**Prasanna Gaekwad Rollno:-04**

**Practical No: 01**

**Aim:** To design EER diagrams using StarUML

**Description:**

An **Entity-Relationship (ER) Diagram** is a visual representation of the structure of a database. It shows the major entities in a system, their attributes, and the relationships between those entities.

• **Entities** represent real-world objects or concepts, such as *Student*, *Customer*, *Product*, etc.

• **Attributes** are the properties or details of each entity, like *name*, *ID*, *address*, etc.

• **Relationships** illustrate how entities are connected, for example, a *Student enrolls in a Course*.

ER diagrams help in designing a clear and logical database structure before actual implementation. They are widely used in database design to ensure data consistency, normalization, and integrity.

1) For a given Scenario draw EER diagram and convert entities and relationship into tables

**Scenario:** Hospital Management System

The system keeps records of patients, hospitals, doctors, and

their medical records. Patients are admitted to hospitals. Each

hospital employs multiple doctors. Each patient can have

multiple medical records**.**

**Step1:Entities and Attributes Identified**

1.**Patient**

o Pat\_id *(Primary Key)*

o PName

o PDiagnosis

o PAddress

2.**Hospital**

o Hos\_id *(Primary Key)*

o HName

o HAddress

o HCity

1

**Prasanna Gaekwad Rollno:-04**

3.**Doctor**

o Doc\_id *(Primary Key)*

o DName

o Qualification

o Salary

4.**Medical Record**

o Precord\_id *(Primary Key)*

o Date\_of\_examination

o Problem

**Step2: Relationships Identified**

**1. Admitted\_in** (Patient - Hospital)

o Many-to-One (M:1)

o One hospital can have many patients. o Foreign Key: Hos\_id in Patient table.

**2. Has** (Hospital - Doctor)

o One-to-Many (1:N)

o One hospital has many doctors.

o Foreign Key: Hos\_id in Doctor table.

**3. Has** (Patient - Medical Record)

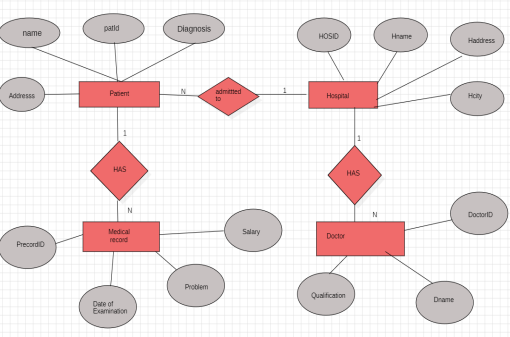
o One-to-Many (1:N)

o One patient has multiple medical records. o Foreign Key: Pat\_id in MedicalRecord table.

2

**Prasanna Gaekwad Rollno:-04**

**Step 3: Draw EER Diagram in StarUML**

****

Description: This ER diagram represents a **Hospital Management System** involving entities like Patient, Hospital, Doctor, and Medical Record.

• **Patient** has attributes like patId, name, Addresss, and Diagnosis. • Each **Patient** is *admitted to* one **Hospital**, but a **Hospital** can admit many patients (1:N).

• **Hospital** has attributes like HOSID, Hname, Haddress, and Hcity. • A **Hospital** *has* many **Doctors** (1:N), and each **Doctor** is associated with one Hospital.

• **Doctor** has attributes like DoctorID, Dname, Qualification, and Salary. • A **Patient** *has* many **Medical Records** (1:N), but each record is tied to one patient.

• **Medical Record** has attributes such as PrecordID, Date of Examination, Problem, and Salary.

Each relationship (like *admitted to* or *has*) shows cardinality and links relevant entities clearly.

3

**Prasanna Gaekwad Rollno:-04**

**Step 4: Convert Entities and Relationships into Tables using SQL Write** SQL CREATE TABLE queries to represent entities and relationships. CREATE TABLE Patient (

Pat\_id INT PRIMARY KEY,

PName VARCHAR(50),

PDiagnosis VARCHAR(100),

PAddress VARCHAR(100),

Hos\_id INT,

FOREIGN KEY (Hos\_id) REFERENCES Hospital(Hos\_id)

);

CREATE TABLE Hospital (

Hos\_id INT PRIMARY KEY,

HName VARCHAR(50),

HAddress VARCHAR(100),

HCity VARCHAR(50)

);

CREATE TABLE Doctor (

Doc\_id INT PRIMARY KEY,

DName VARCHAR(50),

Qualification VARCHAR(50),

Salary DECIMAL(10,2),

Hos\_id INT,

FOREIGN KEY (Hos\_id) REFERENCES Hospital(Hos\_id)

);

CREATE TABLE MedicalRecord (

Precord\_id INT PRIMARY KEY,

Date\_of\_examination DATE,

Problem VARCHAR(100),

Pat\_id INT,

FOREIGN KEY (Pat\_id) REFERENCES Patient(Pat\_id) );

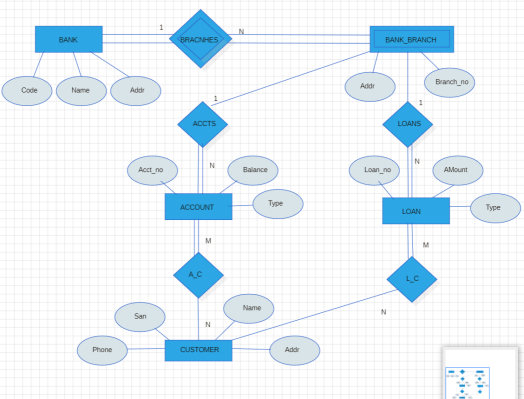
4

**Prasanna Gaekwad Rollno:-04**

2) For a given scenario, draw an EER diagram and show multiple entities and relationships

**Scenario:** Bank Management System

A bank has multiple branches. Each branch manages several accounts and issues multiple loans. Customers can hold multiple accounts and take multiple loans. Each account or loan can be shared by more than one customer.



Description: This ER diagram represents a **Banking System** with entities like Bank, Branch, Customer, Account, and Loan.

• A **Bank** (Code, Name, Addr) has multiple **Bank Branches**, each with a Branch\_no and Addr. • Each **Branch** can have multiple **Accounts** and **Loans** (1:N with both ACCTS and LOANS). • **Account** (Acct\_no, Balance, Type) and **Loan** (Loan\_no, Amount, Type) are managed per branch. • A **Customer** (San, Name, Phone, Addr) can hold multiple accounts (A\_C M:N) and take multiple loans (L\_C M:N).

• The many-to-many relationships between **Customer–Account** and **Customer–Loan** are managed through associative entities A\_C and L\_C.

5

**Prasanna Gaekwad Rollno:-04**

**Practical No.2**

**Aim:** To perform basic Create, Read, Update, and Delete operations using MongoDB.

MongoDB is a NoSQL document-oriented database that stores data in JSON-like BSON documents. It allows fast development, horizontal scaling, and flexible schema design. The four fundamental operations used in any database system are CRUD : Create, Read, Update, and Delete.

**1. Create / Switch to Database:**

The use command is used to create a new database or switch to an existing one. In MongoDB, if the database does not exist, it will be created only after inserting some data.

**Query:**

| use Practical2 |
| --- |

**Output:**

****

**2. Create Collection:**

Collections in MongoDB are similar to tables in relational databases. This command creates a collection named students. However, MongoDB allows you to insert directly into a collection even if it doesn’t exist; it will automatically create it on insertion.

**Query:**

| db.createCollection("students") |
| --- |

**Output:**

****

**3. Insert Data (Create):**

The insertMany() function is used to insert multiple documents at once into the students collection. Each document is a JSON-like object (BSON). This represents the Create operation in CRUD.

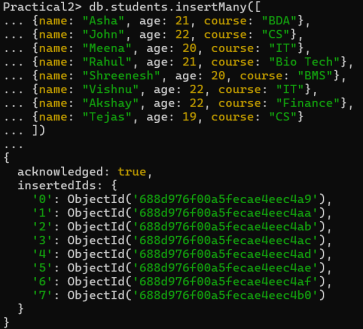
**Query:**

6

**Prasanna Gaekwad Rollno:-04**

| db.students.insertMany([  {name: "Asha", age: 21, course: "BDA"},  {name: "John", age: 22, course: "CS"},  {name: "Meena", age: 20, course: "IT"},  {name: "Rahul", age: 21, course: "Bio Tech"},  {name: "Shreenesh", age: 20, course: "BMS"},  {name: "Vishnu", age: 22, course: "IT"},  {name: "Akshay", age: 22, course: "Finance"},  {name: "Tejas", age: 19, course: "CS"}  ]) |
| --- |

**Output:**

**4. Read Data:**

a) Fetch all documents:

find() retrieves all documents from the collection. **Query:**

7

**Prasanna Gaekwad Rollno:-04**

| db.students.find() |
| --- |

**Output:**

****

b) Filter documents:

Using a filter, you can narrow results (e.g., only students from BDA course)

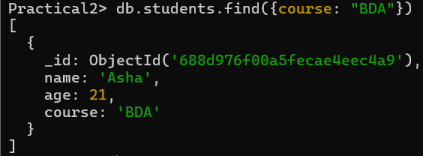
8

**Prasanna Gaekwad Rollno:-04**

Query:

| db.students.find({course: "BDA"}) |
| --- |

Output:



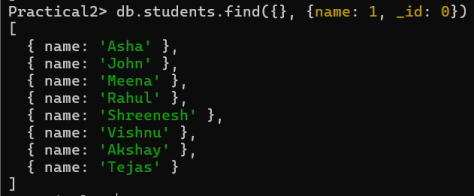
c) Project specific fields:

The projection part lets you control which fields to display. **{name: 1, \_id: 0}** means only name will be shown, and \_id will be excluded.

Query:

| db.students.find({}, {name: 1, \_id: 0}) |
| --- |

Output:

**4. Update Data:**

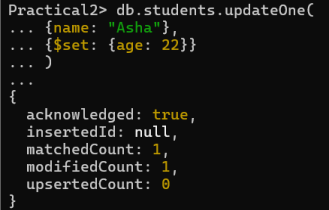
9

**Prasanna Gaekwad Rollno:-04 updateOne()** finds the first matching document and updates the specified fields. $set is an update operator that modifies the value of a field. Asha’s age will be updated from 21 to 22.

Query:

| db.students.updateOne(  {name: "Asha"},  {$set: {age: 22}}  ) |
| --- |

Output:



**5. Delete Data:**

deleteOne() removes the first document that matches the filter condition. This command deletes the student document where name is "John".

Query:

| db.students.deleteOne({name: "John"}) |
| --- |

Output:

10

**Prasanna Gaekwad Rollno:-04 Practical No:03**

Aim: To implement and indexing and sorting (ordering) in MongoDB

**1. Create Index:**

- Creating an index in MongoDB helps speed up search queries by organizing data for faster access. It works like an index in a book, letting MongoDB find results without scanning every document. Indexes can be created on one or more fields in ascending or descending order.

| db.students.createIndex({name: 1}) |
| --- |

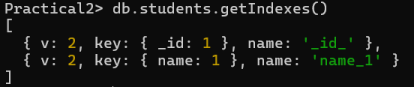


**2. View Existing Indexes:**

- It is used to view all indexes on a MongoDB collection.

It shows details like index fields, type (e.g., single-field, compound), and whether it’s unique or default.

| db.students.getIndexes() |
| --- |

**3. Drop an Index:**

- dropIndex() is used to delete a specific index from a MongoDB collection. This is useful if an index is no longer needed or is affecting write performance.

| db.students.dropIndex({name: 1}) |
| --- |

11

**Prasanna Gaekwad Rollno:-04 4. Ordering Operations (Sorting):**

**-** sort() in MongoDB is used to order query results based on one or more fields. You can sort in ascending (1) or descending (-1) order.

| db.students.find().sort({age: 1}) |
| --- |



| db.students.find().sort({name: -1}) |
| --- |



12

**Prasanna Gaekwad Rollno:-04 5. Limit and Skip Results:**

**-** limit() in MongoDB restricts the number of documents returned by a query. -skip() is used to bypass a specified number of documents, useful for pagination.

| db.students.find().sort({age: 1}).limit(2) |
| --- |



| db.students.find().sort({age: 1}).skip(1) |
| --- |



13

**Prasanna Gaekwad Rollno:-04**

**Practical No: 04**

**Aim:** To perform Create, Read, Update, and Delete operations on a CouchDB database through the Futon web interface

**Description:**

CouchDB is an open-source NoSQL database that stores data as flexible JSON documents. It uses HTTP and RESTful APIs to interact with the database, making it easy to work with over the web. CouchDB supports master-master replication, fault tolerance, and conflict resolution, making it great for distributed systems. It also includes a web-based UI called Fauxton for managing databases and documents.

**Step 1 : Install CouchDB**

Search CouchDB on Google

Click on the first link.



Click on the download button and download CouchDb



14

**Prasanna Gaekwad Rollno:-04**

Select the appropriate version and click on the appropriate icon of your operating system.

Click on Download CouchDb



Open the file explorer of your Computer/Laptop.Go to Downloads and click on the windows install packager of couchdb



15

**Prasanna Gaekwad Rollno:-04**

Click on next





16

**Prasanna Gaekwad Rollno:-04**

Create CouchDB Admin Account by giving account name and password 

Click on Random Cookie.It will randomly generate a cookie value

17

**Prasanna Gaekwad Rollno:-04** Click on Install



Let the system setup the wizard installs of Apache CouchDB

18

**Prasanna Gaekwad Rollno:-04** Click on Finish



Open Google. Type the url “http://localhost:5984/\_utils” on the address bar. 

Login with the username and password which we had set during the installation.

19

**Prasanna Gaekwad Rollno:-04**

**Step 2: Click on “Create Database”**

****

Give the database name and then click on create.



**Step 3: Create 5 Documents (CRUD: Create)**

Manually create 5 student documents using the Futon interface. 1. Click on the student database.

2. Click “+ New Document”

3. Click “Edit” and enter the following JSON fields:

20

**Prasanna Gaekwad Rollno:-04 **

Create 5 Documents (CRUD : Create)

| {  "RollNo": 101,  "Name": "Alice",  "Dept": "CSE",  "Year": 2  } |
| --- |

| {  "RollNo": 102,  "Name": "Bob",  "Dept": "IT",  "Year": 1  } |
| --- |

| {  "RollNo": 103,  "Name": "Carol",  "Dept": "ECE",  "Year": 3  } |
| --- |

| {  "RollNo": 104,  "Name": "David",  "Dept": "MECH",  "Year": 4  } |
| --- |

{

"RollNo": 105,

"Name": "Eva",

"Dept": "Civil",

21

**Prasanna Gaekwad Rollno:-04** "Year": 2

}







22

**Prasanna Gaekwad Rollno:-04 **

**Step 4: Read Documents (CRUD: Read)**

View and verify the inserted documents.

● Open the student database.

● Click on “All Documents”.

● Click on any document to view the contents.

Each document will display the fields entered in JSON format. 

**Step 5: Update a Document (CRUD: Update)**

Update RollNo 103 (Carol) to change the year from 3 to 4. 1. Click on the document for Carol (RollNo 103).

2. Click Edit.

3. Change the Year field from 3 to 4.

4. Click Save Document.

Document is updated and a new \_rev value will be generated automacally.

23

**Prasanna Gaekwad Rollno:-04 **

**Step 6: Delete a Specific Document (CRUD: Delete)**

Delete the document of RollNo 104 (David).

1. Open the student database.

2. Click on the document where RollNo = 104.

3. Click the Delete (trash icon) on top-right.

4. Confirm the prompt: *“Are you sure you want to delete this doc?”* 5. Click Delete Document.

David’s document is now removed. Only 4 documents should remain.



**Step 7: Upload an Excel File to a Document (Attachments)**

Attach a sample Excel file (e.g., student\_marks.xlsx) to Roll No 105 (Eva). 1. Open the document for Roll No 105.

2. Click “Upload Attachment”.

3. Browse and select the Excel file (e.g., student\_marks.xlsx).

4. Click Upload.

24

**Prasanna Gaekwad Rollno:-04** 5. The file will appear under the \_attachments field.

You can now download or preview the attached Excel file



25

**Prasanna Gaekwad Rollno:-04**

**PRACTICAL NO: 5**

**Aim**: To demonstrate creation, insertion, update, deletion, and retrieval operations on a Cassandra database using CQL

**Description:**

• Apache Cassandra is a highly scalable, distributed NoSQL database designed to handle large amounts of data across many servers.

• It provides high availability with no single point of failure, making it ideal for mission-critical applications.

• Cassandra uses a peer-to-peer architecture and supports replication across multiple data centers.

• Its write-optimized design ensures fast performance for insert-heavy workloads. • Data is stored in a column-family format, which allows flexible schema design. Popular use cases include real-time analytics, IoT data, and time-series applications.

**Login and Database Setup**

❖ Go to: hps://auth.cloud.datastax.com/

❖ Sign in with your email ID.

❖ Create a new database:

➢ **Database Name**: bigdata\_db

➢ **Keyspace Name**: bigdata

➢ **Region**

26

**Prasanna Gaekwad Rollno:-04 **

**Open**

**CQL Console**

Once the database is active, click **"CQL Console"** from the dashboard to execute your queries.

**1.Use Keyspace**

| USE bigdata; |
| --- |

A **keyspace** in Cassandra is the top-level namespace that defines how data is stored on the cluster. It holds one or more tables and defines replication settings like the strategy and replication factor. Think of it as similar to a database in traditional RDBMS systems.

**2. Create Table**

27

**Prasanna Gaekwad Rollno:-04**

| CREATE TABLE users (  userid UUID PRIMARY KEY, name TEXT,  age INT,  email TEXT,  city TEXT  ); |
| --- |



**3.Insert Data (CRUD: Create)**

Insert 5 user records into the table.

| INSERT INTO users (userid, name, age, email, city)  VALUES (2734f2e4-726e-445e-97d3-6f25a7f06afa, 'Alice', 25, 'alice@gmail.com', 'Chennai');  INSERT INTO users (userid, name, age, email, city)  VALUES (8e3503f2-1722-4d12-af25-a0859136ea85, 'Bob', 30, 'bob@gmail.com', 'Mumbai');  INSERT INTO users (userid, name, age, email, city)  VALUES (cee9efa1-2b6f-4555-a880-43b20783ae36, 'Carol', 28, 'carol@gmail.com', 'Bangalore');  INSERT INTO users (userid, name, age, email, city)  VALUES (943270f2-51e8-49c5-b0c6-e0dfcb9fca40, 'David', 35, 'david@gmail.com', 'Delhi');  INSERT INTO users (userid, name, age, email, city)  VALUES (915a4c1c-2d47-4e98-b394-29cd48e51b8f,'Eva', 27, 'eva@gmail.com', 'Hyderabad'); |
| --- |

28

**Prasanna Gaekwad Rollno:-04 **

You may copy the UUID from one of the inserted rows to use in UPDATE/DELETE commands.

**4. Display Records (CRUD: Read)**

| SELECT \* FROM users; |
| --- |



**5.Update Record (CRUD: Update)**

| UPDATE users  SET age = 31, city = 'Pune'  WHERE userid = 915a4c1c-2d47-4e98-b394-29cd48e51b8f; |
| --- |



29

**Prasanna Gaekwad Rollno:-04**

**6. Delete Record (CRUD: Delete)**

Delete the record of user named **Carol** (replace UUID):

| DELETE FROM users  WHERE userid = cee9efa1-2b6f-4555-a880-43b20783ae36; |
| --- |

Confirm deletion:

| SELECT \* FROM users; |
| --- |



**7.Drop Table**

| DROP TABLE users; |
| --- |

30

**Prasanna Gaekwad Rollno:-04**

**Practical No. 6**

**Aim:** To perform various operations in Redis using CLI.

**Description:**

Redis is an open-source, in-memory data store that works as a database, cache, and message broker. It delivers extremely low latency since operations run directly in RAM. It supports rich data structures like strings, hashes, lists, sets, sorted sets, and streams. Persistence options let you save snapshots or logs to disk for durability. Common use cases include caching, real-time leaderboards, chat systems, and analytics. Its simplicity and speed make it a core tool for high-performance applications.

• Installation and Setup :

Step 1 : Search for “Redis Github Download” on any Browser. And Open the first link.



Step 2 : Download the **.msi** file



31

**Prasanna Gaekwad Rollno:-04** Step 3 : Run the msi file and finish the setup by just clicking next for every window within the redis setup wizard.



Step 4: Go to program files in the drive you installed Redis and open the Redis folder. Step 5: Run the file “redis-cli” to open the cli of redis.



**PING**

Tests the connection with Redis server.

| PING |
| --- |



**CRUD with SET and GET (Strings)**

**1. Create (Insert Records)**

Store simple values like name, age, and course for 3 students.

| SET student:101:name "Shrinesh"  SET student:101:course "BMS"  SET student:102:name "Shweta"  SET student:102:course "AI"  SET student:103:name "Kiran"  SET student:103:course "DBMS" |
| --- |

32

**Prasanna Gaekwad Rollno:-04 **

**2. Read (Retrieve Records)**

| GET student:101:name  GET student:102:course |
| --- |



**3. Update (Modify Records)**

| SET student:103:course "Deep Learning" |
| --- |



**4. Delete (Remove Data)**

| DEL student:102:course  DEL student:103:name |
| --- |



**Limitation:**

With SET/GET, each field needs a separate key (student:201:name, student:201:course).

For large records, this becomes hard to manage. We use **Hashes (HMSET, HGETALL).**

33

**Prasanna Gaekwad Rollno:-04**

**CRUD with HM**

**1. Create (Insert 5 New Student Records)**

| HMSET student:201 id 201 name "Amit" course "Data Science"  HMSET student:202 id 202 name "Sneha" course "Artificial Intelligence" HMSET student:203 id 203 name "Vikram" course "Machine Learning" HMSET student:204 id 204 name "Pooja" course "Operating Systems" HMSET student:205 id 205 name "Rohan" course "Computer Networks" |
| --- |



**2. Read (Retrieve Records)**

Get all fields of student 201

| HGETALL student:201 |
| --- |



Get only the name of student 202

| HGET student:202 name |
| --- |



Get multiple fields (id and course) of student 203

| HMGET student:203 id course |
| --- |

Show all keys (all students)

34

**Prasanna Gaekwad Rollno:-04**

| KEYS student:\* |
| --- |



**3. Update (Modify Records)**

Update course of student 201 to "Cloud Computing"

| HSET student:201 course "Cloud Computing" |
| --- |



Change name of student 205 to "Rohan Sharma"

| HSET student:205 name "Rohan Sharma" |
| --- |



**4. Delete (Remove Data)**

Remove the course field of student 204

| HDEL student:204 course |
| --- |



Delete full record of student 203

| DEL student:203 |
| --- |



Verify deletion

| EXISTS student:203 |
| --- |

35

**Prasanna Gaekwad Rollno:-04 **

**Common Utility Commands in Redis**

**1. KEYS**

Lists all keys matching a pattern.

| KEYS student:\* |
| --- |



**2. EXISTS**

Checks whether a key exists in Redis.

| EXISTS student:201 |
| --- |



**3. TYPE**

Shows what type of data is stored in a key (string, hash, list, set, etc.).

| TYPE student:201 |
| --- |



**4. EXPIRE**

Set a time-to-live (in seconds) for a key, after which it will be deleted automatically.

| EXPIRE student:202 60 |
| --- |



→ student:202 will be deleted after 60 seconds.

**5. TTL (Time To Live)**

36

**Prasanna Gaekwad Rollno:-04**

Shows the remaining expiry time of a key in seconds.

| TTL student:202 |
| --- |



**Practical No. 07**

**Aim:** To implement MongoDB aggregation using the aggregate() method with operators such as SUM, AVG, MIN, MAX, push, addToSet, first, and last.

**Description:**

MongoDB uses a powerful aggregation framework to process data like a pipeline. Instead of row-by-row queries, it transforms documents through stages. Stages include $match, $group, $project, $sort, $lookup, etc. This allows filtering, reshaping, and combining data efficiently. It works like SQL’s GROUP BY + joins, but with JSON-style flexibility. Optimized for handling large datasets with complex transformations.

**1) Create DB And Collection**

| use Stores |
| --- |



| db.createCollection(“Sales”) |
| --- |



**2) Inserting values**

db.Sales.insert({"id":12,"item":"Soap","price":10,"Quantity":2,"date":ISODate("2014- 01-01T08:00:00Z")})

db.Sales.insert({"id":13,"item":"Soap","price":20,"Quantity":2,"date":ISODate("2014- 02-01T09:00:00Z")})

37

**Prasanna Gaekwad Rollno:-04** db.Sales.insert({"id":17,"item":"Shampoo","price":50,"Quantity":1,"date":ISODate("2 015-01-01T09:00:00Z")})

db.Sales.insert({"id":18,"item":"Biscuit","price":90,"Quantity":4,"date":ISODate("2016 -01-01T09:00:00Z")})

db.Sales.insert({"id":8,"item":"Toffee","price":80,"Quantity":7,"date":ISODate("2016- 03-01T09:00:00Z")})

db.Sales.insert({"id":9,"item":"Facewash","price":100,"Quantity":1,"date":ISODate("2 016-09-01T09:00:00Z")})

db.Sales.insert({"id":10,"item":"Facewash","price":100,"Quantity":1,"date":ISODate(" 2017-09-01T09:00:00Z")})

db.Sales.insert({"id":23,"item":"Toffee","price":50,"Quantity":5,"date":ISODate("2016- 04-01T09:00:00Z")})

db.Sales.insert({"id":98,"item":"Biscuit","price":290,"Quantity":14,"date":ISODate("20 16-05-01T09:00:00Z")})

db.Sales.insert({"id":27,"item":"Shampoo","price":100,"Quantity":2,"date":ISODate(" 2015-01-01T09:00:00Z")})

**3) find():**

To select data from a collection in MongoDB, we can use the find() method.

| db.Sales.find() |
| --- |

38

**Prasanna Gaekwad Rollno:-04 **

**The aggregate() Method**

For the aggregation in MongoDB, you should use aggregate() method. Syntax:

Basic syntax of aggregate() method is as follows −

| >db.COLLECTION\_NAME.aggregate(AGGREGATE\_OPERATION) |
| --- |

**4)SUM:**

$sum: MongoDB $sum returns the sum of numeric values

Syntax: { $sum: <expression> }

Code:

| db.Sales.aggregate([{$group:{\_id:"$item",sumPrice:{$sum:"$price"}}}]) |
| --- |

**5)Average:**

$avg: The MongoDB $avg returns the average value of numeric values. Syntax: { $avg: <expression> }

Code:

| db.Sales.aggregate([{$group:{\_id:"$item",AVG:{$avg:"$price"}}}]) |
| --- |

39

**Prasanna Gaekwad Rollno:-04 6)Minimum:**

$min: The MongoDB $min returns the minimum value.

Syntax: { $min: <expression> }

Code:

| db.Sales.aggregate([{$group:{\_id:"$item",Min:{$min:"$price"}}}]) |
| --- |



**7)Maximum:**

$max: The MongoDB $max returns the maximum value.

Syntax: { $max: <expression> }

Code:

| db.Sales.aggregate([{$group:{\_id:"$item",Max:{$max:"$price"}}}]) |
| --- |

**8)addToSet**

$addToSet: The $addToSet operator adds a value to an array unless the value is already present, in which case $addToSet does nothing to that array. Syntax: { $addToSet: { <field1>: <value1>, ... } }

Code:

| db.Sales.aggregate([{$group:{\_id:"$item",Price:{$addToSet:"$price"},Quantity:{$add ToSet:"$Quantity"}}}]) |
| --- |

40

**Prasanna Gaekwad Rollno:-04 **

| db.Sales.aggregate([{$group:{\_id:{Day:{$dayOfYear:"$date"},Year:{$year:"$date"}},Ite m\_sold:{$addToSet:"$item"}}}]) |
| --- |

**9)push**

$push: The $push operator appends a specified value to an array

Syntax: { $push: { <field1>: <value1>, ... } }

Code:

| db.Sales.aggregate([{$group:{\_id:"$item",Pushq:{$push:"$Quantity"}}}]) |
| --- |

**10)first**

$first: Returns the result of an expression for the first document in a group of documents. Only meaningful when documents are in a defined order Syntax: { $first: <expression> }

Code:

| db.Sales.aggregate([{$group:{\_id:"$item","Date":{$first:"$date"}}}]) |
| --- |

41

**Prasanna Gaekwad Rollno:-04 **

**11)last**

$last: Returns the result of an expression for the last document in a group of documents. Only meaningful when documents are in a defined order. Syntax: { $last: <expression> }

Code:

| db.Sales.aggregate([{$group:{\_id:"$item","Date":{$last:"$date"}}}]) |
| --- |

42

**Prasanna Gaekwad Rollno:-04**

**Practical 8**

**Aim:** Implementation of MapReduce using PySpark in Google Colab.

**Description:** MapReduce in PySpark is done with RDD transformations. map() applies a function to each element, creating key-value pairs if needed. reduceByKey() or groupByKey() then combines values with the same key. This mimics the map (parallel data prep) and reduce (aggregation) stages. Unlike Hadoop’s MapReduce, PySpark runs in memory, so it’s faster. You can chain multiple transformations and then call an action like collect() or saveAsTextFile().

**Install and import PySpark.**

| !pip install pyspark |
| --- |



| from pyspark.sql import SparkSession  spark = SparkSession.builder.master("local[\*]").appName("MapReduceExample").getOrCreate() sc = spark.sparkContext |
| --- |

**Upload your text file into Colab.**

File Icon in top right -> upload icon -> upload the desired file.

43

**Prasanna Gaekwad Rollno:-04 **

****

| Words = sc.textFile("Data Analytics.txt") |
| --- |

**Split lines into words.**

**Map each word into key-value pairs (word, 1).**

| WordPairs = WordsCount.map(lambda word: (word, 1))  WordPairs.take(10) # Display first 10 pairs |
| --- |



**Reduce by key to count occurrences of each word.**

| DistinctWordsCount = WordPairs.reduceByKey(lambda a, b: a + b)  print("Distinct Words:", DistinctWordsCount.count()) |
| --- |

44

**Prasanna Gaekwad Rollno:-04 Collect the results.**

| DistinctWordsCount.collect() |
| --- |



**Sort the words by frequency (top 5).**

| SortedWordsCount = DistinctWordsCount.map(lambda a: (a[1], a[0])).sortByKey(False) SortedWordsCount.top(5) |
| --- |

45