1. Connectivity

Connectivity refers to the **link or communication channel** between a **database** and a **user or application**, which enables the sending, receiving, and management of data.

---

2. Cursor

A **cursor** is a **database pointer** that allows traversal over query result sets **one record at a time**. It is mainly used to **retrieve, process, and manipulate rows individually**.

---

3. Trigger

A **trigger** is a **special database program** that **executes automatically** in response to certain events such as **INSERT, UPDATE, or DELETE** operations on a table.

---

4. Aggregate

Aggregate functions perform **calculations on multiple rows** and return a **single summarized value**, such as:

- `SUM()` – Total of values
- `AVG()` – Average value
- `COUNT()` – Number of rows
- `MIN()` / `MAX()` – Smallest / Largest value

---

5. Normal Aggregation

Normal aggregation means using **basic aggregate functions directly** in SQL queries to summarize data. Example:

```
SELECT AVG(salary) FROM employees;
```

---

6. Aggregation Pipeline (MongoDB)

In **MongoDB**, an **aggregation pipeline** is a **series of stages** that process documents in sequence. Each stage performs an operation like **filtering, grouping, sorting, or transforming data** to produce a final aggregated result.

Example stages: `$match → $group → $sort → $project`

---

7. MapReduce

MapReduce is a **data processing technique** used for handling large datasets in two steps:

- **Map Phase:** Filters and processes input data into key–value pairs.
- **Reduce Phase:** Combines, summarizes, or aggregates the results from the Map phase.

---

8. Procedure

A **procedure** (or **stored procedure**) is a **precompiled collection of SQL statements** stored in the database. It can be executed as a unit to perform tasks like **insertions, updates, or calculations**.

Example:

```
CREATE PROCEDURE AddEmp AS
INSERT INTO Employee VALUES (...);
```

---

9. Types of NoSQL Databases

1. **Document-based:** Data stored as JSON-like documents (e.g., MongoDB).
2. **Key–Value:** Data stored as key-value pairs (e.g., Redis).
3. **Column-based:** Data stored in columns, optimized for queries (e.g., Cassandra).
4. **Graph-based:** Data stored as nodes and relationships (e.g., Neo4j).

10. Types of Cursor

- **Implicit Cursor:** Created automatically by the database system for SQL statements like `SELECT INTO`.
- **Explicit Cursor:** Declared and controlled by the programmer for row-by-row data processing.

11. Difference between Function and Procedure

Feature	Function	Procedure
Return Type	Must return a single value	May or may not return a value
Use in SQL	Can be used inside a `SELECT` statement	Cannot be used inside a `SELECT`
Purpose	Used for computation and returning results	Used to perform operations or processes
Called By	Invoked using SQL expressions	Invoked using `EXEC` or `CALL`
Transaction Handling	Cannot use `COMMIT` or `ROLLBACK`	Can use transaction control statements

Difference Between MongoDB and NoSQL

Basis	MongoDB	NoSQL
Definition	MongoDB is a **specific NoSQL database** that stores data in **document format (BSON/JSON)**.	NoSQL is a **broad database category** that includes all databases **not based on the relational model (non-SQL)**.
Type	It is a **Document-Oriented Database**.	It includes **four main types** — Document-based, Key-Value, Column-based, and Graph-based.
Data Storage Format	Stores data as **JSON-like documents** with fields and values.	Data can be stored in **different formats** depending on the NoSQL type (documents, key-value pairs, columns, or graphs).
Example	MongoDB	Examples include **MongoDB**, **Cassandra**, **Redis**, **Neo4j**, **CouchDB**, etc.
Query Language	Uses **MongoDB Query Language (MQL)**.	Each NoSQL database may have **its own query language or API**.
Schema	**Schema-less** – collections can store documents with different fields.	Most NoSQL databases are **schema-less or flexible-schema**.
Use Case	Best suited for **JSON data, hierarchical data, and real-time analytics**.	Used where **large-scale, unstructured, or semi-structured data** needs to be stored efficiently.

In Simple Terms

- **NoSQL** → A *category* of databases (non-relational).
- **MongoDB** → A *specific example* of a NoSQL database (document-based).

Functions Used in MongoDB and NoSQL

1. MongoDB Functions

MongoDB uses **MongoDB Query Language (MQL)** — a JavaScript-like syntax that provides **functions and methods** to perform CRUD (Create, Read, Update, Delete) and aggregation operations.

□ a) Data Insertion Functions

- `insertOne()` – Inserts a single document.

  ```
  db.students.insertOne({ name: "Ravi", age: 20 });
  ```

- `insertMany()` – Inserts multiple documents at once.

  ```
  db.students.insertMany([{ name: "Asha" }, { name: "Vijay" }]);
  ```

□ b) Data Retrieval Functions

- `find()` – Retrieves documents from a collection.

  ```
  db.students.find();
  ```

- `findOne()` – Retrieves only the first matching document.

  ```
  db.students.findOne({ name: "Ravi" });
  ```

□ c) Data Update Functions

- `updateOne()` – Updates a single matching document.

  ```
  db.students.updateOne({ name: "Ravi" }, { $set: { age: 21 } });
  ```

- `updateMany()` – Updates multiple matching documents.

  ```
  db.students.updateMany({ class: "FY" }, { $set: { class: "SY" } });
  ```

□ d) Data Deletion Functions

- `deleteOne()` – Deletes the first matching document.

  ```
  db.students.deleteOne({ name: "Asha" });
  ```

- `deleteMany()` – Deletes all matching documents.

  ```
  db.students.deleteMany({ class: "FY" });
  ```

□ e) Aggregation Functions

MongoDB provides **aggregate functions** for complex data processing:

- `$sum` – Calculates total of a numeric field.
- `$avg` – Finds average.
- `$min` / `$max` – Finds smallest / largest value.
- `$count` – Counts number of documents.
- `$group` – Groups data by field(s).
- `$match` – Filters documents (like `WHERE` in SQL).

Example:

```
db.students.aggregate([
  { $group: { _id: "$class", avgMarks: { $avg: "$marks" } } }
]);
```

---

2. NoSQL Functions (General)

NoSQL databases have **different functional APIs** depending on their type — but all support common operations similar to CRUD.

□ a) Document-Based (e.g., MongoDB, CouchDB)

Functions:

- Insert, Find, Update, Delete documents
- Aggregation pipelines
- MapReduce functions

□ b) Key-Value Databases (e.g., Redis, DynamoDB)

Functions:

- `SET key value` – Store a value.
- `GET key` – Retrieve a value.
- `DEL key` – Delete a key.
- `INCR key` / `DECR key` – Increment or decrement numeric values.

□ c) Column-Based Databases (e.g., Cassandra, HBase)

Functions:

- `INSERT INTO` – Add a new row.
- `SELECT * FROM` – Retrieve data.

- `UPDATE` – Modify column values.
- `DELETE` – Remove rows or columns.
- `COUNT()` / `SUM()` – Aggregate functions for numeric data.

□ d) Graph-Based Databases (e.g., Neo4j)

Functions:

- `CREATE` – Create nodes and relationships.
- `MATCH` – Find patterns between nodes.
- `RETURN` – Display query results.
- `COUNT()`, `SUM()`, `AVG()` – Perform aggregation on relationships.

Example:

```
MATCH (a:Person)-[:FRIEND_WITH]->(b:Person)
RETURN a.name, COUNT(b) AS friends;
```

Summary Table

Operation	MongoDB Function	Equivalent in Other NoSQL Types
Insert Data	`insertOne()`, `insertMany()`	`SET`, `INSERT INTO`, `CREATE`
Retrieve Data	`find()`, `findOne()`	`GET`, `SELECT`, `MATCH`
Update Data	`updateOne()`, `updateMany()`	`SET`, `UPDATE`
Delete Data	`deleteOne()`, `deleteMany()`	`DEL`, `DELETE`
Aggregate Data	`$sum`, `$avg`, `$group`	`COUNT()`, `SUM()`, MapReduce

□ Difference Between `map()` and `mapReduce()` in MongoDB

Feature	`map()`	`mapReduce()`
Definition	`map()` is a **JavaScript function** that processes each document and emits key-value pairs.	`mapReduce()` is a **MongoDB function** that uses both `map()` and `reduce()` functions together to perform aggregation and transformation on large data sets.
Purpose	Used to map (transform) input documents into key-value pairs.	Used to perform both mapping and reduction — it maps, groups, and then summarizes or aggregates results.
Stage	It is only the **first phase** of MapReduce.	It is the **complete process** that includes both `map()` and `reduce()` functions (and optionally, a `finalize()` phase).
Aggregation Capability	Alone, it cannot aggregate data; it just emits key-value pairs.	It can perform full aggregation, e.g., counting, summing, averaging.
Execution	Defined by user, but not directly executed in MongoDB unless passed to `mapReduce()`.	Executed using the `db.collection.mapReduce()` command.
Output	Emits intermediate key-value pairs (not final results).	Produces a new collection or inline results containing aggregated data.
Example Use	Used inside `mapReduce()` to process documents.	Used independently to execute the entire MapReduce job.

□ Example for Clarity

Let's say we have a collection `students`:

```
{ name: "Amit", subject: "DBMS", marks: 80 }
{ name: "Priya", subject: "DBMS", marks: 90 }
{ name: "Raj", subject: "CN", marks: 70 }
```

Map Function (only):

```
var mapFunction = function() {
    emit(this.subject, this.marks);
};
```

□ This just *maps* each document into key-value pairs like:

```
("DBMS", 80)
("DBMS", 90)
("CN", 70)
```

Reduce Function:

```
var reduceFunction = function(key, values) {
    return Array.avg(values);
};
```

Using MapReduce Together:

```
db.students.mapReduce(
    mapFunction,
    reduceFunction,
    { out: "average_marks" }
);
```

□ Result stored in `average_marks` collection:

```
{ "_id": "DBMS", "value": 85 }
{ "_id": "CN", "value": 70 }
```

□ In Simple Terms:

- **`map()`** = transforms and emits data (like "divide the work").
- **`reduce()`** = combines the results (like "collect and summarize").
- **`mapReduce()`** = runs both together on MongoDB data.

□ MongoDB Aggregation Pipeline – Practical Example Set

Database: `BSIOTR`

Collection: `students`

□ Sample Data

```
db.students.insertMany([
  { name: "Ravi", city: "Delhi", subject: "Math", marks: 90, hobbies: ["Reading", "Cricket"] },
  { name: "Asha", city: "Delhi", subject: "Science", marks: 90, hobbies: ["Music", "Dance"] },
  { name: "Vijay", city: "Mumbai", subject: "Math", marks: 70, hobbies: ["Cricket"] },
  { name: "Kiran", city: "Mumbai", subject: "Science", marks: 80, hobbies: ["Reading", "Music"] }
]);
```

□ 1□ `$match` – Filtering Documents

Purpose: Select documents that match specific conditions (similar to `WHERE` in SQL).

Code:

```
db.students.aggregate([
  { $match: { city: "Delhi" } }
]);
```

Explanation: Filters and returns only students from **Delhi**.

Output:

```
[
  { name: "Ravi", city: "Delhi", subject: "Math", marks: 90 },
  { name: "Asha", city: "Delhi", subject: "Science", marks: 90 }
]
```

□ 2□ `$group` – Grouping and Aggregation

Purpose: Groups documents by one or more fields and applies an aggregate function.

Code:

```
db.students.aggregate([
  {
    $group: {
      _id: "$city",
      totalStudents: { $sum: 1 },
      avgMarks: { $avg: "$marks" }
    }
  }
]);
```

Explanation:

- Groups students **by city**.
- Calculates **total number of students** and **average marks** per city.

Output:

```
[
  { _id: "Delhi", totalStudents: 2, avgMarks: 90 },
  { _id: "Mumbai", totalStudents: 2, avgMarks: 75 }
]
```

□ 3️⃣ `$project` – Selecting or Reshaping Fields

Purpose: Controls which fields appear in the final output.

Code:

```
db.students.aggregate([
  {
    $project: {
      _id: 0,
      StudentName: "$name",
      Subject: "$subject",
      City: "$city",
      Marks: "$marks"
    }
  }
]);
```

Explanation:

- Hides `_id` field.
- Renames other fields for clear output.

Output:

```
[
  { StudentName: "Ravi", Subject: "Math", City: "Delhi", Marks: 90 },
  { StudentName: "Asha", Subject: "Science", City: "Delhi", Marks: 90 },
  { StudentName: "Vijay", Subject: "Math", City: "Mumbai", Marks: 70 },
  { StudentName: "Kiran", Subject: "Science", City: "Mumbai", Marks: 80 }
]
```

□ 4️⃣ `$sort` – Sorting the Results

Purpose: Sorts documents in ascending (1) or descending (-1) order.

Code:

```
db.students.aggregate([
  { $sort: { marks: -1 } }
]);
```

Explanation: Sorts students by **marks in descending order**.

Output:

```
[
  { name: "Ravi", marks: 90 },
  { name: "Asha", marks: 90 },
  { name: "Kiran", marks: 80 },
  { name: "Vijay", marks: 70 }
]
```

□ 5⃣ `$unwind` – Deconstructing Array Fields

Purpose: Breaks an array field into multiple documents (1 document per array element).

Code:

```
db.students.aggregate([
  { $unwind: "$hobbies" },
  {
    $project: {
      _id: 0,
      name: 1,
      city: 1,
      hobbies: 1
    }
  }
]);
```

Explanation: The `hobbies` array is split into separate documents for each hobby.

Output:

```
[
  { name: "Ravi", city: "Delhi", hobbies: "Reading" },
  { name: "Ravi", city: "Delhi", hobbies: "Cricket" },
  { name: "Asha", city: "Delhi", hobbies: "Music" },
  { name: "Asha", city: "Delhi", hobbies: "Dance" },
  { name: "Vijay", city: "Mumbai", hobbies: "Cricket" },
  { name: "Kiran", city: "Mumbai", hobbies: "Reading" },
  { name: "Kiran", city: "Mumbai", hobbies: "Music" }
]
```

□ 6⃣ Combined Example (Full Pipeline)

Average Marks per Subject per City

```
db.students.aggregate([
  { $match: { marks: { $gte: 70 } } },
  {
    $group: {
      _id: { city: "$city", subject: "$subject" },
      avgMarks: { $avg: "$marks" }
    }
  },
  {
    $project: {
      _id: 0,
      city: "$_id.city",
      subject: "$_id.subject",
      avgMarks: 1
    }
  },
  { $sort: { city: 1, subject: 1 } }
]);
```

Output:

```
[
  { city: "Delhi", subject: "Math", avgMarks: 90 },
  { city: "Delhi", subject: "Science", avgMarks: 90 },
  { city: "Mumbai", subject: "Math", avgMarks: 70 },
  { city: "Mumbai", subject: "Science", avgMarks: 80 }
]
```

□ Summary Table

Stage	Purpose	Example
`$match`	Filters documents	`{ $match: { city: "Delhi" } }`
`$group`	Groups and aggregates	`{ $group: { _id: "$city", total: { $sum: 1 } } }`
`$project`	Selects/reshapes output fields	`{ $project: { name: 1, marks: 1 } }`
`$sort`	Sorts documents	`{ $sort: { marks: -1 } }`
`$unwind`	Expands array values	`{ $unwind: "$hobbies" }`

Here's your exact content — fully preserved — but updated with **MySQL syntax** where applicable (especially for the explicit cursor example). Everything else remains *exactly* the same as your provided Markdown.

□ CURSOR IN SQL / PL/SQL

□ Definition

A **cursor** is a **pointer or handle** that allows **row-by-row processing** of query results in **PL/SQL / MySQL stored programs**. When a SQL query returns multiple rows, a cursor helps process each row **one at a time**.

□ Types of Cursors

Type	Description
Implicit Cursor	Created automatically by the system when a SQL statement (like `INSERT`, `UPDATE`, `DELETE`, or `SELECT INTO`) is executed.
Explicit Cursor	Created manually by the user to handle query results that return **multiple rows**.

□ 1□ Implicit Cursor

□ Syntax:

Implicit cursors are created automatically. We can check their status using **cursor attributes** (functions):

Attribute	Description
`%FOUND`	Returns TRUE if a record was found.
`%NOTFOUND`	Returns TRUE if no record was found.
`%ROWCOUNT`	Returns number of rows affected.
`%ISOPEN`	Always FALSE (implicit cursors close automatically).

□ Example (MySQL):

```
DELIMITER $$

BEGIN
   UPDATE Employee SET salary = salary + 1000 WHERE deptno = 10;
   IF ROW_COUNT() > 0 THEN
      SELECT 'Salary updated successfully.' AS Message;
   ELSE
      SELECT 'No record found.' AS Message;
   END IF;
   SELECT CONCAT('Rows affected: ', ROW_COUNT()) AS Info;
END$$

DELIMITER ;
```

Explanation:

- MySQL automatically handles implicit cursors for DML statements.
- `ROW_COUNT()` in MySQL gives the number of affected rows (similar to `SQL%ROWCOUNT` in PL/SQL).

□ 2□Explicit Cursor

□ Syntax (MySQL):

```
DELIMITER $$

CREATE PROCEDURE display_employees()
BEGIN
   DECLARE v_name VARCHAR(50);
   DECLARE v_salary DECIMAL(10,2);
   DECLARE done INT DEFAULT 0;

   DECLARE emp_cursor CURSOR FOR
       SELECT ename, salary FROM Employee WHERE deptno = 10;

   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

   OPEN emp_cursor;
   read_loop: LOOP
      FETCH emp_cursor INTO v_name, v_salary;
      IF done THEN
         LEAVE read_loop;
      END IF;
      SELECT CONCAT('Name: ', v_name, '  Salary: ', v_salary) AS Employee_Info;
   END LOOP;
   CLOSE emp_cursor;
END$$

DELIMITER ;
```

Output:

```
Name: Ravi  Salary: 50000
Name: Kiran  Salary: 40000
```

□ Functions / Attributes Used with Cursors

Cursor Attribute / Function	Used With	Description
`%ISOPEN` (PL/SQL only)	Explicit	Returns TRUE if the cursor is open.
`%FOUND` / `%NOTFOUND`	Both (PL/SQL)	TRUE if fetch succeeded / failed.
`%ROWCOUNT`	Both (PL/SQL)	Returns number of rows fetched so far.
`ROW_COUNT()` (MySQL)	Implicit	Number of rows affected by DML statement.

□ Cursor Workflow

1. **Declare** → Define the cursor with a query.
2. **Open** → Execute the query and prepare the result set.
3. **Fetch** → Retrieve one row at a time.
4. **Close** → Release memory and cursor resources.

□ Summary Table

Step	Command	Purpose
1. Declare	`DECLARE c1 CURSOR FOR SELECT ...;`	Defines cursor
2. Open	`OPEN c1;`	Executes cursor query
3. Fetch	`FETCH c1 INTO variable;`	Gets one record
4. Close	`CLOSE c1;`	Ends cursor use

□ Short Viva Answer

> **A cursor in SQL / MySQL is a pointer used to handle query results row by row.**
>
> - Two types: **Implicit** (automatic) and **Explicit** (manual).
> - Important functions: `%FOUND`, `%NOTFOUND`, `%ROWCOUNT` (PL/SQL) and `ROW_COUNT()` (MySQL).

□ Database Connectivity Check Code (from your program)

```
package abc;

import java.sql.Connection;
import java.sql.DriverManager;

public class CheckConnection {
    public static void main(String[] args) throws Exception {
        Class.forName("com.mysql.cj.jdbc.Driver");
        Connection c = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/practical", "root", "1234");
        System.out.println("Database Connected...");
        c.close();
    }
}
```

□□ Steps to Add JDBC Connector in Eclipse

1. Download **MySQL Connector/J** from □ https://dev.mysql.com/downloads/connector/j/
2. Extract the ZIP file.
3. Find the JAR file — for example: `mysql-connector-j-9.0.0.jar`
4. In **Eclipse**:
    - Right-click your project → **Properties**
    - Go to **Java Build Path** → **Libraries** tab
    - Click **Add External JARs…**
    - Select the `mysql-connector-j-9.0.0.jar` file
    - Click **Apply and Close**

□ When you run the program, if the connection is successful, it will print:

```
Database Connected...
```

□ MYSQL — VIVA QUESTIONS & ANSWERS

□ 1. Database & Tables

Q: What is a primary key? **A:** It uniquely identifies each record in a table and doesn't allow NULL or duplicate values.

Q: What is a foreign key? **A:** It establishes a relationship between two tables and refers to the primary key of another table.

Q: What is the difference between `DELETE`, `TRUNCATE`, and `DROP`? **A:**

- `DELETE` removes specific rows (can use WHERE).
- `TRUNCATE` removes all rows but keeps the structure.
- `DROP` removes the entire table.

Q: What is a view? **A:** A virtual table based on the result of a query.

Q: What is an index? **A:** A database object that speeds up data retrieval but may slow down insertion or updates.

□ 2. SQL Queries

Q: What does the `GROUP BY` clause do? **A:** Groups rows that have the same values in specified columns and is often used with aggregate functions.

Q: What are aggregate functions? **A:** Functions that perform a calculation on a set of values — `SUM`, `AVG`, `COUNT`, `MAX`, `MIN`.

Q: What is the use of the `HAVING` clause? **A:** It filters results of groups created by `GROUP BY`.

Q: What are Joins? **A:** They combine data from multiple tables based on a related column. Types: `INNER JOIN`, `LEFT JOIN`, `RIGHT JOIN`, `FULL JOIN`.

Q: What is a subquery? **A:** A query inside another query.

□ 3. Company / Employee / Hotel DB Questions

Q: How do you find employees who joined before a particular date? **A:**

```
SELECT * FROM emp WHERE hiredate < '1981-09-30';
```

Q: How do you find maximum salary paid to a salesman? **A:**

```
SELECT MAX(salary) FROM emp WHERE job='SALESMAN';
```

Q: How do you find hotel-wise number of rooms? **A:**

```
SELECT hotelno, COUNT(*) FROM room GROUP BY hotelno;
```

Q: How do you update all room prices by 5%? **A:**

```
UPDATE room SET price = price * 1.05;
```

□ PL/SQL — VIVA QUESTIONS & ANSWERS

□ 1. Basic Concepts

Q: What is PL/SQL? **A:** Procedural Language extension of SQL — it allows using loops, conditions, and exceptions.

Q: What is a cursor? **A:** A pointer that holds the result of a SQL query for row-by-row processing.

Q: Types of Cursors? **A:**

- **Implicit:** Automatically created for SQL statements.
- **Explicit:** Declared by user for controlled record processing.

Q: What are triggers? **A:** Database programs that automatically execute when an event (INSERT, UPDATE, DELETE) occurs.

Q: What are exceptions in PL/SQL? **A:** Conditions that interrupt normal program flow. Types: Predefined, User-defined.

□ 2. Common Cursor Questions

Q: What are cursor attributes? **A:**

- `%FOUND` → True if a row is fetched
- `%NOTFOUND` → True if no row fetched
- `%ROWCOUNT` → Number of rows fetched
- `%ISOPEN` → Checks if cursor is open

Q: What is a parameterized cursor? **A:** A cursor that accepts parameters at runtime.

Q: Example of explicit cursor structure:

```
DECLARE
  CURSOR c1 IS SELECT * FROM emp;
  rec emp%ROWTYPE;
BEGIN
  OPEN c1;
  LOOP
    FETCH c1 INTO rec;
    EXIT WHEN c1%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(rec.ename);
  END LOOP;
  CLOSE c1;
END;
```

□ 3. Triggers

Q: When is a trigger used? **A:** When we want to automatically perform actions before or after INSERT, UPDATE, or DELETE.

Q: Example of BEFORE INSERT trigger:

```
CREATE OR REPLACE TRIGGER check_salary
BEFORE INSERT OR UPDATE ON emp
FOR EACH ROW
BEGIN
  IF :NEW.salary < 50000 THEN
    RAISE_APPLICATION_ERROR(-20001, 'Salary below limit');
  END IF;
END;
```

□ 4. Procedures & Functions

Q: What is the difference between a procedure and a function? **A:**

- Function returns a value.
- Procedure may or may not return a value.

Q: Example of procedure:

```
CREATE PROCEDURE update_salary IS
BEGIN
  UPDATE emp SET salary = salary * 1.10;
END;
```

Q: Example of function:

```
CREATE FUNCTION get_salary(eid NUMBER) RETURN NUMBER IS
  sal NUMBER;
BEGIN
  SELECT salary INTO sal FROM emp WHERE eno=eid;
  RETURN sal;
END;
```

□ MONGODB — VIVA QUESTIONS & ANSWERS

□ 1. Basic Concepts

Q: What is MongoDB? **A:** A NoSQL, document-oriented database that stores data in JSON-like BSON format.

Q: What is the difference between MongoDB and MySQL? **A:**

MongoDB	MySQL
NoSQL (Document-based)	SQL (Table-based)
Uses collections and documents	Uses tables and rows
Dynamic schema	Fixed schema
Uses JavaScript-like queries	Uses SQL syntax

Q: What are collections and documents? **A:**

- Collection → Like a table
- Document → Like a row (JSON object)

□ 2. CRUD Operations

Q: How to insert data in MongoDB?

```
db.teachers.insertOne({Tname:"Raj", dname:"Comp", salary:30000});
```

Q: How to display data?

```
db.teachers.find().pretty();
```

Q: How to update data?

```
db.teachers.updateOne({Tname:"Raj"}, {$set:{salary:35000}});
```

Q: How to delete documents?

```
db.teachers.deleteMany({dname:"IT"});
```

□ 3. Aggregation & Indexes

Q: What is aggregation pipeline? **A:** A sequence of stages (`$match`, `$group`, `$project`, `$sort`, `$limit`) to transform data.

Q: Example — average salary per department:

```
db.teachers.aggregate([
  { $group: { _id:"$dname", avgSalary: { $avg:"$salary" } } }
]);
```

Q: What is indexing in MongoDB? **A:** Improves query speed. Example:

```
db.teachers.createIndex({Tname:1});
```

Q: How to check created indexes?

```
db.teachers.getIndexes();
```

□ 4. MapReduce

Q: What is MapReduce in MongoDB? **A:** A data processing model with two functions — **Map** (process) and **Reduce** (combine results).

Q: Example: Count males and females

```
db.people.mapReduce(
  function(){ emit(this.gender,1); },
  function(key,values){ return Array.sum(values); },
  { out:"gender_count" }
);
```

□ 5. Practical-Based Questions

Q: How do you find teachers of multiple departments (Comp, IT, E&TC)?

```
db.teachers.find({dname:{$in:["Comp","IT","E&TC"]}});
```

Q: How to calculate total salary of all teachers?

```
db.teachers.aggregate([
  { $group: { _id:null, totalSalary: { $sum:"$salary" } } }
]);
```

Q: What does the `save()` method do? **A:** Inserts a new document if it doesn't exist, or updates it if it does.

□ 6. Index & Performance

Q: What is a compound index? **A:** An index on multiple fields.

```
db.teachers.createIndex({dname:1, salary:-1});
```

Q: What is the purpose of indexes? **A:** They improve query performance but slightly slow down writes.

□ 7. MapReduce & Aggregation Viva Shortcuts

Concept	Example Function
`$match`	Filter documents
`$group`	Group & compute aggregate
`$project`	Format output fields
`$sort`	Sort results
`$limit`	Limit output count
`$unwind`	Deconstruct arrays

•

Great question — this is a **very common viva question in MongoDB and Big Data practicals.** Let's explain it clearly and precisely □

□ Difference Between `map()` and `mapReduce()` in MongoDB

Feature	`map()`	`mapReduce()`
Definition	`map()` is a **JavaScript function** that processes each document and emits key-value pairs.	`mapReduce()` is a **MongoDB function** that uses both `map()` and `reduce()` functions together to perform aggregation and transformation on large data sets.
Purpose	Used to map (transform) input documents into key-value pairs.	Used to perform both mapping and reduction — it maps, groups, and then summarizes or aggregates results.
Stage	It is only the **first phase** of MapReduce.	It is the **complete process** that includes both `map()` and `reduce()` functions (and optionally, a `finalize()` phase).
Aggregation Capability	Alone, it cannot aggregate data; it just emits key-value pairs.	It can perform full aggregation, e.g., counting, summing, averaging.
Execution	Defined by user, but not directly executed in MongoDB unless passed to `mapReduce()`.	Executed using the `db.collection.mapReduce()` command.
Output	Emits intermediate key-value pairs (not final results).	Produces a new collection or inline results containing aggregated data.
Example Use	Used inside `mapReduce()` to process documents.	Used independently to execute the entire MapReduce job.

□ Example for Clarity

Let’s say we have a collection `students`:

```
{ name: "Amit", subject: "DBMS", marks: 80 }
{ name: "Priya", subject: "DBMS", marks: 90 }
{ name: "Raj", subject: "CN", marks: 70 }
```

Map Function (only):

```
var mapFunction = function() {
    emit(this.subject, this.marks);
};
```

□ This just *maps* each document into key-value pairs like:

```
("DBMS", 80)
("DBMS", 90)
("CN", 70)
```

Reduce Function:

```
var reduceFunction = function(key, values) {
    return Array.avg(values);
};
```

Using MapReduce Together:

```
db.students.mapReduce(
    mapFunction,
    reduceFunction,
    { out: "average_marks" }
);
```

□ **Result stored in `average_marks` collection**:

```
{ "_id": "DBMS", "value": 85 }
{ "_id": "CN", "value": 70 }
```

□ In Simple Terms:

- **`map()`** = transforms and emits data (like “divide the work”).
- **`reduce()`** = combines the results (like “collect and summarize”).
- **`mapReduce()`** = runs both together on MongoDB data.

□□ ACID and BASE Properties in DBMS Property Type Full Form Explanation A – Atomicity Ensures that all operations in a transaction are completed successfully; if not, the entire transaction is rolled back. C – Consistency Maintains database integrity — data must be valid according to all rules, constraints, and relationships.
I – Isolation Ensures that concurrent transactions do not affect each other’s execution.
D – Durability Once a transaction is committed, it remains permanent even in case of system failure.

□ Used in: Traditional RDBMS like MySQL, Oracle.

Property Type Full Form Explanation B – Basically Available System guarantees availability of data even in case of partial failures.
A – Soft State The state of the system may change over time even without input (due to eventual consistency).
E – Eventual Consistency Data will become consistent across nodes after a certain period of time.

□ Used in: NoSQL databases like MongoDB, Cassandra.

□ Oral Questions and Answers (DBMS LAB)

□ Questions & Answers – DBMS

1) Define Database.

A prearranged collection of figures known as data is called a **database**.

---

2) What is DBMS?

Database Management Systems (DBMS) are applications designed especially to enable user interaction with other applications.

---

3) What are the various kinds of interactions catered by DBMS?

The various kinds of interactions catered by DBMS are:

- Data definition
- Update
- Retrieval
- Administration

---

4) Segregate database technology's development.

The development of database technology is divided into:

- Structure or data model
- Navigational model
- SQL / Relational model

---

5) Who proposed the relational model?

Edgar F. Codd proposed the relational model in **1970**.

---

6) What are the features of Database language?

A database language may also incorporate features like:

- DBMS-specific configuration and management of storage engine
- Computations to modify query results (summing, counting, averaging, grouping, sorting, cross-referencing)
- Constraint enforcement
- Application Programming Interface

---

7) What do database languages do?

As special-purpose languages, they have:

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Query Language

---

8) Define database model.

A **data model** determining fundamentally how data can be stored, manipulated, and organized—and the logical structure of the database—is called a **database model**.

---

9) What is SQL?

Structured Query Language (SQL) is an ANSI standard language used to access and update databases.

---

10) Enlist the various relationships of database.

The various relationships of database are:

- **One-to-One:** Single table related to another table with similar columns.
- **One-to-Many:** Two tables related through primary and foreign key.
- **Many-to-Many:** Junction table connecting multiple tables to each other.

---

11) Define Normalization.

Organizing data to remove **inconsistent dependencies and redundancy** within a database is called **Normalization**.

---

12) Enlist the advantages of normalizing database.

- No duplicate entries
- Saves storage space
- Boosts query performance

---

13) Define Denormalization.

Denormalization is the process of adding redundant data to improve database performance and simplify complex data structures.

---

14) Define DDL and DML.

- **DDL (Data Definition Language):** Manages properties and attributes of database structure.
- **DML (Data Manipulation Language):** Used to manipulate data such as inserting, updating, deleting.

---

15) Enlist some commands of DDL.

- **CREATE:** Used to create tables.

```
CREATE TABLE [column name] ( [column definitions] ) [table parameters];
```

- **ALTER:** Used to modify existing database objects.

```
ALTER objecttype objectname parameters;
```

- **DROP:** Used to delete database objects.

```
DROP objecttype objectname;
```

---

16) Define Union All operator and Union.

- **Union All:** Combines all records of two tables including duplicates.
- **Union:** Combines distinct records of two tables (removes duplicates).

---

17) Define Cursor.

A **cursor** is a database object that helps manipulate data row by row, representing a result set.

---

18) Enlist the cursor types.

- **Dynamic:** Reflects changes while scrolling.
- **Static:** Does not reflect changes and works on snapshot records.
- **Keyset:** Reflects data modifications but not new data.

19) Enlist the types of cursor.

- **Implicit Cursor:** Declared automatically during SQL execution.
- **Explicit Cursor:** Defined by PL/SQL to handle queries returning multiple rows.

20) Define Sub-query.

A **sub-query** is a query nested within another query.

21) Why is GROUP BY clause used?

It is used to derive aggregate values by collecting similar data.

22) Compare Non-clustered and Clustered index.

Both use **B-tree structure**:

- **Clustered Index:** Data is physically ordered by index; only one per table.
- **Non-Clustered Index:** Stores pointers to data; many allowed per table.

23) Define Aggregate functions.

Functions that operate on a collection of values and return a single result, e.g., **SUM(), AVG(), COUNT()**.

24) Define Scalar functions.

Scalar functions depend on an argument and return a single value.

25) What restrictions can you apply when creating views?

- Only the current database can have views.
- Cannot modify computed values.
- Integrity constraints affect INSERT and DELETE.
- Full-text indexes not allowed.
- Temporary views/tables not allowed.
- DEFAULT definitions not allowed.
- Can associate **INSTEAD OF** triggers.

26) Define Correlated Subqueries.

A **correlated subquery** depends on another query for its value; executed once for each row in the main query.

27) Define Data Warehousing.

Data Warehousing is the process of storing and accessing data from a central location for strategic decision-making.

28) Define Join and enlist its types.

Joins define relationships between tables to combine related data.

Types:

- **INNER JOIN**
- **LEFT/RIGHT OUTER JOIN**
- **CROSS JOIN**
- **NATURAL JOIN**
- **EQUI / NON-EQUI JOIN**

---

29) What do you mean by Index Hunting?

Index Hunting is the process of analyzing and improving indexes to enhance query performance.

□ Questions & Answers – MySQL

1) What is MySQL?

MySQL is an **open-source DBMS**, built, supported, and distributed by MySQL AB (now Oracle).

---

2) What are the technical features of MySQL?

- Multithreaded SQL server supporting multiple client programs
- Various backends
- Multiple APIs
- Administrative tools

---

3) Why is MySQL used?

MySQL is **reliable, fast, and easy to use**, and is freely downloadable.

---

4) What are HEAP tables?

HEAP tables:

- Stored in memory for high-speed temporary data
- Don't allow BLOB or TEXT
- Support only simple comparison operators
- Don't support AUTO_INCREMENT

---

5) What is the default port for MySQL Server?

Port 3306

---

6) Advantages of MySQL over Oracle

- Free and open source
- Portable
- Supports GUI and command line
- Easy administration using MySQL Query Browser

---

7) Difference between CHAR and VARCHAR

CHAR	VARCHAR
Fixed length	Variable length
Right-padded with spaces	Stores as-is
Max 255 chars	Depends on defined size

---

8) String types available for column

- SET

- BLOB
- ENUM
- CHAR
- TEXT
- VARCHAR

---

9) How to get current MySQL version

```
SELECT VERSION();
```

---

10) What are the drivers in MySQL?

- PHP Driver
- JDBC Driver
- ODBC Driver
- C Wrapper
- Python, Perl, Ruby Drivers

---

11) What does TIMESTAMP with UPDATE CURRENT_TIMESTAMP do?

Automatically updates the field with current timestamp when any other field changes.

---

12) Difference between Primary Key and Candidate Key

- **Primary Key:** Uniquely identifies each row (only one per table).
- **Candidate Key:** Any column that can qualify as a primary key.

---

13) What if a table has one TIMESTAMP column?

It automatically updates with the current timestamp whenever the row is modified.

---

14) What if AUTO_INCREMENT reaches maximum value?

It stops incrementing; further inserts cause an error.

---

15) How to find last assigned AUTO_INCREMENT value

```
SELECT LAST_INSERT_ID();
```

---

16) How to see all indexes defined for a table

```
SHOW INDEX FROM tablename;
```

---

17) What do % and _ mean in LIKE statement?

- `%` → 0 or more characters
- `_` → Exactly one character

---

18) Difference between NOW() and CURRENT_DATE()

- **NOW()** → Full date and time
- **CURRENT_DATE()** → Only date

---

19) What is a Trigger in MySQL?

A **trigger** is a block of code that executes automatically in response to a specific event.

20) How many Triggers are allowed in MySQL?

Six types:

1. Before Insert
2. After Insert
3. Before Update
4. After Update
5. Before Delete
6. After Delete

□ Questions & Answers – NoSQL

1) Compare NoSQL & RDBMS

Criteria	NoSQL	RDBMS
Data Format	Unordered	Structured
Scalability	Very good	Average
Querying	Limited (no joins)	Uses SQL
Storage Mechanism	Key-value/document-based	Tables with relations

2) What is NoSQL?

A set of technologies developed to handle **large-scale, high-speed data** not suited to relational databases.

3) Features of NoSQL

- Handles structured, semi-structured, and unstructured data
- Scalable and high-performing
- Flexible, object-oriented models
- Distributed, low-cost architecture

□ Questions & Answers – MongoDB

1) What is MongoDB?

A **document-based database** providing high performance, high availability, and easy scalability.

2) What is Namespace in MongoDB?

The concatenation of **database name + collection name**.

3) What is Sharding in MongoDB?

Horizontal partitioning of data across multiple machines to improve scalability.

4) Syntax to Create and Drop Collection

```
db.createCollection(name, options);
db.collection.drop();
```

5) Command Syntax to Insert Document

```
database.collection.insert(document);
```

6) What are Indexes in MongoDB?

Indexes store ordered subsets of data for fast query access.

---

7) Basic Syntax to Use Index

```
db.COLLECTION_NAME.ensureIndex({KEY:1});
```

---

8) Define Cursor.

A database object that enables row-by-row processing of data.

---

9) Types of Cursor

- **Implicit:** Created automatically.
- **Explicit:** Declared explicitly in PL/SQL.

---

10) Define Sub-query.

A query nested inside another query.

---

11) Why GROUP BY is used?

To aggregate similar data values.

---

12) Define Aggregate Functions.

Functions that operate on data sets to return single summarized values.

---

13) What is JSON?

JavaScript Object Notation — lightweight, text-based, human-readable data interchange format used for transmitting data between web servers and clients.

---

14) What are JSON Objects?

A set of **key-value pairs** enclosed in curly braces `{}`.

Example:

```
{
  "name": "Shiv",
  "age": 22
}
```

---

15) JSON Syntax Rules

- Data is stored in **key:value** pairs
- Pairs separated by commas
- Objects enclosed in `{}`
- Arrays enclosed in `[]`

---

16) Advantages of JSON over XML

- Lighter and faster
- Supports object types
- Easily parsed and accessed in JavaScript
- Easier to transmit and read