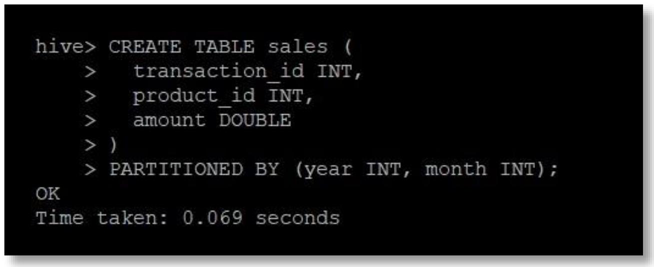
1. **Create a partitioned table sales partitioned by year and month. (2 marks) Code:**

```
CREATE TABLE sales (
  transaction_id INT,
product_id INT,   amount
DOUBLE
)
PARTITIONED BY (year INT, month INT);
```

Screenshot:



```
hive> CREATE TABLE sales (
    >   transaction_id INT,
    >   product_id INT,
    >   amount DOUBLE
    > )
    > PARTITIONED BY (year INT, month INT);
OK
Time taken: 0.069 seconds
```

Inference:

A partitioned table is divided into segments, called partitions that make it easier to manage and query your data. By dividing a large table into smaller partitions, you can improve query performance and control costs by reducing the number of bytes read by a query.

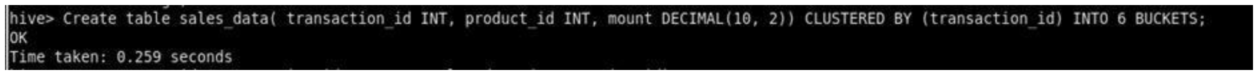
Here in our table data to be inserted will get partitioned by Year and Month columns.

2. **A. Write a HiveQL query to create a table named sales_data with columns transaction_id, product_id, and amount clustered into 6 buckets based on transaction_id.**

Code:

```
CREATE TABLE sales_data (
transaction_id INT,
product_id INT,   mount
DECIMAL(10, 2)
)
CLUSTERED BY (transaction_id) INTO 6 BUCKETS;
```

Screenshot:



```
hive> Create table sales_data( transaction_id INT, product_id INT, mount DECIMAL(10, 2)) CLUSTERED BY (transaction_id) INTO 6 BUCKETS;
OK
Time taken: 0.259 seconds
```

Inference:

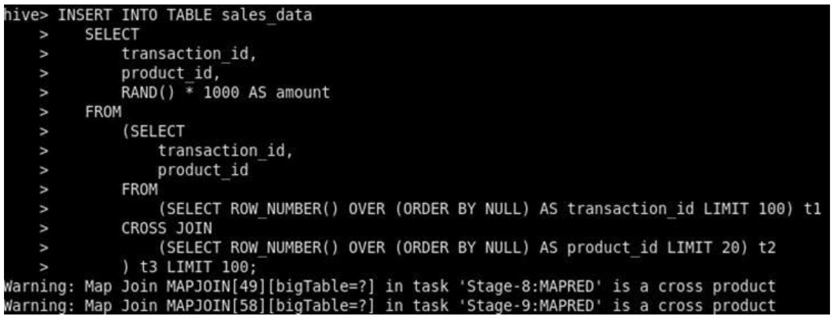
It is a technique to decompose data into buckets. In bucketing, Hive splits the data into a fixed number of buckets, according to a hash function over some set of columns. Hive ensures that all rows that have the same hash will be stored in the same bucket.

B. Insert data into the sales_data table you created, inserting records for transactions with IDs 1 to 100, products ranging from 1 to 20, and random transaction amounts.

Code:

```
 INSERT INTO TABLE sales_data SELECT
  transaction_id,
  CAST(RAND() * 20 + 1 AS INT) AS product_id,
  CAST(RAND() * 1000 AS DECIMAL(10, 2)) AS amount
FROM
  (
    SELECT EXPLODE(sequence(1, 100)) AS transaction_id
  ) ;
```

Screenshot:



```
hive> INSERT INTO TABLE sales_data
    >     SELECT
    >         transaction_id,
    >         product_id,
    >         RAND() * 1000 AS amount
    >     FROM
    >         (SELECT
    >             transaction_id,
    >             product_id
    >         FROM
    >             (SELECT ROW_NUMBER() OVER (ORDER BY NULL) AS transaction_id LIMIT 100) t1
    >         CROSS JOIN
    >             (SELECT ROW_NUMBER() OVER (ORDER BY NULL) AS product_id LIMIT 20) t2
    >         ) t3 LIMIT 100;
Warning: Map Join MAPJOIN[49][bigTable=?] in task 'Stage-8:MAPRED' is a cross product
Warning: Map Join MAPJOIN[58][bigTable=?] in task 'Stage-9:MAPRED' is a cross product
```

Inference:

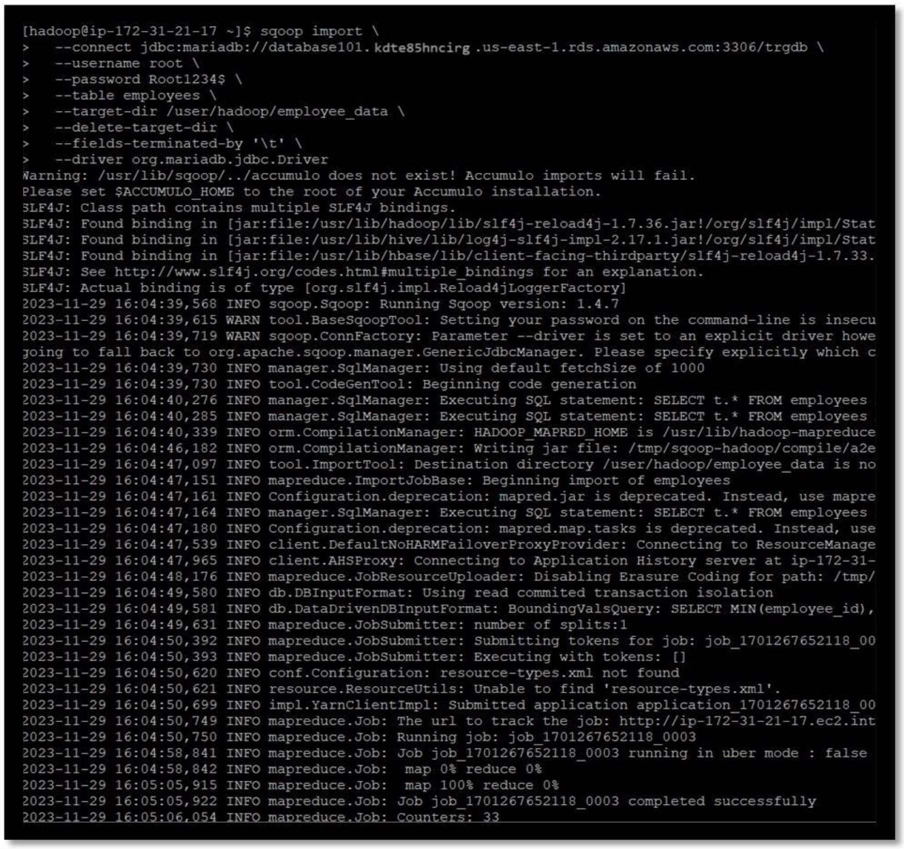
Inserting data in to table required above commands and rand() returns a random number (that changes from row to row) that is distributed uniformly across all rows of each column.

3. **Write a Sqoop command to import data from a MySQL database table named "employees" into the Hadoop Distributed File System (HDFS). The data should be saved in a directory named "employee_data" in HDFS**

Code:

```
sqoop import \
  --connect jdbc:mysql://your_mysql_host:your_mysql_port/your_database \
  --username your_username \
  --password your_password \
  --table employees \
  --target-dir /user/hadoop/employee_data \
  --m 1
```

Screenshot:



```
[hadoop@ip-172-31-21-17 ~]$ sqoop import \
>   --connect jdbc:mariadb://database101. kdte85hncirg .us-east-1.rds.amazonaws.com:3306/trgdb \
>   --username root \
>   --password Root1234$ \
>   --table employees \
>   --target-dir /user/hadoop/employee_data \
>   --delete-target-dir \
>   --fields-terminated-by '\t' \
>   --driver org.mariadb.jdbc.Driver
Warning: /usr/lib/sqoop/../accumulo does not exist! Accumulo imports will fail.
Please set $ACCUMULO_HOME to the root of your Accumulo installation.
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/usr/lib/hadoop/lib/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/Stat
SLF4J: Found binding in [jar:file:/usr/lib/hive/lib/log4j-slf4j-impl-2.17.1.jar!/org/slf4j/impl/Stat
SLF4J: Found binding in [jar:file:/usr/lib/hbase/lib/client-facing-thirdparty/slf4j-reload4j-1.7.33.
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [org.slf4j.impl.Reload4jLoggerFactory]
2023-11-29 16:04:39,568 INFO sqoop.Sqoop: Running Sqoop version: 1.4.7
2023-11-29 16:04:39,615 WARN tool.BaseSqoopTool: Setting your password on the command-line is insecu
2023-11-29 16:04:39,719 WARN sqoop.ConnFactory: Parameter --driver is set to an explicit driver howe
going to fall back to org.apache.sqoop.manager.GenericJdbcManager. Please specify explicitly which c
2023-11-29 16:04:39,730 INFO manager.SqlManager: Using default fetchSize of 1000
2023-11-29 16:04:39,730 INFO tool.CodeGenTool: Beginning code generation
2023-11-29 16:04:40,276 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM employees
2023-11-29 16:04:40,285 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM employees
2023-11-29 16:04:40,339 INFO orm.CompilationManager: HADOOP_MAPRED_HOME is /usr/lib/hadoop-mapreduce
2023-11-29 16:04:46,182 INFO orm.CompilationManager: Writing jar file: /tmp/sqoop-hadoop/compile/a2e
2023-11-29 16:04:47,097 INFO tool.ImportTool: Destination directory /user/hadoop/employee_data is no
2023-11-29 16:04:47,151 INFO mapreduce.ImportJobBase: Beginning import of employees
2023-11-29 16:04:47,161 INFO Configuration.deprecation: mapred.jar is deprecated. Instead, use mapre
2023-11-29 16:04:47,164 INFO manager.SqlManager: Executing SQL statement: SELECT t.* FROM employees
2023-11-29 16:04:47,180 INFO Configuration.deprecation: mapred.map.tasks is deprecated. Instead, use
2023-11-29 16:04:47,539 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManage
2023-11-29 16:04:47,965 INFO client.AHSProxy: Connecting to Application History server at ip-172-31-
2023-11-29 16:04:48,176 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/
2023-11-29 16:04:49,580 INFO db.DBInputFormat: Using read commited transaction isolation
2023-11-29 16:04:49,581 INFO db.DataDrivenDBInputFormat: BoundingValsQuery: SELECT MIN(employee_id),
2023-11-29 16:04:49,631 INFO mapreduce.JobSubmitter: number of splits:1
2023-11-29 16:04:50,392 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1701267652118_00
2023-11-29 16:04:50,393 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-11-29 16:04:50,620 INFO conf.Configuration: resource-types.xml not found
2023-11-29 16:04:50,621 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2023-11-29 16:04:50,699 INFO impl.YarnClientImpl: Submitted application application_1701267652118_00
2023-11-29 16:04:50,749 INFO mapreduce.Job: The url to track the job: http://ip-172-31-21-17.ec2.int
2023-11-29 16:04:50,750 INFO mapreduce.Job: Running job: job_1701267652118_0003
2023-11-29 16:04:58,841 INFO mapreduce.Job: Job job_1701267652118_0003 running in uber mode : false
2023-11-29 16:04:58,842 INFO mapreduce.Job:  map 0% reduce 0%
2023-11-29 16:05:05,915 INFO mapreduce.Job:  map 100% reduce 0%
2023-11-29 16:05:05,922 INFO mapreduce.Job: Job job_1701267652118_0003 completed successfully
2023-11-29 16:05:06,054 INFO mapreduce.Job: Counters: 33
```

Inference:

Step 1: Create a database and table in the hive.
Step 2: Insert data into the hive table.
Step 3: Create a database and table in MySQL in which data should be exported.
Step 4: Run the command

4. **Using MongoDB Compass, create a new document in a collection named "students" with the following fields: "name," "age," and "grade".**

Steps:

Steps to create a new document in a collection named “students” with the fields “name”, “age”, and “grade” using MongoDB Compass: Open MongoDB Compass and connect to your MongoDB server.

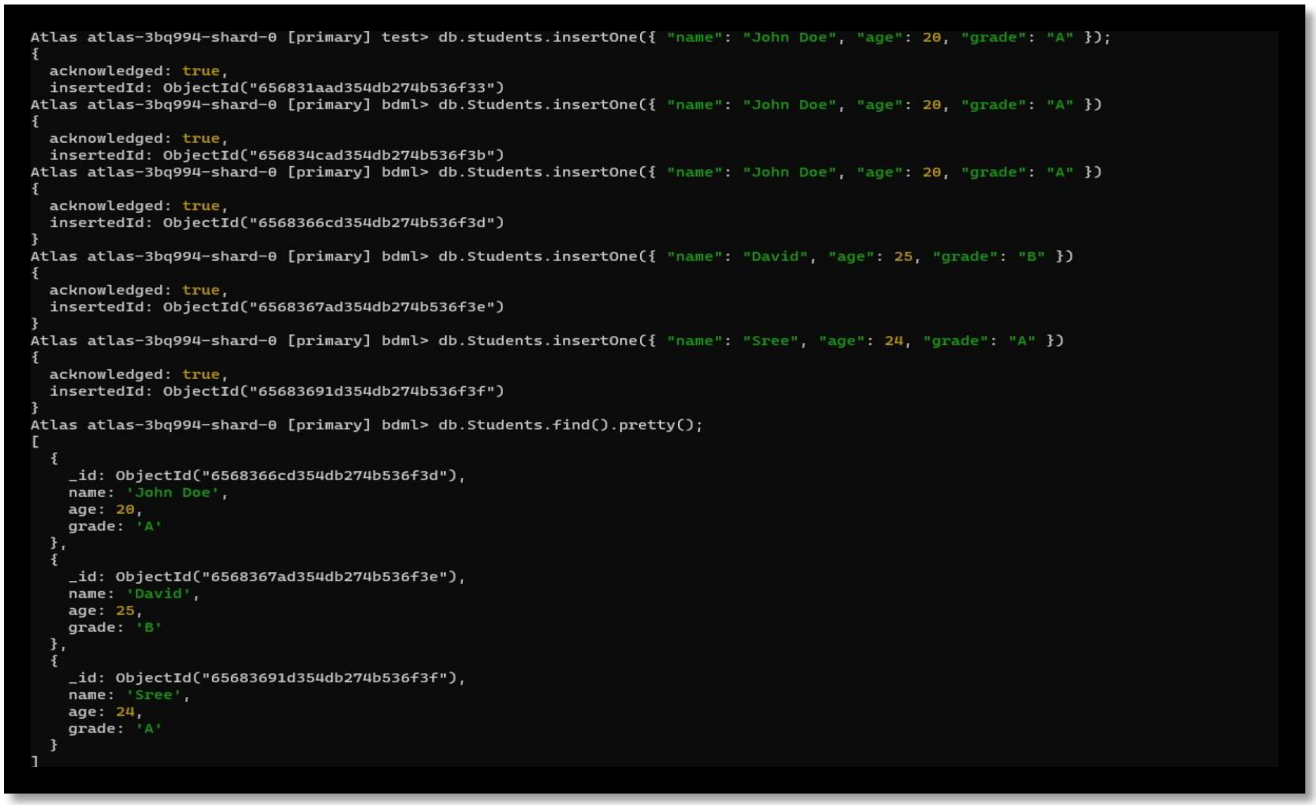
- On the left navigation panel, select the database and the collection where you want to insert the document. If the collection does not exist, you can create it by clicking the Create Collection button.

- Click the Add Data dropdown and select Insert Document.
- Set the View to JSON ({}).
- Write or paste the JSON document that you want to insert in the editor. For example, you can use this document:

```
{
 "name": "James",
 "age": 31,
 "grade": "A"
}
```

- Click Insert.

Screenshot:



```
Atlas atlas-3bq994-shard-0 [primary] test> db.students.insertOne({ "name": "John Doe", "age": 20, "grade": "A" });
{
  acknowledged: true,
  insertedId: ObjectId("656831aad354db274b536f33")
Atlas atlas-3bq994-shard-0 [primary] bdml> db.Students.insertOne({ "name": "John Doe", "age": 20, "grade": "A" })
{
  acknowledged: true,
  insertedId: ObjectId("656834cad354db274b536f3b")
Atlas atlas-3bq994-shard-0 [primary] bdml> db.Students.insertOne({ "name": "John Doe", "age": 20, "grade": "A" })
{
  acknowledged: true,
  insertedId: ObjectId("6568366cd354db274b536f3d")
}
Atlas atlas-3bq994-shard-0 [primary] bdml> db.Students.insertOne({ "name": "David", "age": 25, "grade": "B" })
{
  acknowledged: true,
  insertedId: ObjectId("6568367ad354db274b536f3e")
}
Atlas atlas-3bq994-shard-0 [primary] bdml> db.Students.insertOne({ "name": "Sree", "age": 24, "grade": "A" })
{
  acknowledged: true,
  insertedId: ObjectId("65683691d354db274b536f3f")
}
Atlas atlas-3bq994-shard-0 [primary] bdml> db.Students.find().pretty();
[
  {
    _id: ObjectId("6568366cd354db274b536f3d"),
    name: 'John Doe',
    age: 20,
    grade: 'A'
  },
  {
    _id: ObjectId("6568367ad354db274b536f3e"),
    name: 'David',
    age: 25,
    grade: 'B'
  },
  {
    _id: ObjectId("65683691d354db274b536f3f"),
    name: 'Sree',
    age: 24,
    grade: 'A'
  }
]
```

5. **Use MongoDB Compass to perform an aggregation operation that calculates the average "price" of products in the "products" collection**

Steps:

Steps to perform an aggregation operation that calculates the average “price” of products in the “products” collection using MongoDB Compass: Open MongoDB Compass and connect to your MongoDB server.

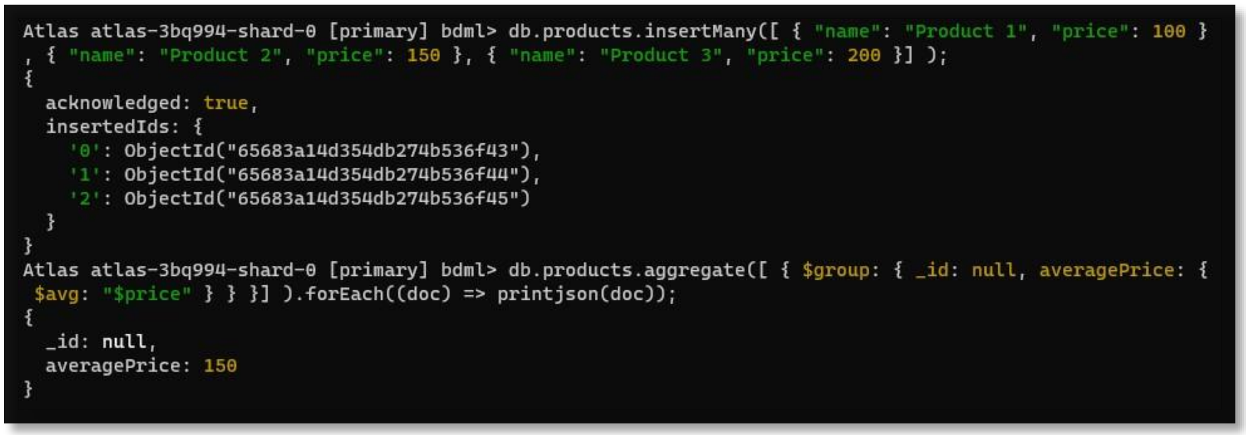
- On the left navigation panel, select the database and the collection where you want to perform the aggregation. In this case, select the “products” collection.
- Click the Aggregations tab to open the Aggregation Pipeline Builder.
- Click the Add Stage button to add a new stage to the pipeline. A blank stage with a dropdown menu appears.
- From the dropdown menu, select the $group stage. This stage groups documents by a specified expression and applies an accumulator function to each group.
- In the editor, type or paste the following JSON document:

```
{
 _id: null, // This means group all documents together
 averagePrice: { $avg: "$price" } // This means calculate the average of the price field for each
group }
```

- Click Apply to run the pipeline. You should see the output of the aggregation operation in the Documents section. It should look something like this:

```
{
 _id: null,
 averagePrice: 12.34 // This is the average price of all products }
```

Screenshot:



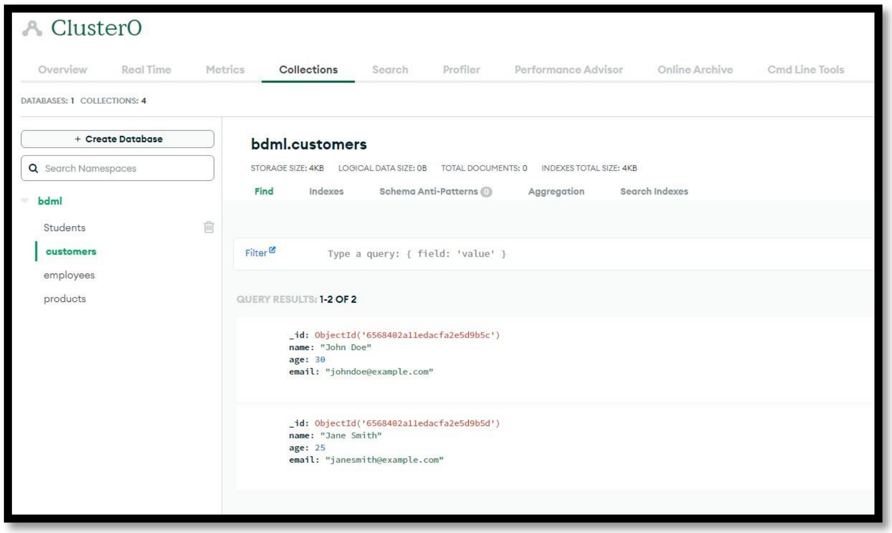
```
Atlas atlas-3bq994-shard-0 [primary] bdml> db.products.insertMany([ { "name": "Product 1", "price": 100 }
, { "name": "Product 2", "price": 150 }, { "name": "Product 3", "price": 200 }] );
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("65683a14d354db274b536f43"),
    '1': ObjectId("65683a14d354db274b536f44"),
    '2': ObjectId("65683a14d354db274b536f45")
  }
}
Atlas atlas-3bq994-shard-0 [primary] bdml> db.products.aggregate([ { $group: { _id: null, averagePrice: {
 $avg: "$price" } } }] ).forEach((doc) => printjson(doc));
{
  _id: null,
  averagePrice: 150
}
```

6. Import a JSON file containing customer data into a new collection named "customers" in your MongoDB database using MongoDB Compass.

Steps:

- Open MongoDB Compass and connect to your MongoDB server.
- On the left navigation panel, select the database where you want to create the new collection. If the database does not exist, you can create it by clicking the Create Database button.
- Click the Create Collection button and enter “customers” as the collection name. Click Create Collection to confirm.
- Click the Add Data dropdown and select Import JSON or CSV file.
- Select JSON as the file type and click Select.
- Choose the JSON file that contains the customer data and click Open.
- Configure the import options as needed. For example, you can check Ignore empty strings to skip fields with empty string values in the imported documents.
- Click Import. A progress bar shows the status of the import. If the import is successful, the dialog closes and you can see the imported documents in the Documents tab. If there is an error, the progress bar turns red and an error message appears. You can click View Log to see the details of the error.

Screenshot:


Cluster0
Overview Real Time Metrics Collections Search Profiler Performance Advisor Online Archive Cmd Line Tools
DATABASES: 1 COLLECTIONS: 4
+ Create Database
Search Namespaces
bdml
Students
customers
employees
products
Find Indexes Schema Anti-Patterns Aggregation Search Indexes
Filter

bdml.customers

STORAGE SIZE: 4KB LOGICAL DATA SIZE: 0B TOTAL DOCUMENTS: 0 INDEXES TOTAL SIZE: 4KB

Type a query: { field: 'value' }

QUERY RESULTS: 1-2 OF 2

```
_id: ObjectId('6568402a11edacfa2e5d9b5c')
name: "John Doe"
age: 30
email: "johndoe@example.com"
```

```
_id: ObjectId('6568402a11edacfa2e5d9b5d')
name: "Jane Smith"
age: 25
email: "janesmith@example.com"
```

7. Delete a document in the "employees" collection where the "salary" field is less than $40,000 using MongoDB Compass.

Steps: 	Open MongoDB Compass and connect to your

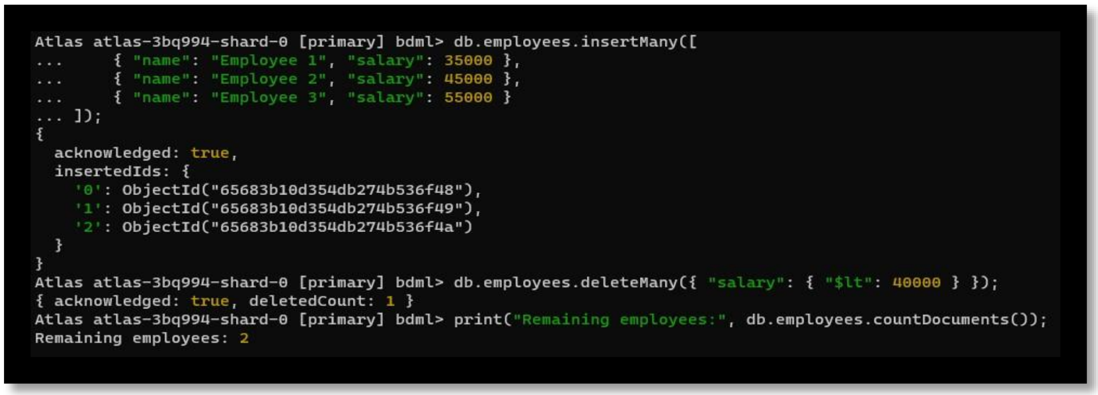
MongoDB server.

- On the left navigation panel, select the database and the collection where you want to delete the document. In this case, select the “employees” collection.
- In the Filter field, enter the following query to find the documents that match the condition:

```
{
 "salary": { "$lt": 60000 }
}
```

- Click Apply to run the query. You should see the documents that have a salary less than $40,000 in the Documents tab.
- For the document that you want to delete, hover over the document and click the trash icon that appears on the right-hand side.
- A confirmation dialog will appear. Click Delete to confirm your selection. The document will be deleted from the collection.

Screenshot:



```
Atlas atlas-3bq994-shard-0 [primary] bdml> db.employees.insertMany([
...     { "name": "Employee 1", "salary": 35000 },
...     { "name": "Employee 2", "salary": 45000 },
...     { "name": "Employee 3", "salary": 55000 }
... ]);
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("65683b10d354db274b536f48"),
    '1': ObjectId("65683b10d354db274b536f49"),
    '2': ObjectId("65683b10d354db274b536f4a")
  }
}
Atlas atlas-3bq994-shard-0 [primary] bdml> db.employees.deleteMany({ "salary": { "$lt": 40000 } });
{ acknowledged: true, deletedCount: 1 }
Atlas atlas-3bq994-shard-0 [primary] bdml> print("Remaining employees:", db.employees.countDocuments());
Remaining employees: 2
```