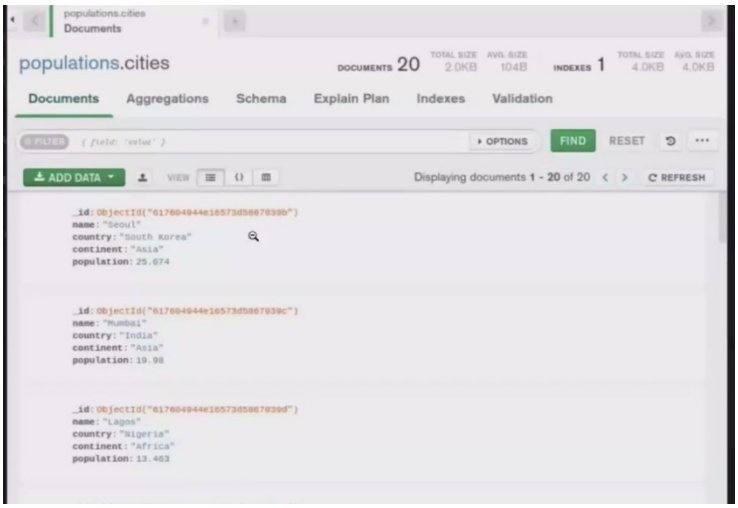
To identify who is using mongodb, firebase



Go to api calls and see the id field if the id field contains __id: after this there is a object id then

See the eg

Mongo db has horizontal scaling

POSTGRESS WE HAVE TO REALLY GOOD AT

MONGO DB is a no sql databases , they are schemaless (we can change the fields in one data)

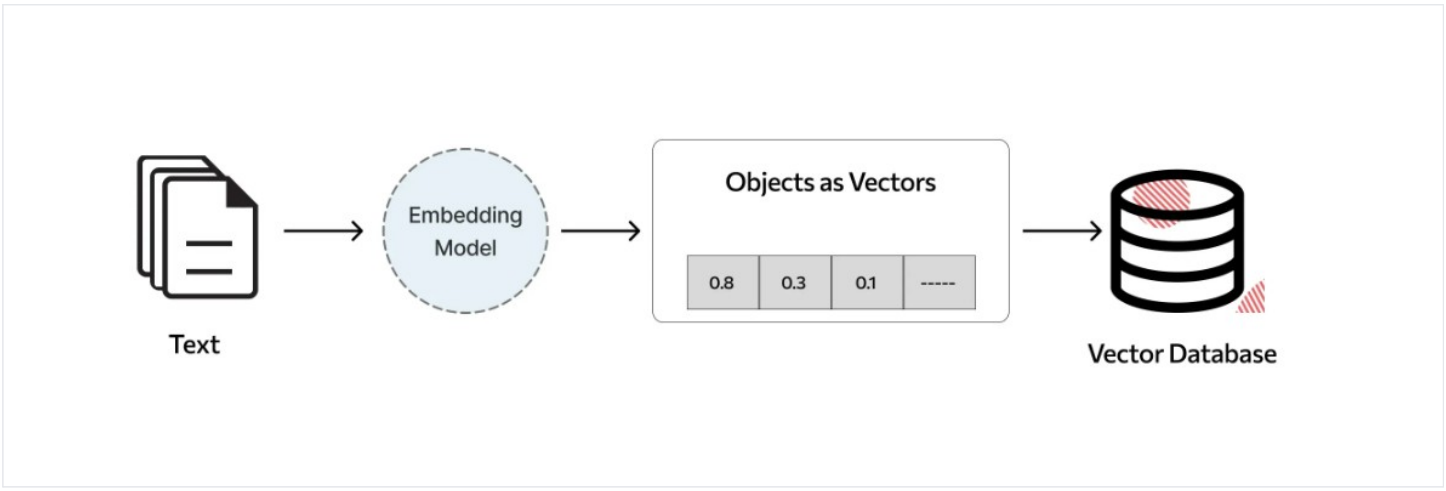
But in node js we have mongoose which prevents adding other data it defines a proper schema but then also we can edit from mongodb BUT HAVIND SCHEMA/STRUCTURE IS GOOD AND HAVING TYPES IS ALSO GOOD

Like in facebook a people can have like 50 friends so how the database stores that for that we use

Graph Databases: eg **Neo4j** which helps us to store the data in the form of graph

Vector databases

1. Stores data in the form of vectors
2. Useful in Machine learning
3. Examples – Pinecone
4. Stores data in the form of embeddings like converting the texts into embeddings only



SQL databases

1. Stores data in the form of rows
2. Most full stack applications will use this
3. Examples - MySQL, Postgres, Promql

Table: customers

customer_id	first_name	last_name	phone	country
1	John	Doe	817-646-8833	USA
2	Robert	Luna	412-862-0502	USA
3	David	Robinson	208-340-7906	UK
4	John	Reinhardt	307-242-6285	UK
5	Betty	Taylor	806-749-2958	UAE

Why not NoSQL

You might've used `MongoDB`

It's `schemaless` properties make it ideal to for bootstraping a project fast.

But as your app grows, this property makes it very easy for data to get `curropted`

What is schemaless?

Different rows can have different `schema` (keys/types)

Anyone can change the fields as there is no schema

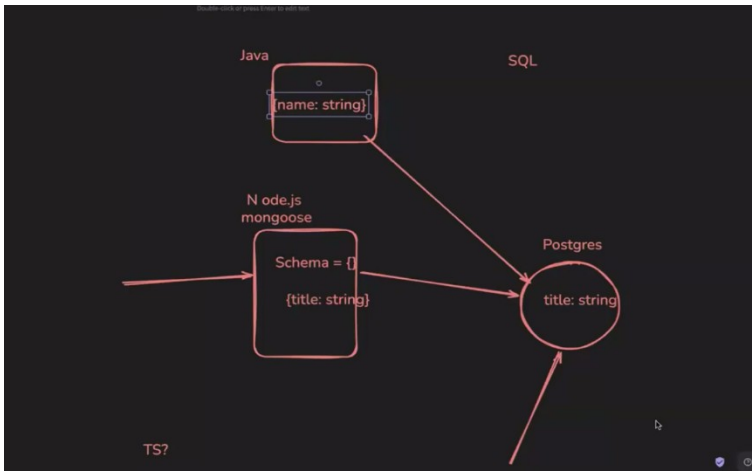
Problems?

1. Can lead to inconsistent database
2. Can cause runtime errors
3. Is too flexible for an app that needs strictness

Upsides?

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

You might think that `mongoose` does add strictness to the codebase because we used to define a schema there. That strictness is present at the Node.js level, not at the DB level. You can still put in erroneous data in the database that doesn't follow that schema.



Main difference between the mongo and postgress is this that eg in java

Why SQL?

SQL databases have a strict schema. They require you to

1. Define your schema (necessary condition)
2. Put in data that follows that schema
3. Update the schema as your app changes and perform `migrations (WE NEED TO UPDATE THE DATA AS THE DATA INCREASES)`

So there are 4 parts when using an SQL database (not connecting it to Node.js, just running it and putting data in it)

1. Running the database.
2. Using a library that let's you connect and put data in it.
3. Creating a table and defining it's `schema`.
4. Run queries on the database to interact with the data (Insert/Update/Delete)

To make changes in the sql schema we have to do **DB MIGRATION** which will like help me to create new field and data

To setup a postgress db we can use like neon.tech or docker

On neon.tech you can copy the connetion string it same like mongodb

Pdadmin is like a mongodb compass in mongodb

Step 5 - Using a library that let's you connect and put data in it.

1. psql

`psql` is a terminal-based front-end to PostgreSQL. It provides an interactive command-line interface to the PostgreSQL (or TimescaleDB) database. With psql, you can type in queries interactively, issue them to PostgreSQL, and see the query results.

How to connect to your database?

psql Comes bundled with postgresql. You don't need it for this tutorial. We will directly be communicating with the database from Node.js

```
psql -h p-broken-frost-69135494.us-east-2.aws.neon.tech -d database1 -U
```

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.

IT IS SAME AS SQL AND MYSQL

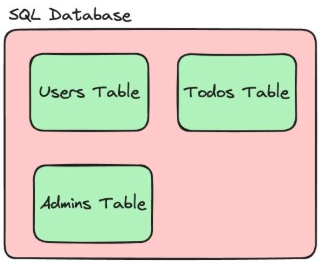
Some commands:

\dt; , SQL Editor inside the neon tech

STEP 6 Creating a table and defining a schema

Tables in SQL

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



Until now, we have a database that we can interact with. The next step in case of postgres is to define the `schema` of your tables.

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.

To create a table, the command to run is –

```
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
);
```

There are a few parts of this SQL statement, let’s decode them one by one

1. CREATE TABLE users

`CREATE TABLE users`: This command initiates the creation of a new table in the database named `users`.

2. id SERIAL PRIMARY KEY

- `id`: The name of the first column in the `users` table, typically used as a unique identifier for each row (user). Similar to `_id` in mongodb
- `SERIAL`: A PostgreSQL-specific data type for creating an auto-incrementing integer. Every time a new row is inserted, this value automatically increments, ensuring each user has a unique `id`.
- `PRIMARY KEY`: This constraint specifies that the `id` column is the primary key for the table, meaning it uniquely identifies each row. Values in this column must be unique and not null.

3. email VARCHAR(255) UNIQUE NOT NULL,

- `email`: The name of the second column, intended to store the user's username.
- `VARCHAR(50)`: A variable character string data type that can store up to 50 characters. It's used here to limit the length of the username.
- `UNIQUE`: This constraint ensures that all values in the `username` column are unique across the table. No two users can have the same username.
- `NOT NULL`: This constraint prevents null values from being inserted into the `username` column. Every row must have a username value.

4. password VARCHAR(255) NOT NUL

Same as above, can be non uniqye

5. created_at TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP

- `created_at`: The name of the fifth column, intended to store the timestamp when the user was created.
- `TIMESTAMP WITH TIME ZONE`: This data type stores both a timestamp and a time zone, allowing for the precise tracking of when an event occurred, regardless of the user's or server's time zone.
- `DEFAULT CURRENT_TIMESTAMP`: This default value automatically sets the `created_at` column to the date and time at which the row is inserted into the table, using the current timestamp of the database server.

If you have access to a database right now, try running this command to create a simple table in there 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); Then try running \dt; to see if the table has been created or not

CRUD Application = Create, Read, Update,Delete

STEP – 7 Interacting with the database

There are 4 things you'd like to do with a database

1. INSERT

INSERT INTO users (username, email, password)

VALUES ('username_here', 'user@example.com', 'user_password');

Notice how you didn't have to specify the id because it auto increments

2. UPDATE

UPDATE users

SET password = 'new_password'

WHERE email = 'user@example.com';

3. DELETE

DELETE FROM users

WHERE id = 1;

4. Select

SELECT * FROM users

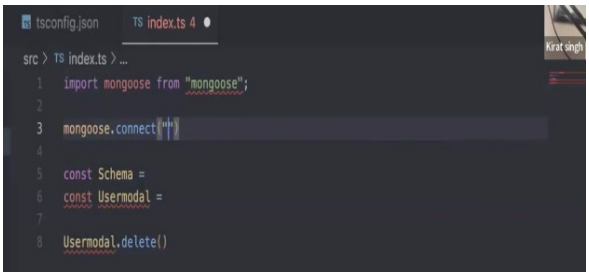
WHERE id = 1;

Try running all 4 of these in your terminal if you have psql installed locally.

If not, that's fine we'll eventually be doing these through the pg library.

STEP – 8 How to do queries from a Node.js app?

Firstly setup the tsc file and then install npm install pg @types/pg



This is how we used the mongoose

Two ways of connecting to db of postgress:

```
const pgClient = new Client({
    user: "neondb_owner",
    password: "wrWG5KI1ziYB",
    port: 5432,
    host: "ep-lucky-snow-a50il0b5.us-east-2.aws.neon.tech",
    database: "neondb"
})
```

```
const pgClient = new Client("postgresql://
neondb_owner:npg_Pdt9EbN3mWuH@ep-frosty-grass-ai9krbOj-pooler.c-4.us-east-1.
neon.tech/neondb?sslmode=require&channel_binding=require")
```

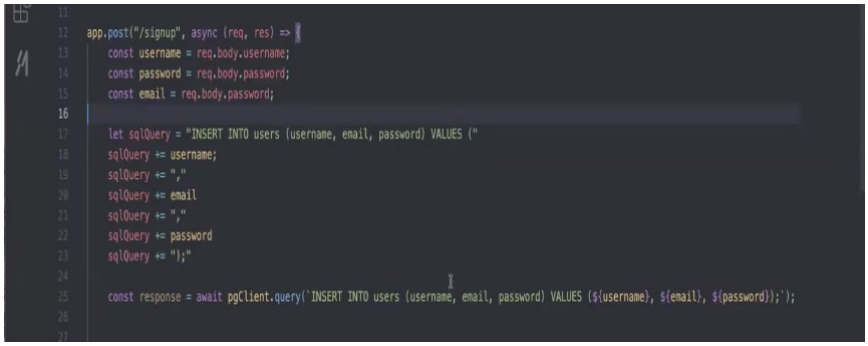
Updating the data from the ts file and accessing the files and data in sql database

```
import { Client } from "pg";

const pgClient = new Client("postgresql://
neondb_owner:npg_Pdt9EbN3mWuH@ep-frosty-grass-ai9krbOj-pooler.c-4.us-east-1.a
neon.tech/neondb?sslmode=require&channel_binding=require")

async function main(){
    await pgClient.connect();
    const response = await pgClient.query("SELECT * FROM users;")
    console.log(response.rows);
}
```

Updated the username

```
async function main(){
    await pgClient.connect();
    const response = await pgClient.query("UPDATE users SET username='harkirat'
    where id=10;")
    console.log(response.rows);
}
main()
```

Two meathods to make a sql query



```
import { Client } from "pg";
import express from "express";

const app = express();
app.use(express.json());

const pgClient = new Client("postgresql://
neondb_owner:npg_Pdt9EbN3mWuH@ep-frosty-grass-ai9krb0j-pooler.c-4.us-east-1.aws.
neon.tech/neondb?sslmode=require&channel_binding=require")

pgClient.connect();

app.post("/signup",async (req ,res) =>{
    const username = req.body.username;
    const password = req.body.password;
    const email = req.body.email;

    const insertQuery = `INSERT INTO users(username,email,password) VALUES ('$
    {username}', '${email}', '${password}');`

    const response = await pgClient.query(insertQuery);
    res.json({
        message:"You have signed up"
    })
})

app.listen(3000)

```

in this there are some problem like when we send a same id pass then the backend crashes so we have to fix that

put it inside a try catch to fix the crashing

```
{
    "username": "asdasdaddsa",
    "email": "asdasd@gmail.com",
    "password": "123123'); DELETE * FROM users; INSERT INTO users (username, password,
        email) VALUES ('123123', '123123', '123123"
}
```

we can do **sql injection**

it means like someone try to inject there own sql query into the code using the password or username to exploit the data and make changes to the db (DELETE FROM users)

To fix the SQL INJECTION

```
try{
    const insertQuery = `INSERT INTO users(username,email,password) VALUES
    ($1,$2,$3);`
    const response = await pgClient.query(insertQuery,[username, email,
    password]);
    res.json({
        message:"You have signed up"
    })
}
```

in this the values are inserted as text not as a command like before the data was inserted directly and was running like sql commands but in this if they send any command it will act as a whole text

SQL DATABASES SCALES VERTICALLY LIKE YOU CAN UPGRADE THE SYSTEM YOU CANNOT ADD MORE SYSTEMS UNLESS IF YOU WANT TO EXPAND HORIZONTALLY YOU NEED TO USE SHARDS

Step 10 - Relationships and Transactions

Relationships let you store data in different tables and relate it with each other.

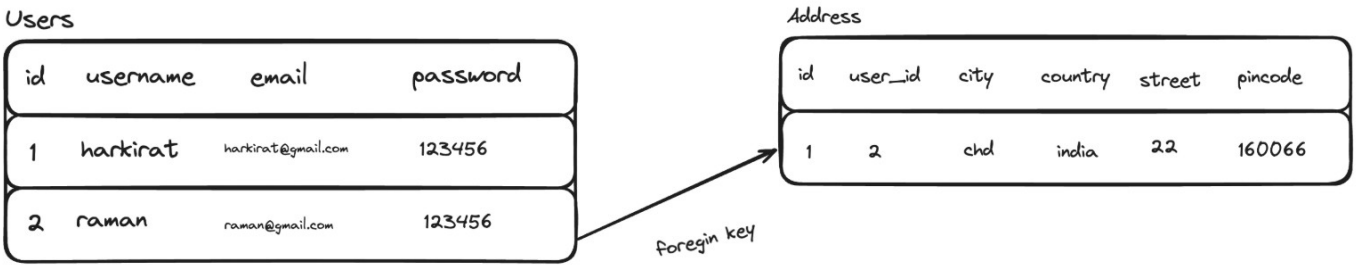
Relationships in Mongodb

Since mongodb is a NoSQL database, you can store any shape of data in it.

If I ask you to store a users details along with their address, you can store it in an object that has the address details.

Relationships in SQL

Since SQL cannot store objects as such, we need to define two different tables to store this data in.



This is called a relationship, which means that the Address table is related to the Users table.

When defining the table, you need to define the relationship

```
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
);
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
);
```

CASCADE IS LIKE IF THE KEY GETS DELETED IN THE PARENT TABLE THEN THE CHILDREN TABLE DATA IS ALSO GET DELETED

ON DELTE CASCADE MEANS IN THIS IF ANY VALUE DELETED IN THE PARENT TABLE IT SHOULD NOT HAVE ANY CONNECTED TERM IN THE CHILD TABLE

EG LIKE THIS IS AMAZON S3 BUCKET

SQL query

To insert the address of a user -

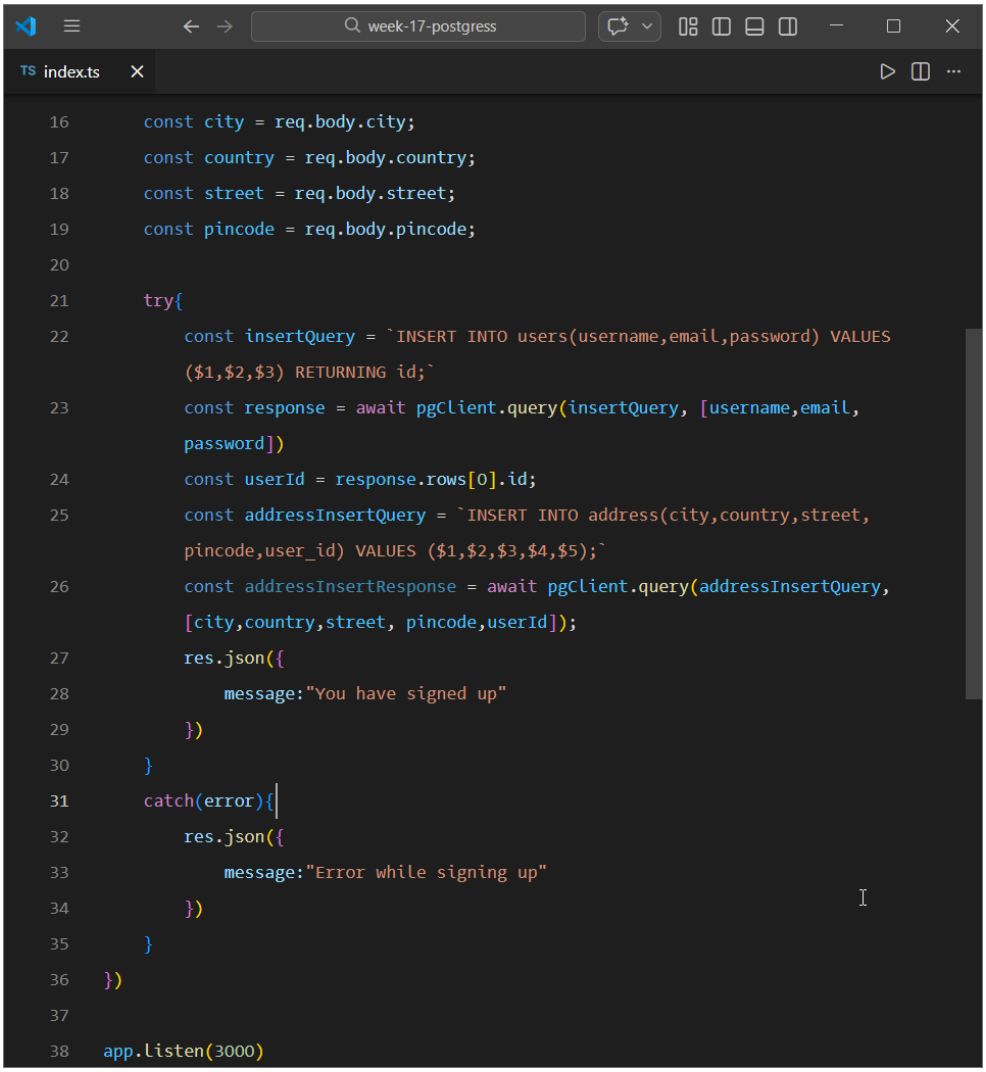
```
INSERT INTO addresses (user_id, city, country, street, pincode)
VALUES (1, 'New York', 'USA', '123 Broadway St', '10001');
```

Now if you want to get the address of a user given an id , you can run the following query -

SELECT city, country, street, pincode

FROM addresses

WHERE user_id = 1;



now in this we can add the data from the node js file only but the issue is if somehow our db breaks while we were making the address table then only partial table exist for the user so

this is fixed using **transactions**

if you want to run multiple queries one after other then put them in a transaction

its like someone is sending money and someone sends the money the other person should for sure receive that or should both fails this is done using transaction

Extra - Transactions in SQL

Good question to have at this point is what queries are run when the user signs up and sends both their information and their address in a single request.

Do we send two SQL queries into the database? What if one of the queries (address query for example) fails?

This would require transactions in SQL to ensure either both the user information and address goes in, or neither does

BEGIN; -- Start transaction

INSERT INTO users (username, email, password)

VALUES ('john_doe', 'john_doe1@example.com', 'securepassword123');

INSERT INTO addresses (user_id, city, country, street, pincode)

VALUES (currval('users_id_seq'), 'New York', 'USA', '123 Broadway St', '10001');

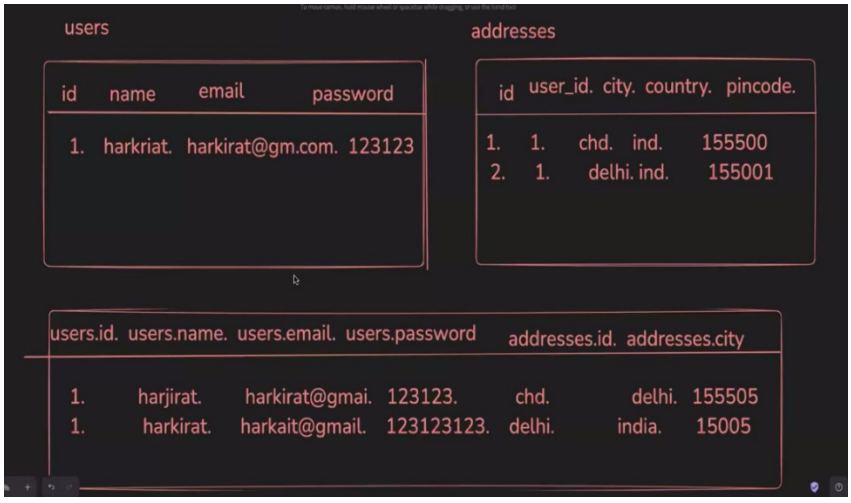
COMMIT; (**if the first query rans and other does not it ensures the code will not work either both of them will run or none of them will not work**)

JOINS

Defining relationships is easy.

What’s hard is joining data from two (or more) tables together.

For example, if I ask you to fetch me a users details and their address, what SQL would you run?



Litteraly joining two tables based on one same id

Approach 2 (using joins)

SELECT users.id, users.username, users.email, addresses.city, addresses.country, addresses.street, addresses.pincode

FROM users JOIN addresses ON users.id = addresses.user_id

WHERE users.id = '12';

Benefits of using a join -

Reduced Latency

Simplified Application Logic

Transactional Integrity

Types of Joins

1. INNER JOIN

Returns rows when there is at least one match in both tables. If there is no match, the rows are not returned. It's the most common type of join.

Use Case: Find All Users With Their Addresses. If a user hasn’t filled their address, that user shouldn’t be returned

SELECT users.username, addresses.city, addresses.country, addresses.street, addresses.pincode

FROM users

INNER JOIN addresses ON users.id = addresses.user_id;

2. LEFT JOIN

Returns all rows from the left table, and the matched rows from the right table.

Use case - To list all users from your database along with their address information (if they've provided it), you'd use a LEFT JOIN. Users without an address will still appear in your query result, but the address fields will be NULL for them.

SELECT users.username, addresses.city, addresses.country, addresses.street, addresses.pincode

FROM users

LEFT JOIN addresses ON users.id = addresses.user_id;

3. RIGHT JOIN

Returns all rows from the right table, and the matched rows from the left table.

Use case - Given the structure of the database, a RIGHT JOIN would be less common since the addresses table is unlikely to have entries not linked to a user due to the foreign key constraint. However, if you had a situation where you start with the addresses table and optionally include user information, this would be the theoretical use case.

SELECT users.username, addresses.city, addresses.country, addresses.street, addresses.pincode

FROM users

RIGHT JOIN addresses ON users.id = addresses.user_id;

4. FULL JOIN

Returns rows when there is a match in one of the tables. It effectively combines the results of both LEFT JOIN and RIGHT JOIN.

Use case - A FULL JOIN would combine all records from both users and addresses, showing the relationship where it exists. Given the constraints, this might not be as relevant because every address should be linked to a user, but if there were somehow orphaned records on either side, this query would reveal them.

SELECT users.username, addresses.city, addresses.country, addresses.street, addresses.pincode

FROM users

FULL JOIN addresses ON users.id = addresses.user_id;

MY LEARNINGS

So imagine you have to get all the data of a user like in /metadata?id=12 (this id is a query parameter)

```
app.get("/metadata", async (req, res) => {
    const id = req.query.id;
    const query1 = `SELECT username,email,id FROM users WHERE id=$1`;
    const response1 = await pgClient.query(query1, [id]);
    const query2 = `SELECT * FROM addresses WHERE user_id=$1`;
    const response2 = await pgClient.query(query2, [id]);
    res.json({
        user: response1.rows[0],
        address: response2.rows
    });
});

app.get("/better-metadata",async (req,res)=>{
    const id = req.query.id;
    const query = `SELECT users.id, users.username, users.email, addresses.
    city, addresses.country, addresses.street, addresses.pincode
    FROM users JOIN addresses ON users.id = addresses.user_id
    WHERE users.id = $1;`
    const response = await pgClient.query(query,[id]);
    res.json({
        response : response.rows
    })
})
```

Upper one uses like two queries and below one uses joins and see reduces code a lot

BUT THE PROBLEM IS THE JOINS ARE VERY EXPENSIVE AS THEY ARE DOING LIKE N X M MATRIX MULTIPLICATION

INCREASES THE COMPUTATION POWER SO THAT'S THE ISSUE

TYPES OF JOINS

Now there is a issue suppose the users table has a user with no address then if we try to run the better-metadata it will not return anything and if we try to run without using joins it will work

1. Inner Join: in this if we want to access a data from a 2 tables and they are connected using a foreign key and like for the userid like 13 if the data is present in the user table and not in the address table then the inner join will not return any data it will return empty array

2. Left Join: In this if the user table is in left and the address table is in right then if the data exist in the user table it will show that and also the address data columns with filled null in it

3. Right Join: Imagine if the address table contain some data and the user table does not then also it will return the data but with null values for the users columns

4. FULL JOIN : This is the best one mixture of both the joins Left join and the right join like if sometime the right side table is empty it will show the left part with empty values for right table and vice versa and if both table exist then also it will print the data

FOR CODE JUST CHANGE THE JOIN WITH LEFT JOIN, RIGHT JOIN, FULL JOIN ETC

CHROME USES V8 ENGINE

FIREFOX USES SPIDERMONKEY