Introduction to HTML, Javascript, PostgreSQL and PostGIS

Presented By
Siddhartha Bhuyan
siddhartha.bhuyan1@nesac.gov.in
Contact No.: 03642308734
Mobile: 9085995599

HTML

What is HTML?

- Stands for Hyper Text Markup Language
- Standard markup language for creating Web pages
- Describes the structure of a Web page
- Consists of a series of elements
- Elements tell the browser how to display the content
- Elements label pieces of content such as "this is a heading", "this is a paragraph", "this is a link", etc.

History of HTML

1989	Tim Berners-Lee invented www
1991	Tim Berners-Lee invented HTML
1993	Dave Raggett drafted HTML+
1995	HTML Working Group defined HTML 2.0
1997	W3C Recommendation: HTML 3.2
1999	W3C Recommendation: HTML 4.01
2000	W3C Recommendation: XHTML 1.0
2008	WHATWG HTML5 First Public Draft
2012	WHATWG HTML5 Living Standard
2014	W3C Recommendation: HTML5
2016	W3C Candidate Recommendation: HTML 5.1
2017	W3C Recommendation: HTML5.1 2nd Edition
2017	W3C Recommendation: HTML5.2

A Simple HTML Document

```
<!DOCTYPE html>
<html>
	<head>
		<title>Page Title</title>
	</head>
	<body>

		<h1>My First Heading</h1>
		<p>My first paragraph.</p>

	</body>
</html>
```

Example Explained

- The <!DOCTYPE html> declaration defines that this document is an HTML5 document
- The <html> element is the root element of an HTML page
- The <head> element contains meta information about the HTML page
- The <title> element specifies a title for the HTML page (which is shown in the browser's title bar or in the page's tab)
- The <body> element defines the document's body, and is a container for all the visible contents, such as headings, paragraphs, images, hyperlinks, tables, lists, etc.
- The <h1> element defines a large heading
- The <p> element defines a paragraph

The HTML Element

An HTML element is defined by a start tag, some content, and an end tag:

<tagname> Content goes here... </tagname>

The HTML element is everything from the start tag to the end tag:

<h1>My First Heading</h1>

My first paragraph.

Nesting Elements

Elements can be placed within other elements. This is called *nesting*. If we wanted to state that a cat is **very** grumpy, we could wrap the word *very* in a element, which means that the word is to have strong(er) text formatting:

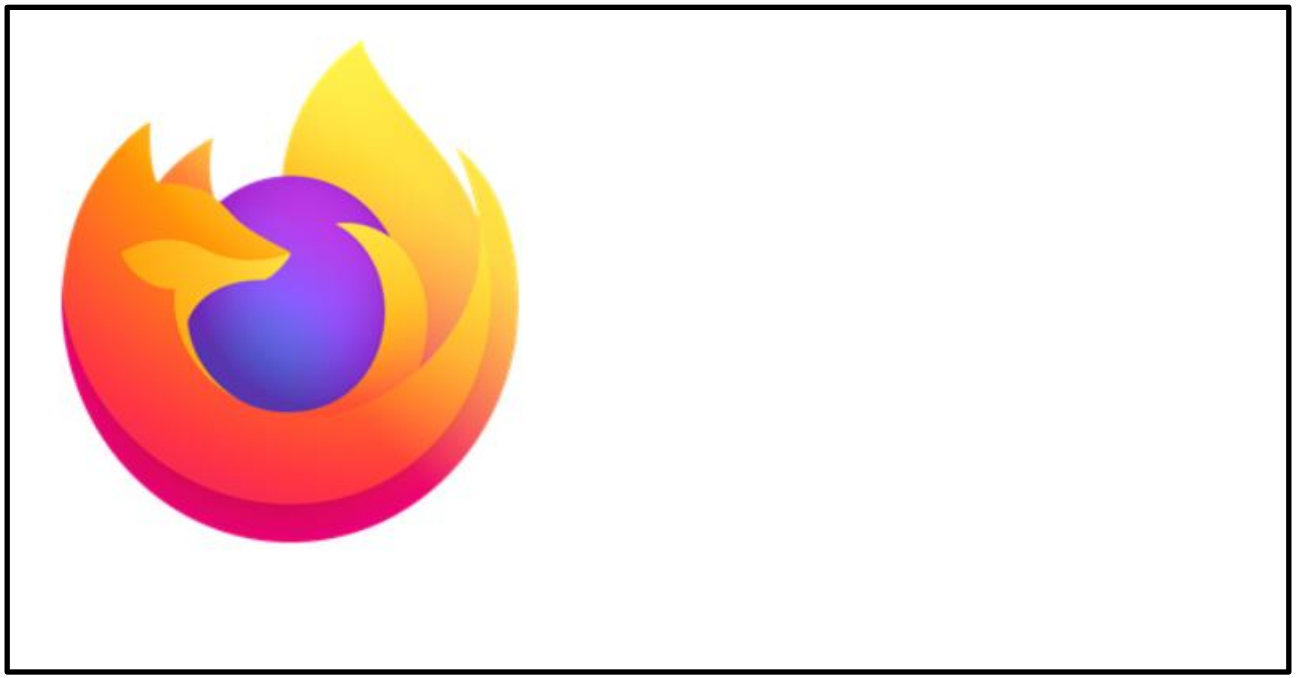
My cat is very grumpy.

*IMP: The tags have to open and close in a way that they are inside or outside one another.

Void Elements

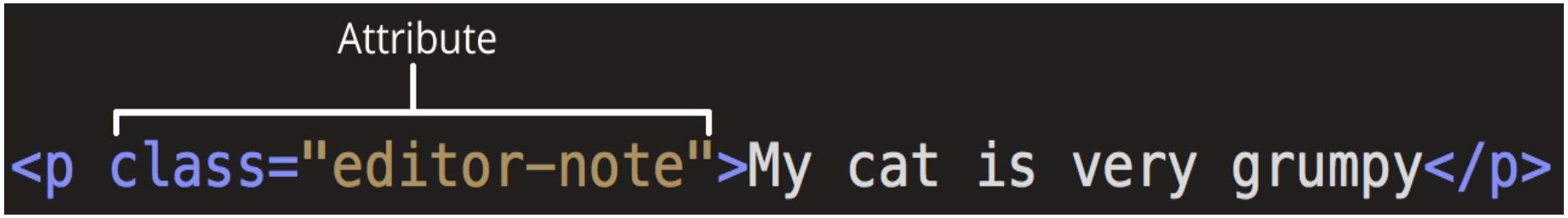
Some elements consist of a single tag, which is typically used to insert/embed something in the document. Such elements are called void elements. For example, the <img> element embeds an image file onto a page:

```
<img src="https://raw.githubusercontent.com/mdn/beginner-html-site/gh-pages/images/firefox-icon.png" alt="Firefox icon" />
```



Attributes

Elements can also have attributes. Attributes look like this:



Attributes contain extra information about the element that won't appear in the content. In this example, the **class** attribute is an identifying name used to target the element with style information.

Attributes

An attribute should have:

- A space between it and the element name. (For an element with more than one attribute, the attributes should be separated by spaces too.)
- The attribute name, followed by an equal sign.
- An attribute value, wrapped with opening and closing quote marks.

Boolean Attributes

Attributes written without values are called Boolean attributes. They can only have one value, which is generally the same as the attribute name. For example, the disabled attribute:

```
<input type="text" disabled="disabled" />
```

OR

```
<input type="text" disabled />
```

Single or Double Quotes?

Both are equivalent for example:

A link to my example.

A link to my example.

*IMP: Don’t mix quotes for e.g. href="https://www.example.com’

But the following is alright

 A link to my example.

What if you wanted to use double quotes inside double quotes? Use *HTML Entities*

A link to my example.

Whitespaces in HTML

These two code snippets are equivalent:

```
<p id="noWhitespace">Dogs are silly.</p>
```

```
<p id="whitespace">Dogs
are                                    silly.</p>
```

No matter how much whitespace you use inside HTML element content (which can include one or more space characters, but also line breaks), the HTML parser reduces each sequence of whitespace to a single space when rendering the code. So why have this facility? The answer is readability. Formatting your HTML code is also very important for debugging and code intelligibility

Entity References: Including Special Characters in HTML

In HTML, the characters **<**, **>**,",' and **&** are special characters. They are parts of the HTML syntax itself. So how do you include one of these special characters in your text?

Using Character References – special codes that represent characters, to be used in these exact circumstances. Each character reference starts with an ampersand (&), and ends with a semicolon (;)

Literal character	Character reference equivalent
<	<
>	>
"	"
'	'
&	&

In HTML, you define a paragraph using the element.

In HTML, you define a paragraph using the <p> element.

HTML Comments

To write an HTML comment, wrap it in the special markers <!-- and -->. For example:

```
<p>I'm not inside a comment</p>
<!-- <p>I am!</p> -->
```

Other Important Things

- Metadata such as charset, name:content, http-equiv:content
- Script and Link elements
- Style tags
- Primary language and span elements
- Headings and hierarchy (and SEO and screen readers)
- Ordered and unordered lists, nesting of lists
- Emphasis and importance using the and tags
- Hyperlinks using the <a> tag
- Title attribute in anchor tags
- File hierarchy and anchor tags
- Document fragments
- Absolute and relative URLs
- Download attribute in <a> tags

Other Important Things

- mailto attribute
- Description lists
- Block quotes and inline quotations
- Citations
- Abbreviations
- Address tags
- Superscripts and subscripts
- <code>, <pre>, <var>, <kbd>, <samp> tags
- <time> tag and datetime attribute
- HTML templates and page design
- Markup validation service (W3C) and debugging
- Multimedia and embedding – img, figure, video, audio, iframe, embed, object, SVG
- HTML tables and divs

Javascript

What is Javascript?

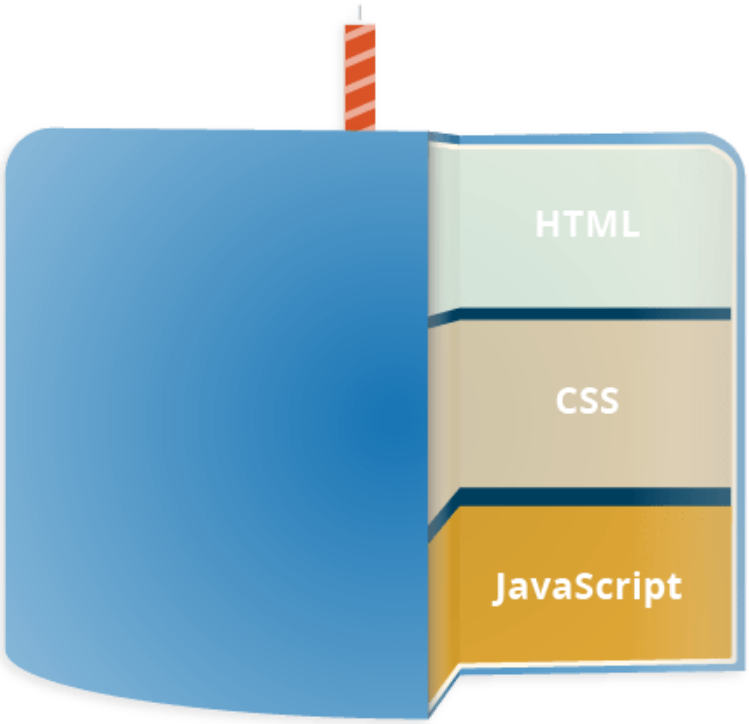
- A scripting or programming language that allows you to implement complex features on web pages

- Used for displaying timely content updates, interactive maps, animated 2D/3D graphics, scrolling video jukeboxes, etc.

- Can change HTML content, attributes, styles etc. that is brings dynamism to static HTML pages

Role of Javascript

•HTML is the markup language that we use to structure and give meaning to our web content, for example defining paragraphs, headings, and data tables, or embedding images and videos in the page.

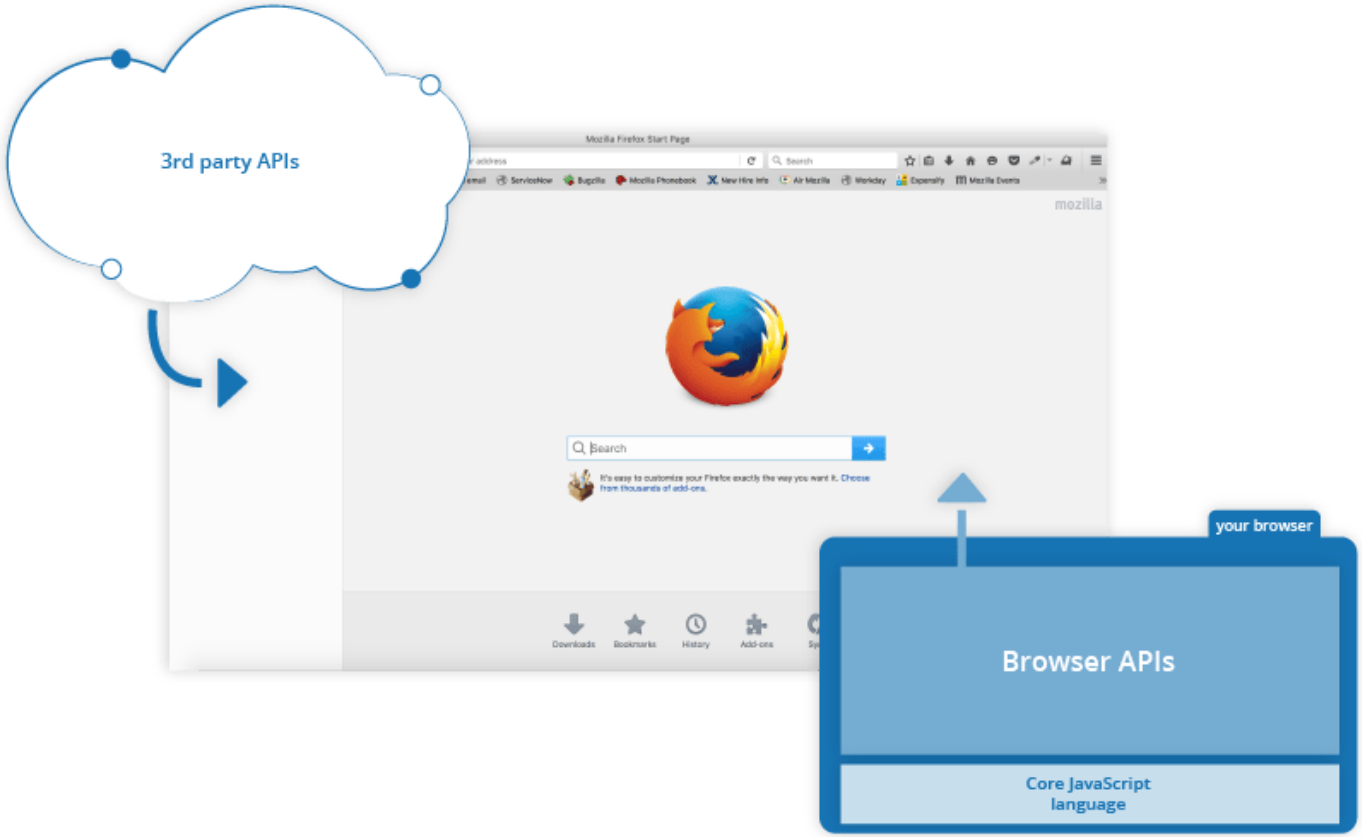


•CSS is a language of style rules that we use to apply styling to our HTML content, for example setting background colors and fonts, and laying out our content in multiple columns.

•JavaScript is a scripting language that enables you to create dynamically updating content, control multimedia, animate images, and pretty much everything else.

APIs and Javascript Functionality

•Application Programming Interfaces or APIs are ready-made sets of code building blocks that allow a developer to implement programs that would otherwise be hard or impossible to implement and fall broadly into two categories



Browser APIs

These are built into your web browser, and are able to expose data from the surrounding computer environment, or do useful complex things. For example:

- The DOM (Document Object Model) API allows you to manipulate HTML and CSS, creating, removing and changing HTML, dynamically applying new styles to your page, etc.
- The Geolocation API retrieves geographical information.
- The Canvas and WebGL APIs allow you to create animated 2D and 3D graphics.
- Audio and Video APIs like HTMLMediaElement and WebRTC allow you to do really interesting things with multimedia, such as play audio and video right in a web page, or grab video from your web camera and display it on someone else's computer

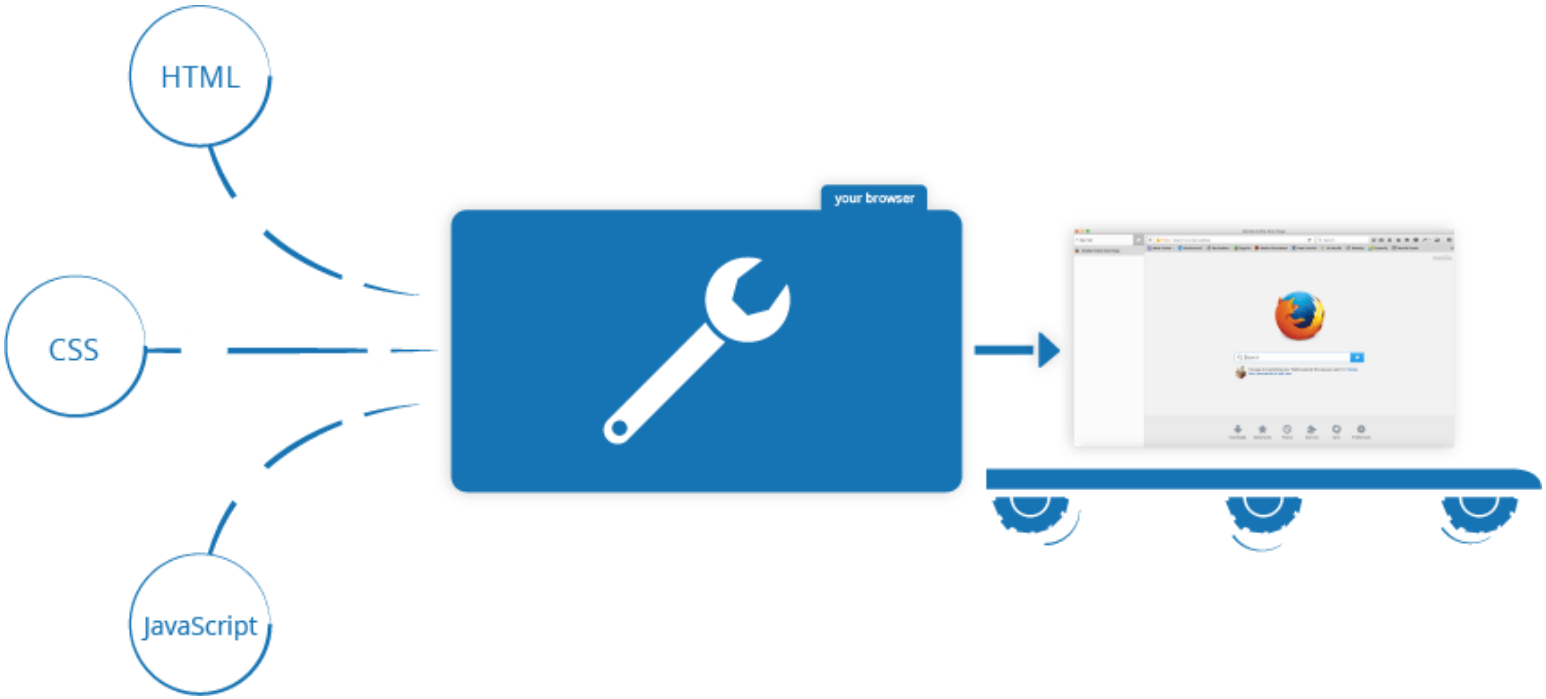
Third Party APIs

These are not built into the browser by default, and you generally have to grab their code and information from somewhere on the Web. For example:

- The Twitter API allows you to do things like displaying your latest tweets on your website.
- The Google Maps API and OpenStreetMap API allows you to embed custom maps into your website, and other such functionality.

Loading a Page in the Browser

When you load a web page in your browser, you are running your code (the HTML, CSS, and JavaScript) inside an execution environment (the browser tab). Note that the code in your web documents is generally loaded and executed in the order it appears on the page.



Using Javascript

Two ways:

- Internal Javascript
- External Javascript

Inline javascript handlers – bac

JS

```
function createParagraph() {
  const para = document.createElement("p");
  para.textContent = "You clicked the button!";
  document.body.appendChild(para);
}
```

HTML

```
<button onclick="createParagraph()">Click me!</button>
```

Instead use addEventListener as follows

```
const buttons = document.querySelectorAll("button");

for (const button of buttons) {
  button.addEventListener("click", createParagraph);
}
```

Script Loading

Consider the following

```
document.addEventListener("DOMContentLoaded", () => {
  // javascript code block
});
```

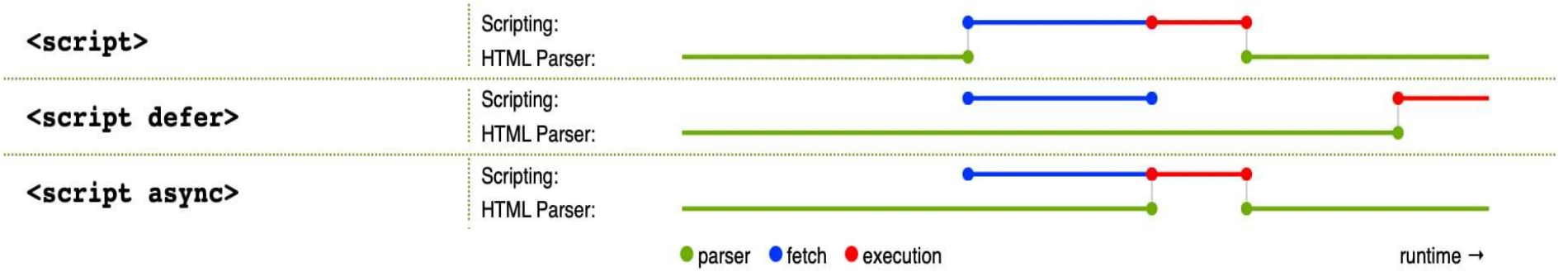
This is an event listener, which listens for the browser's DOMContentLoaded event, which signifies that the HTML body is completely loaded and parsed. The JavaScript inside this block will not run until after that event is fired. Can be put towards end of <body> but will cause slowness (why?)

Async and Defer

The defer attribute, which tells the browser to continue downloading the HTML content once the <script> tag element has been reached

```
<script src="script.js" defer></script>
```

Using async attribute will download the script without blocking the page while the script is being fetched. However, once the download is complete, the script will execute, which blocks the page from rendering. Scripts won't run in any specific order. Use async when the scripts in the page run independently from each other and depend on no other script on the page.



Async and Defer

- async and defer both instruct the browser to download the script(s) in a separate thread, while the rest of the page (the DOM, etc.) is downloading, so the page loading is not blocked during the fetch process.
- Scripts with an async attribute will execute as soon as the download is complete. This blocks the page and does not guarantee any specific execution order.
- Scripts with a defer attribute will load in the order they are in and will only execute once everything has finished loading.
- If your scripts should be run immediately and they don't have any dependencies, then use async.
- If your scripts need to wait for parsing and depend on other scripts and/or the DOM being in place, load them using defer and put their corresponding <script> elements in the order you want the browser to execute them.

Comments in Javascript

- A single line comment is written after a double forward slash (//), e.g.

  // I am a comment

- A multi-line comment is written between the strings /* and */, e.g.

  /* I am
  also a
  comment */

Javascript Operators

Operator	Name	Purpose	Example
`+`	Addition	Adds two numbers together.	`6 + 9`
`-`	Subtraction	Subtracts the right number from the left.	`20 - 15`
`*`	Multiplication	Multiplies two numbers together.	`3 * 7`
`/`	Division	Divides the left number by the right.	`10 / 5`
`%`	Remainder (sometimes called modulo)	Returns the remainder left over after you've divided the left number into a number of integer portions equal to the right number.	`8 % 3` (returns 2, as three goes into 8 twice, leaving 2 left over).
`**`	Exponent	Raises a `base` number to the `exponent` power, that is, the `base` number multiplied by itself, `exponent` times.	`5 ** 2` (returns `25`, which is the same as `5 * 5`).

Javascript Operators

Operator	Name	Purpose	Example	Shortcut for
`+=`	Addition assignment	Adds the value on the right to the variable value on the left, then returns the new variable value	`x += 4;`	`x = x + 4;`
`-=`	Subtraction assignment	Subtracts the value on the right from the variable value on the left, and returns the new variable value	`x -= 3;`	`x = x - 3;`
`*=`	Multiplication assignment	Multiplies the variable value on the left by the value on the right, and returns the new variable value	`x *= 3;`	`x = x * 3;`
`/=`	Division assignment	Divides the variable value on the left by the value on the right, and returns the new variable value	`x /= 5;`	`x = x / 5;`

Javascript Operators

Operator	Name	Purpose	Example
`===`	Strict equality	Tests whether the left and right values are identical to one another	`5 === 2 + 4`
`!==`	Strict-non-equality	Tests whether the left and right values are **not** identical to one another	`5 !== 2 + 3`
`<`	Less than	Tests whether the left value is smaller than the right one.	`10 < 6`
`>`	Greater than	Tests whether the left value is greater than the right one.	`10 > 20`
`<=`	Less than or equal to	Tests whether the left value is smaller than or equal to the right one.	`3 <= 2`
`>=`	Greater than or equal to	Tests whether the left value is greater than or equal to the right one.	`5 >= 4`

Javascript Operators

Operator	Name	Example

`===`	Strict equality (is it exactly the same?)	JS
		`5 === 2 + 4 // false`
		`'Chris' === 'Bob' // false`
		`5 === 2 + 3 // true`
		`2 === '2' // false; number versus string`

`!==`	Non-equality (is it not the same?)	JS
		`5 !== 2 + 4 // true`
		`'Chris' !== 'Bob' // true`
		`5 !== 2 + 3 // false`
		`2 !== '2' // true; number versus string`

`<`	Less than	JS
		`6 < 10 // true`
		`20 < 10 // false`

`>`	Greater than	JS
		`6 > 10 // false`
		`20 > 10 // true`

Loops in Javascript

```
const fruits = ["apples", "bananas", "cherries"];
for (const fruit of fruits) {
        console.log(fruit);
}
```

Can also use a while loop (look it up)

Variable Types

- Numbers
- Strings
- Booleans
- Arrays
- Objects

JavaScript is a "dynamically typed language", which means that, unlike some other languages, you don't need to specify what data type a variable will contain (numbers, strings, arrays, etc.). Just use the keywords var or let to declare / initialize the values.

Constants: like variables, except that:

- you must initialize them when you declare them
- you can't assign them a new value after you've initialized them

Strings

- Creation
- Escaping characters
- Concatenation
- Multiline strings
- String methods

Objects

An object is a collection of related data and/or functionality

```
const person = {
        name: ["Bob", "Smith"],
        age: 32,
        bio: function () {
                console.log(`${this.name[0]} ${this.name[1]} is
                ${this.age} years old.`); },
        introduceSelf: function () {
                console.log(`Hi! I'm ${this.name[0]}.`);
        },
};
```

Objects

person.name;

person.name[0];

person.age;

person.bio(); // "Bob Smith is 32 years old.“

person.introduceSelf(); // "Hi! I'm Bob.“

Dot and Bracket notation – for accessing object properties

const person = { name: { first: "Bob", last: "Smith", }, age: 32, // … };

person.name.first; person.name.last;

person["age"]; person["name"]["first"];

Constructors

JSON Objects

- JSON is purely a string with a specified data format — it contains only properties, no methods.
- JSON requires double quotes to be used around strings and property names. Single quotes are not valid other than surrounding the entire JSON string.
- Even a single misplaced comma or colon can cause a JSON file to go wrong, and not work. You should be careful to validate any data you are attempting to use (although computer-generated JSON is less likely to include errors, as long as the generator program is working correctly). You can validate JSON using an application like JSONLint.
- JSON can actually take the form of any data type that is valid for inclusion inside JSON, not just arrays or objects. So for example, a single string or number would be valid JSON.
- Unlike in JavaScript code in which object properties may be unquoted, in JSON only quoted strings may be used as properties.

An Example With Javascript + HTML

```
<!DOCTYPE html>
<html lang="en-us">
 <head>
  <meta charset="utf-8">

  <title>Number guessing game</title>
  <script src="javascript_code.js">
  <style>
   html {
    font-family: sans-serif;
   }

   body {
    width: 50%;
    max-width: 800px;
    min-width: 480px;
    margin: 0 auto;
   }

   .form input[type="number"] {
    width: 200px;
   }

   .lastResult {
    color: white;
    padding: 3px;
   }
  </style>
 </head>
<body>
  <h1>Number guessing game</h1>

  <p>We have selected a random number between 1 and 100. See if you can guess it in 10 turns or fewer. We'll tell you if your guess was too high or too low.</p>

  <div class="form">
   <label for="guessField">Enter a guess: </label>
   <input type="number" min="1" max="100" required id="guessField" class="guessField">
   <input type="submit" value="Submit guess" class="guessSubmit">
  </div>

  <div class="resultParas">
   <p class="guesses"></p>
   <p class="lastResult"></p>
   <p class="lowOrHi"></p>
  </div>
</body>
</html>
```

An Example With Javascript + HTML

```
let randomNumber = Math.floor(Math.random() * 100) + 1;
const guesses = document.querySelector('.guesses');
const lastResult = document.querySelector('.lastResult');
const lowOrHi = document.querySelector('.lowOrHi');
const guessSubmit = document.querySelector('.guessSubmit');
const guessField = document.querySelector('.guessField');
let guessCount = 1;
let resetButton;
```

```
function checkGuess() {
    const userGuess = Number(guessField.value);
    if (guessCount === 1) {
      guesses.textContent = 'Previous guesses: ';
    }

    guesses.textContent += userGuess + ' ';

    if (userGuess === randomNumber) {
      lastResult.textContent = 'Congratulations! You got it right!';
      lastResult.style.backgroundColor = 'green';
      lowOrHi.textContent = '';
      setGameOver();
    } else if (guessCount === 10) {
      lastResult.textContent = '!!!GAME OVER!!!';
      lowOrHi.textContent = '';
      setGameOver();
    } else {
      lastResult.textContent = 'Wrong!';
      lastResult.style.backgroundColor = 'red';
      if(userGuess < randomNumber) {
        lowOrHi.textContent = 'Last guess was too low!' ;
      } else if(userGuess > randomNumber) {
        lowOrHi.textContent = 'Last guess was too high!';
      }
    }

    guessCount++;
    guessField.value = '';
    guessField.focus();
  }
```

An Example With Javascript + HTML

```
guessSubmit.addEventListener('click', checkGuess);

  function setGameOver() {
    guessField.disabled = true;
    guessSubmit.disabled = true;
    resetButton = document.createElement('button');
    resetButton.textContent = 'Start new game';
    document.body.appendChild(resetButton);
    resetButton.addEventListener('click', resetGame);
  }

  function resetGame() {
    guessCount = 1;
    const resetParas = document.querySelectorAll('.resultParas p');
    for (const resetPara of resetParas) {
      resetPara.textContent = '';
    }

    resetButton.parentNode.removeChild(resetButton);
    guessField.disabled = false;
    guessSubmit.disabled = false;
    guessField.value = '';
    guessField.focus();
    lastResult.style.backgroundColor = 'white';
    randomNumber = Math.floor(Math.random() * 100) + 1;
  }
```

PostgreSQL

What is PostgreSQL?

An open source relational database management system (DBMS) developed by a worldwide team of volunteers. PostgreSQL is not controlled by any corporation or other private entity and the source code is available free of charge.

History of PostgreSQL

- A Brief History of PostgreSQL
- PostgreSQL, originally called Postgres, was created at UCB by a computer science professor named Michael Stonebraker. Stonebraker started Postgres in 1986 as a follow-up project to its predecessor, Ingres, now owned by Computer Associates.
- **1977-1985** – A project called INGRES was developed.
  - Proof-of-concept for relational databases
  - Established the company Ingres in 1980
  - Bought by Computer Associates in 1994
- **1986-1994** – POSTGRES
  - Development of the concepts in INGRES with a focus on object orientation and the query language - Quel
  - The code base of INGRES was not used as a basis for POSTGRES
  - Commercialized as Illustra (bought by Informix, bought by IBM)
- **1994-1995** – Postgres95
  - Support for SQL was added in 1994
  - Released as Postgres95 in 1995
  - Re-released as PostgreSQL 6.0 in 1996
  - Establishment of the PostgreSQL Global Development Team

Database Operations

- Creation

CREATE DATABASE db_name

OWNER = role_name

TEMPLATE = template

ENCODING = encoding

LC_COLLATE = collate

LC_CTYPE = ctype

TABLESPACE = tablespace_name

CONNECTION LIMIT = max_concurrent_connection

- **db_name:** It is the name of the new database that you want to create. It must always be a unique name.
- **role_name:** It is the role name of the user who will own the new database.
- **template:** It is the name of the database template from which the new database gets created.
- **encoding:** It specifies the character set encoding for the new database. By default, it is the encoding of the template database.
- **collate:** It specifies a collation for the new database.
- **ctype:** It specifies the character classification for the new database like digit, lower and upper.
- **tablespace_name:** It specifies the tablespace name for the new database.
- **max_concurrent_connection:** It specifies the maximum concurrent connections to the new database.

Datatypes

- **Boolean**
- **Character** Types *[such as char, varchar, and text]*
- **Numeric** Types *[such as integer and floating-point number]*
- **Temporal** Types *[such as date, time, timestamp, and interval]*
- **UUID** *[for storing UUID (Universally Unique Identifiers)]*
- **Array** *[for storing array strings, numbers, etc.]*
- **JSON** *[stores JSON data]*
- **hstore** *[stores key-value pair]*
- **Special** Types *[such as network address and geometric data]*

Querying Data

The various clauses that can be used with the SELECT statement are listed below:

- **DISTINCT operator:** It is used to select distinct rows from a table.
- **ORDER BY clause:** It is used to sort table rows.
- **WHERE clause:** It is used to filter rows from a table. LIMIT clause: It is used to select a subset of rows from the table
- **FETCH clause**: It is also used to select subset of rows from the table.
- **GROUP BY clause:** It is used to group different rows into a single group.
- **HAVING clause:** It is used to filter rows from a table with specified attributes or features.
- **FROM clause:** It is used to specify a column in a table.
- **joins:** It is used to join two or more tables together using joins such as **INNER JOIN, LEFT JOIN, FULL OUTER JOIN, CROSS JOIN** clauses.
- **Set operators:** These operators such as **UNION, INTERSECT,** and **EXCEPT** are used to manipulate the different sets of data.
- **IN operator:** Used to test whether value matches any in a list of values or subquery

Modifying Data

- INSERT INTO table(column1, column2, …) VALUES (value1, value2, …);
- UPDATE table SET column1 = value1, column2 = value2, ... WHERE condition;
- DELETE FROM table WHERE condition;
- INSERT INTO table_name(column_list) VALUES(value_list) ON CONFLICT target action;
  - The target can be :
    - (column_name) – any column name.
    - ON CONSTRAINT constraint_name – where the constraint name could be the name of the UNIQUE constraint.
    - WHERE predicate – a WHERE clause with a boolean condition.
  - The action can be :
    - DO NOTHING – If the row already exists in the table, then do nothing.
    - DO UPDATE SET columnA = valueA, …. WHERE condition – update some fields in the table depending upon the condition.

Keys and Constraints

•**Primary Key**: A primary key is a column or a group of columns used to identify a row uniquely in a table. Technically speaking a primary key constraint is the blend of a not-null constraint and a UNIQUE constraint. Only one primary key must exist in a table.

•**Foreign Key**: A foreign key is a column or a group of columns used to identify a row uniquely of a different table. The table that comprises the foreign key is called the referencing table or child table. And the table to that the foreign key references is known as the referenced table or parent table. A table can possess multiple foreign keys according to its relationships with other tables.

•**Check Constraint:** The CHECK constraint is primarily used to specify if a value in a column necessarily meets a specific requirement. The CHECK constraint utilizes a Boolean expression to assess the values before performing an insert or update operation to the column. If the values pass the check, PostgreSQL allows the insertion or update of those values into the column. It is primarily used while creating a table. Syntax is: variable_name Data-type CHECK(condition);

•**Unique Constraint:** Used to make sure that values stored in a column or a group of columns are unique across rows in a table. Every time the user inserts a new row, PostgreSQL checks if the value already exists in the table if UNIQUE constraints are used. If it discovers that the new value is already present, it denies the change and issues an error. A similar process is carried out for updating existing data.

•**Not NULL Constraint**: Not-Null constraint as the name suggests is used to ensure that any value in the respective column is not null. In the world of database, NULL is unknown or missing information. The NULL value is separate from an empty string or the number zero.

Joins

There are 4 basic types of joins supported by PostgreSQL, namely:

- Inner Join
- Left Join
- Right Join
- Full Outer Join

Some special PostgreSQL joins are below:

- Natural Join
- Cross Join
- Self Join

Operators

- **UNION**: Used to combine result sets of multiple queries into a single set of result. It is used to combine result sets of two or more SELECT statements into a single result set. **Syntax:** SELECT column_1, column_2 FROM table_name_1 UNION SELECT column_1, column_2 FROM table_name_2;
- **INTERSECT**: used to combine two or more result sets returned by the SELECT statement and provide with the common data among the tables into a single result set. **Syntax:** SELECT column_list FROM A INTERSECT SELECT column_list FROM B; The below rules must be followed while using the INTERSECT operator with the SELECT statement:
  - The number of columns and their order in the SELECT clauses must be the same.
  - The data types of the columns must be compatible.
- **EXCEPT**: used to return distinct rows from the first (left) query that are not in the output of the second (right) query while comparing result sets of two or more queries. **Syntax:** SELECT column_list FROM A WHERE condition_a EXCEPT SELECT column_list FROM B WHERE condition_b;

Operators

- **ANY**: used to compare a scalar value with a set of values returned by a subquery. **Syntax:** expression operator ANY(subquery)The below rules must be followed while using PostgreSQL ANY operator:
  - The subquery must return exactly one column.
  - The ANY operator must be preceded by one of the following comparison operator =, <=, >, <, > and <>
  - The ANY operator returns true if any value of the subquery meets the condition, otherwise, it returns false.
- **ALL**: used for comparing a value with a list of values returned by a subquery. **Syntax:** comparison_operator ALL (subquery)The below rules need to be followed while using the ALL operator:
  - The ALL operator always needs to be preceded by a comparison operator(=, !=, <, >, >=, <=).
  - It must always be followed by a subquery surrounded by parentheses.
- **EXISTS**: used to test for the existence of rows in a subquery.It is generally used with correlated subqueries. If the subquery returns at least one row, the result of EXISTS is true. In case the subquery returns no row, the result is of EXISTS is false. **Syntax:** EXISTS (subquery)

PostGIS

What is PostGIS?

PostGIS is a Postgres extension that adds support for storing and manipulating spatial data types. We would typically need to use spatial data storage when we are building software applications that store, manipulate and visualize data on a map.

To avail its functionalities, first download and install PostGIS from the website *https://postgis.net/*

Then run the following command in a query window against the database for which postGIS is required:

CREATE EXTENSION IF NOT EXISTS postgis;

Spatial Data Types

- PostGIS supports a few different types of geospatial data types
- Geospatial coordinates reference points on the surface of the Earth
- This makes it possible to not just render such objects on the map, but also to analyze interactions between them
- PostGIS helps us in using spatial objects and operations to solve real-world problems

Vector Data

- Spatial vector data supports basic geometrical shapes, like point, line string, and polygon. In addition to basic geometries, PostGIS supports some more advanced ones:
  - Multi-version of basic geometries — a homogeneous collection of points, line strings or, polygons
  - 3D versions of basic geometries — the same as basic geometries with the Z-coordinate added
  - Geometry collection — a collection of any geometries, homogeneous or heterogeneous
  - Polyhedral surface — complex 3D surface

Vector Data

Maps and navigation applications heavily rely on vector objects to model features of the map. Looking at the screenshot, most objects on Google Maps can be represented as polygons (e.g., buildings) or points (e.g., businesses) or lines (e.g., roads).

When viewing a map in 3D mode, buildings are often represented as polyhedral surfaces.

Vector Data

- To create a table using the "geometry" data type, we can run the statement below
  - CREATE TABLE building (
    id UUID PRIMARY KEY,
    geom geometry
    );
  - CREATE TABLE building (
    id UUID PRIMARY KEY,
    geom geometry(Polygon)
    );
  - CREATE TABLE building (
    id UUID PRIMARY KEY,
    geom geometry(Polygon,4326)
    );

This creates a table with column "geom" of type geometry which is a generic type for all vector objects. That means that we can combine points, lines, polygons, and other vector objects in the same column. If we specify the geometry (specified as 'Polygon' in second query) as a part of the column type definition PostGIS will not allow other geometry types to be inserted in the same column.

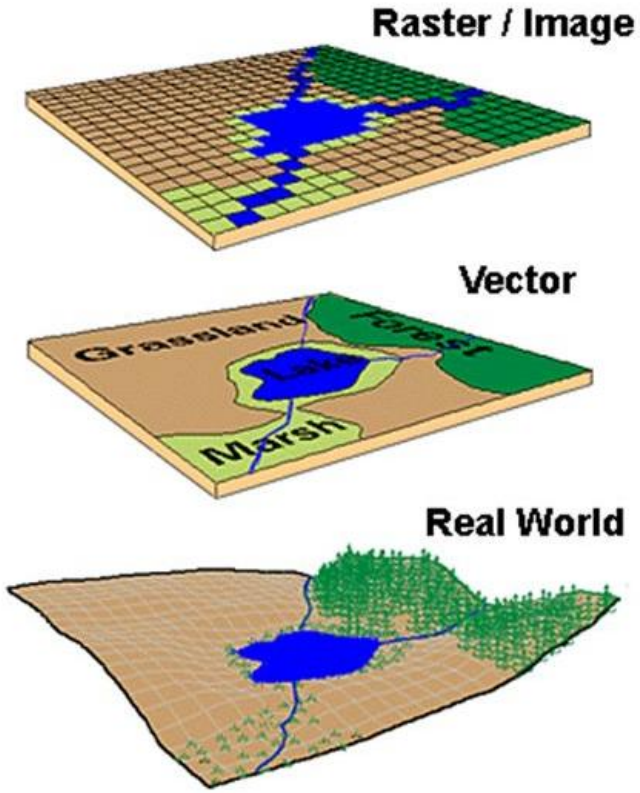
We can also include the SRID (spatial reference identifier – 4326 in this example) in the column type definition, forcing all values to conform to the same SRID

Raster Data

The spatial raster data type is also similar to its graphical cousins (JPEG, PNG, TIFF, and other raster files we use in our daily lives), but there are some differences.

- Unlike the regular rasters, where one pixel is a dot on a screen, spatial rasters have spatial resolution that defines pixel width and height. So each pixel of a spatial raster covers a uniformly sized rectangle on the map.
- Spatial rasters have one or more bands, and each band has a matrix of values for all "pixels". Data type of each band is set separately, and it can be almost any numeric type — binary (useful for masking), integers or floating-point values.
- A spatial equivalent of a 24bit RGB raster would be a 3-band raster where each band is defined as an unsigned 8bit int. However, with the flexibility of storing any numerical values in addition to colors, we can leverage rasters to store various information — terrain elevation, population density, vegetation information or metrics, etc.

Raster Data



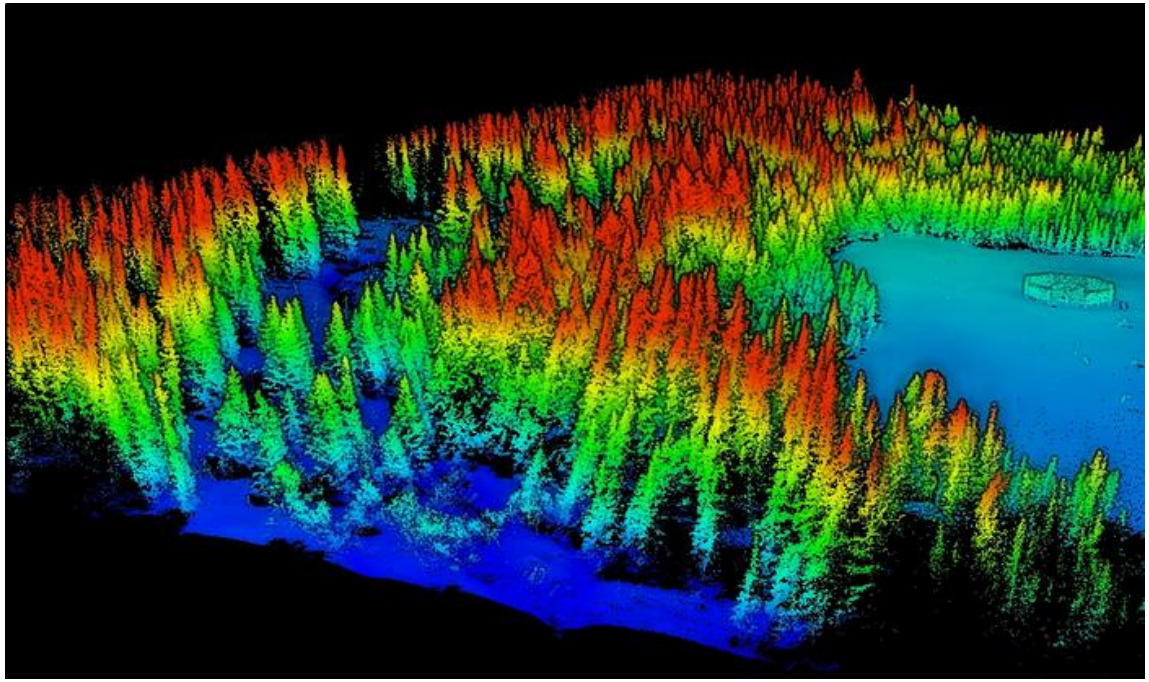
Raster data support is included in a separate postgis extension that needs to be installed before we can work with rasters. Run the command

CREATE EXTENSION IF NOT EXISTS postgis_raster;

Then we can create a table containing a column of type raster

```
CREATE TABLE satellite_image (
id UUID PRIMARY KEY,
rast raster
);
```

Point Clouds



- Point cloud data format can be seen as a mix between rasters and vectors
- It's similar to rasters in a way that it represents a discrete data set, composed of individual points rather than shapes. However, unlike rasters, there is no resolution or density, so points can be anywhere in the 3D space

- Comparing point clouds to vector types — it's similar to a collection of 3D vector points.
- Point cloud data is typically obtained from LiDAR, 3D scanners or similar devices that measure physical properties of objects in the 3D space
- When visualized, it looks similar to the image above. The trees (or any other objects) look like continuous 3D objects, but they are all made up of discrete points in space

Point cloud support is included in a separate postgis extension that needs to be installed before we can work with rasters.

```
CREATE EXTENSION pointcloud;
CREATE EXTENSION
pointcloud_postgis;
```

Spatial Operations

When dealing with the “regular” non-spatial data, we typically join and filter tables based on exact values in a column containing primitive values representing object identifiers (integers, strings or maybe UUIDs). Our typical queries on a non-spatial dataset may look something like this

```
SELECT *
FROM book b
INNER JOIN publisher p ON p.id = b.publisher_id;
```

```
SELECT *
FROM book b
WHERE b.publisher_id = 12345;
```

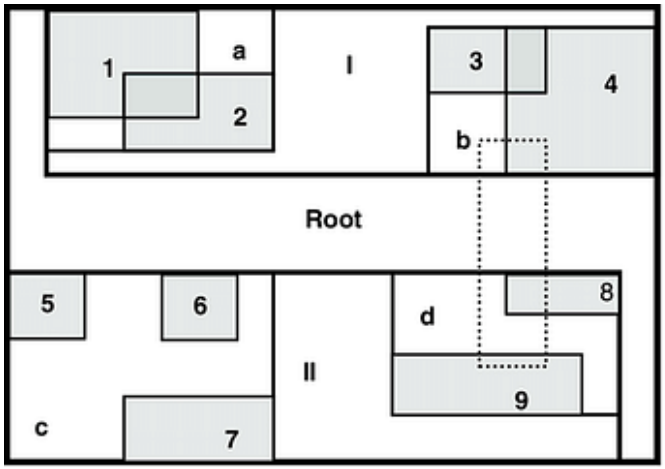
Now if we think for a moment about how Google Maps works — zoom, pan, click on objects, we can deduce that the most commonly used operation on spatial data is the intersection. Whenever we pan or zoom the map, the system needs to figure out which objects should be fetched and shown by intersecting objects with the rectangle that represents our visible portion of the map. The query below finds buildings that intersect a given rectangle on the map

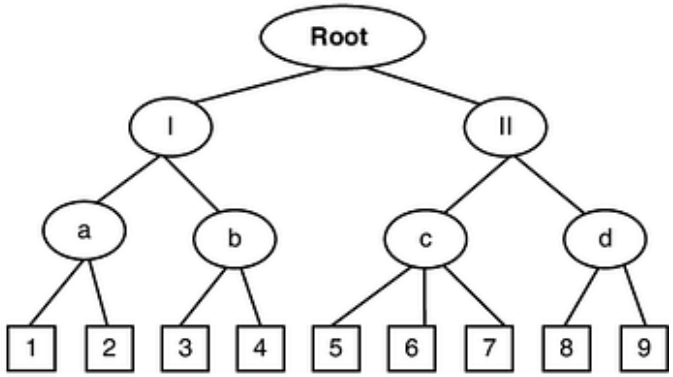
```
SELECT *
FROM building b
WHERE ST_Intersects(b.geom, ST_MakeEnvelope(24, 47, 25, 48, 4326));
```

Another commonly used operation is distance calculation which is often used to determine which objects are located in the proximity of a given point on the map

Spatial Indexing

- When indexing primitive values (like integer, character, string etc.) the database typically uses Hash or B-Tree to construct an index
- Due to the difference in operations that are typically used with spatial data, this approach cannot be applied here
- Spatial index needs to be constructed in a way that would allow us to efficiently find spatial objects from a collection of spatial objects that intersect with a given spatial object
- To solve this problem, spatial indexes use the R-Tree ("R" as in Rectangle) structure, which builds a tree of rectangles where each of the child node rectangles are contained in the parent node rectangle
- At the leaf of the tree are the rectangles that represent bounding boxes of our spatial objects in a PostGIS column





Spatial Indexing

- That way, we can quickly traverse the tree to find which objects intersect with a given object instead of checking each object for intersection. This reduces the time complexity of the filtering operation from O(N) to O(logN)
- SQL command that creates a spatial index is very similar to the "regular" index creation
  - CREATE INDEX building_geom_idx ON building USING GIST(geom);
- The only difference here is the "GIST" part, which signals PostGIS that we require a "generic index structure" for this index. PostGIS supports three spatial indexes (GIST, SPGIST and BRIN) but in most cases, GIST is a decent choice
- Spatial indices can also be used for raster data, as we often need to get to the relevant rasters quickly. The same syntax can be applied to a raster column, but in this case we are indexing bounding boxes around raster images, so the statement needs to include ST_ConvexHull function
  - CREATE INDEX satellite_image_rast_idx ON satellite_image USING GIST(ST_ConvexHull(rast));
- As with any indexing, there is a performance tradeoff at the time objects are inserted into the database, as PostGIS needs to insert new objects into the R-Tree index. But whenever we plan to use spatial operations, we should consider adding an index to the columns used in the query, as it dramatically improves performance

SRIDs

- SRID (spatial reference identifier) is an important piece of information that we need to attribute to each spatial object. It includes information about the coordinate system, where the (0, 0) point is on the globe, the resolution of the coordinates, and how coordinates on the map correspond with actual points on the globe

- The most commonly used SRID is **WGS84 — SRID 4326** which is used for GPS tracking, Google Maps and many other applications, however there are many more SRIDs that are popular, some offering higher precision than WGS84 in certain areas on the globe. So we always need to be aware of the SRID of the data coming into the system

- PostGIS is very flexible when it comes to SRIDs. In the previous example, we created a table “building” with a geometry column without SRID specification. This means that PostGIS will allow polygons with any SRID to be inserted. This can sometimes be useful, even necessary, in cases when we cannot predict or change the SRID of the incoming data, but it should be avoided whenever possible

SRIDs

- The first reason for having uniform SRIDs across all objects is that spatial queries require the objects of the same SRID and will fail if we try to intersect objects of different SRIDs
  - SELECT ST_Intersects(
    ST_MakeEnvelope(24, 47, 25, 48, 4979),
    ST_MakeEnvelope(24, 47, 25, 48, 4326)
    );

    ERROR: ST_Intersects: Operation on mixed SRID geometries (Polygon, 4979) != (Polygon, 4326)
- Whenever we have mismatched SRIDs, we can convert one of the spatial objects to the SRID of the other object
  - SELECT ST_Intersects(
    ST_Transform(ST_MakeEnvelope(24, 47, 25, 48, 4979), 4326),
    ST_MakeEnvelope(24, 47, 25, 48, 4326)
    );
- In this query, ST_Transform transforms all the coordinates from the source SRID to the destination SRID and outputs a polygon with SRID 4326 that can be intersected with the other polygon with no errors

SRIDs

However, there's a performance penalty that comes with this approach

- Firstly, this transformation will take some time. And more importantly, we will be unable to use spatial indexing to improve the performance of ST_Intersects operation because spatial index applies to the geometries in the original SRIDs, not to the transformed geometries in the target SRIDs
- The query execution plan will need to perform a table scan on the first table to figure out which of the objects intersect the objects from the second table, AFTER being transformed to the destination SRID

One way to deal with this problem is to perform ST_Transform on all objects at the time they are inserted into the database and maintain consistency between SRIDs at all times. **This has many benefits, but it is worth noting that object transformation is not always exact, and we will lose some precision when transforming from one SRID to another.** If the precision is critical for the software, it may be a good idea to store both the original and transformed object in the database and use them interchangeably.

Interesting Links and Info

For learning and trying out more about HTML, Javascript, CSS, PostgreSQL etc. visit

https://www.w3schools.com/

For formatting your code online try

https://formatter.org/

Press F12 after opening your browser to see the developer options

Right click on any webpage, choose 'View Source' to see the entire HTML code of that page

Right click on any webpage, choose 'Inspect' to see the code and styles etc. associated with the element where your mouse is hovering over

For very useful sessions on Remote Sensing, GIS and programming visit ISRO's channel

https://www.youtube.com/@iirsoutreachdehradun/playlists

For code related queries

https://stackoverflow.com/

https://www.geeksforgeeks.org/

About web services

https://en.wikipedia.org/wiki/Web_service

For PostgreSQL and PostGIS

https://www.postgresql.org/

https://postgis.net/

Leafletjs

https://leafletjs.com/