SQL in Practice

Overview

What We'll Cover

- JOIN
- Aliases
- DROP
- Connecting Databases to Applications

Objectives

1. Student understands where and how databases fit into full stack applications.
2. Student can execute database queries from an application using an ORM.
3. Student can analyze real world problem and define requirements for a SQL report that would solve that problem.

Intro

Let's get started with SQL syntax for more advanced queries.

Then we'll jump into the what, where, when, and how of using databases.

JOIN

Once table relationships have been created, `JOIN` statements can be used to collect data from tables that relate to each other.

Example: JOIN

Given this data, how will we select data from both tables?

Dept_Code	Department
legal	Legal
mktg	Marketing
fin	Finance

Id	Name	Dept_Code
1	Liz	legal
2	Maggie	mktg
3	Leonard	legal
4	Nadine	NULL

Syntax: JOIN

Basic syntax: `SELECT ... FROM t1 JOIN t2 ON ...`

Query for the employees and departments:

```
-- for repeating column names, we need to specify which table each column comes from using a dot between the table name and the
column name
SELECT name, employees.dept_code, departments.dept_code, dept
FROM employees
  JOIN departments
    ON employees.dept_code = departments.dept_code;
```

Result:

Name	e.Dept_Code	d.Dept_Code	Department
Liz	legal	legal	Legal
Maggie	mktg	mktg	Marketing
Leonard	legal	legal	Legal

Note: Inner Join

The type of join we just used is called an inner join. (There are many others!) Inner joins require that the field being joined on has a value in both tables, it can’t be NULL in either place.

Aliases

An alias is like a nickname that we assign to columns or tables. It makes writing longer queries more convenient.

Example: Aliases

Given the following two tables, let's look at how could we get the titles and authors in one query using a `JOIN` and ***aliases***.

author_id	author
1	Hawkins
2	Gold
3	Williams
4	Anderson

book_id	title	pages	author_id
1	Big Book	2300	2
2	Book	400	4
3	Tiny Book	8	1
4	The Book	1500	3

Syntax: Aliases

We have two options when creating an alias:

- use the keyword `AS`

```
SELECT fname AS first_name, city...
```

- simply put a space between the real name and the alias

```
SELECT fname first_name, city...
```

Examples of how to get the titles and authors:

```
SELECT a.author AS author_last_name, b.title
FROM authors AS a
  JOIN books AS b
    ON a.author_id = b.author_id;
```

```
SELECT a.author author_last_name, b.title
FROM authors a
  JOIN books b
    ON a.author_id = b.author_id;
```

The result would look like this (we don't get the `author_id` in this case because we didn't select either of them at the beginning of the query):

author_last_name	title
Gold	Big Book
Anderson	Book
Hawkins	Tiny Book
Williams	The Book

Note: Use Before Defining

Aliases can be used before they're defined. We can see this in the select above. We reference `b.title` before giving books the alias `b` later in the query.

DROP

Drop statements are for deleting, or 'dropping', database tables.

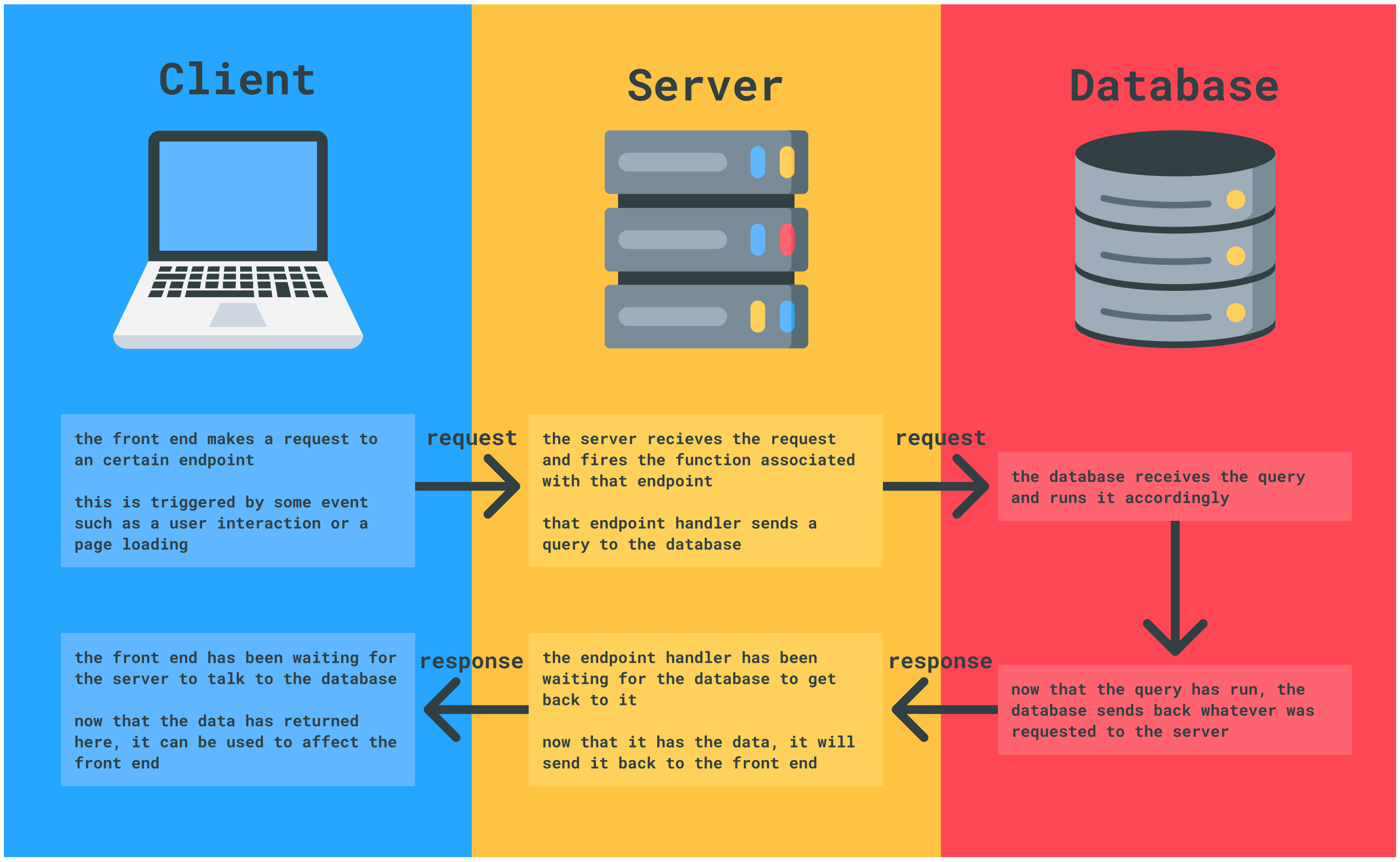
Be careful: Dropping a table is irreversible.

Syntax: DROP

```
DROP TABLE ratings;
```

Connecting Databases to Applications

Where do databases fit in to a fullstack app?


Client
Server
Database
the front end makes a request to an certain endpoint
this is triggered by some event such as a user interaction or a page loading
request
the server recieves the request and fires the function associated with that endpoint
that endpoint handler sends a query to the database
request
the database receives the query and runs it accordingly
now that the query has run, the database sends back whatever was requested to the server
response
the endpoint handler has been waiting for the database to get back to it
now that it has the data, it will send it back to the front end
response
the front end has been waiting for the server to talk to the database
now that the data has returned here, it can be used to affect the front end

Overview

- `dotenv`: a package that allows you to create local environment variables to store sensitive information (like a database URI)
- `sequelize`: an object relational mapping tool that allows us to connect to and interact with databases through our server
- these tools make it super simple to execute queries from our server

Dotenv

- in your project, install using `npm install dotenv`
- create a `.env` file at the root of your project
- **important**: include `.env` in your `.gitignore` file every time so that your private info doesn't get pushed to GitHub

```
SERVER_PORT=4444
CONNECTION_STRING=postgres://vfndos:a48g0408hg
```

- environment variable names should be capitalized
- do not put quotes around strings
- do not put semicolons at the end of lines

Note: Naming Environment Variables

It's typical to name your port `SERVER_PORT`, but you can also just use `PORT`.

`CONNECTION_STRING` is where we store the `URI` of our databases.

- in node files (like our server) we can access environment variables by bringing dotenv in and configuring it like so:

```
require('dotenv').config()
```

- once we require and configure `dotenv`, we can access our variables anywhere in that file using `process.env.VAR_NAME`:

```
app.listen(process.env.SERVER_PORT, () => console.log(`server running on port ${process.env.SERVER_PORT}`))
```

Sequelize

- in your project, install using `npm install sequelize`
- since we're using PostgreSQL, we'll also need to `npm install pg pg-hstore` so that Sequelize can interact with our database properly

Note: Installing Multiple Dependencies

You can install multiple dependencies in the same command, just put a space between the names.

`pg` and `pg-hstore` are separate dependencies in the example above.

```
const Sequelize = require('sequelize')
```

- in your server, require `sequelize`
- the "S" is capitalized because the default export is a class
- then create an instance like this:

```
const sequelize = new Sequelize(process.env.CONNECTION_STRING, {
    dialect: 'postgres',
    dialectOptions: {
        ssl: {
            rejectUnauthorized: false
        }
    }
})
```

- now we can execute SQL in our endpoint handler functions using Sequelize's `query` method

```
app.get('/people', (req, res) => {
  sequelize.query('select * from people;')
    .then(dbRes => res.send(dbRes))
})
```

Fullstack

This is super exciting! Now you can write fullstack apps!

The End