Databases

Types of Databases

Simple: SQL vs NoSQL, how to create Postgres Databases, How to do Crud on them.

Advance: Relationships, joins, Transaction

There are a few types of databases, all servicing different types of use-cases.

NoSQL Databases

1. Store data in a schema-less fashion. Extremely lean and fast way to store data.

Example - MongoDB

2 Graph Databases

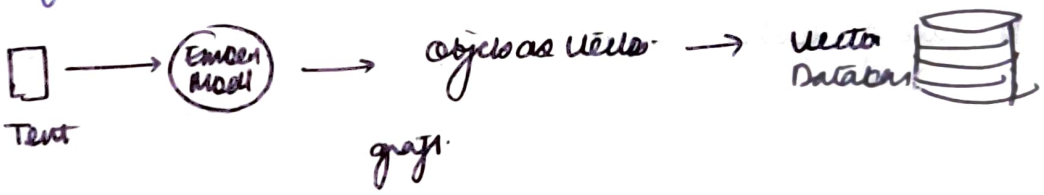
Data is stored in the form of graph. Specially useful in cases where relationship need to be stored (social Networks)

Example: Neo4j

3 Vector data

Stores in form of vectors, useful in machine learning.

Example: pinecone.



4. SQL databases

stores data in rows and Columns, most full stack application will use this

Example. MySQL, postgres.

Step 2 Why not Not SQL

- It is good for bootstrapping the project.
- There is no schema to be followed

Problems

→ Can lead to inconsistent databases

→ Can cause runtime errors.

→

Upsides

1. Can move very fast
2. Can change schema very easily.

Step-3 why SQL

SQL databases have a strict schema

1. Define your schema
2. put in data that follows that schema
3. updates the schema as your app changes and perform migrations.

4 parts to run the SQL DB.

1. start the database
2. using a library that lets you connect and put data in it.
3. Creating a table and defining its schema.
4. Run queries on the database to interact with the data (insert/update/Delete)

step-4 Creating a database

1. **using neondb** it is a decent service that lets you create a server.

2. **using docker locally**: docker run --name my-postgres -e POSTGRES_password=mysecretpassword -d -p 5432:5432 postgres.

3. using docker on windows

postgresql://username:password@host/database

---

step 5: using a library that lets you connect and put data in it.

1. **psql**: psql is a terminal-based front-end to postgresql.
provides an interactive command-line interface to the PostgreSQL.
with psql we can type in queries interactively.

How to connect to your database?

psql comes bundled with postgresql. you don't need it for this tutorial, we will directly communicating with the database from nodejs.

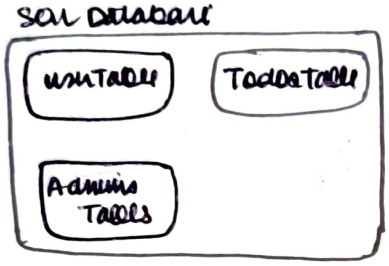
2. Pg

Pg is a Node.js library that you can use in your backend app to store data in the postgres DB (similar to mongoose). We will be installing this eventually in our app.

Creating a table and defining it's schema.

Tables in SQL

A single database can have multiple tables inside. Think of them as collections in a mongoDB database.



SQL: SQL stands for structured query language. It is a language in which you can describe what / how you want to put data in the database.

```
Create table users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(255), UNIQUE NOT NULL,
    password VARCHAR(255), NOT NULL,
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
);
```

\dt : to see if table has been created or not.

Querying in SQL

1. Insert

```
insert into users ( username, email, password ) values ('username_here',
'user@examples.com', 'user_password');
```

2. UPDATE

```
update users
SET password = 'new_password'
WHERE email = 'user@example.com';
```

3. DELETE

```
DELETE FROM USERS
WHERE id=1;
```

1. SELECT

```
SELECT * FROM USERS WHERE id=1;
```

Relationship and Transaction (MONGODB)

User can store any shape of data in it.

```
address : object
{
    street: "1 west NW"
    city : "Chandigarh"
    country : "India"
}
```

Relationship in SQL

users

id	name	email	password
1	rahul	alg...a	—
2	raman	—	—

Address.

id	user	city	country	pin

Foreign key.

This is called relation where address table is related to the users table.

Transaction in SQL

Transaction ensure both user info and address goes in or neither data, if user send both the information simultaneously.

Joins

Defining relationship is easy.
What is hard is joining data from two (or more) tables together.

Example

```
select user.id, user.name, user.email, address from users.
JOIN address ON users.id = address.user_id
WHERE users.id = 1
```

```
SELECT u.id, u.username, u.email, a.city a.country
from users u
JOIN address a ON u.id = a.user_id
WHERE uid = YOUR_USER_ID;
```

BENEFITS OF using a join

1. Reduced latency
2. simplified Application logic
3. Transactional integrity

Types of joins

1. **INNER JOINS** (Return rows when there is at least one match in both tables)

```
SELECT users.username, address.city, address.country From users INNER JOIN addresses
                                                    ON usr.id = address.usr_id,
```

2. **LEFT JOINS** (Return rows all left side & matching rows in the right side).

```
select users.username, address, city, address.county from users
LEFT JOIN addresses ON user id = address.user id
```

3. **RIGHT JOINS** [Return all row from right table and match row from left table).

```
select users.username, address.city ——————— from users
RIGHT JOIN address ON user.id = address.user.id.
```

4. **FOUL JOINS** Return rows when there is a match in one of the tables

It effectively combines the result of both LEFT and RIGHT joins

```
select users.username, address.city, address.country —— from users.
Full JOIN addresses ON usr.id = address.usr_id.
```

postgres and sql Databases

MongoDB is schemaless

Mongoose is having schema (at nodejs level)

CRUD Application

→ Create

→ Read

→ Update

→ Delete

```
const insertQuery = 'insert into users (username,
email, password) values ($1, $2, $3);

const response = await pgClient.query(insertQuery,
[username, email, password]);
```

{ master - slave Architecture in Database }

Foreign key Constraint

```
Create table users(
    id SERIAL PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL
    email VARCHAR(255) UNIQUE NOT NULL
    password VARCHAR(255) NOT NULL
    created_at TIMESTAMP WITH THE ZONE DEFAULT CURRENT_TIMESTAMP.
);

Create table addresses (
    id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL
    city VARCHAR(100) NOT NULL
    country VARCHAR (100) NOT NULL
    street VARCHAR (255) NOT NULL
    pincode VARCHAR (20)
    created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE
```

for running multiple Queries in database.

We use.

await pg.client.query("BEGIN;"). ← In the beginig of the Query.

(put Queries here) (that needs to be executed simultaneously)

await pg.client.query("COMMIT;") ← at the end of the Queries