

Slip 1 - MongoDB CRUD Operations

Using Mongo Shell:

// 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()
```

Using Python (PyMongo):

```
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)

```

Slip 2 - Querying JSON Data

Using Mongo Shell:

```

// Load JSON file (run from command line first)
// mongoimport --db product_db --collection products --file products.json

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

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

```

Using Python:

```

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
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}")
```

Slip 3 - Aggregation Pipeline

Using Mongo Shell:

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

Using Python:

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

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

Slip 4 - PyMongo Operations

```

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
high_earners = employees.find({"salary": {"$gt": 50000}})
print("Employees with salary > 50,000:")
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)

```

Slip 5 - Hive Basic Querying

```

-- Create table
CREATE TABLE movies (
    title STRING,
    type STRING,
    release_year INT,
    country STRING
)

```

```

ROW FORMAT DELIMITED
FIELDS TERMINATED BY ','
STORED AS TEXTFILE;

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

-- 1. Number of movies per country
SELECT country, COUNT(*) as movie_count
FROM movies
GROUP BY country
ORDER BY movie_count DESC;

-- 2. Top 5 recent release years
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

```

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

-- 1. Total sales per region
SELECT region, SUM(amount) as total_sales
FROM sales_data
GROUP BY region;

-- 2. 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

```
-- Create tables
CREATE TABLE customers (
    cust_id INT,
    name STRING,
    city STRING
);

CREATE TABLE orders (
    order_id INT,
    cust_id INT,
    amount DOUBLE
);

-- Inner join to find total order amount per customer
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

```
-- Create UDF (Java implementation needed first)
-- Compile and add JAR to Hive
ADD JAR /path/to/uppercase-udf.jar;

-- Create temporary function
CREATE TEMPORARY FUNCTION uppercase AS 'com.example.UpperCaseUDF';

-- Apply UDF on movies table
SELECT uppercase(title) as upper_title, type, release_year
FROM movies;
```

Slip 9 - Pig Basic Operations

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

```
-- Filter students with marks > 70
good_students = FILTER students BY marks > 70;

-- Display names and marks
result = FOREACH good_students GENERATE name, marks;

-- Store or display results
STORE result INTO 'output';
DUMP result;
```

Slip 10 - Pig Grouping and Aggregation

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

-- Group by category and calculate average
grouped_sales = GROUP sales BY category;
avg_sales = FOREACH grouped_sales
            GENERATE group as category, AVG(sales.amount) as avg_amount;

-- Display results
DUMP avg_sales;
```

Slip 11 - Pig Join Operation

```
-- 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);

-- Perform join
joined_data = JOIN employees BY dept_id, departments BY dept_id;

-- Generate result with employee names and department names
result = FOREACH joined_data GENERATE employees::name, departments::dept_name
;

-- Display results
```

```
DUMP result;
```

Slip 12 - Pig Sorting and Filtering

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

-- Filter only TV Shows
tv_shows = FILTER movies BY type == 'TV Show';

-- Sort by release year descending
sorted_shows = ORDER tv_shows BY release_year DESC;

-- Get top 10
top_10 = LIMIT sorted_shows 10;

-- Display results
DUMP top_10;
```