component in the web application's functionality, ensuring the seamless processing of album reviews through messaging queues.

MySQLService:

The MySQLService class is a critical component in managing database connections for the MySQL database in the web application. A Hikari connection pool is initialized in the constructor, providing an efficient and reliable way to handle database connections. The class exposes a method, getConnectionPool(), to retrieve the established pool of connections.

The underlying connect() method configures the Hikari connection pool by specifying the MySQL database URL, username, password, and connection pool parameters, such as minimum and maximum idle connections. Upon successful connection, a message is printed to the console.

The close() method allows for the graceful closure of the connection pool when needed, ensuring proper resource management.

Overall, the MySQLService class encapsulates the functionality for establishing and closing connections to the MySQL database, contributing to the robustness and scalability of the web application's data access layer.

This class is used in the AlbumsServlet class to POST new albums to the MySQL database.

RabbitMQService:

The RabbitMQService class is pivotal in facilitating communication with RabbitMQ, serving as a dedicated service for managing channels within the web application. In the constructor, default values are set for the channel pool size, including methods to create and close a pool of channels for RabbitMQ.

The createChannelPool method establishes a connection to RabbitMQ using the provided host information and creates a pool of channels. The method iteratively creates channels, declares the specified exchange with its type, and adds each channel to the ConcurrentLinkedDeque, forming a pool. Any encountered exceptions during the connection or channel creation process result in a runtime exception being thrown.

The closeChannelPool method gracefully closes each channel within the provided pool, handling potential IOExceptions and TimeoutExceptions by encapsulating them in a runtime exception. This ensures proper cleanup of resources associated with RabbitMQ channels when needed.

Overall, the RabbitMQService class encapsulates the logic for establishing and closing a pool of channels to interact with RabbitMQ, enhancing the efficiency and reliability of asynchronous messaging within the web application.

This class is used in the ReviewServlet class to get channels to publish reviews on the RabbitMQ Exchange.

AlbumController:

The AlbumController class serves as a controller component responsible for managing interactions between the AlbumsServlet and the MySQL database concerning album-related operations. The class includes a method, postToDatabase, which updates the database with album information provided, using a UUID as a key. The method takes a connection to the MySQL database, a UUID, information on the uploaded image (imageData), and the album profile (albumProfile). It constructs and executes a SQL INSERT query to store the album details, including the image data and album profile, in the 'albumRequests' table. The method returns the row count of the update, indicating the success of the database update. Any potential SQLException is thrown, signaling a database access error. Overall, the AlbumController class encapsulates the logic for posting album-related data to the MySQL database, contributing to the data management functionality of the web application.

Packages

The Server imports and utilizes multiple existing Java packages and dependencies. They are listed and described below.

- javax.servlet-api - Allows us to create our servlet web applicaiton.
- swagger-java-client - Allows us to use the Default and Like APIs generated by SwaggerAPI.
- gson - Converts Album and Image data into JSON format for JSON serialization.
- mysqp-connector-java - Used to create MySQL connections and database queries.
- HikariCP - Used to create a connection pool of MySQL connections.
- amqp-client - Used to create and connect to a RabitMQ messaging service.

Relationships

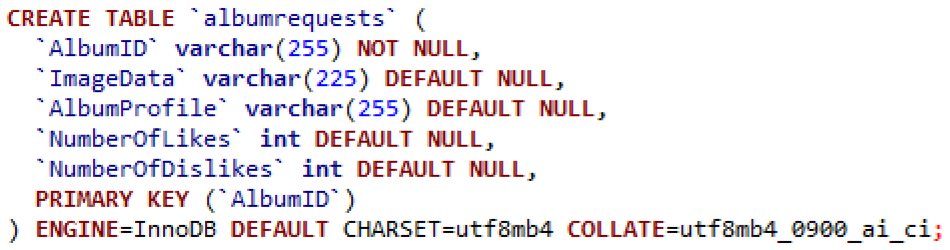
The relationships on my server side are fairly simple.

My ReviewServlet uses an instance of my RabbitMQService to create a pool of RabbitMQ channels from a single connection. When my ReviewServlet POST method is called, a channel is removed from the pool, used to send the message to the exchange, and then returned to the pool.

My AlbumsServlet uses an instance of my MySQLService to create a pool of connections to my MySQL database. When an album needs to be posted, a connection is taken from the pool, then an instance of my AlbumController is used to call postToDatabase, which inserts the information passed to the database. After insertion, the connection is returned to the pool.

Data Model

Creating a data model for this assignment was fairly straightforward. In my servlet, the request process is as follows: send a POST request, create a UUID inside the servlet, return the UUID in the POST response (per API spec), parse that response in my client to get the UUID, and immediately use the UUID to send two like and one dislike review requests with that UUID. In my database, each UUID is used as a key to retrieve my ImageData, AlbumProfile, NumberOfLikes, and NumberOfDislikes information. This is explained in further detail in the following sections. See Image 1 for a model example.



```
CREATE TABLE `albumrequests` (
  `AlbumID` varchar(255) NOT NULL,
  `ImageData` varchar(225) DEFAULT NULL,
  `AlbumProfile` varchar(255) DEFAULT NULL,
  `NumberOfLikes` int DEFAULT NULL,
  `NumberOfDislikes` int DEFAULT NULL,
  PRIMARY KEY (`AlbumID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Image 1: Database data model

Image

For this assignment, I used the image provided to us by Ian. The image size is 3,475 bytes. See the Image 2 below.

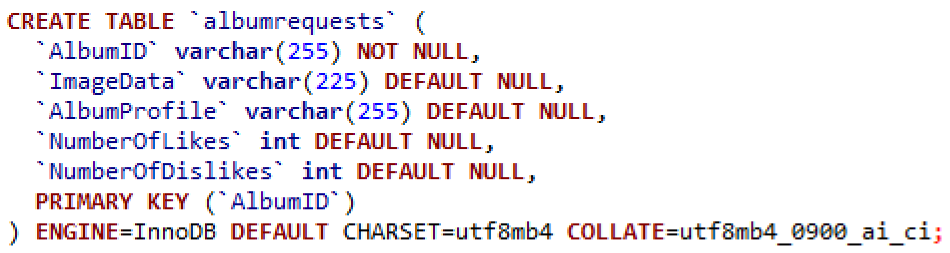
Image 2: The image used in Assignment 2.

Database Choice

For this assignment, I continued to use AWS’s RDS service to create a MySQL database.

MySQL Configuration

Typically, NoSQL databases are used with data in a flat key-value structure. However, with simple data like the ImageData, AlbumProfile, NumberOfLikes, and NumberOfDislikes, a correctly configured MySQL table suffices. To create a fast and efficient low-latency key-value store using Amazon RDS for MySQL, each entry to my table had a UUID named AlbumID defined as a Primary Key. I created my table, as seen below.



```
CREATE TABLE `albumrequests` (
  `AlbumID` varchar(255) NOT NULL,
  `ImageData` varchar(225) DEFAULT NULL,
  `AlbumProfile` varchar(255) DEFAULT NULL,
  `NumberOfLikes` int DEFAULT NULL,
  `NumberOfDislikes` int DEFAULT NULL,
  PRIMARY KEY (`AlbumID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Image 3: SQL statement used to create albumRequests table.

Defining AlbumID as a Primary Key, an associated index is created, leading to fast query performance. ImageData and AlbumProfile are stored as JSON strings rather than the JSON data type to improve performance. NumberOfLikes and NumberOfDislikes are both stored as integers. Then, as long as I have a valid UUID, I can query any of my columns. My Review POST requests query either NumberOfLikes or NumberOfDislikes and then increment the current value by 1. When an album is first created, both values are set to zero.

RabbitMQ Message Processing

Sending Messages

As mentioned, my ReviewServlet establishes a connection to a RabbitMQ message broker using the RabbitMQService class. Upon initialization, a channel pool is created to manage communication with the RabbitMQ Exchange named "REVIEW_EXCHANGE efficiently." The Exchange Type is defined as “direct” since, on my consumer side, I have a queue to handle “like” reviews and one to handle “dislike” reviews. In the doPost method, the servlet processes the incoming request by extracting relevant information from the URL path, such as the review type ("like" or "dislike") and the album ID. It then constructs a JSON-formatted message containing this data and publishes it to the RabbitMQ Exchange using a channel from the pool and listing the review type as the routing key.

Receiving Messages

My RabbitMQ consumers handle messages published to the queues and are in my ReviewService Project. The Project follows a similar architecture to my Server and has the following four classes: ReviewController, MySQLService, ReviewConsumer, and ReviewRunnable.

The ReviewConsumer and ReviewRunnable classes collectively constitute a RabbitMQ message consumer system designed to handle and process album review messages asynchronously.

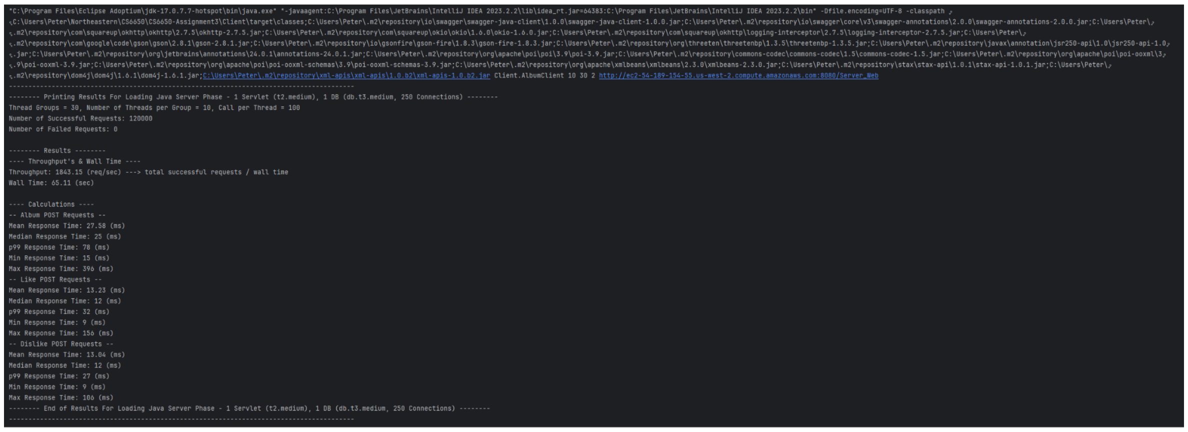
In ReviewConsumer, a fixed-size thread pool is created for both "like" and "dislike" queues, and multiple instances of ReviewRunnable are executed concurrently to consume messages from these queues. The ReviewRunnable class encapsulates the logic for receiving and processing individual messages. Each thread within the pool establishes a RabbitMQ connection, declares the necessary exchange and queues, and then enters a waiting state for incoming messages. Upon receiving a message, the handleMessage callback method is invoked, which deserializes the JSON content of the message, extracts the album ID and review type, and increments the corresponding review count in the MySQL database using the ReviewController. In case of exceptions during message processing or database interactions, appropriate error handling is implemented. Each message is auto-acknowledged from the queue to prevent queue buildup while waiting for database updates.

The setup ensures that multiple consumers are concurrently processing messages from the "like" and "dislike" queues, allowing for efficient and parallelized handling of album review updates. Thread pools facilitate concurrency and responsiveness, enabling the system to scale with increased message load. A shutdown hook is also registered to close the MySQL database connection pool upon application termination. Overall, this architecture adheres to the principles of asynchronous message processing, enhancing the responsiveness and scalability of the system.

Results

Output Window Results

10/30/2 Configuration Run 1



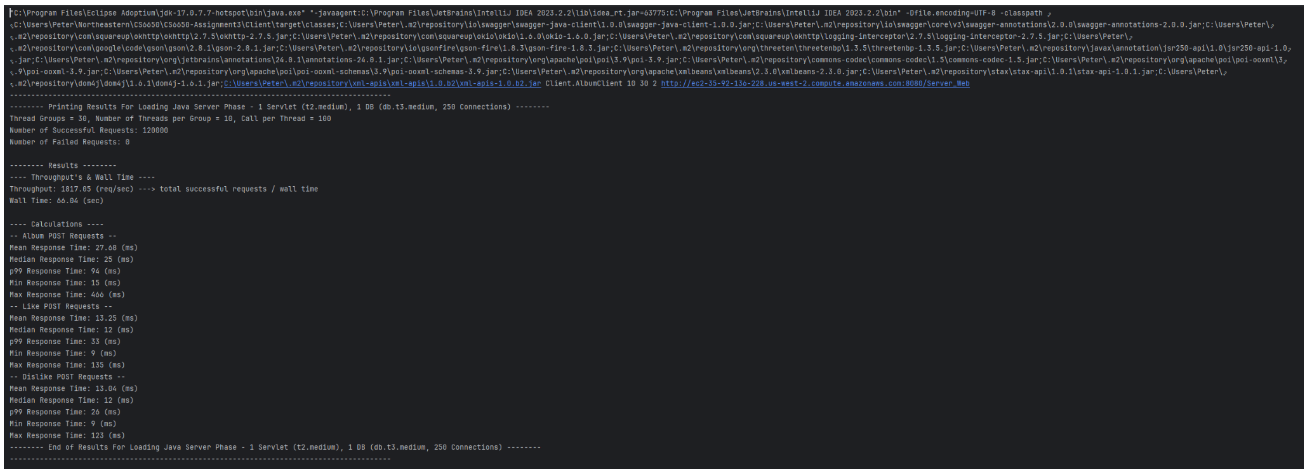
```
"C:\Program Files\Eclipse Adoptium\jdk-17.0.7.7-hotspot\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.2\lib\idea_rt.jar=64383:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.2\bin" -Dfile.encoding=UTF-8 -classpath 
C:\Users\Peter\Northeastern\CS6650\CS6650-Assignment3\Client\target\classes;C:\Users\Peter\.m2\repository\io\swagger\swagger-java-client\1.0.0\swagger-java-client-1.0.0.jar;C:\Users\Peter\.m2\repository\io\swagger\core\v3\swagger-annotations\2.0.0\swagger-annotations-2.0.0.jar;C:\Users\Peter\
.m2\repository\com\squareup\okhttp\okhttp\2.7.5\okhttp-2.7.5.jar;C:\Users\Peter\.m2\repository\com\squareup\okio\okio\1.6.0\okio-1.6.0.jar;C:\Users\Peter\.m2\repository\com\squareup\okhttp\logging-interceptor\2.7.5\logging-interceptor-2.7.5.jar;C:\Users\Peter\
.m2\repository\com\google\code\gson\gson\2.8.1\gson-2.8.1.jar;C:\Users\Peter\.m2\repository\io\gsonfire\gson-fire\1.8.3\gson-fire-1.8.3.jar;C:\Users\Peter\.m2\repository\org\threeten\threetenbp\1.3.5\threetenbp-1.3.5.jar;C:\Users\Peter\.m2\repository\javax\annotation\jsr250-api\1.0\jsr250-api-1.0
.jar;C:\Users\Peter\.m2\repository\org\jetbrains\annotations\24.0.1\annotations-24.0.1.jar;C:\Users\Peter\.m2\repository\org\apache\poi\poi\3.9\poi-3.9.jar;C:\Users\Peter\.m2\repository\commons-codec\commons-codec\1.5\commons-codec-1.5.jar;C:\Users\Peter\.m2\repository\org\apache\poi\poi-ooxml\3
.9\poi-ooxml-3.9.jar;C:\Users\Peter\.m2\repository\org\apache\poi\poi-ooxml-schemas\3.9\poi-ooxml-schemas-3.9.jar;C:\Users\Peter\.m2\repository\org\apache\xmlbeans\xmlbeans\2.3.0\xmlbeans-2.3.0.jar;C:\Users\Peter\.m2\repository\stax\stax-api\1.0.1\stax-api-1.0.1.jar;C:\Users\Peter\
.m2\repository\dom4j\dom4j\1.6.1\dom4j-1.6.1.jar;C:\Users\Peter\.m2\repository\xml-apis\xml-apis\1.0.b2\xml-apis-1.0.b2.jar Client.AlbumClient 10 30 2 http://ec2-54-189-154-55.us-west-2.compute.amazonaws.com:8080/Server_Web
------------------------------------------------------------------------------------
-------- Printing Results For Loading Java Server Phase - 1 Servlet (t2.medium), 1 DB (db.t3.medium, 250 Connections) --------
Thread Groups = 30, Number of Threads per Group = 10, Call per Thread = 100
Number of Successful Requests: 120000
Number of Failed Requests: 0

-------- Results --------
---- Throughput's & Wall Time ----
Throughput: 1843.15 (req/sec) ---> total successful requests / wall time
Wall Time: 65.11 (sec)

---- Calculations ----
-- Album POST Requests --
Mean Response Time: 27.58 (ms)
Median Response Time: 25 (ms)
p99 Response Time: 78 (ms)
Min Response Time: 15 (ms)
Max Response Time: 396 (ms)
-- Like POST Requests --
Mean Response Time: 13.23 (ms)
Median Response Time: 12 (ms)
p99 Response Time: 32 (ms)
Min Response Time: 9 (ms)
Max Response Time: 156 (ms)
-- Dislike POST Requests --
Mean Response Time: 13.04 (ms)
Median Response Time: 12 (ms)
p99 Response Time: 27 (ms)
Min Response Time: 9 (ms)
Max Response Time: 106 (ms)
-------- End of Results For Loading Java Server Phase - 1 Servlet (t2.medium), 1 DB (db.t3.medium, 250 Connections) --------
------------------------------------------------------------------------------------
```

Image 4: Screenshot of output window for 10/30/2 run configuration.

10/30/2 Configuration - Run 2



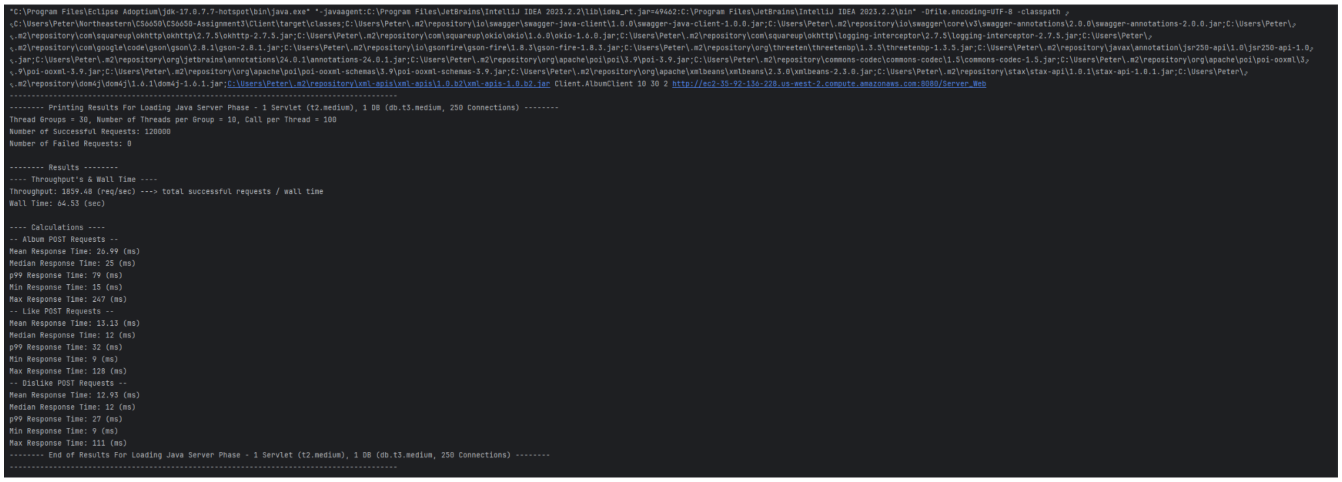
```
"C:\Program Files\Eclipse Adoptium\jdk-17.0.7.7-hotspot\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.2\lib\idea_rt.jar=63775:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.2\bin" -Dfile.encoding=UTF-8 -classpath 
C:\Users\Peter\Northeastern\CS6650\CS6650-Assignment3\Client\target\classes;C:\Users\Peter\.m2\repository\io\swagger\swagger-java-client\1.0.0\swagger-java-client-1.0.0.jar;C:\Users\Peter\.m2\repository\io\swagger\core\v3\swagger-annotations\2.0.0\swagger-annotations-2.0.0.jar;C:\Users\Peter\
.m2\repository\com\squareup\okhttp\okhttp\2.7.5\okhttp-2.7.5.jar;C:\Users\Peter\.m2\repository\com\squareup\okio\okio\1.6.0\okio-1.6.0.jar;C:\Users\Peter\.m2\repository\com\squareup\okhttp\logging-interceptor\2.7.5\logging-interceptor-2.7.5.jar;C:\Users\Peter\
.m2\repository\com\google\code\gson\gson\2.8.1\gson-2.8.1.jar;C:\Users\Peter\.m2\repository\io\gsonfire\gson-fire\1.8.3\gson-fire-1.8.3.jar;C:\Users\Peter\.m2\repository\org\threeten\threetenbp\1.3.5\threetenbp-1.3.5.jar;C:\Users\Peter\.m2\repository\javax\annotation\jsr250-api\1.0\jsr250-api-1.0
.jar;C:\Users\Peter\.m2\repository\org\jetbrains\annotations\24.0.1\annotations-24.0.1.jar;C:\Users\Peter\.m2\repository\org\apache\poi\poi\3.9\poi-3.9.jar;C:\Users\Peter\.m2\repository\commons-codec\commons-codec\1.5\commons-codec-1.5.jar;C:\Users\Peter\.m2\repository\org\apache\poi\poi-ooxml\3
.9\poi-ooxml-3.9.jar;C:\Users\Peter\.m2\repository\org\apache\poi\poi-ooxml-schemas\3.9\poi-ooxml-schemas-3.9.jar;C:\Users\Peter\.m2\repository\org\apache\xmlbeans\xmlbeans\2.3.0\xmlbeans-2.3.0.jar;C:\Users\Peter\.m2\repository\stax\stax-api\1.0.1\stax-api-1.0.1.jar;C:\Users\Peter\
.m2\repository\dom4j\dom4j\1.6.1\dom4j-1.6.1.jar;C:\Users\Peter\.m2\repository\xml-apis\xml-apis\1.0.b2\xml-apis-1.0.b2.jar Client.AlbumClient 10 30 2 http://ec2-35-92-136-228.us-west-2.compute.amazonaws.com:8080/Server_Web
--------------------------------------------------------------------------------
-------- Printing Results For Loading Java Server Phase - 1 Servlet (t2.medium), 1 DB (db.t3.medium, 250 Connections) --------
Thread Groups = 30, Number of Threads per Group = 10, Call per Thread = 100
Number of Successful Requests: 120000
Number of Failed Requests: 0

-------- Results --------
---- Throughput's & Wall Time ----
Throughput: 1817.05 (req/sec) ---> total successful requests / wall time
Wall Time: 66.04 (sec)

---- Calculations ----
-- Album POST Requests --
Mean Response Time: 27.68 (ms)
Median Response Time: 25 (ms)
p99 Response Time: 94 (ms)
Min Response Time: 15 (ms)
Max Response Time: 466 (ms)
-- Like POST Requests --
Mean Response Time: 13.25 (ms)
Median Response Time: 12 (ms)
p99 Response Time: 33 (ms)
Min Response Time: 9 (ms)
Max Response Time: 135 (ms)
-- Dislike POST Requests --
Mean Response Time: 13.04 (ms)
Median Response Time: 12 (ms)
p99 Response Time: 26 (ms)
Min Response Time: 9 (ms)
Max Response Time: 123 (ms)
-------- End of Results For Loading Java Server Phase - 1 Servlet (t2.medium), 1 DB (db.t3.medium, 250 Connections) --------
--------------------------------------------------------------------------------
```

Image 5: Screenshot of output window for 10/30/2 run configuration.

10/30/2 Configuration - Run 3



```
"C:\Program Files\Eclipse Adoptium\jdk-17.0.7.7-hotspot\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.2\lib\idea_rt.jar=49462:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.2\bin" -Dfile.encoding=UTF-8 -classpath 
C:\Users\Peter\Northeastern\CS6650\CS6650-Assignment3\Client\target\classes;C:\Users\Peter\.m2\repository\io\swagger\swagger-java-client\1.0.0\swagger-java-client-1.0.0.jar;C:\Users\Peter\.m2\repository\io\swagger\core\v3\swagger-annotations\2.0.0\swagger-annotations-2.0.0.jar;C:\Users\Peter\
.m2\repository\com\squareup\okhttp\okhttp\2.7.5\okhttp-2.7.5.jar;C:\Users\Peter\.m2\repository\com\squareup\okio\okio\1.6.0\okio-1.6.0.jar;C:\Users\Peter\.m2\repository\com\squareup\okhttp\logging-interceptor\2.7.5\logging-interceptor-2.7.5.jar;C:\Users\Peter\
.m2\repository\com\google\code\gson\gson\2.8.1\gson-2.8.1.jar;C:\Users\Peter\.m2\repository\io\gsonfire\gson-fire\1.8.3\gson-fire-1.8.3.jar;C:\Users\Peter\.m2\repository\org\threeten\threetenbp\1.3.5\threetenbp-1.3.5.jar;C:\Users\Peter\.m2\repository\javax\annotation\jsr250-api\1.0\jsr250-api-1.0
.jar;C:\Users\Peter\.m2\repository\org\jetbrains\annotations\24.0.1\annotations-24.0.1.jar;C:\Users\Peter\.m2\repository\org\apache\poi\poi\3.9\poi-3.9.jar;C:\Users\Peter\.m2\repository\commons-codec\commons-codec\1.5\commons-codec-1.5.jar;C:\Users\Peter\.m2\repository\org\apache\poi\poi-ooxml\3
.9\poi-ooxml-3.9.jar;C:\Users\Peter\.m2\repository\org\apache\poi\poi-ooxml-schemas\3.9\poi-ooxml-schemas-3.9.jar;C:\Users\Peter\.m2\repository\org\apache\xmlbeans\xmlbeans\2.3.0\xmlbeans-2.3.0.jar;C:\Users\Peter\.m2\repository\stax\stax-api\1.0.1\stax-api-1.0.1.jar;C:\Users\Peter\
.m2\repository\dom4j\dom4j\1.6.1\dom4j-1.6.1.jar;C:\Users\Peter\.m2\repository\xml-apis\xml-apis\1.0.b2\xml-apis-1.0.b2.jar Client.AlbumClient 10 30 2 http://ec2-35-92-136-228.us-west-2.compute.amazonaws.com:8080/Server_Web
--------------------------------------------------------------------------------
-------- Printing Results For Loading Java Server Phase - 1 Servlet (t2.medium), 1 DB (db.t3.medium, 250 Connections) --------
Thread Groups = 30, Number of Threads per Group = 10, Call per Thread = 100
Number of Successful Requests: 120000
Number of Failed Requests: 0

-------- Results --------
---- Throughput's & Wall Time ----
Throughput: 1859.48 (req/sec) ---> total successful requests / wall time
Wall Time: 64.53 (sec)

---- Calculations ----
-- Album POST Requests --
Mean Response Time: 26.99 (ms)
Median Response Time: 25 (ms)
p99 Response Time: 79 (ms)
Min Response Time: 15 (ms)
Max Response Time: 247 (ms)
-- Like POST Requests --
Mean Response Time: 13.13 (ms)
Median Response Time: 12 (ms)
p99 Response Time: 32 (ms)
Min Response Time: 9 (ms)
Max Response Time: 128 (ms)
-- Dislike POST Requests --
Mean Response Time: 12.93 (ms)
Median Response Time: 12 (ms)
p99 Response Time: 27 (ms)
Min Response Time: 9 (ms)
Max Response Time: 111 (ms)
-------- End of Results For Loading Java Server Phase - 1 Servlet (t2.medium), 1 DB (db.t3.medium, 250 Connections) --------
--------------------------------------------------------------------------------
```

Image 6: Screenshot of output window for 10/30/2 run configuration.

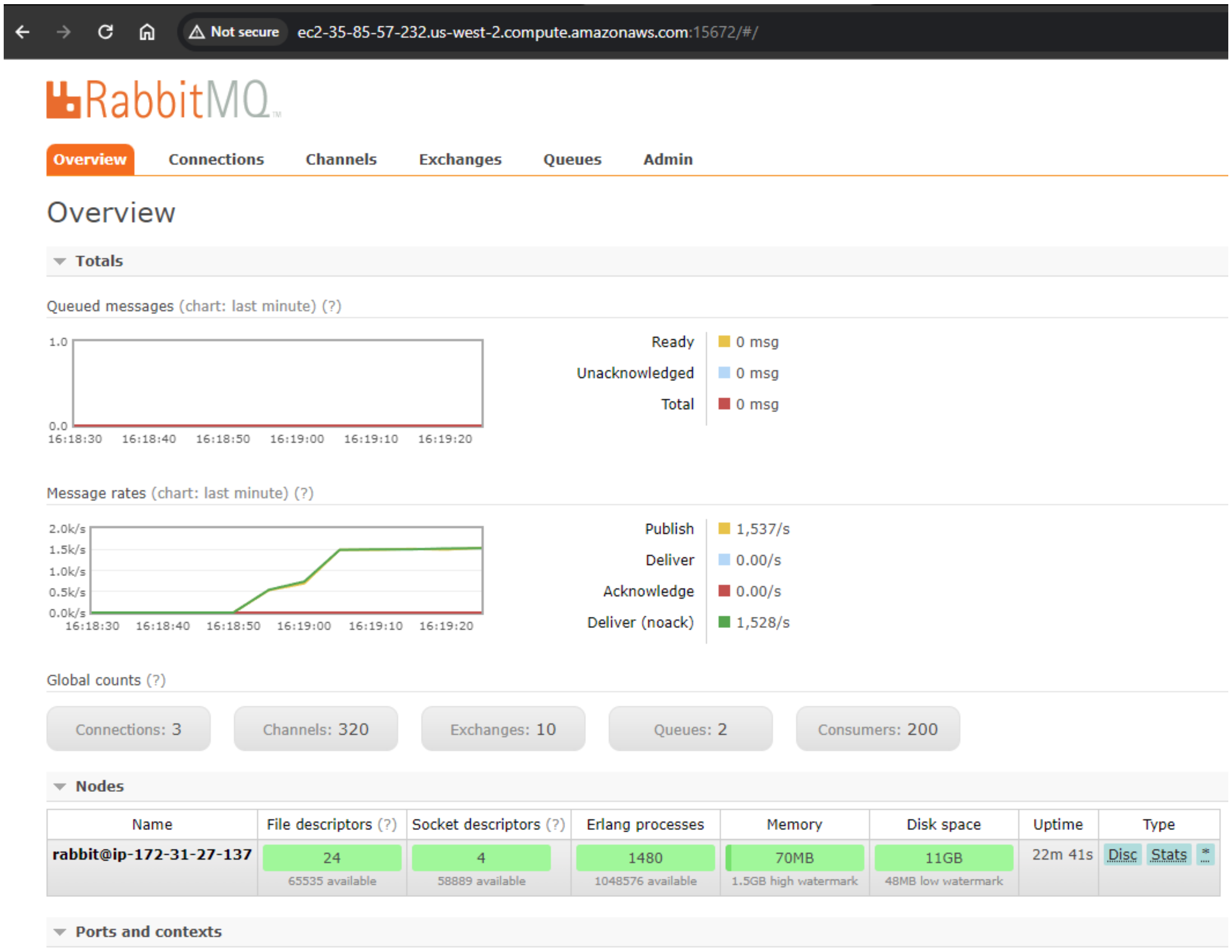
Output Window - Table Results

Table 1: Results for Part 3 of Assignment 3

	Overall Results				Album POST Calculated Response Times (ms)					Like POST Calculated Response Times (ms)					Dislike POST Calculated Response Times (ms)				
10/30/20 Config	# Successful Requests	# Failed Reqs	Through put (req/sec)	Wall Time (sec)	Mean	Median	p99	Min	Max	Mean	Median	p99	Min	Max	Mean	Median	p99	Min	Max
Run 1	120,000	0	1843	65.11	27	25	78	15	396	13	12	32	9	156	13	12	27	9	106
Run 2	120,000	0	1817	66.04	54	50	135	16	607	13	12	33	9	135	13	12	26	9	123
Run 3	120,000	0	1859	64.53	74	62	301	17	988	13	12	32	9	128	12	12	27	9	111

RabbitMQ 10/30/2 Run 1 Screenshots

Run 1 Overview Page



Overview

Totals

Queued messages (chart: last minute) (?)

Ready	🟨 0 msg
Unacknowledged	🟦 0 msg
Total	🟥 0 msg

Message rates (chart: last minute) (?)

Publish	🟨 1,537/s
Deliver	🟦 0.00/s
Acknowledge	🟥 0.00/s
Deliver (noack)	🟩 1,528/s

Global counts (?)

Connections: 3 | Channels: 320 | Exchanges: 10 | Queues: 2 | Consumers: 200

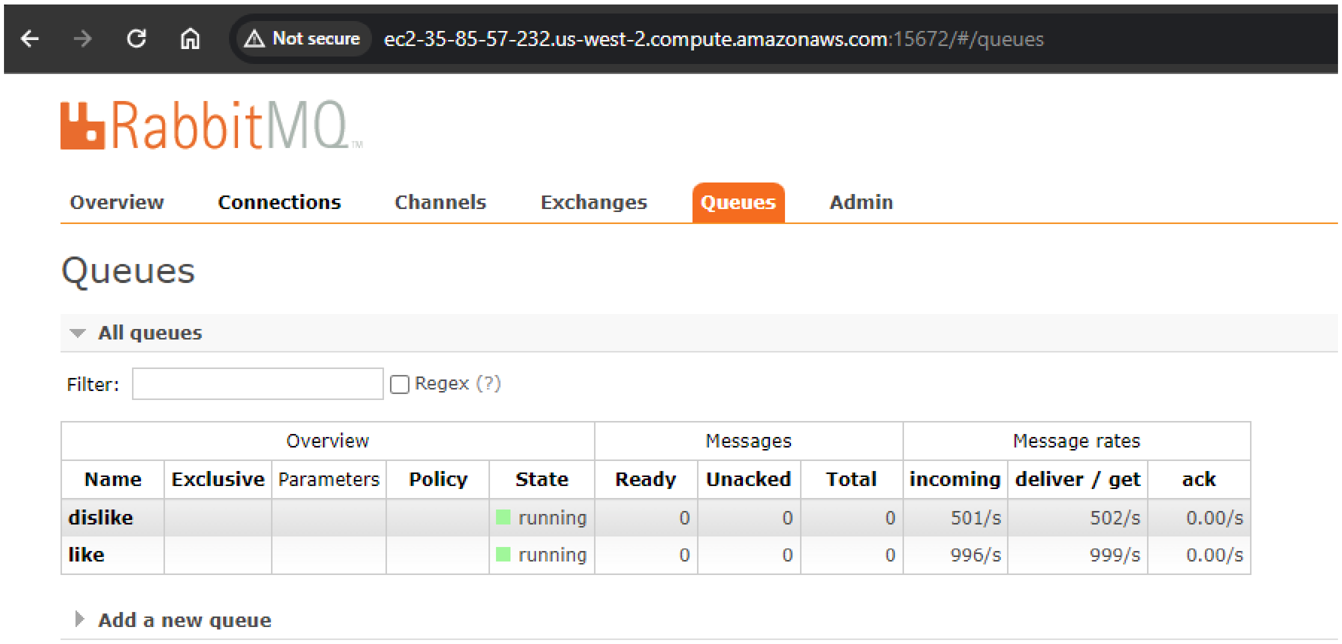
Nodes

Name	File descriptors (?)	Socket descriptors (?)	Erlang processes	Memory	Disk space	Uptime	Type
rabbit@ip-172-31-27-137	24 65535 available	4 58889 available	1480 1048576 available	70MB 1.5GB high watermark	11GB 48MB low watermark	22m 41s	Disc Stats *

Ports and contexts

Image 7: Screenshot of RabbitMQ Overview window for 10/30/2 run 1.

Run 1 Queue Page



Queues

All queues

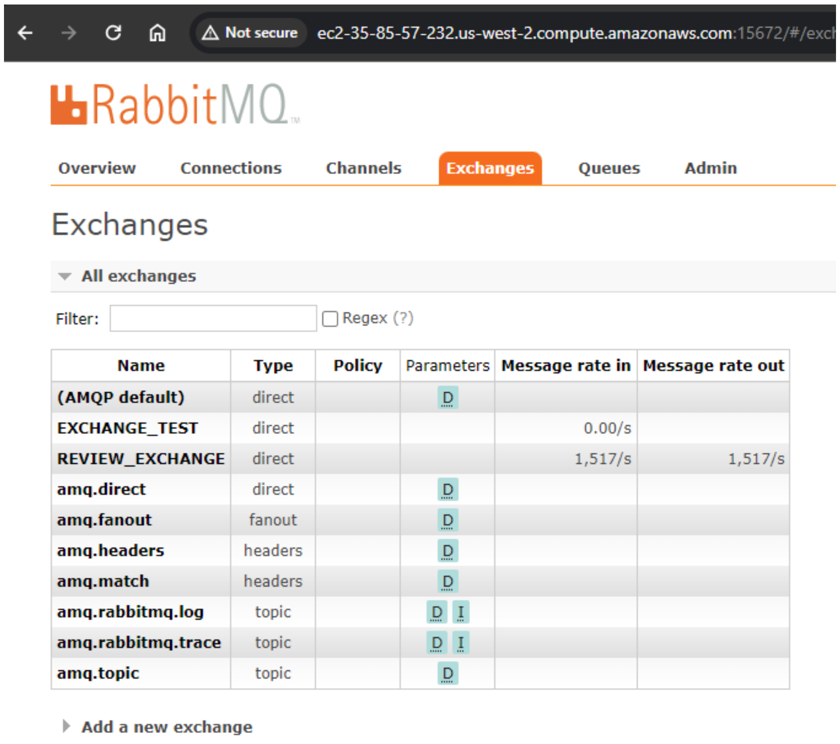
Filter: ☐ Regex (?)

Overview					Messages			Message rates		
Name	Exclusive	Parameters	Policy	State	Ready	Unacked	Total	incoming	deliver / get	ack
dislike				🟩 running	0	0	0	501/s	502/s	0.00/s
like				🟩 running	0	0	0	996/s	999/s	0.00/s

Add a new queue

Image 8: Screenshot of RabbitMQ Queues window for 10/30/2 run 1.

Run 1 Exchange Page



Exchanges

All exchanges

Filter: ☐ Regex (?)

Name	Type	Policy	Parameters	Message rate in	Message rate out
(AMQP default)	direct		D		
EXCHANGE_TEST	direct			0.00/s	
REVIEW_EXCHANGE	direct			1,517/s	1,517/s
amq.direct	direct		D		
amq.fanout	fanout		D		
amq.headers	headers		D		
amq.match	headers		D		
amq.rabbitmq.log	topic		D I		
amq.rabbitmq.trace	topic		D I		
amq.topic	topic		D		

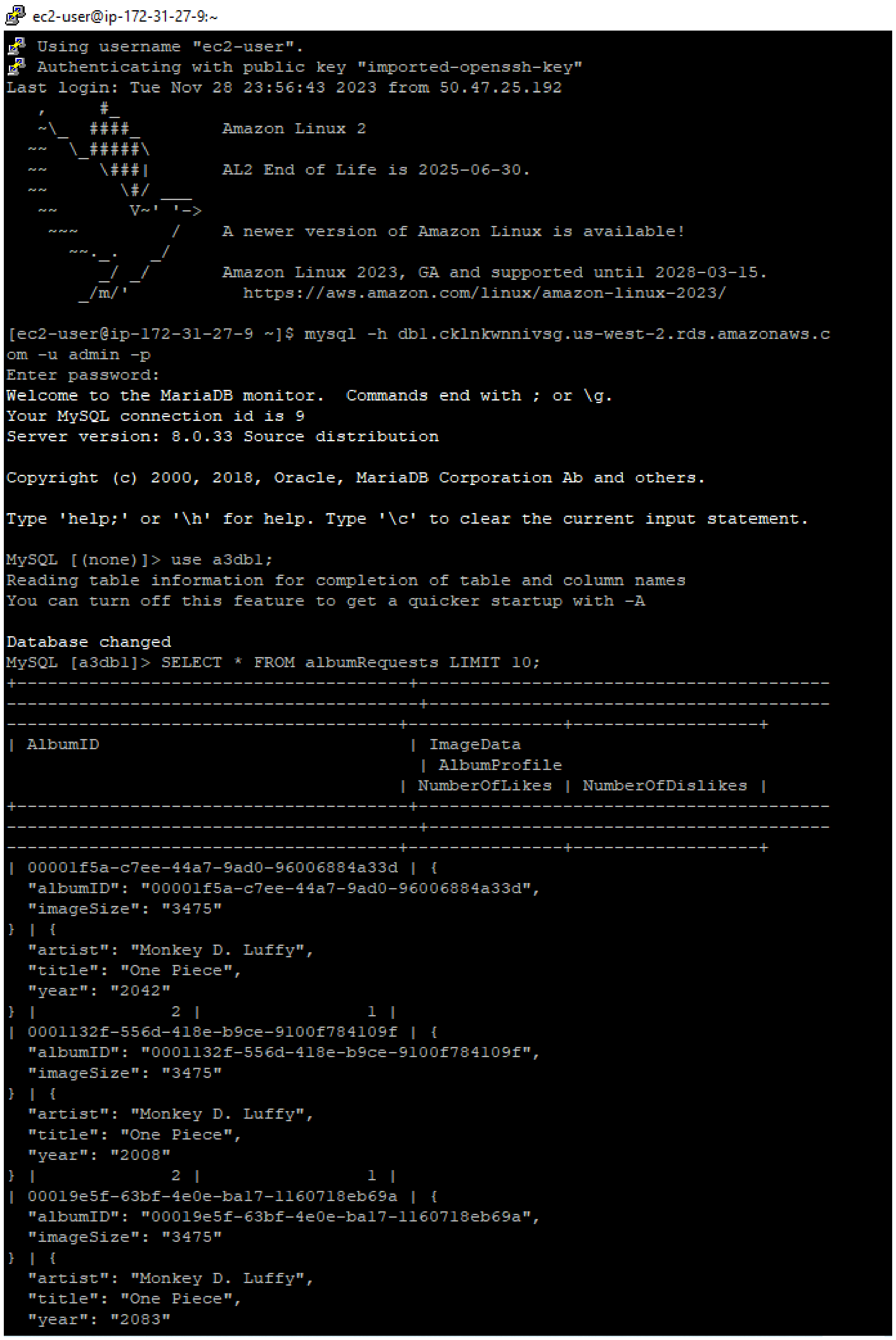
Add a new exchange

Image 9: Screenshot of RabbitMQ Exchange window for 10/30/2 run 1.

As seen in the three RabbitMQ images above, the queue does not grow at all. The queue message rates slowly build to a consistent 1.5k msg/sec. Additional screenshots are included in the Client Project results folder.

MySQL Review Updates

All the review updates to the MySQL take some time to complete. Since the ReviewServlet returns a response after publishing a message to the RabbitMQ Exchange, it returns a response very quickly. Similarly, since the RabbitMQ consumer queues are created with auto acknowledgment enabled, messages are consumed almost immediately. Thus, the database updates happen asynchronously as intended, but we currently have no way set up to track when all updates are complete without manually checking the database. Here is a sample of my db after all updates are completed.



```
ec2-user@ip-172-31-27-9:~
 Using username "ec2-user".
 Authenticating with public key "imported-openssh-key"
Last login: Tue Nov 28 23:56:43 2023 from 50.47.25.192
   ,     #_
   ~\_  ####_        Amazon Linux 2
  ~~  \_#####\
  ~~     \###|       AL2 End of Life is 2025-06-30.
  ~~       \#/ ___
   ~~       V~' '->
    ~~~         /    A newer version of Amazon Linux is available!
      ~~._.   _/
         _/ _/       Amazon Linux 2023, GA and supported until 2028-03-15.
       _/m/'           https://aws.amazon.com/linux/amazon-linux-2023/

[ec2-user@ip-172-31-27-9 ~]$ mysql -h db1.cklnkwnnivsg.us-west-2.rds.amazonaws.c
om -u admin -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 8.0.33 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]> use a3db1;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
MySQL [a3db1]> SELECT * FROM albumRequests LIMIT 10;
+--------------------------------------+----------------------------------------
----------------------------------------+---------------------------------------
--------------------------------------+---------------+------------------+
| AlbumID                              | ImageData
                                        | AlbumProfile
                                      | NumberOfLikes | NumberOfDislikes |
+--------------------------------------+----------------------------------------
----------------------------------------+---------------------------------------
--------------------------------------+---------------+------------------+
| 00001f5a-c7ee-44a7-9ad0-96006884a33d | {
  "albumID": "00001f5a-c7ee-44a7-9ad0-96006884a33d",
  "imageSize": "3475"
} | {
  "artist": "Monkey D. Luffy",
  "title": "One Piece",
  "year": "2042"
} |             2 |                1 |
| 0001132f-556d-418e-b9ce-9100f784109f | {
  "albumID": "0001132f-556d-418e-b9ce-9100f784109f",
  "imageSize": "3475"
} | {
  "artist": "Monkey D. Luffy",
  "title": "One Piece",
  "year": "2008"
} |             2 |                1 |
| 00019e5f-63bf-4e0e-ba17-1160718eb69a | {
  "albumID": "00019e5f-63bf-4e0e-ba17-1160718eb69a",
  "imageSize": "3475"
} | {
  "artist": "Monkey D. Luffy",
  "title": "One Piece",
  "year": "2083"
```

Image 10: Screenshot of MySQL Database after Review Updates for 10/30/2 run 1.