
Big Data Analytics (BDA) Experiments – Complete Slips

💻 MongoDB Slips (NoSQL)

Slip 1 – MongoDB CRUD Operations

Problem Statement

Create a MongoDB collection named **Students**. Perform the following operations:

1. Insert five student documents (fields: name, dept, marks).
2. Update marks of one student.
3. Delete one record.
4. Display all records.

Solution & Commands

Mongo Shell Commands

```
use student_db

// 1. Create collection and insert 5 documents
db.Students.insertMany([
  {name: "John", dept: "CS", marks: 85},
  {name: "Alice", dept: "IT", marks: 78},
  {name: "Bob", dept: "CS", marks: 92},
  {name: "Carol", dept: "ECE", marks: 65},
  {name: "David", dept: "IT", marks: 88}
]);

// 2. Update marks of one student
db.Students.updateOne({name: "Carol"}, {$set: {marks: 75}})

// 3. Delete one record
db.Students.deleteOne({name: "Bob"})
```

```
// 4. Display all records
db.Students.find().pretty()
```

Python (PyMongo) Code

```
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')
db = client['student_db']
collection = db['Students']

# 1. Insert documents
students = [
  {"name": "John", "dept": "CS", "marks": 85},
  {"name": "Alice", "dept": "IT", "marks": 78},
  {"name": "Carol", "dept": "ECE", "marks": 65},
  {"name": "David", "dept": "IT", "marks": 88}
]
collection.insert_many(students)

# 2. Update marks
collection.update_one({"name": "Carol"}, {"$set": {"marks": 75}})

# 3. Delete record
collection.delete_one({"name": "Bob"})

# 4. Display all records
for student in collection.find():
    print(student)
```

Slip 2 – Querying Unstructured JSON Data

Problem Statement

Load a JSON file of product details into MongoDB. Write queries to:

1. Display all products in the **Electronics** category.
2. Count the total number of items priced above ₹10,000.

Sample Data File (products.json)

```
[
 { "name": "Laptop Pro", "category": "Electronics", "price": 45000, "in_stock": true },
 { "name": "Desk Chair", "category": "Furniture", "price": 8500, "in_stock": true },
 { "name": "Smartphone X", "category": "Electronics", "price": 12000, "in_stock": false },
 { "name": "Coffee Maker", "category": "Appliances", "price": 6000, "in_stock": true },
 { "name": "4K Monitor", "category": "Electronics", "price": 15000, "in_stock": true }
]
```

Solution & Commands

Command Line Import

```
mongoimport --db product_db --collection products --file products.json --jsonArray
```

Mongo Shell Queries

```
use product_db

// 1. Display electronics products
db.products.find({category: "Electronics"})

// 2. Count items priced above 10,000
db.products.countDocuments({price: {$gt: 10000}})
```

Python (PyMongo) Code

```
import json
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')
db = client['product_db']
collection = db['products']
```

```
# Load JSON file (if not using mongoimport)
with open('products.json') as f:
    data = json.load(f)
collection.insert_many(data)

# 1. Electronics products
for product in collection.find({"category": "Electronics"}):
    print(product)

# 2. Count expensive items
count = collection.count_documents({"price": {"$gt": 10000}})
print(f"Items above 10,000: {count}")
```

Slip 3 – Aggregation Pipeline

Problem Statement

Use MongoDB aggregation pipeline to **group employees by department** and calculate **average salary per department**, displaying results in **descending order of average salary**.

Sample Data

```
use company_db
db.employees.insertMany([
  {name: "Emp A", department: "IT", salary: 60000},
  {name: "Emp B", department: "HR", salary: 45000},
  {name: "Emp C", department: "IT", salary: 80000},
  {name: "Emp D", department: "Finance", salary: 75000},
  {name: "Emp E", department: "HR", salary: 55000}
])
```

Mongo Shell Query

```
db.employees.aggregate([
  { $group: {
    _id: "$department",
    averageSalary: { $avg: "$salary" }
  }},
  { $sort: { averageSalary: -1 } }
])
```

Python (PyMongo) Code

```
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')
collection = client['company_db']['employees']

pipeline = [
  {"$group": {"_id": "$department", "averageSalary": {"$avg": "$salary"}}},
  {"$sort": {"averageSalary": -1}}
]

for result in collection.aggregate(pipeline):
    print(f"Department: {result['_id']}, Avg Salary: {result['averageSalary']:.2f}")
```

Slip 4 – API Operations using PyMongo

Problem Statement

Using PyMongo:

1. Connect to MongoDB
2. Insert 3 employee documents
3. Retrieve records where salary > 50,000
4. Update one record and print all documents

Solution Code

```
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')
```

```
db = client['company_db']
employees = db['employees']

employees.delete_many({})  # Optional clean start

employees.insert_many([
  {"name": "John", "department": "IT", "salary": 60000},
  {"name": "Alice", "department": "HR", "salary": 45000},
  {"name": "Bob", "department": "Finance", "salary": 75000}
])

print("\nEmployees with salary > 50,000:")
for emp in employees.find({"salary": {"$gt": 50000}}):
    print(emp)

employees.update_one({"name": "Alice"}, {"$set": {"salary": 52000}})
print("\nUpdated Alice's salary.")

print("\nAll employees:")
for emp in employees.find():
    print(emp)
```

🐘 Hive Slips (HiveQL)

Slip 5 – Basic Querying

Problem Statement

Create a Hive table movies with fields (title, type, release_year, country).
Perform queries to:

1. Display the number of movies per country.
2. Show top 5 recent release years.

Sample CSV: movies.csv

```
The Social Network,Movie,2010,USA
Midnight in Paris,Movie,2011,France
Stranger Things,TV Show,2016,USA
Money Heist,TV Show,2017,Spain
Parasite,Movie,2019,South Korea
Nomadland,Movie,2020,USA
The Crown,TV Show,2016,UK
Dune,Movie,2021,USA
Squid Game,TV Show,2021,South Korea
```

HiveQL Code

```
CREATE DATABASE IF NOT EXISTS bda_hive;
USE bda_hive;

CREATE TABLE movies (
  title STRING,
  type STRING,
  release_year INT,
  country STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '/path/to/movies.csv' INTO TABLE movies;

-- Query 1
SELECT country, COUNT(*) AS movie_count
FROM movies
GROUP BY country
ORDER BY movie_count DESC;
```

```
-- Query 2
SELECT release_year, COUNT(*) AS movie_count
FROM movies
GROUP BY release_year
ORDER BY release_year DESC
LIMIT 5;
```

Slip 6 – Sorting and Aggregation

Sample Data: sales_data.txt

```
East	Laptop	12000.00
West	Keyboard	150.00
East	Monitor	3000.00
South	Laptop	15000.00
West	Mouse	20.00
East	Keyboard	100.00
```

HiveQL Code

```
CREATE TABLE sales_data (
  region STRING,
  product STRING,
  amount DOUBLE
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t';

LOAD DATA LOCAL INPATH '/path/to/sales_data.txt' INTO TABLE sales_data;

SELECT region, SUM(amount) AS total_sales
FROM sales_data
GROUP BY region
ORDER BY total_sales DESC;
```

Slip 7 – Join Operation

Problem Statement

Join two tables **customers** and **orders** to find **total order amount per customer**.

Sample Files

customers.txt

```
1,John,New York

2,Alice,London

3,Bob,Paris
```

orders.txt

```
101,1,500.00

102,2,120.00

103,1,300.00

104,3,450.50

105,2,80.00
```

HiveQL Code

```
CREATE TABLE customers (

  cust_id INT,

  name STRING,

  city STRING

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ',';


CREATE TABLE orders (

  order_id INT,

  cust_id INT,

  amount DOUBLE

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ',';
```

```
LOAD DATA LOCAL INPATH '/path/to/customers.txt' INTO TABLE customers;
LOAD DATA LOCAL INPATH '/path/to/orders.txt' INTO TABLE orders;

SELECT c.name, SUM(o.amount) AS total_amount
FROM customers c
JOIN orders o ON c.cust_id = o.cust_id
GROUP BY c.name
ORDER BY total_amount DESC;
```

Slip 8 – Hive UDF

Problem: Create Hive UDF to convert movie titles to uppercase.

Java Code: UpperCaseUDF.java

```
import org.apache.hadoop.hive.ql.exec.UDF;
import org.apache.hadoop.io.Text;

public class UpperCaseUDF extends UDF {
  public Text evaluate(Text input) {
    if (input == null) return null;
    return new Text(input.toString().toUpperCase());
  }
}
```

Commands

```
javac -cp "$(hadoop classpath):/usr/lib/hive/lib/*:." UpperCaseUDF.java
jar -cf uppercase-udf.jar UpperCaseUDF.class
```

Hive Commands

```
ADD JAR /home/student/ESE/slip8/uppercase-udf.jar;
CREATE TEMPORARY FUNCTION uppercase AS 'UpperCaseUDF';

SELECT uppercase(title), type, release_year FROM movies;
```

🐖 Pig Slips (Pig Latin)

Slip 9 – Basic Filtering

Sample: students.txt

```
Rajiv,85
Siddharth,68
Rajesh,92
Preethi,72
Trupthi,55
Archana,70
```

Pig Script

```
students = LOAD 'students.txt' USING PigStorage(',') AS (name:chararray, marks:int);
good_students = FILTER students BY marks > 70;
result = FOREACH good_students GENERATE name, marks;
DUMP result;
```

Slip 10 – Grouping and Aggregation

Sample: sales_data.txt

```
Electronics	Laptop	12000.00
Furniture	Desk Chair	8500.00
Electronics	Monitor	3000.00
Appliances	Coffee Maker	6000.00
Electronics	Phone	15000.00
```

Pig Script

```
sales = LOAD 'sales_data.txt' USING PigStorage('\t')
  AS (category:chararray, product:chararray, amount:double);


grouped_sales = GROUP sales BY category;
avg_sales = FOREACH grouped_sales GENERATE group AS category, AVG(sales.amount) AS avg_amount;
DUMP avg_sales;
```

Slip 11 – Join Operation

employee_details.txt

```
1,John Doe,101

2,Alice Smith,102

3,Bob Johnson,101

4,Carol White,103
```

department.txt

```
101,IT

102,HR

103,Finance
```

Pig Script

```
employees = LOAD 'employee_details.txt' USING PigStorage(',')

  AS (emp_id:int, name:chararray, dept_id:int);

departments = LOAD 'department.txt' USING PigStorage(',')

  AS (dept_id:int, dept_name:chararray);

joined_data = JOIN employees BY dept_id, departments BY dept_id;

result = FOREACH joined_data GENERATE employees::name, departments::dept_name;

DUMP result;
```

Slip 12 – Sorting and Filtering

Sample: movies.txt

```
The Crown,TV Show,2016,8.5

Money Heist,TV Show,2017,8.3

Game of Thrones,TV Show,2011,9.3

Chernobyl,TV Show,2019,9.4

The Queens Gambit,TV Show,2020,8.6

Ozark,TV Show,2017,8.4

The Witcher,TV Show,2019,8.2

Derry Girls,TV Show,2018,8.4

Succession,TV Show,2018,8.8
```

```
Squid Game,TV Show,2021,8.0

Peaky Blinders,TV Show,2013,8.8
```

Pig Script

```
movies = LOAD 'movies.txt' USING PigStorage(',')

  AS (title:chararray, type:chararray, release_year:int, rating:double);

tv_shows = FILTER movies BY type == 'TV Show';

sorted_shows = ORDER tv_shows BY release_year DESC;

top_10 = LIMIT sorted_shows 10;

DUMP top_10;
```

✅ **All Slips Complete (MongoDB, Hive, Pig)**
Includes **data files, queries, and code** ready for practical execution.