Lab Session 10

10. NoSQL Experiment

Aim:

To understand the basics of NoSQL databases, design a schema, model data, and perform CRUD (Create, Read, Update, Delete) operations using a NoSQL database.

Description:

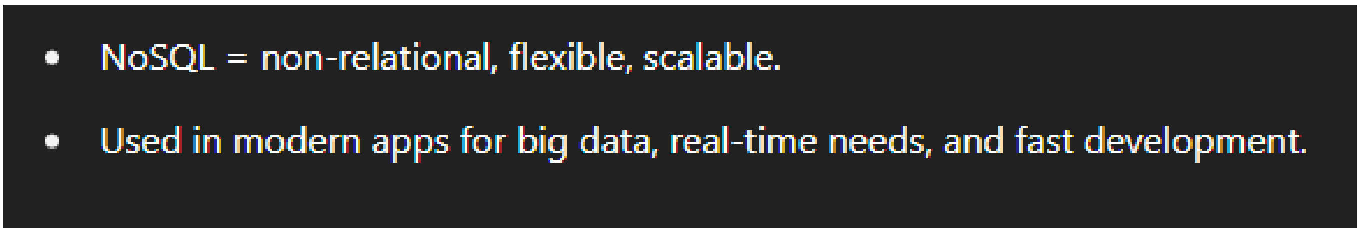
NoSQL databases are a type of database management system that diverges from the traditional relational database model used in SQL databases. They provide flexible data models and are designed to handle large volumes of data and high concurrency. NoSQL databases are commonly used in modern web applications, big data analytics, and real-time data processing scenarios.

Pre-Requisites:

- Choose a NoSQL database to work with (e.g., MongoDB, Cassandra, Redis)
- Install the database software on your local machine or use a cloud-based service.
- Set up a development environment with appropriate drivers or libraries for your chosen NoSQL database.
- Access the database through a command-line interface or a graphical user interface.

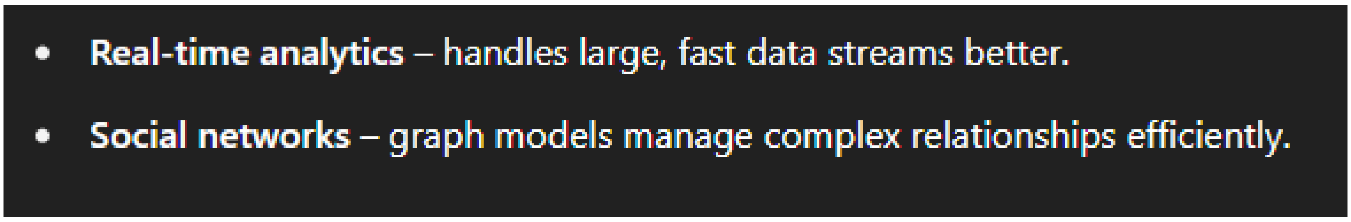
Pre-Lab:

1) What distinguishes NoSQL databases from traditional SQL databases, and why are they increasingly used in modern application development?



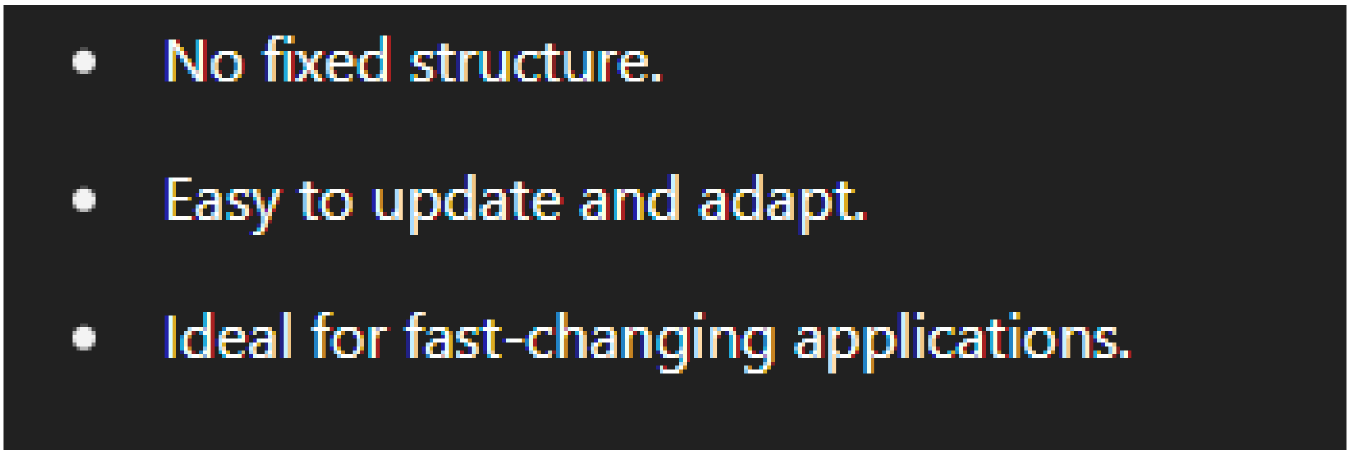
- NoSQL = non-relational, flexible, scalable.
- Used in modern apps for big data, real-time needs, and fast development.

2) Can you name at least two real-world scenarios where NoSQL databases might outperform SQL databases, and explain why?



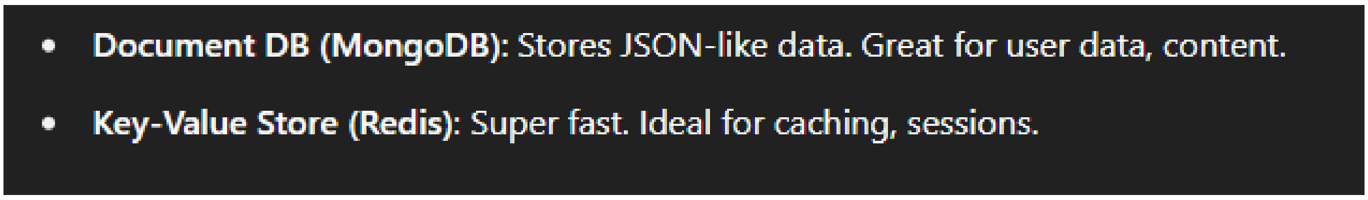
- **Real-time analytics** – handles large, fast data streams better.
- **Social networks** – graph models manage complex relationships efficiently.

3) Describe the concept of schema flexibility in NoSQL databases and discuss its implications for database design and development?



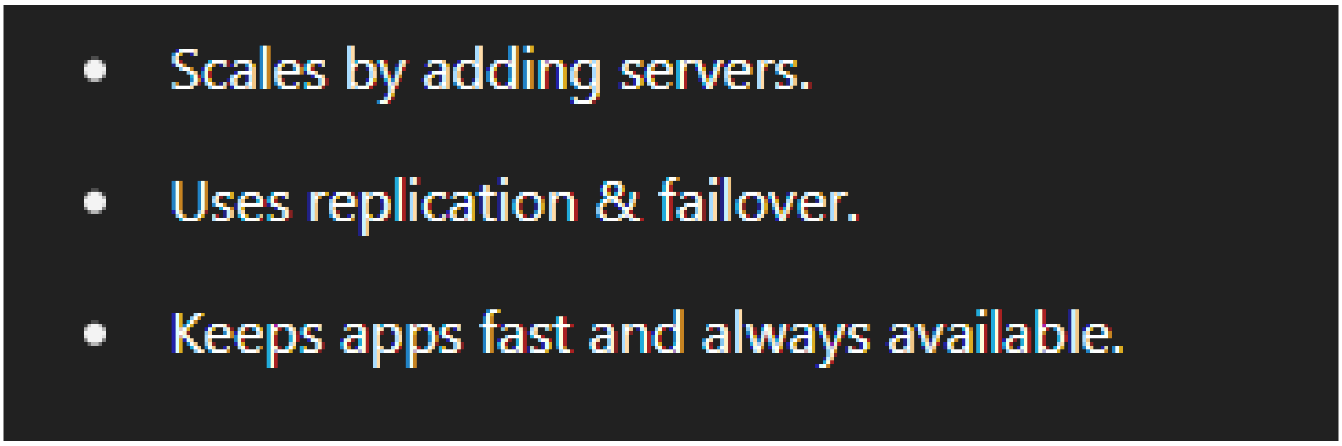
- No fixed structure.
- Easy to update and adapt.
- Ideal for fast-changing applications.

4) Name and briefly explain two types of NoSQL databases, highlighting their respective strengths and ideal use cases?



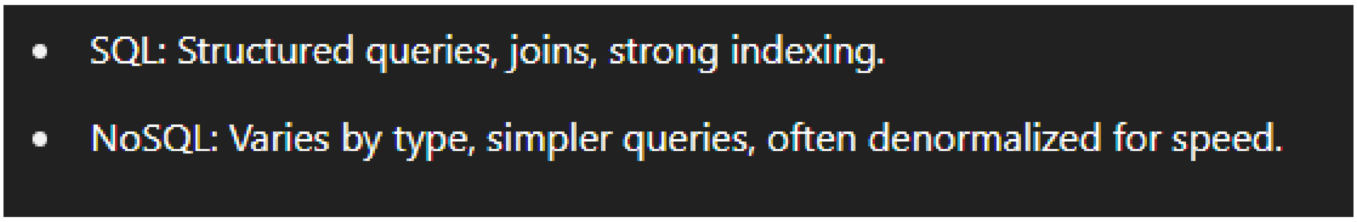
- **Document DB (MongoDB):** Stores JSON-like data. Great for user data, content.
- **Key-Value Store (Redis):** Super fast. Ideal for caching, sessions.

5) How do NoSQL databases typically handle scalability and high availability, and why are these features important for modern applications?



- Scales by adding servers.
- Uses replication & failover.
- Keeps apps fast and always available.

6) Compare and contrast the querying capabilities of NoSQL databases with those of SQL databases, considering factors such as query language and indexing.



- SQL: Structured queries, joins, strong indexing.
- NoSQL: Varies by type, simpler queries, often denormalized for speed.

<u>In-Lab:</u>

E-commerce Platform

You are developing an e-commerce platform where users can browse products, add items to their cart, and place orders. Design a MongoDB schema and implement CRUD operations to manage products, user carts, and orders? Design appropriate schemas and implementing CRUD operations to below mentioned questions and meet the needs of each use case.

1) Develop a query to store a product information including name, description, price, and stock quantity?

```
db.products.insertOne({
  name: "Laptop",
  description: "A powerful gaming laptop",
  price: 1500,
  stockQuantity: 10
});
```

2) Write a query to create accounts with usernames, email addresses, and passwords?

```
db.users.insertOne({
  username: "john_doe",
  email: "john@example.com",
  password: "hashedPassword123",
  cart: []
});
```

3) Develop a query to perform operation on Users can add/remove products from their shopping carts?

✅ **Add to cart**

```
db.users.updateOne(
 { _id: ObjectId("USER_ID_HERE") },
 {
   $push: {
     cart: {
       productId: ObjectId("PRODUCT_ID_HERE"),
       quantity: 1
     }
   }
 }
);
```

❌ **Remove from cart**

```
db.users.updateOne(
 { _id: ObjectId("USER_ID_HERE") },
 {
   $pull: {
     cart: {
       productId: ObjectId("PRODUCT_ID_HERE")
     }
   }
 }
);
```

4) Write a query where Users can place orders containing items from their cart?

```
const user = db.users.findOne({ _id: ObjectId("USER_ID_HERE") });

db.orders.insertOne({
  userId: user._id,
  items: user.cart,
  createdAt: new Date()
});

db.users.updateOne(
  { _id: user._id },
  { $set: { cart: [] } }
);
```

5) Develop a query for to Delete products, user carts, and orders.?

❌ Delete a product

```
db.products.deleteOne({ _id: ObjectId("PRODUCT_ID_HERE") });
```

❌ Delete a user's cart (empty cart)

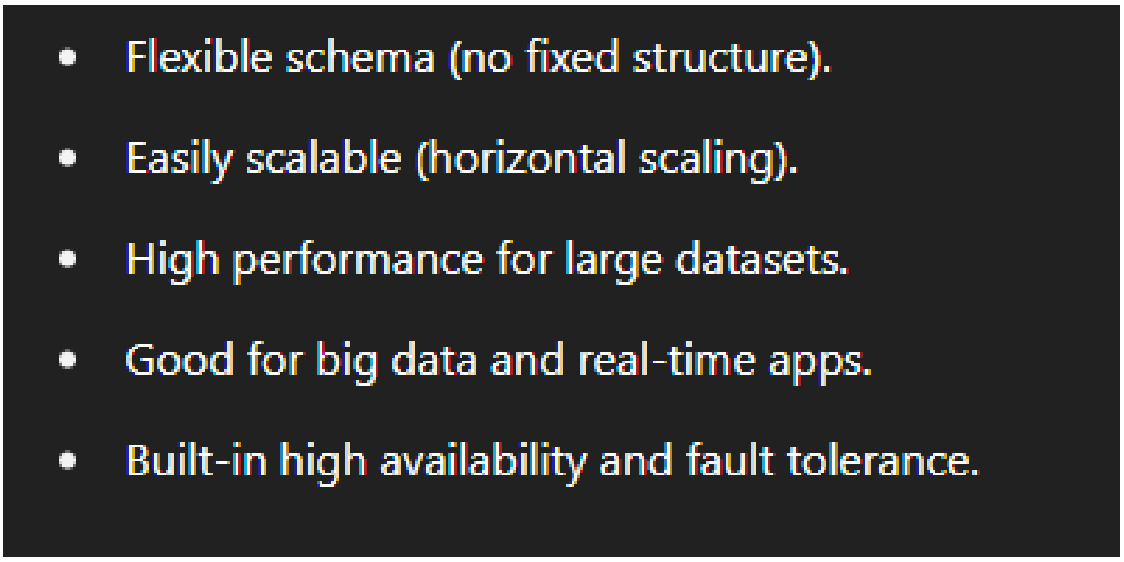
```
db.users.updateOne(
  { _id: ObjectId("USER_ID_HERE") },
  { $set: { cart: [] } }
);
```

❌ Delete an order

```
db.orders.deleteOne({ _id: ObjectId("ORDER_ID_HERE") });
```

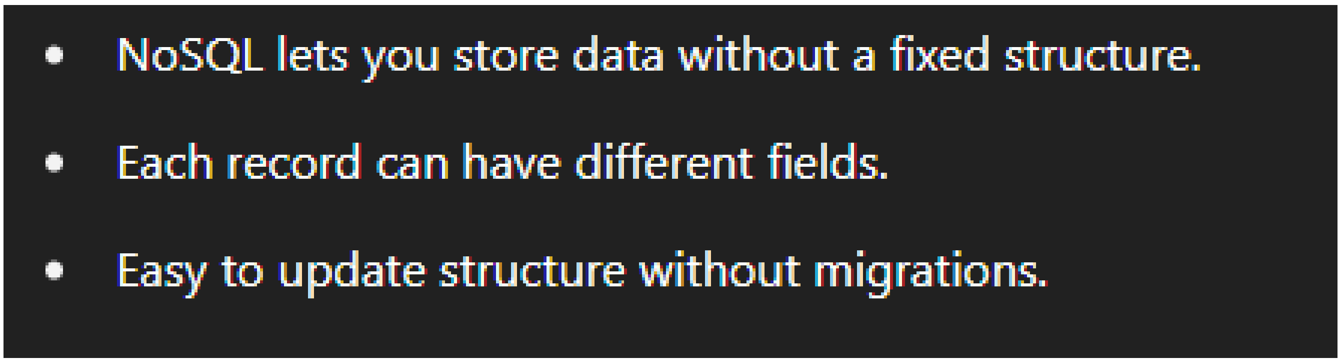
Viva-Voice Questions (In-Lab):

1) What are the advantages of using NoSQL databases over traditional SQL databases?



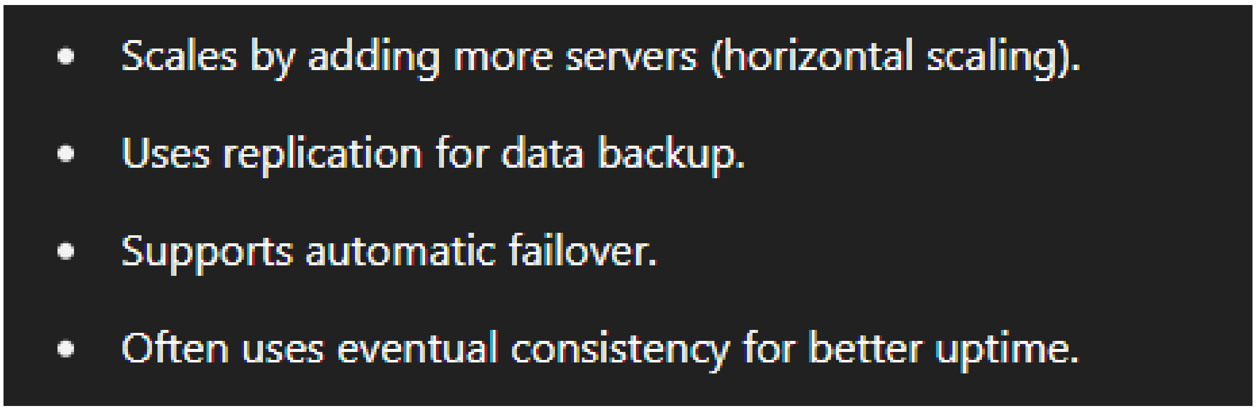
- Flexible schema (no fixed structure).
- Easily scalable (horizontal scaling).
- High performance for large datasets.
- Good for big data and real-time apps.
- Built-in high availability and fault tolerance.

2) Can you explain the concept of schema flexibility in NoSQL databases?



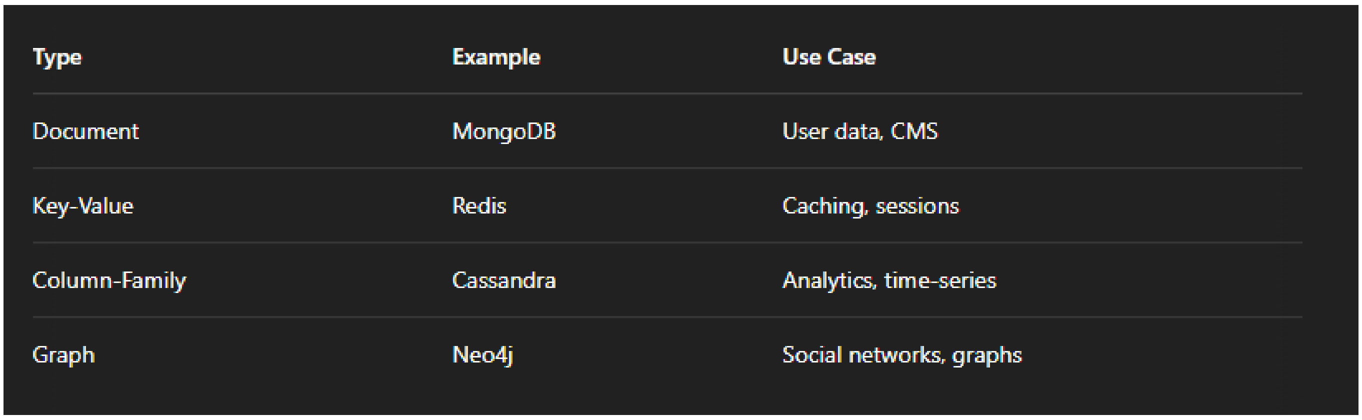
- NoSQL lets you store data without a fixed structure.
- Each record can have different fields.
- Easy to update structure without migrations.

3) How do NoSQL databases handle scalability and high availability?



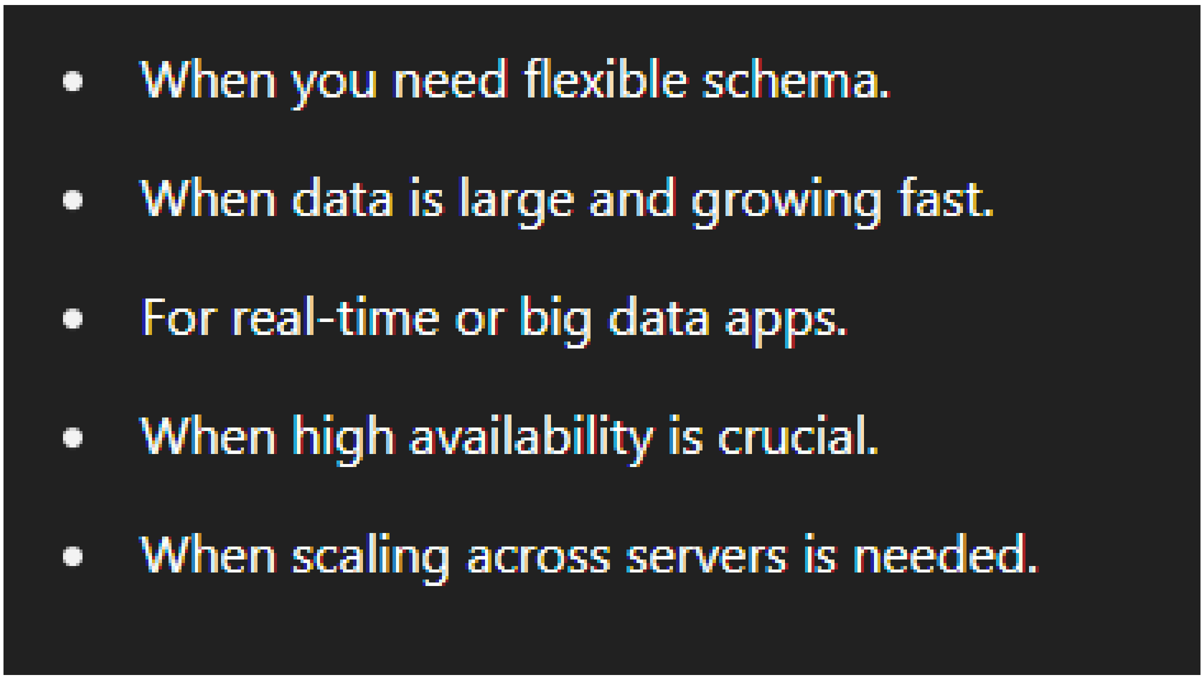
- Scales by adding more servers (horizontal scaling).
- Uses replication for data backup.
- Supports automatic failover.
- Often uses eventual consistency for better uptime.

4) What are some common types of NoSQL databases, and how do they differ from each other?



Type	Example	Use Case
Document	MongoDB	User data, CMS
Key-Value	Redis	Caching, sessions
Column-Family	Cassandra	Analytics, time-series
Graph	Neo4j	Social networks, graphs

5) In what scenarios would you recommend using a NoSQL database instead of a SQL database?



- When you need flexible schema.
- When data is large and growing fast.
- For real-time or big data apps.
- When high availability is crucial.
- When scaling across servers is needed.

Post Lab:

You are developing a task tracking application for managing project tasks and deadlines. Design a MongoDB schema and implement CRUD operations to manage projects, tasks, and user assignments. Design appropriate schemas and implementing CRUD operations to below mentioned questions and meet the needs of each use case.

1) Develop a query where Users can create accounts with usernames, email addresses, and passwords?

```
db.users.insertOne({
  username: "alice_wonder",
  email: "alice@example.com",
  password: "secureHashedPassword123"
});
```

2) Write a query to Users can create projects and assign tasks to team members.

✅ **Create Project**

```
db.projects.insertOne({
  name: "Website Redesign",
  description: "Redesign the company website for 2025",
  members: [ObjectId("USER_ID_1"), ObjectId("USER_ID_2")]
});
```

✅ **Assign Task to Team Member**

```
db.tasks.insertOne({
  projectId: ObjectId("PROJECT_ID_HERE"),
  assignedTo: ObjectId("USER_ID_HERE"),
  title: "Design Homepage",
  description: "Create a responsive homepage UI",
  dueDate: new Date("2025-05-15"),
  status: "pending"
});
```

3) Create a query to find Tasks have details such as title, description, due date, and status (e.g., pending, in progress, completed)?

```
db.tasks.find({
  status: "pending"
}, {
  title: 1,
  description: 1,
  dueDate: 1,
  status: 1
});
```

To list all tasks for a project:

```
db.tasks.find({ projectId: ObjectId("PROJECT_ID_HERE") });
```

4) Write a query to Update task details and project assignments?

✏️ Update Task Details

```
db.tasks.updateOne(
  { _id: ObjectId("TASK_ID_HERE") },
  {
    $set: {
      title: "Updated Homepage Design",
      status: "in progress",
      dueDate: new Date("2025-05-20")
    }
  }
);
```

✏️ **Add New Member to Project**

```
db.projects.updateOne(
  { _id: ObjectId("PROJECT_ID_HERE") },
  {
    $addToSet: {
      members: ObjectId("NEW_USER_ID")
    }
  }
);
```

5) Write a query to Perform operations to Delete tasks and projects.

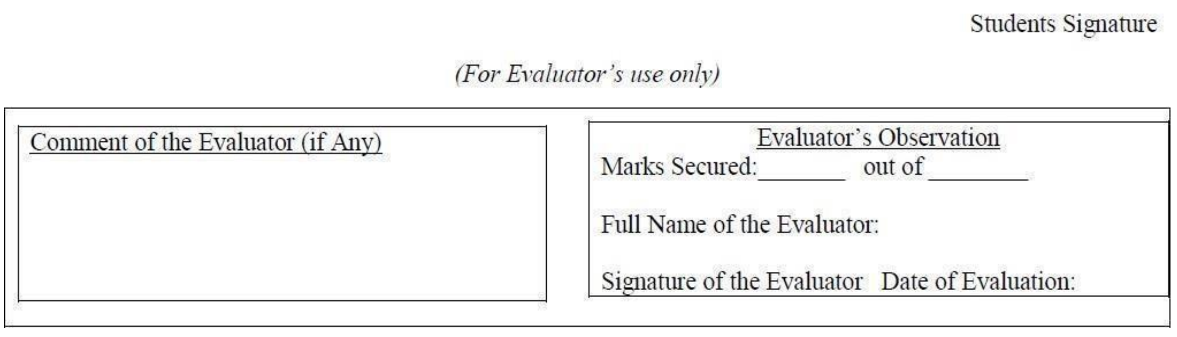
❌ Delete a Task

```
db.tasks.deleteOne({ _id: ObjectId("TASK_ID_HERE") });
```

❌ Delete a Project (and optionally its tasks)

```
db.projects.deleteOne({ _id: ObjectId("PROJECT_ID_HERE") });


db.tasks.deleteMany({ projectId: ObjectId("PROJECT_ID_HERE") });
```



Students Signature

(For Evaluator's use only)

Comment of the Evaluator (if Any)	Evaluator's Observation Marks Secured:_______ out of ________ Full Name of the Evaluator: Signature of the Evaluator Date of Evaluation: