

Data transmission between server and client & database management

Sending data to server or vice versa

Client side data storage

Database

Database management systems

SQL

Previous week

HTML

- To design webpages

CSS

- For styles

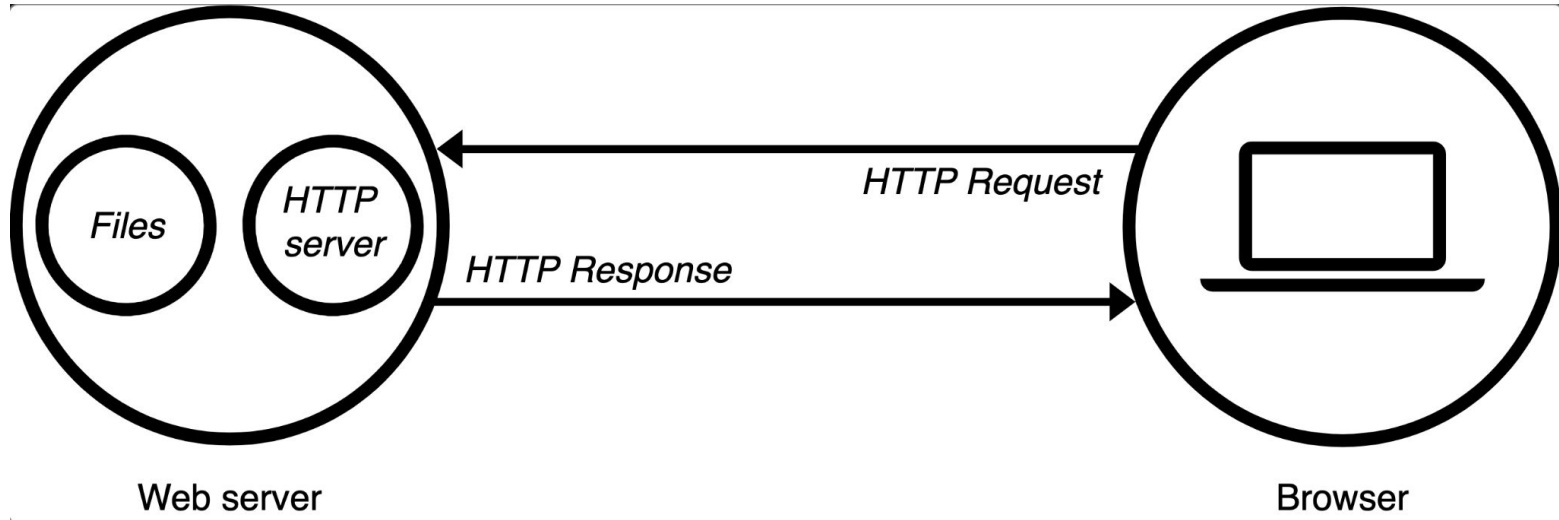
Javascript

- A programming language,
- We use to modify/add/remove HTML objects and website data

Event based programming

- E.g. onclick run a Javascript function

A web server



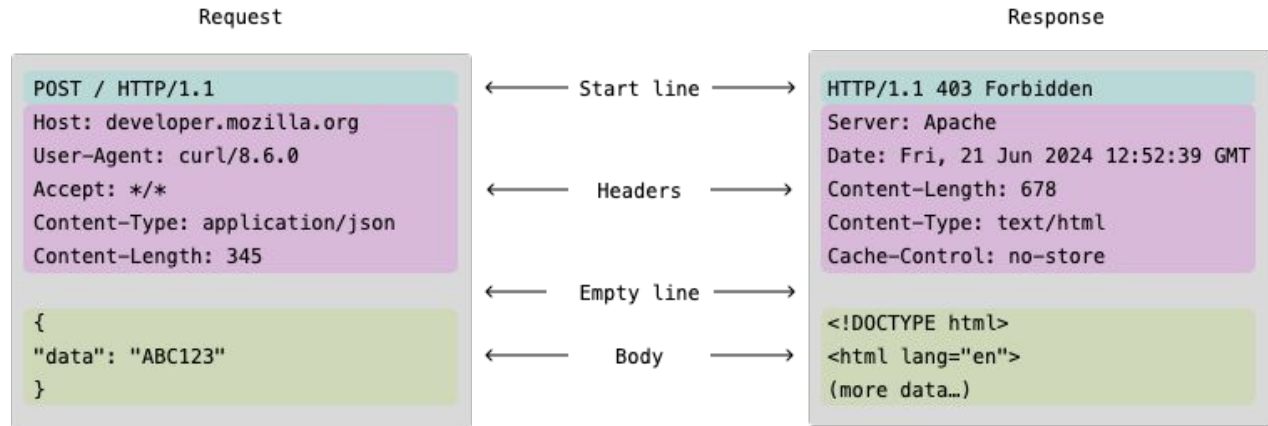
https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Web_mechanics/What_is_a_web_server

HTTP

A **Protocol** is a set of rules for communication between two computers.

HTTP specifies how to transfer hypertext (linked web documents) between two computers.

example:



<https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages>

Javascript Fetch API

The [Fetch API](#) provides a JavaScript interface for making HTTP requests and processing the responses.

Fetch is the modern replacement for [XMLHttpRequest](#):

unlike XMLHttpRequest, which uses callbacks, Fetch is promise-based and is integrated with features of the modern web such as [service workers](#) and [Cross-Origin Resource Sharing \(CORS\)](#).

Example

```
<!DOCTYPE html>
<html>
<head><style>
    .fetch:before {
        content: 'fetch-----: ';
    }
    .xhr:before {
        content: 'xhr-----: ';
    }
</style></head>
<body>
<h1>The XMLHttpRequest Object and Fetch</h1>
<button type="button" onclick="getData()">Fetch</button>
<button type="button" onclick="loadDoc()">XHR</button>
<p id="fetch" class="fetch"></p>
<p id="xhr" class="xhr"></p>
```

The XMLHttpRequest Object and Fetch

Fetch XHR

fetch-----:

xhr-----:

I would like to fetch/get content of

https://www.w3schools.com/jsref/fetch_info.txt

```
<h1>Fetch API</h1>
```

```
<p>The Fetch API interface allows web browser to make HTTP requests to web servers.</p>
```

```
<p>If you use the XMLHttpRequest Object, Fetch can do the same in a simpler way.</p>
```

Note: for cross origin requests; use `JSONP-script.src`, `form`, etc. See [Fetch: Cross-Origin Requests](#)

```
async function getData() {
    document.getElementById("fetch").innerHTML = "send";
    let response = await fetch("https://www.w3schools.com/jsref/fetch_info.txt");
    let text = await response.text();
    document.getElementById("fetch").innerHTML = text;
}

function loadDoc() {
    var xhttp = new XMLHttpRequest();
    xhttp.onreadystatechange = function () {
        if (this.readyState ==XMLHttpRequest.DONE
            && this.status == 200) {
            document.getElementById("xhr").innerHTML = this.responseText;
        }
    };
    xhttp.open("GET", "https://www.w3schools.com/jsref/fetch_info.txt", true);
    xhttp.send();
}

Note: you can only run this on www.w3schools.com
```


How to send formatted data?

XML

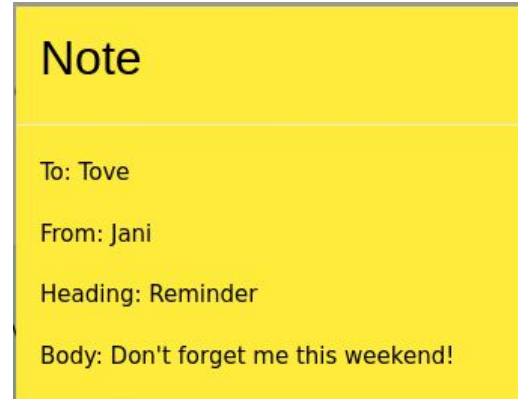
XML stands for eXtensible Markup Language.

- designed to store and transport data.
- designed to be both human- and machine-readable.

<https://www.w3schools.com/xml/default.asp>

note.xml

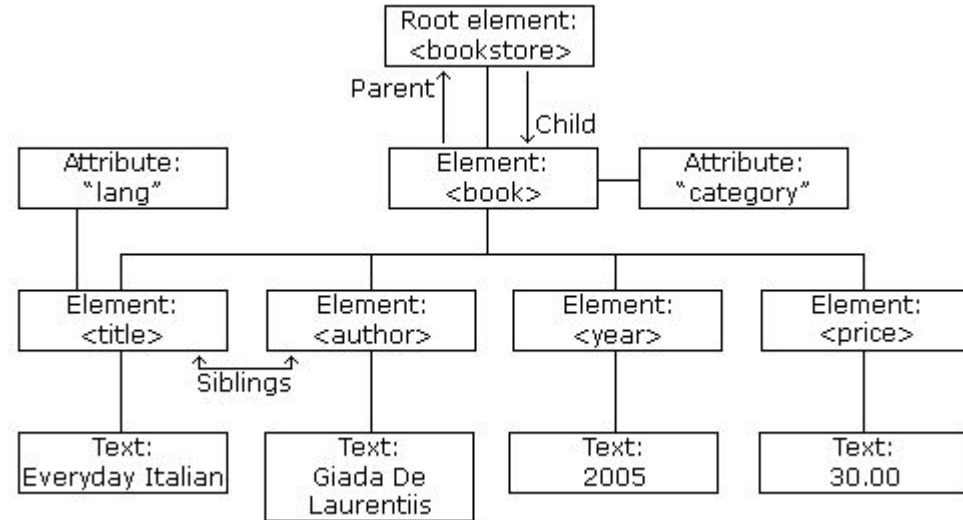
```
<note>  
  <to>Tove</to>  
  <from>Jani</from>  
  <heading>Reminder</heading>  
  <body>Don't forget me this weekend!</body>  
</note>
```



XML tree

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="children">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
```

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```



Javascript JSON

JSON stands for **J**ava**S**cript **O**bject **N**otation

```
{  
  "employees": [  
    {"firstName": "John", "lastName": "Doe"},  
    {"firstName": "Anna", "lastName": "Smith"},  
    {"firstName": "Peter", "lastName": "Jones"}  
  ]  
}
```

https://www.w3schools.com/js/js_json.asp

String to JSON & JSON to string

```
<p id="demo"></p>
<p id="demo2"></p>
<script>
let text = '{ "employees" : [' +
'{ "firstName":"John" , "lastName":"Doe" },' +
'{ "firstName":"Anna" , "lastName":"Smith" },' +
'{ "firstName":"Peter" , "lastName":"Jones" } ]}';
const obj = JSON.parse(text);

document.getElementById("demo").innerHTML =
obj.employees[1].firstName + " " + obj.employees[1].lastName;
document.getElementById("demo2").innerHTML = JSON.stringify(obj);
</script>
```

https://www.w3schools.com/js/js_json.asp

Exercise

Design a weather page by using any of the free weather services

<https://www.weather.gov/documentation/services-web-api>

<https://www.metoffice.gov.uk/services/data/datapoint/uk-observations-detailed-documentation>

Note: Some servers may not allow cross origin requests; you can use `JSONP-script.src`, `form`, etc. See also [Fetch: Cross-Origin Requests](#)

Client-side storage of website data

Cookies

Web Storage API

IndexedDB (indexed database)

Cookies

HTTP cookies, web cookies, Internet cookies, browser cookies, or simply **cookies**

A **cookie** is a small piece of information left on a visitor's computer by a website, via a web browser.

Cookies are used to personalize a user's web experience with a website.

- It may contain the user's preferences or inputs when accessing that website. A user can customize their web browser to accept, reject, or delete cookies.

[HTTP cookie - Wikipedia](#)

[Cookie - MDN Web Docs Glossary: Definitions of Web-related terms](#)

```
document.cookie = "yummy_cookie=chocolate";  
document.cookie = "tasty_cookie=strawberry";  
console.log(document.cookie);  
// logs "yummy_cookie=chocolate;  
tasty_cookie=strawberry"
```

```
document.cookie = "yummy_cookie=blueberry";  
  
console.log(document.cookie);  
// logs "tasty_cookie=strawberry;  
yummy_cookie=blueberry"  
allCookies = document.cookie;
```

Ctrl+shift+c

Then open applications to see storages

Web Storage API

The **Web Storage API** provides mechanisms by which browsers can store key/value pairs, in a much more intuitive fashion than using [cookies](#).

Two mechanisms

- `sessionStorage` maintains a separate storage area as long as the browser tab is open, including page reloads and restores.
- `localStorage` does the same thing, but persists even when the browser is closed and reopened.

https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API

https://www.w3schools.com/html/html5_webstorage.asp

Web Storage API

```
// Save data to sessionStorage
sessionStorage.setItem("key1", "value1");

// Get saved data from sessionStorage
let data = sessionStorage.getItem("key1");

// Remove saved data from sessionStorage
sessionStorage.removeItem("key1");

// Remove all saved data from sessionStorage
sessionStorage.clear();
```

Web Storage API

```
localStorage.setItem("myCat", "Tom");
```

```
const user1 = {  
  name: 'Jane Smith',  
  age: 30,  
  email: 'jane@example.com'  
};
```

```
localStorage.setItem('user1',  
  JSON.stringify(user1));
```

```
// Retrieving values  
const cat = localStorage.getItem("myCat");  
const name = localStorage.getItem('name');  
const user =  
JSON.parse(localStorage.getItem('user1'));
```

```
//to clear everything  
//localStorage.clear();
```

IndexedDB

A **database** is an organized collection of structured information, or data, typically stored electronically in a computer system.

A database is usually controlled by a database management system (DBMS).

A database management system provides a mechanism for the storage and retrieval of data.

IndexedDB is a **NoSQL database management system**

We use on the client side!

Database Management Systems

- Hierarchical databases
- Network databases
- Object-oriented databases(object database management system)
 - Data is stored and managed as objects
 -
- Relational databases
 - Store data using tables, rows, and columns
- NoSQL databases
 - Document databases
 - **Key, value stores**
 - Column oriented databases
 - Graph databases

Example

We have a JSON data, key-values

To store this in **indexedDB**

- **Open a database**

- request =

- ```
indexedDB.open("mydatabase", 3);
```

- 3 version of the database schema
    - E.g. new version of the app and changed schema

```
const customerData = [
 {
 ssn: "444-44-4444",
 name: "Bill",
 age: 35,
 email: "bill@company.com"
 },
 {
 ssn: "555-55-5555",
 name: "Donna",
 age: 32,
 email: "donna@home.org"
 },
];
```

# Checking whether open() request is successful

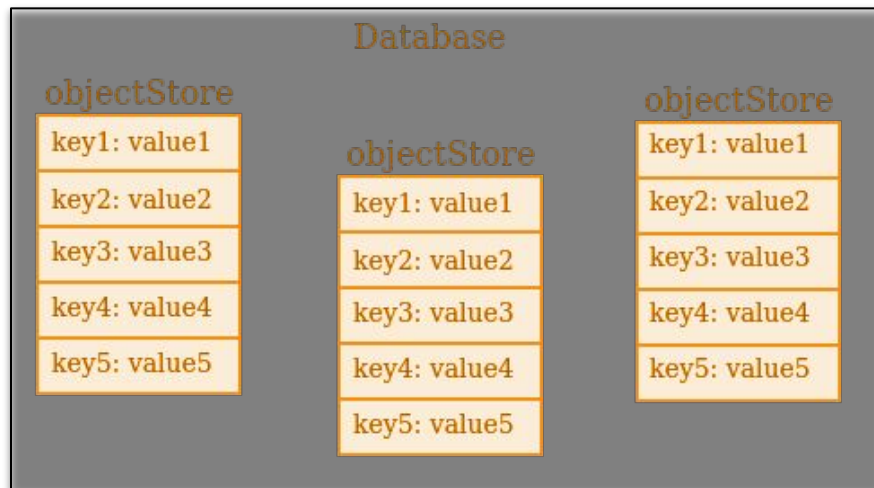
```
let db;
const request = window.indexedDB.open("testDB", 3);
request.onsuccess = function (){
 // Do something with request.result!
 document.write("testDB is created successfully");
}
request.onerror = function (){
 // Do something with request.error!
 document.write("testDB is not created");
}
request.onupgradeneeded = function (event) {
 // // Do something with request.result!
 document.write("testDB is not created");
}
```

# ObjectStore

The data is stored in object stores

We may have multiple object stores in one db

Key is a unique value



<https://javascript.info/indexeddb>

```
db.createObjectStore('students', {keyPath: 'id'});
```

- Create an object store for customerData
  - Object stores are the data storage of IndexedDB
  - `db.createObjectStore("customers", {keyPath: "ssn"});`

## Create an object store in the database.

```
let db;
const request = window.indexedDB.open("testDB", 2);
request.onerror = function (){
 document.write("testDB is not created");
}
request.onupgradeneeded = function (event){
 document.write("testDB is created successfully");
 db = event.target.result;
 // Create an objectStore to hold information about our customers.
 const objectStore = db.createObjectStore("customers", { keyPath: "ssn" });
 // Create an index to search customers by name and email
 objectStore.createIndex("name", "name", { unique: false });
 objectStore.createIndex("email", "email", { unique: true });
}
```

# Database transactions

## A database transaction

symbolizes a **unit of work**, performed against a database

```
const trans1 = db.transaction("foo", "readwrite");
const trans2 = db.transaction("foo", "readwrite");
```

```
const objectStore2 = trans2.objectStore("foo");
const objectStore1 = trans1.objectStore("foo");
```

```
objectStore2.put("2", "key");
objectStore1.put("1", "key");//this writes before trans2
```

```
//you can check whether the transaction is completed
trans2.oncomplete = function(){
 document.write("transactions completed");
 db.close;
};
```



# Add customers

```
let db;
const request = window.indexedDB.open("testDB", 3);
request.onsuccess = function (event){
 db = event.target.result;
 const trans = db.transaction("customers", "readwrite");
 const objectStore = trans.objectStore("customers");
 //adds or error if exist
 objectStore.add(customerData[0]);
 //puts or updates
 objectStore.put(customerData[0]);
 // put each customer to objectstore
 customerData.forEach((customer) => {
 objectStore.put(customer);
 });
}
```

```
const customerData = [
 {
 ssn: "444-44-4444",
 name: "Bill",
 age: 35,
 email: "bill@company.com"
 },
 {
 ssn: "555-55-5555",
 name: "Donna",
 age: 32,
 email: "donna@home.org"
 },
];
```

# Retrieving data

```
<button type="button" onclick="getData()">getdata</button>
```

```
<script>
```

```
...
```

```
function getData() {
 parag = document.getElementById("parag");
 const trans = db.transaction("customers", "readwrite");
 const objectStore = trans.objectStore("customers");

 request = objectStore.get("444-44-4444");
 request.onsuccess = function (event) {
 parag.innerHTML = JSON.stringify(request.result);
 //parag.innerHTML = "email = "+ request.result.email;
 };
}
```

# Remove/delete data

```
<button type="button" onclick="removeData()">removedata</button>
```

```
<script>
```

```
...
```

```
function removeData() {
 parag = document.getElementById("parag");
 const trans = db.transaction("customers", "readwrite");
 const objectStore = trans.objectStore("customers");

 request = objectStore.delete("444-44-4444");
 request.onsuccess = function (event) {
 parag.innerHTML = "444-44-4444 deleted!";
 };
}
```

# server side & backend

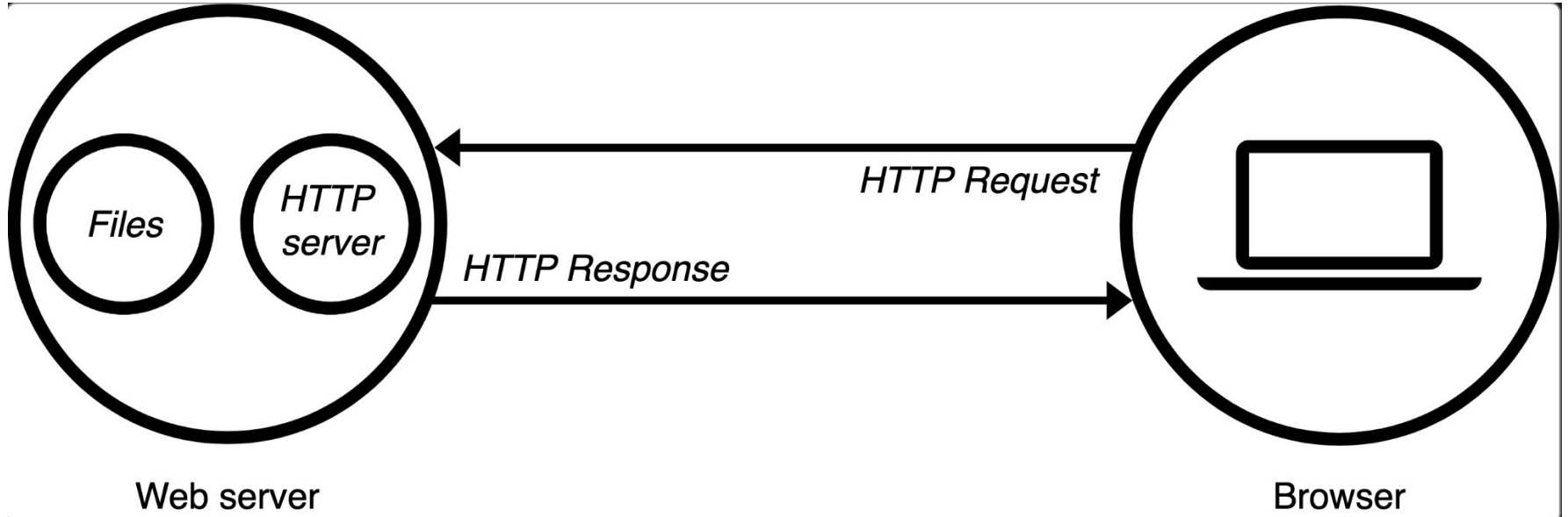
So far we have seen client side:

- Javascript, HTML, CSS
- Client sends/receive **JSON data** from/to server

On server side

- A more comprehensive database management system
  - Relational database or NoSQL database
- A programming language
  - Node.js, Php, Ruby, Java, Python, node.js(javascript) etc..
  - Executed by a webserver and the result is sent to client as HTML

# A web server



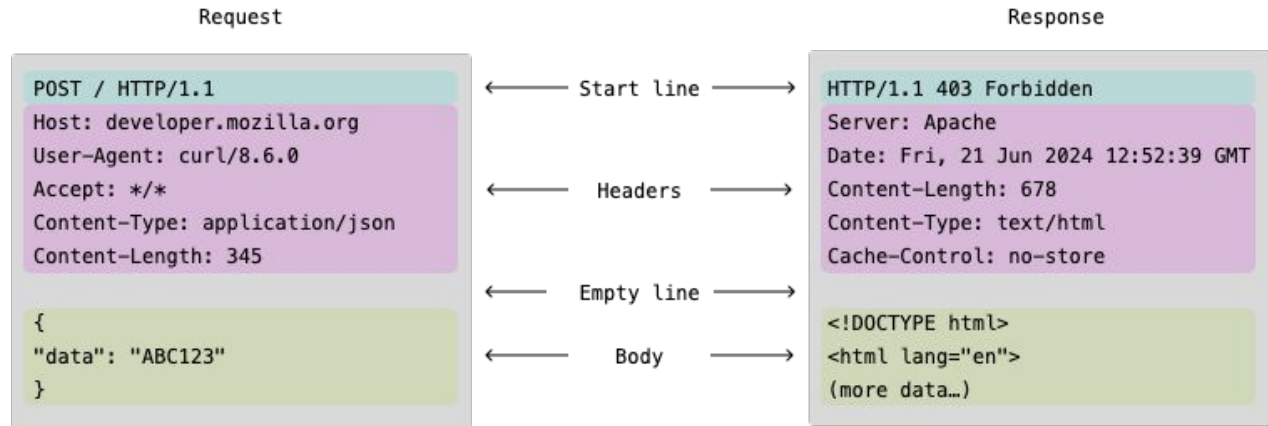
[https://developer.mozilla.org/en-US/docs/Learn/Common\\_questions/Web\\_mechanics/What\\_is\\_a\\_web\\_server](https://developer.mozilla.org/en-US/docs/Learn/Common_questions/Web_mechanics/What_is_a_web_server)

# HTTP

A Protocol is a set of rules for communication between two computers.

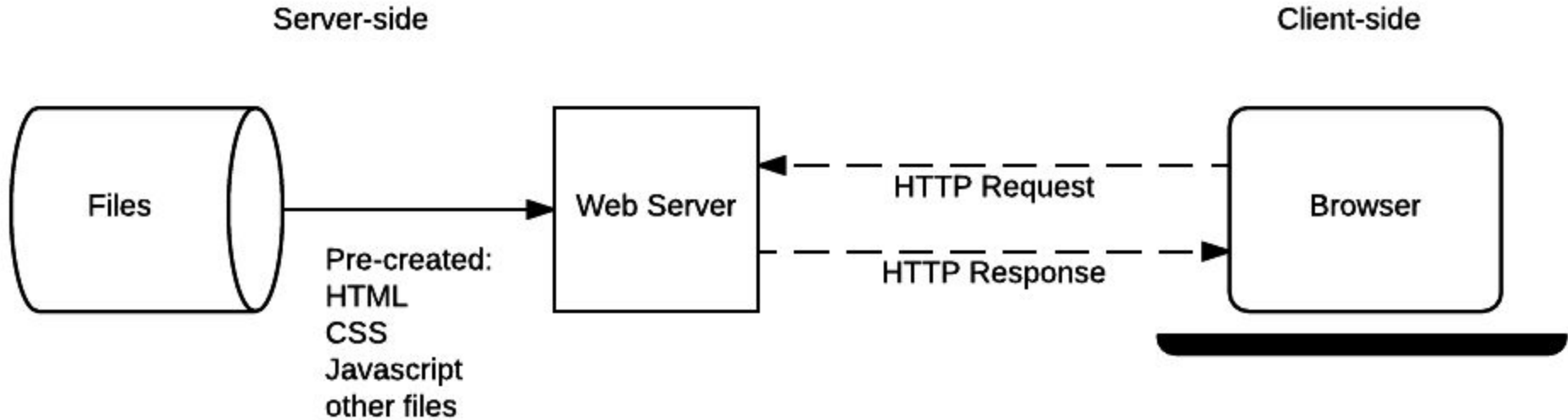
HTTP specifies how to transfer hypertext (linked web documents) between two computers.

example:



<https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages>

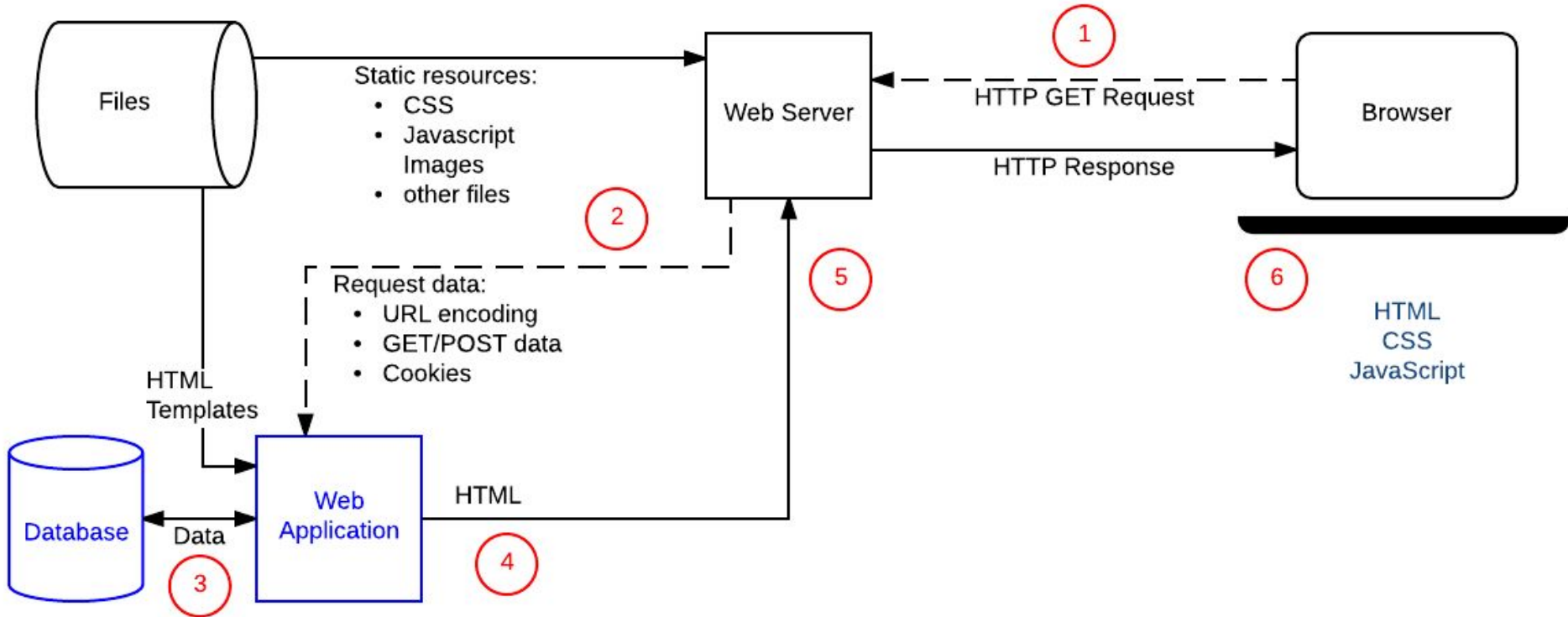
# Static sites



# Dynamic sites

Server-side

Client-side





# Applications of server side scripting

Database Interactions

User Authentication

Form Processing

Content Management Systems

Web Services

More secure

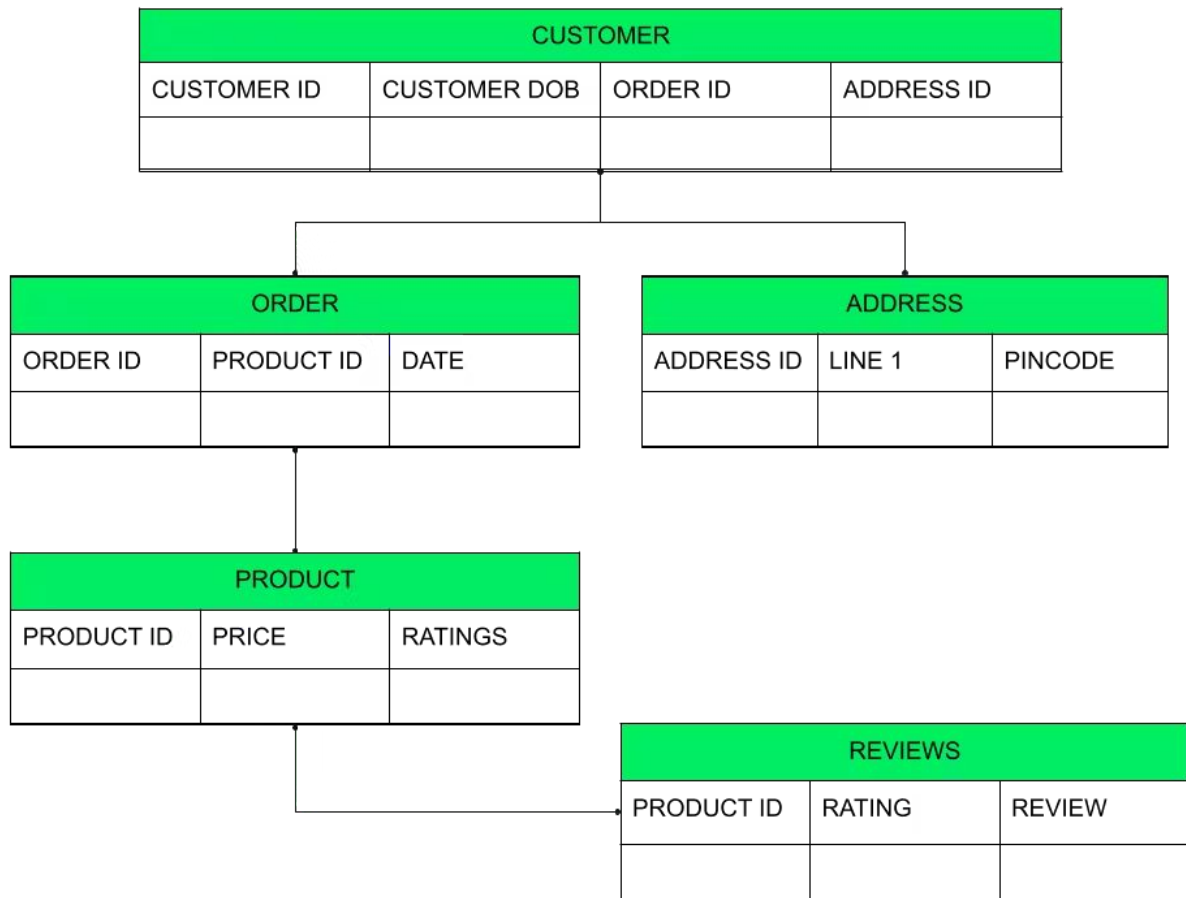
[https://en.wikipedia.org/wiki/Server-side\\_scripting](https://en.wikipedia.org/wiki/Server-side_scripting)

More on databases and SQL

# Relational(SQL) Database

Database schema organizes data in relational, tabular ways

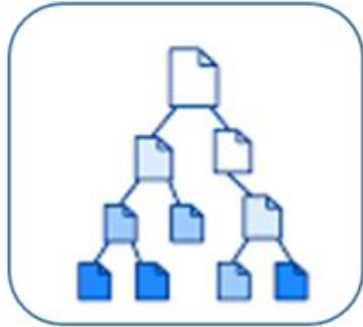
- tables with columns or attributes and rows of records.
- A strictly predefined schema
  - You need to structure data before starting...
- Examples
  - MySQL
  - PostgreSQL
  - Oracle
  - MS SQL Server



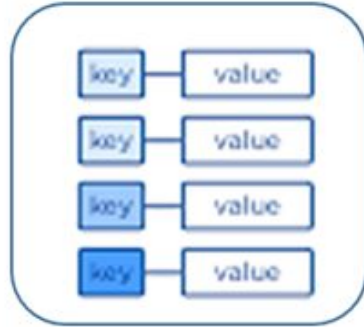
# NoSQL Database

They are not structured as tables with relations

More flexible



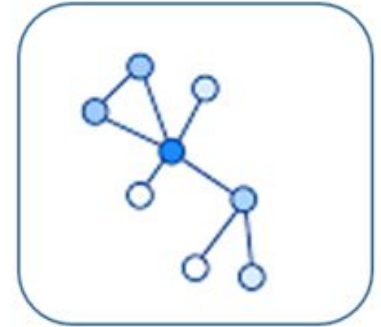
Document  
Store



Key-Value  
Store



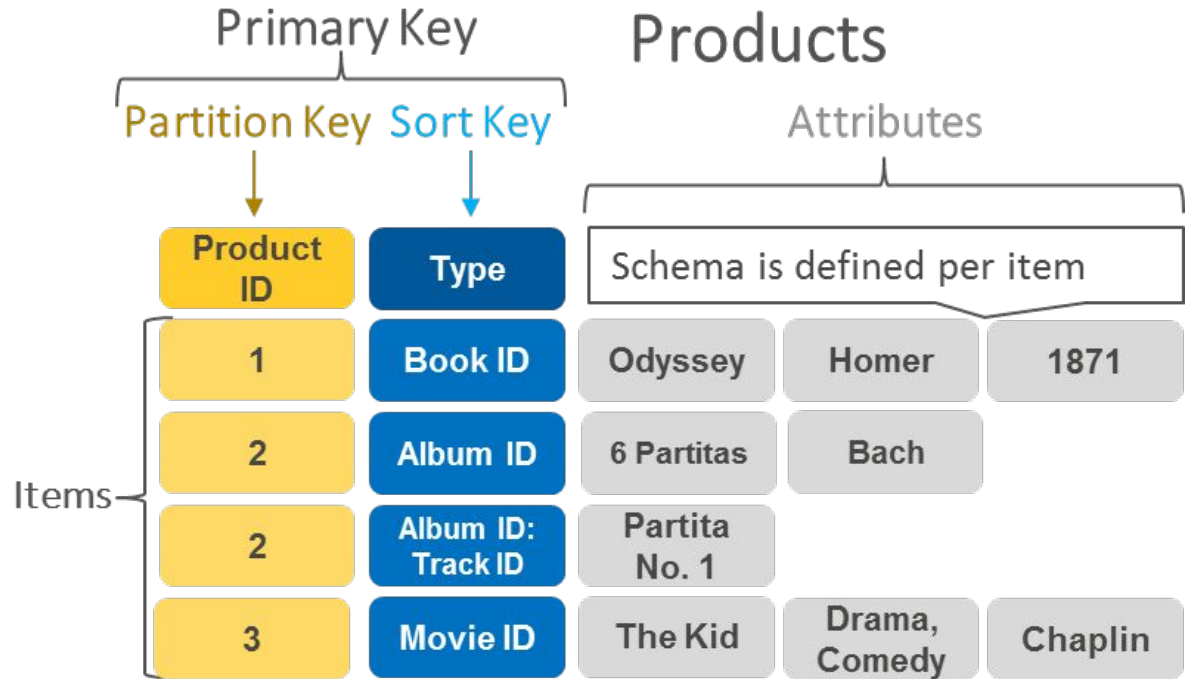
Wide-Column  
Store



Graph  
Store

<https://learn.microsoft.com/en-us/dotnet/architecture/cloud-native/relational-vs-nosql-data>

# NoSQL: Key-Value



<https://aws.amazon.com/compare/the-difference-between-relational-and-non-relational-databases/>

## NoSQL: Document {

XML, YAML, JSON.. etc

```
company_name: "AnyCompany",
address: {street: "1212 Main Street", city: "Anytown"},
phone_number: "1-800-555-0101",
industry: ["food processing", "appliances"]
type: "private",
number_of_employees: 987
}
```

<https://aws.amazon.com/compare/the-difference-between-relational-and-non-relational-databases/>



## Social Network

- Represents **friendships** between people

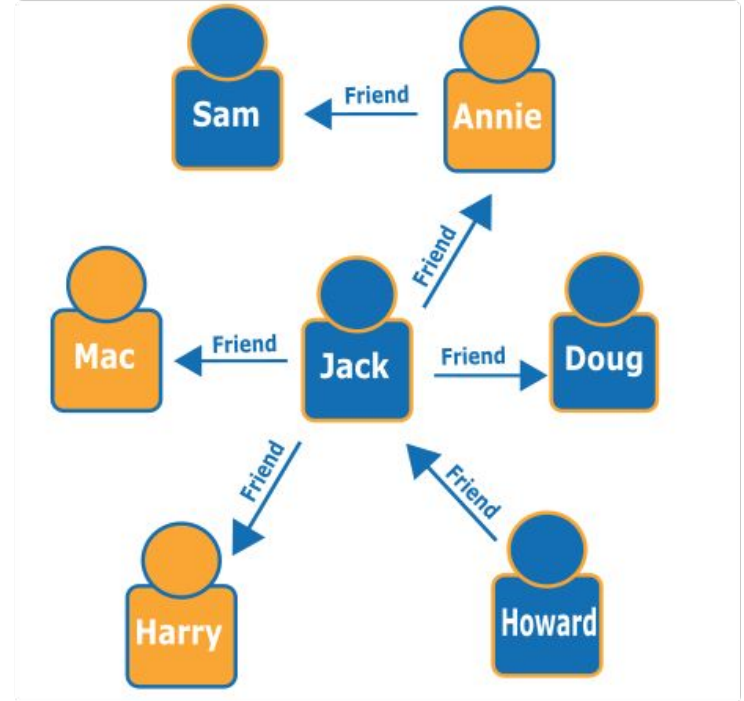
In CS, it is called **Graph**

- Nodes (vertices) = people
- Connections (Edges, links) = friendships

[https://en.wikipedia.org/wiki/Network\\_science](https://en.wikipedia.org/wiki/Network_science)

[https://en.wikipedia.org/wiki/Graph\\_\(discrete\\_mathematics\)](https://en.wikipedia.org/wiki/Graph_(discrete_mathematics))

## NoSQL: Graph



<https://aws.amazon.com/compare/the-difference-between-relational-and-non-relational-databases/>

# Relational database(SQL) vs NoSQL

RDMS must follow ACID

Atomicity: All transactions must succeed or fail completely (**no partial complete**).

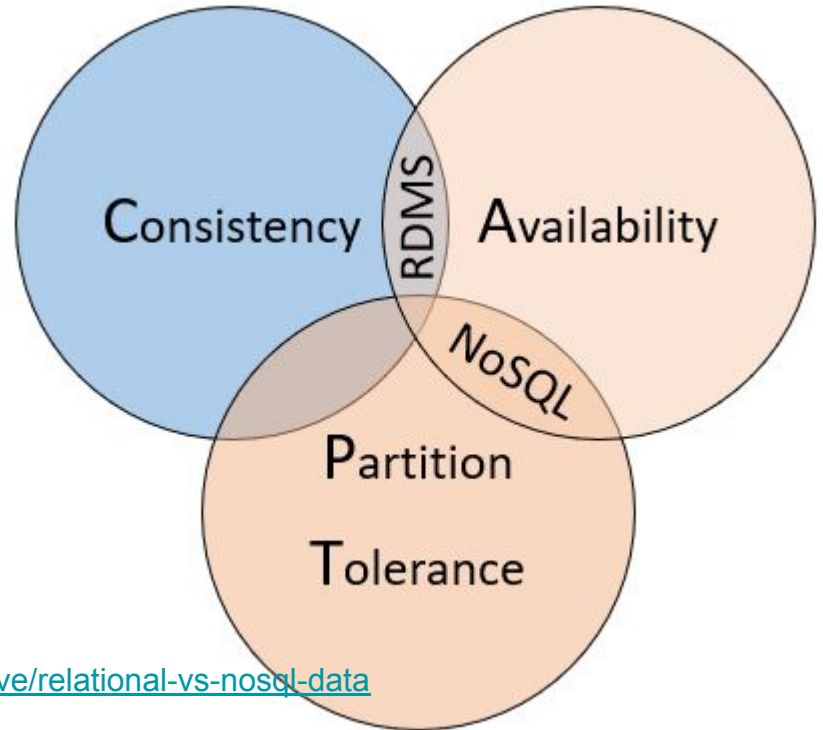
Consistency: The database must follow rules that validate and prevent corruption at every step.

Isolation: Concurrent transactions cannot affect each other.

Durability: Transactions are final, and even system failure cannot “roll back” a complete transaction.

<https://www.ibm.com/think/topics/sql-vs-nosql>

CAP theorem



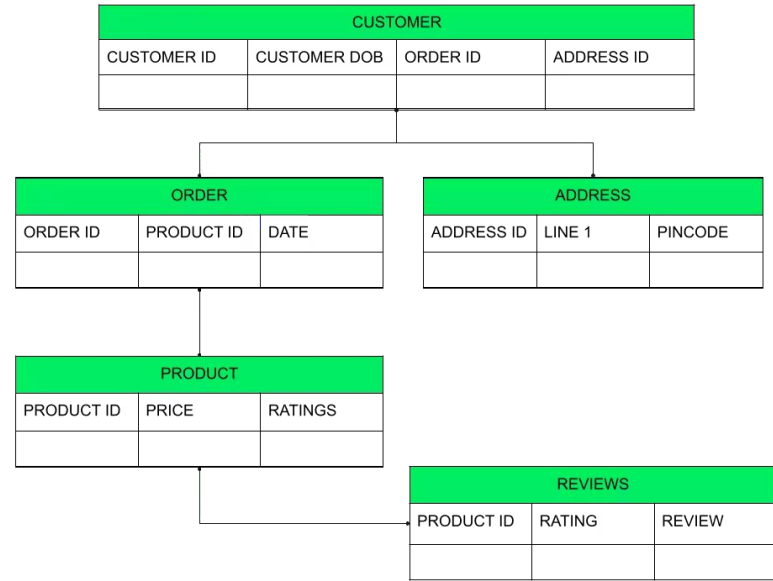
<https://learn.microsoft.com/en-us/dotnet/architecture/cloud-native/relational-vs-nosql-data>



# SQL(Structured Query Language)

SQL is a standard language for storing, manipulating and retrieving data in databases.

You can use in MySQL, SQL Server, MS Access, Oracle, Sybase, Informix, Postgres, and other database systems.



```
sqlite> create table customer(custID INTEGER PRIMARY KEY, name TEXT NOT NULL);
```

```
sqlite> insert into customer values(1, 'aaa');
```

```
sqlite> insert into customer values(2, 'bbb');
```

```
sqlite> select * from customer;
```

```
1|aaa
```

```
2|bbb
```

# Installation

For sql examples used in this class

It is enough to use sqlite: [SQLite Download Page](#)

Then `$ ./sqlite3 test.db`

You can also run from python, C, android

<https://docs.python.org/3/library/sqlite3.html>

<https://www.sqlite.org/c3ref/intro.html>

<https://developer.android.com/training/data-storage/sqlite>

You can also download more complex RDMS

[MySQL](#)

[PostgreSQL](#)

Or use them on azure, google cloud..

# Some important SQL commands

- **SELECT** - extracts data from a database
- **UPDATE** - updates data in a database
- **DELETE** - deletes data from a database
- **INSERT INTO** - inserts new data into a database
- **CREATE DATABASE** - creates a new database
- **ALTER DATABASE** - modifies a database
- **CREATE TABLE** - creates a new table
- **ALTER TABLE** - modifies a table
- **DROP TABLE** - deletes a table
- **CREATE INDEX** - creates an index (search key)
- **DROP INDEX** - deletes an index

SQL keywords are NOT case sensitive: **select** is the same as **SELECT**

[http://www.w3schools.com/sql/sql\\_syntax.asp](http://www.w3schools.com/sql/sql_syntax.asp) Identifiers are case sensitive. ID is not the same as id

```
create database testdb;
```

```
show databases;
```

```
use testdb;
```

```
create table customers(
 id int primary key not null,
 name varchar (20) not null,
 age int not null,
 address varchar (25),
 salary decimal (18, 2)
);
```

# Select more

select

```
select column1, column2, ... from table_name;
```

```
select customer_name, city from customers;
```

```
select * from customers;
```

```
select * from customers where country = 'mexico';
```

```
select * from customers where customer_id = 1;
```

```
select * from customers
```

```
where customer_id > 80;
```

```
select * from customers
```

```
where country = 'germany'
```

```
and city = 'berlin'
```

```
and postal_code > 12000;
```

# Insert into

```
insert into table_name values (value1, value2, value3, ...);
```

```
insert into table_name (column1, column2, column3, ...)
```

```
values (value1, value2, value3, ...);
```

```
insert into customers (customername, contactname, address, city, postalcode, country)
```

```
values ('cardinal', 'tom b. erichsen', 'skagen 21', 'stavanger', '4006', 'norway');
```

[https://www.w3schools.com/sql/sql\\_insert.asp](https://www.w3schools.com/sql/sql_insert.asp)

# Update

```
update table_name
```

```
set column1 = value1, column2 = value2, ... where condition;
```

```
update customers
```

```
set contactname = 'alfred schmidt', city = 'frankfurt'
```

```
where customerid = 1;
```

[https://www.w3schools.com/sql/sql\\_update.asp](https://www.w3schools.com/sql/sql_update.asp)

# Delete from

```
delete from table_name where condition;
```

```
delete from table_name;
```

```
delete from customers where customername='alfreds futterkiste';
```

```
delete from customers;
```

- to delete the table completely, use the **drop table** statement:

```
drop table customers;
```