

# Node-Postgres

---

*RDBMSs, Database Drivers, and Persistent Applications*

# Database Management System (DBMS)

- How to define records (Data Definition)
- CRUD (Create / Read / Update / Delete)
- Performant (B-Trees, etc.)
- Administration
- In short: handle the file system intelligently, without re-inventing the wheel for every project.

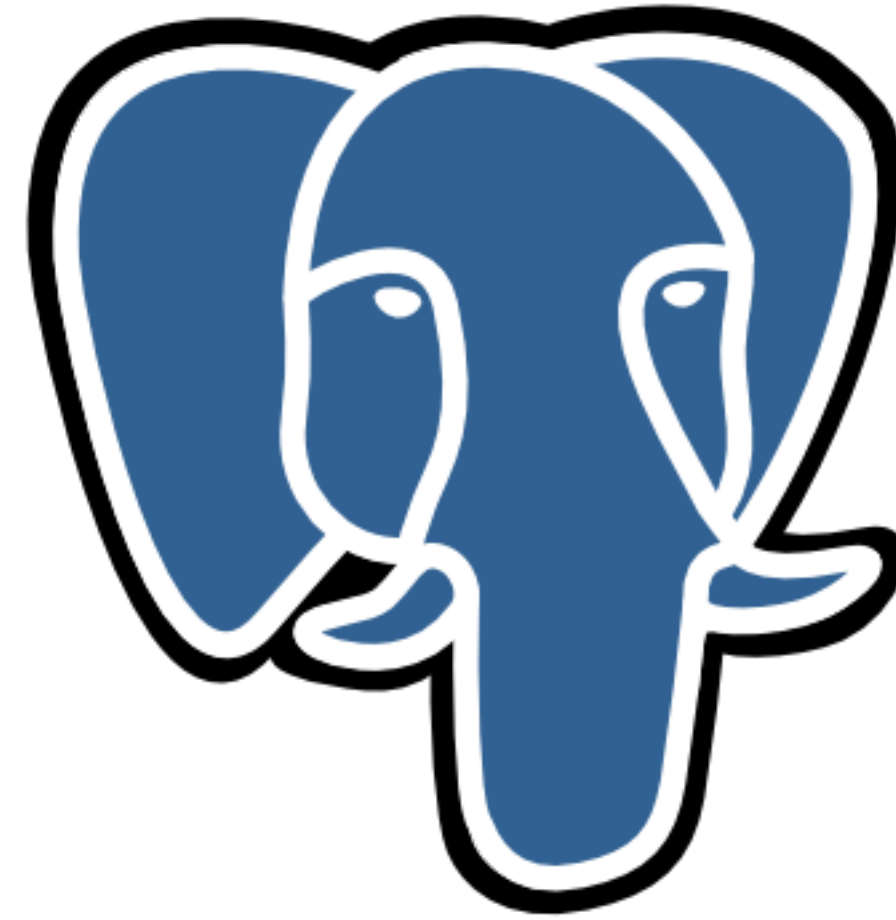
# Progression of Databases

- **Navigational (< 1970s)**
  - More common during tape era; entries had references to next entries.
- **Relational (> 1970s)**
  - Based on relational (table-based) logic, see E.F. Codd.
- **NoSQL (> 2000s)**
  - "Not only SQL" — document storage, for example.



# Some well-known rDBMSs





---

PostgreSQL

# Why PostgreSQL?

- Advanced, powerful, and popular
- Rapid open source development
- Highly extensible (stored procedures)
- Deep SQL standards compliance
- NoSQL ("Not Only SQL"), objective support
- Excellent transactions / ACID reliability; focus on integrity
- Multi-user management / administration

# History of PostgreSQL

- 1970s at UC Berkley:  
**I**Nteractive **G**raphics **R**etrieval **S**ystem (INGRES)
- 1980s: POSTGRES ("Post-Ingres")
- 1995: POSTQUEL and Postgres95.
  - `monitor -> psql`
- 1996: Adopted by the open source community
  - Ongoing: stability, testing, documentation, new features
  - PostgreSQL



# psql

```
1. psql (psql)
X: .../hello-world (zsh) #1 X: .../class-libs (zsh) #2 X: psql (psql) #3 X: postgres (postgres) #4

psql [local]:5432 glebec # ~ \l
List of databases

```

Name	Owner	Encoding	Collate	Ctype	Access privileges
assessmentexpresssequeliza	fullstack	UTF8	en_US.UTF-8	en_US.UTF-8	
author	glebec	UTF8	en_US.UTF-8	en_US.UTF-8	
checkpoint_angular	glebec	UTF8	en_US.UTF-8	en_US.UTF-8	
checkpoint_express_review	glebec	UTF8	en_US.UTF-8	en_US.UTF-8	
glebec	glebec	UTF8	en_US.UTF-8	en_US.UTF-8	
juke	glebec	UTF8	en_US.UTF-8	en_US.UTF-8	
postgres	glebec	UTF8	en_US.UTF-8	en_US.UTF-8	
sequelizecheckpoint	fullstack	UTF8	en_US.UTF-8	en_US.UTF-8	
template0	glebec	UTF8	en_US.UTF-8	en_US.UTF-8	=c/glebec glebec=CTc/glebec
template1	glebec	UTF8	en_US.UTF-8	en_US.UTF-8	=c/glebec glebec=CTc/glebec
tripolanner	glebec	UTF8	en_US.UTF-8	en_US.UTF-8	
twitterdb	glebec	UTF8	en_US.UTF-8	en_US.UTF-8	
wikistack	glebec	UTF8	en_US.UTF-8	en_US.UTF-8	

```
(13 rows)

psql [local]:5432 glebec # ~ \c author
You are now connected to database "author" as user "glebec".

psql [local]:5432 glebec # author \dt
List of relations

```

Schem	Name	Type	Owner
public	stories	table	glebec
public	users	table	glebec

```
(2 rows)

psql [local]:5432 glebec # author
```





# pgcli

```
stayupdated_test> \d
+-----+-----+-----+-----+
| Schema | Name           | Type   | Owner  |
+-----+-----+-----+-----+
| public | admins         | table  | amjith |
| public | cpes           | table  | amjith |
| public | goose_db_version | table  | amjith |
| public | goose_db_version_id_seq | sequence | amjith |
| public | packages       | table  | amjith |
| public | packages_id_seq | sequence | amjith |
| public | users          | table  | amjith |
| public | users_id_seq   | sequence | amjith |
| public | vulnerabilities | table  | amjith |
| public | vulnerabilities_cpes | table  | amjith |
| public | vulnerabilities_id_seq | sequence | amjith |
+-----+-----+-----+-----+
SELECT 11
stayupdated_test> SELECT * FROM users;
+-----+-----+-----+-----+-----+
| id | display_name | password | email           | created_on |
+-----+-----+-----+-----+-----+
| 177 | DisplayName1 | 1024cms  | user@ex.com     | 2014-11-15 15:02:50.094560 |
| 180 | testname2    | pas5w0rd | email@ex.com    | 2014-11-28 10:25:46.170660 |
| 181 | amjith       | password | amjith@amjith.amjith | 2014-11-28 18:39:48.195067 |
+-----+-----+-----+-----+-----+
SELECT 3
stayupdated_test> SELECT * FROM
| admins
| cpes
| goose_db_version
| packages
| users
```

# Postico

The screenshot shows the Postico application window. At the top, there's a navigation bar with tabs for 'Reporting', 'reporting', and 'forex'. The 'forex' tab is selected. To the right of the tabs, it says 'Connected.' and 'PostgreSQL 9.4.5'. Below the navigation bar, there's a sidebar on the left with a list of tables: 'SQL Query', 'currencies', 'daily\_proceeds', 'forex' (selected), 'forex\_sources', 'fs\_daily\_proceeds', 'fs\_daily\_sales', 'fs\_orders', 'fs\_proceeds\_weekly', 'fs\_weekly\_sales', 'itc\_daily\_proceeds', 'itc\_daily\_sales', 'itc\_proceeds\_weekly', 'itc\_reports', 'itc\_reports\_daily', 'itc\_reports\_monthly', and 'itc\_reports\_weekly'. The main area displays a table with the following columns: 'currency', 'base', 'rate', 'date', and 'source\_id'. The table contains 15 rows of data, all for the date '2014-07-25'. The first three rows are highlighted in blue. To the right of the table, there are filters for 'currency' (set to 'MULTIPLE'), 'base' (set to 'USD'), 'rate' (set to 'MULTIPLE'), and 'date' (set to '2014-07-25'). Below these filters, there's a section for 'source\_id' with a dropdown menu showing '1' and a table with columns 'id' and 'name' containing the value '1' and the name 'Open Exchange R...'. At the bottom of the window, there's a search bar and a pagination bar showing 'Page 1 of 342'.

currency	base	rate	date	source_id
AED	USD	3.67291	2014-07-25	1
AFN	USD	56.485726	2014-07-25	1
ALL	USD	103.5838	2014-07-25	1
AMD	USD	410.086	2014-07-25	1
ANG	USD	1.787	2014-07-25	1
AOA	USD	96.952626	2014-07-25	1
ARS	USD	8.169642	2014-07-25	1
AUD	USD	1.062844	2014-07-25	1
AWG	USD	1.79	2014-07-25	1
AZN	USD	0.784067	2014-07-25	1
BAM	USD	1.453024	2014-07-25	1
BBD	USD	2	2014-07-25	1
BDT	USD	77.61971	2014-07-25	1
BGN	USD	1.452543	2014-07-25	1

# Datazenit

The screenshot shows the Datazenit App interface. At the top, there's a navigation bar with tabs for 'postgresql', 'hiburo.com', 'datazenit', and 'local.dev'. Below this, the main interface is divided into several sections. On the left, there's a sidebar with a terminal-like view showing 'root@localhost:3306 | Edit' and a list of tables including 'categories', 'content', 'cust\_hist', and 'emails'. The 'content' table is selected, and its columns are listed: id, title, body, user\_id, created\_at, body, created\_at\_date, foo\_bar, DBL, and amount. The main area is divided into tabs for 'Filter', 'Query', 'Schema', and 'Backups'. The 'Query' tab is active, showing a SQL query: 'Select id, title, body, user\_id, created\_at From content'. Below the query, there are input fields for 'Limit' (20) and 'Offset' (0). At the bottom of the query section, there are buttons for 'Add', 'Condition', 'Sorting', 'History', 'Favorites', 'Save', 'Export', and 'Execute'. Below the query section, there's a 'Content' tab and a 'Visualization' tab. The 'Content' tab is active, showing a table with 7 rows. The first row is highlighted in yellow, and the second, third, and fourth rows are also highlighted in yellow. The table has columns: id, title, body, user\_id, and created\_at. The data is as follows:

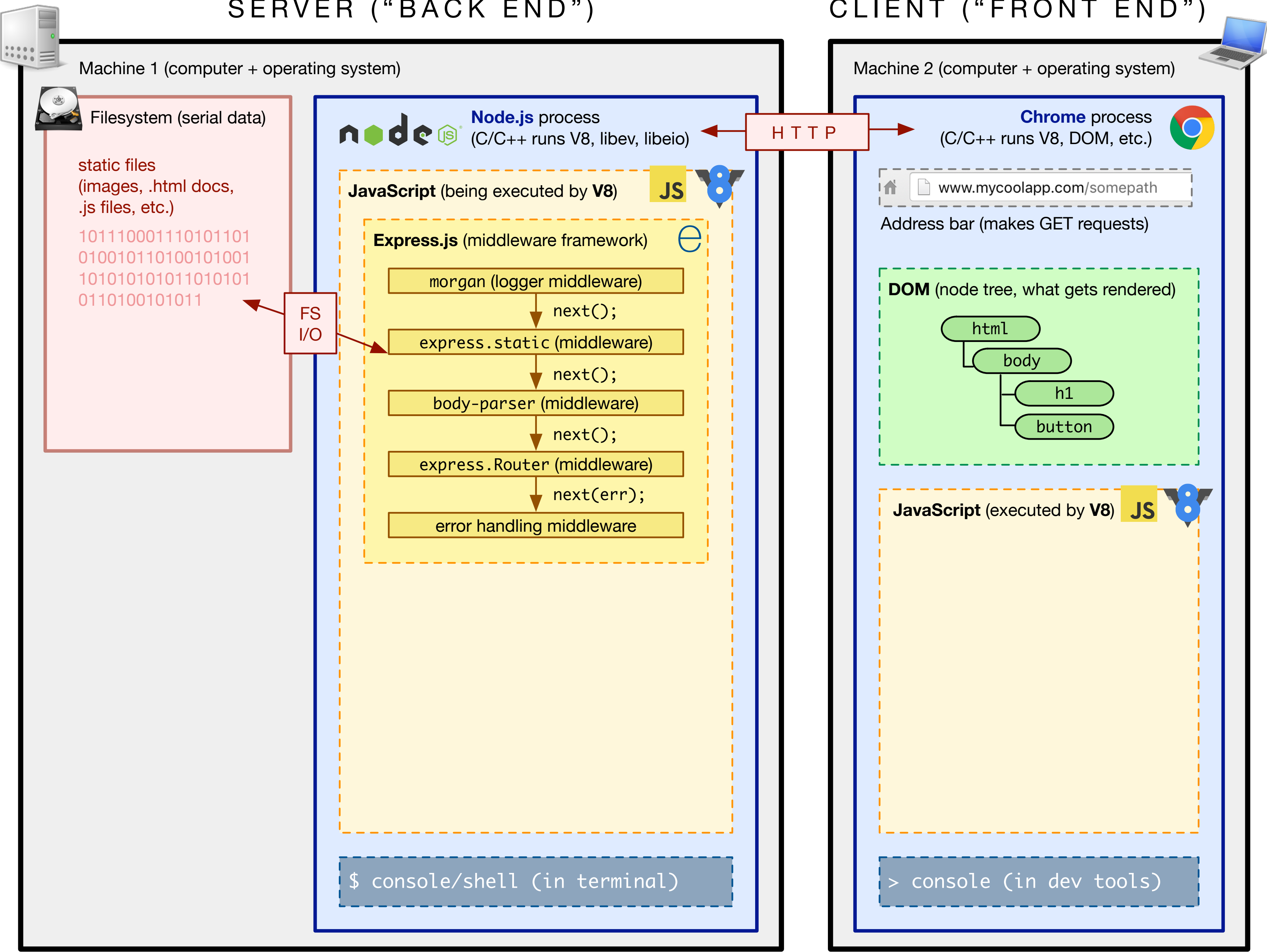
	id	title	body	user_id	created_at
<input type="checkbox"/>	1	Optional actuating utilisation	Inventore quia et et eius. P...	535868	1983-11-11 00:01:22
<input checked="" type="checkbox"/>	2	Expanded dynamic superst...	Corporis et atque qui haru...	949966	1974-04-11 22:10:36
<input checked="" type="checkbox"/>	3	Fully-configurable real-time...	Eaque officia natus porro v...	44126916	2014-10-05 03:29:17
<input checked="" type="checkbox"/>	4	ineered dynamic architecture	Consequatur cumque duci...	0	2012-04-01 05:43:33
<input type="checkbox"/>	5	Customizable heuristic firm...	Est ipsa nobis et omnis. Qu...	89	1992-04-07 15:19:49
<input type="checkbox"/>	6	Total explicit collaboration	Ratione similique omnis au...	8	1978-01-24 10:44:35
<input type="checkbox"/>	7	Facetoface dynamic succe...	Nihil saepe reiciendis labor...	68030159	2006-12-26 20:38:34

At the bottom of the table, there's a 'Delete' button and a status bar showing 'Rows returned: 20' and 'Rows selected: 3'. A tip at the bottom right says 'Tip: press cmd+a to select/deselect all rows'.



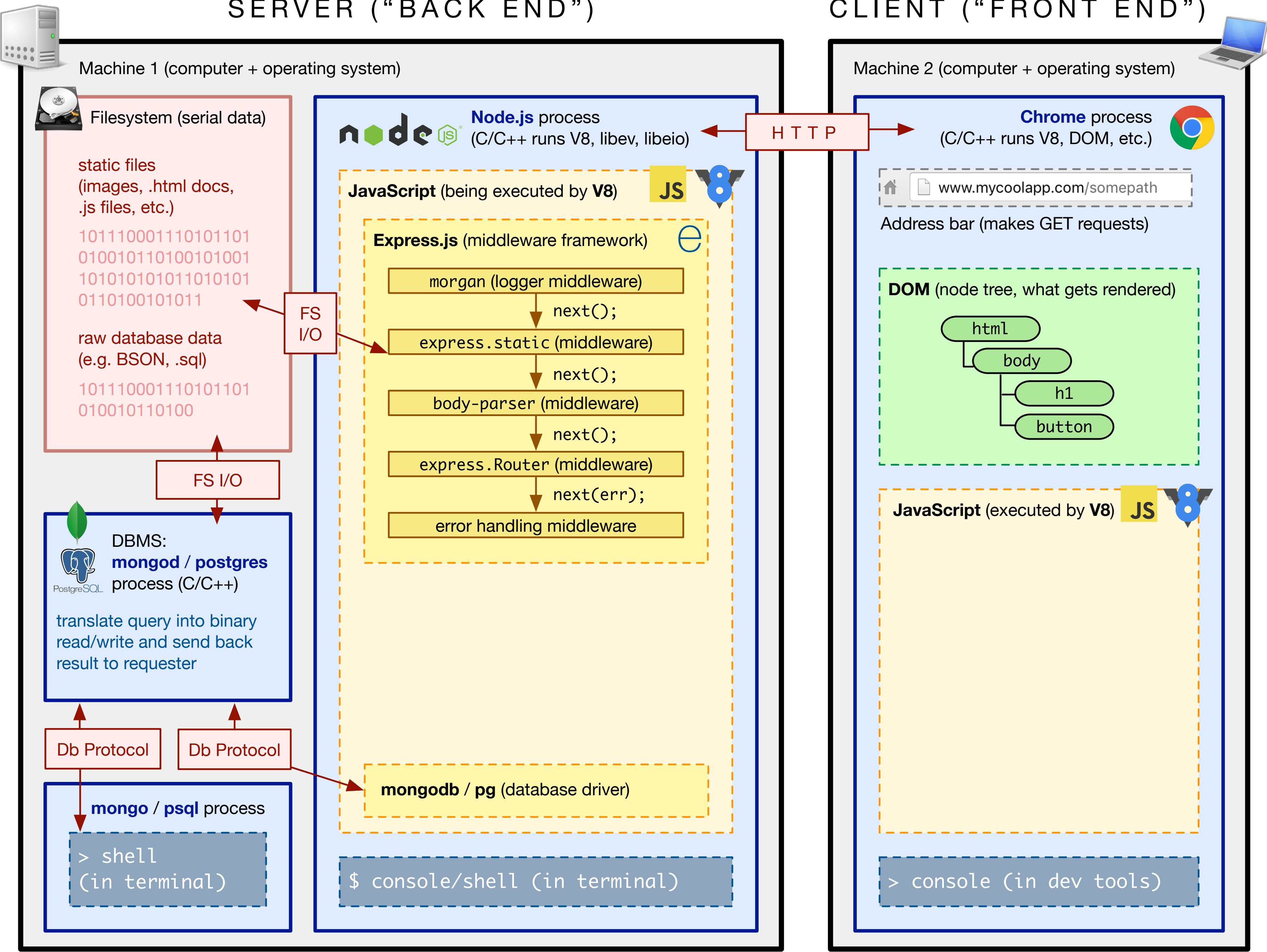
SERVER ("BACK END")

CLIENT ("FRONT END")



SERVER ("BACK END")

CLIENT ("FRONT END")



# postgres process



- The rDBMS itself; a *daemon* (background process)
- Waits for incoming SQL
- Knows how to read/write to disk in a performant way
- Sends back results

**Where does the "incoming  
SQL" come from?**



# Query Sources ("Clients")

- **psql CLI**
  - human input as text
- **GUI like Postico, Datazenit**
  - human actions turned into SQL queries
- **...and other applications**
  - "somehow" communicate with the postgres process

***How to transmit SQL text to app?***  
**How can postgres be "waiting for SQL"?**  
**And how do the results get "sent back"?**

# Postgres is a TCP server!



- Listening on a TCP port (5432 by default) for *requests*
- Does disk access
- Sends back a TCP *response* to the *client* that made the requests

**OK, Postgres is a TCP server.  
Is it... HTTP?**

# Postgres uses the postgres:// protocol

	Transport Protocol	Message Protocol	Content Type
Node + Express	TCP/IP	http://	Anything: HTML, JSON, XML, TXT, etc.
Postgres	TCP/IP	postgres://	SQL

**For HTTP clients, the TCP/IP was handled for you by the browser or Node. How can our JS app communicate with the postgres server?**

*“Let's implement the postgres protocol in  
JavaScript ourselves!”*

– AMBITIOUS MCOVERKILL



<https://www.postgresql.org/docs/current/static/protocol.html>

*“On second thought...  
has anyone done this for us?”*

– SANEY MCREASONABLE

# Node-postgres

- **npm library:** `npm install pg --save`
- *database driver*
- implements the postgres protocol in a Node module (JS!)
- Gives us a `client` object that we can pass SQL to
- Asynchronously talks via postgres protocol / TCP to postgres
- gives us a callback with `rows` array of resulting table

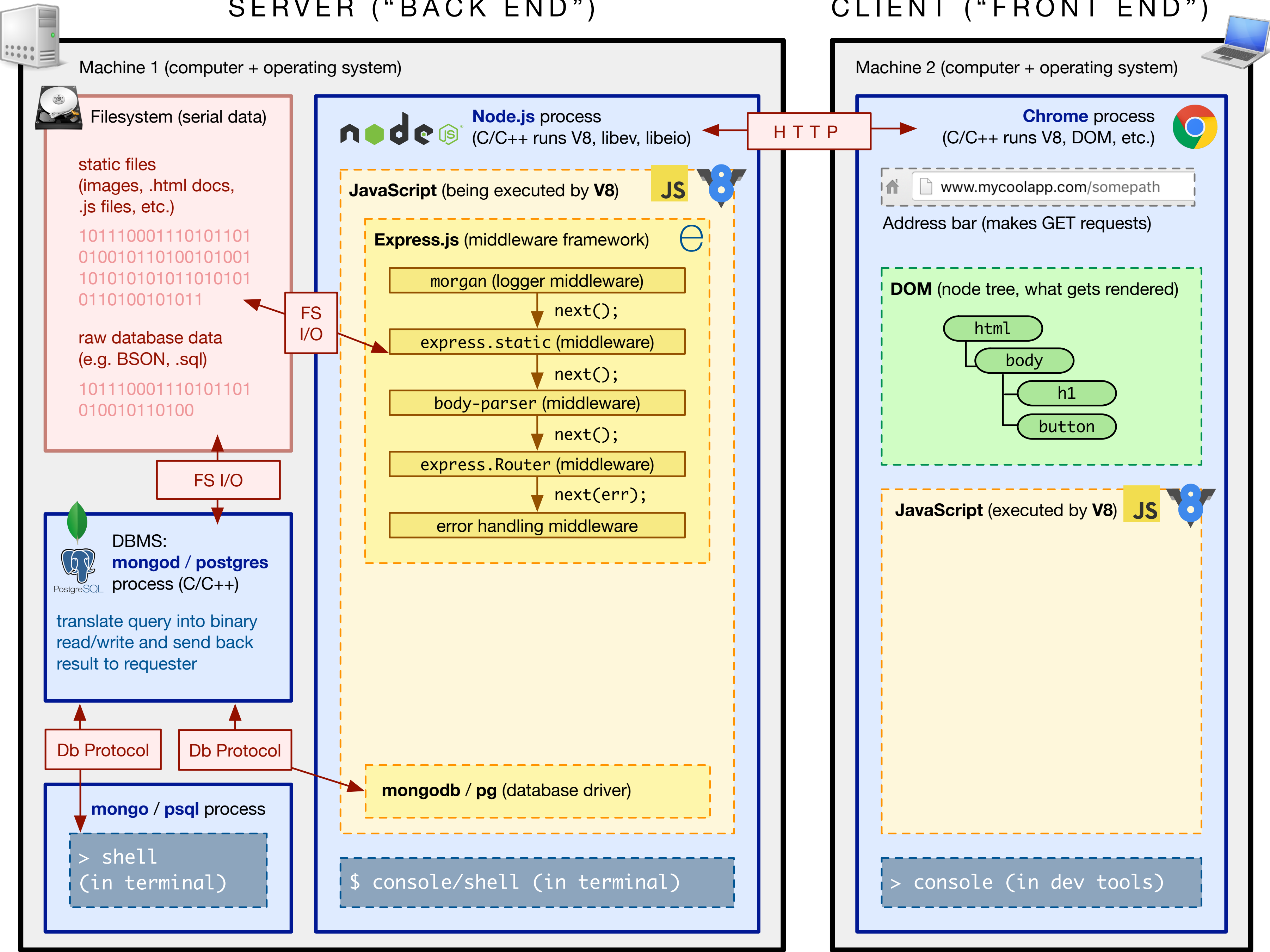


# Example

```
client.query('SELECT * FROM users', function (err, data) {  
  if (err) return console.error(err);  
  data.rows.forEach(function (rowObject) {  
    console.log(rowObject); // { name: 'Claire' }  
  });  
});
```

SERVER ("BACK END")

CLIENT ("FRONT END")



# Final note: `returning`

- SQL comes in slightly different "dialects" depending on your RDBMS of choice (SQLite, MySQL, PostgreSQL etc.)
- PostgreSQL has a very convenient keyword `returning`
- Used during INSERT, UPDATE
- Returns the row(s) inserted/updated
- May come in handy during workshop, so check it out!