BDAL

SLIP-1 - MongoDB CRUD Operations

Using Mongo Shell :

```
# Start MongoDB service
sudo systemctl start mongod
# Open Mongo Shell
mongo
# Execute in Mongo Shell:
# Command 1: Create collection and insert 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}
])
# Command 2: Update marks
db.Students.updateOne(
  {name: "Carol"},
  {$set: {marks: 75}}
)
# Command 3: Delete record
db.Students.deleteOne({name: "Bob"})
# Command 4: Display all records
db.Students.find().pretty()
```

Using Python:

```
# Create Python file
nano slip1_mongodb_crud.py

# Python 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": "Bob", "dept": "CS", "marks": 92},
    {"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)

cd Downloads
sudo apt install python3-pip
pip3 install pymongo
# Execute
python3 slip1_mongodb_crud.py
```

SLIP-2 - Querying JSON Data

```
# create json file
nano products.json
# content of json file
[
  {"name": "Laptop", "category": "Electronics", "price": 55000},
  {"name": "Mouse", "category": "Electronics", "price": 1500},
  {"name": "Chair", "category": "Furniture", "price": 12000},
  {"name": "Smartphone", "category": "Electronics", "price": 25000},
  {"name": "Table", "category": "Furniture", "price": 8000}
]
```

Using Mongo Shell:

```
# Import JSON file
mongoimport --db product_db --collection products --file products.json --jsonArray
# Open Mongo Shell
mongo
# commands
```

```
use product_db
db.products.find().pretty()
// Query 1: Display electronics products
db.products.find({category: "Electronics"})
// Query 2: Count items priced above ₹10,000
db.products.countDocuments({price: {$gt: 10000}})
```

Using python:

```
nano slip2_json_query.py
```

# Python code

```
import json
from pymongo import MongoClient
client = MongoClient('mongodb://localhost:27017/')
db = client['product_db']
collection = db['products']
# Load JSON file
with open('products.json') as f:
   data = json.load(f)
collection.insert_many(data)
# 1. Electronics products
print("Electronics Products:")
electronics = collection.find({"category": "Electronics"})
for product in electronics:
   print(product)
# 2. Count expensive items
count = collection.count_documents({"price": {"$gt": 10000}})
print(f"Items above ₹10,000: {count}")

# Execute
cd Downloads
python3 slip2_json_query.py
```

SLIP-3 - Aggregation Pipeline

Using Mongo Shell :

```
mongo
use employee_db

# insert data
db.employees.insertMany([
  {name: "John", department: "IT", salary: 60000},
  {name: "Alice", department: "HR", salary: 45000},
  {name: "Bob", department: "IT", salary: 75000},
  {name: "Carol", department: "Finance", salary: 80000},
  {name: "David", department: "HR", salary: 50000}
])


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

Using Python:

```
# Create Python file
nano slip3_aggregation.py
```

# **Python code**

```
from pymongo import MongoClient
# 1. Connect to MongoDB
client = MongoClient('mongodb://localhost:27017/')
db = client['company_db']
employees = db['employees']
# 2. Insert 3 employee documents
employee_data = [
   {"name": "John", "department": "IT", "salary": 60000},
   {"name": "Alice", "department": "HR", "salary": 45000},
   {"name": "Bob", "department": "Finance", "salary": 75000}
]
employees.insert_many(employee_data)
# 3. Retrieve records where salary > 50,000
print("Employees with salary > 50,000:")
high_earners = employees.find({"salary": {"$gt": 50000}})
for emp in high_earners:
   print(emp)
# 4. Update one record
employees.update_one(
   {"name": "Alice"},
    {"$set": {"salary": 52000}}
)
# Print all documents
print("\nAll employees:")
for emp in employees.find():
    print(emp)

# Execute
cd Downloads
python3 slip3_aggregation.py
```

SLIP-4 - PyMongo Operations

## Python:

```
nano slip4_pymongo_operations.py
```

# **Python code:**

```
from pymongo import MongoClient
# 1. Connect to MongoDB
client = MongoClient('mongodb://localhost:27017/')
db = client['company_db']
employees = db['employees']
# 2. Insert 3 employee documents
employee_data = [
    {"name": "John", "department": "IT", "salary": 60000},
    {"name": "Alice", "department": "HR", "salary": 45000},
    {"name": "Bob", "department": "Finance", "salary": 75000}
]
employees.insert_many(employee_data)
# 3. Retrieve records where salary > 50,000
print("Employees with salary > 50,000:")
high_earners = employees.find({"salary": {"$gt": 50000}})
for emp in high_earners:
    print(emp)
# 4. Update one record
employees.update_one(
    {"name": "Alice"},
    {"$set": {"salary": 52000}}
)
# Print all documents
print("\nAll employees:")
for emp in employees.find():
    print(emp)

# Execute
cd Downloads
python3 slip4_pymongo_operations.py
```

Slip 5 - Hive Basic Querying

```
sudo apt-get update
ls
./Start-Hadoop-Hive.sh
./Stop-Hadoop-Hive.sh
jps
hive
hive --service metastore &
hive
nano movies.csv
```

Content:

```
Avatar,Movie,2009,USA
Stranger Things,TV Show,2016,USA
Dark,TV Show,2017,Germany
RRR,Movie,2022,India
Money Heist,TV Show,2017,Spain
Squid Game,TV Show,2021,South Korea
```

Command 1: Create table

```
sql
CREATE TABLE movies (
  title STRING,
  type STRING,
  release_year INT,
  country STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
```

```
STORED AS TEXTFILE;
```

Command 2: Load data

sql

```
LOAD DATA LOCAL INPATH '/home/talentum/Downloads/movies.csv' INTO TABLE movies;
```

Command 3: Number of movies per country

sql

```
SELECT country, COUNT(*) as movie_count
FROM movies
GROUP BY country
ORDER BY movie_count DESC;
```

Command 4: Top 5 recent release years

sql

```
SELECT release_year, COUNT(*) as movie_count
FROM movies
GROUP BY release_year
ORDER BY release_year DESC
LIMIT 5;
```

Slip 6 - Hive Sorting and Aggregation

```
nano sales_data.txt
```

Content:

```
North       Laptop      50000
South       Mouse       1500
East        Keyboard    3000
West        Monitor     15000
North       Printer     8000
```

Command 1: Create table

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

Command 2: Load data

```
LOAD DATA LOCAL INPATH '/home/talentum/Downloads/sales_data.txt' INTO TABLE sales_data;
```

Command 3: Total sales per region

```
SELECT region, SUM(amount) as total_sales
FROM sales_data
GROUP BY region;
```

Command 4: Sort by total sales descending

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

Slip 7 - Hive Joins and Filtering

Command 1: Create customers table

```
CREATE TABLE customers (
  cust_id INT,
  name STRING,
  city STRING
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

Command 2: Create orders table

```
CREATE TABLE orders (
  order_id INT,
  cust_id INT,
  amount DOUBLE
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;
```

Command 3: Insert sample data

```
INSERT INTO customers VALUES
(1, 'John', 'Mumbai'),
(2, 'Alice', 'Delhi'),
(3, 'Bob', 'Bangalore');

INSERT INTO orders VALUES
(101, 1, 5000),
(102, 2, 3000),
(103, 1, 2000),
(104, 3, 7000);
```

Command 4: Inner join query

```
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 UDFs

```
cd ~/Downloads
nano UpperCaseUDF.java
```

Java Code:

```
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());
    }
}
```

Step 2: Compile and create JAR in Downloads

```
cd ~/Downloads
javac -cp $(hadoop classpath):/home/talentum/hive/lib/hive-exec-2.3.6.jar:/home/talentum/hive/lib/hive-serde-2.3.6.jar UpperCaseUDF.java
jar cf uppercase-udf.jar UpperCaseUDF.class
```

Hive Commands:

```
hive
```

Command 1: Add JAR and create function

```
ADD JAR /home/talentum/Downloads/uppercase-udf.jar;
CREATE TEMPORARY FUNCTION uppercase AS 'UpperCaseUDF';
```

Command 2: Apply UDF

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

Slip 9 - Pig Basic Operations

```
# Create data file
nano students.txt

John,85
Alice,78
Bob,92
Carol,65
David,88
Emma,72

# Start Pig Grunt shell
pig -x local
```

Command 1: Load data

```
students = LOAD 'students.txt' USING PigStorage(',') AS (name:chararray, marks:int);
```

Command 2: Filter students

```
good_students = FILTER students BY marks > 70;
```

Command 3: Generate result

```
result = FOREACH good_students GENERATE name, marks;
```

Command 4: Display results

```
DUMP result;
```

Slip 10 - Pig Grouping and Aggregation

```
# Create data file
nano sales_data_pig.txt

Electronics	Laptop	50000
Electronics	Mouse	1500
Furniture	Chair	12000
Electronics	Phone	25000
Furniture	Table	8000
```

Pig Commands:

```
pig -x local
```

Command 1: Load data

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

Command 2: Group by category

```
grouped_sales = GROUP sales BY category;
```

Command 3: Calculate average

```
avg_sales = FOREACH grouped_sales GENERATE group as category, AVG(sales.amount) as avg_amount;
```

Command 4: Display results

```
DUMP avg_sales;
```

SLIP-11 Pig Join Operation

```
# Create data files
nano employee_details.txt

101,John,1
102,Alice,2
103,Bob,1
104,Carol,3

nano department.txt
1,IT
2,HR
3,Finance

pig -x local
```

Command 1: Load datasets

```
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);
```

Command 2: Perform join

```
joined_data = JOIN employees BY dept_id, departments BY dept_id;
```

Command 3: Generate result

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

Command 4: Display results

```
DUMP result;
```

SLIP-12 Pig Sorting and Filtering

```
# Create data file
nano movies_pig.txt

Avatar,Movie,2009,8.8
Stranger Things,TV Show,2016,8.7
Dark,TV Show,2017,8.8
RRR,Movie,2022,8.0
```

```
Money Heist,TV Show,2017,8.2
Squid Game,TV Show,2021,8.0
```

```
pig -x local
```

Command 1: Load data

```
movies = LOAD 'movies_pig.txt' USING PigStorage(',') AS (title:chararray, type:chararray, release_year:int, rating:double);
```

Command 2: Filter TV Shows

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

Command 3: Sort by release year

```
sorted_shows = ORDER tv_shows BY release_year DESC;
```

Command 4: Get top 10

```
top_10 = LIMIT sorted_shows 10;
```

Command 5: Display results

```
DUMP top_10;
```