1.what is Nosql database.difference between mysql and Nosql database.

- Nosql is a non relational database to handle large volumes of unstructured,semi-structured or structured data.
- They are ideal for big data,real time web apps and distributed systems.
- There are four types of nosql database.
    - Document:-
        - Store data as json format.
        - ex:-MongoDB
    - Key-pair:-
        - Store data as key-pair format.
        - Ex:-radis
    - Column based:-
        - Store data for column base
        - ex:-Hbase
    - Graph:-
        - Store data in nodes and relationships.
        - Ex:-neo4j
-

Feature	**MySQL (Relational)**	**NoSQL (Non-Relational)**
Data Model	Table-based (rows & columns)	Document, Key-Value, Column, Graph
Schema	Fixed schema	Dynamic/flexible schema
Scalability	Vertical scaling	Horizontal scaling

Transactions	Supports ACID transactions	Limited ACID, but supports high availability
Query Language	SQL	Varies (e.g., MongoDB uses queries in JSON)
Best For	Structured data	Unstructured/semi-structured data
Examples	MySQL, PostgreSQL	MongoDB, Cassandra, Redis, Neo4j

- **Flexible Schema**
  – You don't need to define the structure of data in advance.
  – You can store different types of data in the same database.

- **Handles Large Data**
  – Best for storing huge amounts of data like social media posts, logs, etc.

- **High Performance**
  – Fast read/write operations, especially for big data and real-time apps.

- **Scalable**
  – Easy to add more servers (horizontal scaling) as data grows.

- **Flexible Data Models**
  – Can store documents, key-value pairs, graphs, or

columns.

- **Good for Unstructured Data**
  – Works well with data that doesn’t fit into tables, like images, or videos.

- **Open-source and Cloud-ready:**
  -Many NoSQL databases are open-source and easily deployable on the cloud.

2.what is mongodb?

- Mongodb is an open source document oriented database that is designed for handling large scale data.
- It is one of the most popular Nosql databases.
- The data can not be stored in table format.
- It stores data in BSON format.
- Advantage:-
  - It is a schemaless Nosql database so do not design a schema.
  - It does not support join operations.
  - It easily integrates big data hadoop.
  - It provides high performance,availability and scalability.

- Features:-
  - **Schema-less Database:**
    - MongoDB does **not need a fixed structure** for data.
    - A **single collection** can have **different types of documents**.
    - Each document can have a different number of fields, data types, and sizes.
    - This gives **great flexibility** compared to relational databases.
  - **Document-Oriented:**
    - MongoDB stores data in **documents (key-value pairs)**, not in rows and columns like MySQL.
    - Each document has its **own unique ID**.
    - Data in documents is **easier to read and update**.
  - **Indexing:**
    - MongoDB automatically creates **indexes** on fields (primary and secondary).

    - This helps in **faster searching** of data.
    - Without indexing, it would take longer to find the data.
- **Scalability:**
    - MongoDB supports **horizontal scaling** using **sharding**.
    - Sharding means **splitting large data** into smaller parts (chunks) and storing them on different servers.
    - It also supports **adding more machines** to manage more data.
- **Replication:**
    - MongoDB makes **copies of data** and stores them on different servers.
    - If one server fails, the data is still **available from another server**.
    - This ensures **high availability** and **data safety**.
- **Aggregation:**

        - MongoDB can **group and analyze data** using aggregation.
        - Similar to **SQL's GROUP BY**.
        - It supports three methods:
            - Aggregation pipeline
            - MapReduce
            - Single-purpose methods
    - **High Performance:**
        - MongoDB performs **very fast read and write operations**.
        - Its features like indexing, replication, and sharding help in **handling large data efficiently**.

- Working mongodb:-
    - Mongodb works on the concept of collection and document.
    - Because of its nosql database,the data is stored in the collection and document.
        - Database
        - Collection
            - Document
                - Data

    - The mongodb database contains collections just like other databases create a table.
    - You can allow to create multiple database and collection.
    - Collection is a group of document.
    - Inside a document we store data in bson format.
    - Document have dynamic schema. Dynamic schema means that document in collection do not need to have the same set of structure.
    - Ex:-
        - {
                Fristname:"hjkl"
          }

3.basic operation of mongodb.

- Show database:-
    - Show dbs
- Use database:-
    - Use dbs
- Create collection:-
    - db.createCollection('name');
- Show collections:-
    - Show collections
- Insert one record:-
    - db.collection_name.insertOne({

      })
- Insert many data:-
    - db.student_info.insertMany([
      { SID: 2, sname: "Nita", sclass: "SYBCA" },

{ SID: 3, sname: "Ravi", sclass: "TYBCA" },
  { SID: 4, sname: "Neha", sclass: "FYBCA" }
])

- Find all document:-
    - db.student_info.find()
- Find with condition:-
    - db.student_info.find({ SID: { $gt: 3 } })
    -
- Limited result:-
    - db.collection_name.find().limit(2)
- Update record:-
    - db.student_info.updateOne(
    -   { SID: 3 },
    -   { $set: { sname: "Updated Name" } }
    -)
    -
- Update many:-
    - db.student_info.updateMany(
    -   { sclass: "FYBCA" },
    -   { $set: { sclass: "FYMCA" } }
    -)
    -
- Count the document:-
    - db.student_info.find({ sclass: "SYBCA" }).count()
    -
- Sort ascending and descending order:-
    - db.student_info.find().sort({ SID: -1 })
    -
    - db.student_info.find().sort({ sname: 1 })
    -
- Delete one and all row:-

- db.student_info.deleteOne({ SID: 2 })
-
- db.student_info.deleteMany({})
-

- Delete conditional row:-
  - db.student_info.deleteMany({ SID: { $gt: 5 } })
  -

5.connect mongodb using mongodb driver with step and example.

- A NoSQL database designed for handling large amounts of unstructured data.
- Step1:-install mongodb driver for node js
  - To mongodb in node js then you import the mongodb package.
    - Npm install mongodb
- Step2:-import mongoClient from mongodb package.
  - This main class interacts with the mongodb server.
    - const{MongoClient}=require('mongodb');
- Step 3:-define url
  - url="mongodb://localhost27017" or
  - url="mongodb://127.0.0.1:27017"
- Step 4:- create the mongodb client
  - Const client=new mongodbClient(url);
- Step 5:-connect and close the server
  - client.connect(); to connect server
  - client.close(); to close the server
- Ex:-

  async function run() {

```
  try {
    await client.connect();         // Connect to MongoDB
    console.log("Connected to MongoDB");

    // You can write database operations here

  } finally {
    await client.close();           // Close the connection
    console.log("Connection closed");
  }
}

run();
```

- To run node connect.js
- Connected to mongodb
- Connection is close

5.connect mongodb using mongoose library:-

- Step 1:-install mongoose
  - Npm install mongoose
- Step 2:import mongoose
  - const mongoose = require('mongoose');
  -
- Step 3:-define mongodb url
  - const uri = "mongodb://127.0.0.1:27017";
  -
- Step 4:- connect the mongo db

```
mongoose.connect(uri)
.then(() => {
  console.log("Connected to MongoDB using
Mongoose");
})
.catch((err) => {
        console.error("Connection failed", err);
      });
```

-
- Ex:-

```
// Import mongoose
const mongoose = require('mongoose');

// MongoDB URI
const uri = "mongodb://127.0.0.1:27017/studentDB";

// Connect to MongoDB
mongoose.connect(uri, {
  useNewUrlParser: true,
  useUnifiedTopology: true
})
.then(() => {
  console.log("Connected to MongoDB");
})
.catch((err) => {
  console.error("Connection error", err);
});

// Define Schema
const studentSchema = new mongoose.Schema({
  name: String,
  age: Number,
  class: String
});

// Create Model
const Student = mongoose.model("Student", studentSchema);

// Create a document
const newStudent = new Student({
  name: "Amit",
  age: 20,
  class: "SYBCA"
});
```

```
// Save to database
newStudent.save()
  .then(() => {
    console.log("Student saved successfully!");
    mongoose.connection.close(); // Close connection after operation
  })
  .catch((err) => {
    console.error("Error saving student", err);
  });
```

5.connect database using mysql database.

- Step-1 : Install MySQL Driver To install the "mysql2" module, open the Command Terminal and execute the following:
  - npm install mysql2

- Step- 2 : import the mysql2 module
  - const mysql = require('mysql2');
- Step-3 : Create a connection to the database

  -
    ```
    const connection = mysql.createConnection({
    host: 'localhost',
    user: 'root',
    password: 'your_password',
     database: 'your_database'
    });
    ```
- Step-4 : Connect to the MySQL database

  ```
  connection.connect((err) => {
   if (err) {
                 console.error('Error connecting to the
          database:', err); return;
      }
  console.log('Connected to MySQL database!');
  });
  ```

- Ex:-

```
var mysql = require('mysql2');
var con = mysql.createConnection({
                      host: "localhost",
                     user: "myusername",
                      password: "mypassword",
                     database: "mydb"
                     });
con.connect(function(err) {
 if (err) throw err;
 console.log("Connected!");
 /*Create a table named "customers":*/
var sql = "CREATE TABLE customers (name VARCHAR(255),
address VARCHAR(255))";

con.query(sql, function (err, result) {
 if (err) throw err;
console.log("Table created");
 });
});
```

7.what are cookies in node js.

- The cookies are the data stored in the user’s browser for quick access.
- Cookies are small data that are stored on a client side and sent to the client along with server requests.
- To use cookie in node js then install cookie-parser
  - Npm install cookie-parser
- Step 1:-import cookie-parser
  - Const cookie=require(‘cookie-parser’);

- Step 2:-use cookie-parser middleware
    - app.use(cookie());

- Syntax:-
    - res.cookie(name_of_cookie, value_of_cookie);
- Add cookies expiration time:-
    - After some time cookies will be destroyed automatically.
    - res.cookie(cookie_name, 'value', {expire: 400000 + Date.now()});
- Read the cookie.
    - res.send(req.cookies);
- Destroy cookie:-
    - res.clearCookie(cookieName);

- Ex:-

```
const express = require('express');
const cookieParser = require('cookie-parser');
const app = express();

// Use cookie-parser middleware
app.use(cookieParser());

// Set a cookie
app.get('/setcookie', (req, res) => {
  res.cookie('username', 'Amit');
  res.send('Cookie has been set');
});

// Get a cookie
app.get('/getcookie', (req, res) => {
  let username = req.cookies.username;
  if (username) {
    res.send(`Welcome back, ${username}`);
  } else {
    res.send('No cookie found');
  }
});

// Start server
app.listen(3000, () => {
  console.log('Server running on
http://localhost:3000');
});
```

7.what is body parsing in express.

- body-parser is an npm module used to process data sent in an HTTP request body.
- Using body-parser allows you to access req.body from within routes and use that data.
- This module provides the following parsers:
    -

Parser Type	Description
`json()`	Parses **JSON data**. Use when client sends data as JSON.
`urlencoded()`	Parses **form data.** Use for HTML form submissions.
`text()`	Parses raw **text**. Useful for plain text input.
`raw()`	Parses **raw binary** data (Buffer). Useful for file uploads.

- Ex:- to see

8.what is user authentication?

- Authentication is the process of verifying the identity of a user or system.
- In Express, authentication can be done using various methods like
    - Basic authentication
        - Users provide their username and password for each request
        - Simple to implement.
    - token-based (JWT) authentication:-
        - User receiving json web token after logging
        - Client stores the token and sends it in the `Authorization` header with each request
    - session-based authentication:-
        - To store a user session on the server after login.
    - OAuth:-
        - Lets users log in via third-party services
        - Redirects users to third-party login.

        - On success, a third-party sends an access token.

        - Server uses a token to access user info.
- Ex:- see after
-