Persistent Storage with Databases

Recapped CRUD – different syntax in different languages but the basic type is the same

Create – POST in http
Read - GET in http
Update - PUT in http
Delete - DELETE in http

Database icon is normally a barrel…
Databases were originally big barrels of mercury! And would only hold a few hundred words.
Then moved on to cylinders with lots of magnetic disks (like massive floppy disks)

A database stores information in a structured way that allows us to access it programmatically
There are other ways of storing data: data lakes, data buckets etc.

How are databases relevant in real life?

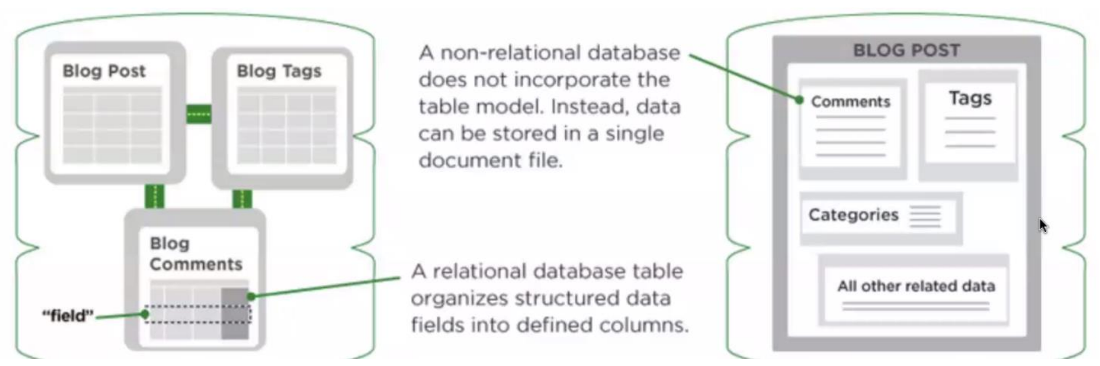
You access them every time you access a website – e.g., all Tweets for Twitter will be stored in a database
Or when you access an API etc.

Databases are not stored on Servers

- Servers are just the communication conduit between the client and the database.
- The server sends on the query to the database – similar to how we separated the actual GET request in to the Models folder – the database handles the data.
- Then the server gets data back from database and serves it back to the client!
- This maintains a "separation of concerns" – each bit of the architecture just does its own job!
- But servers *could* hold data if needed

There is no cloud…just someone else's computer! (that's all the cloud is)

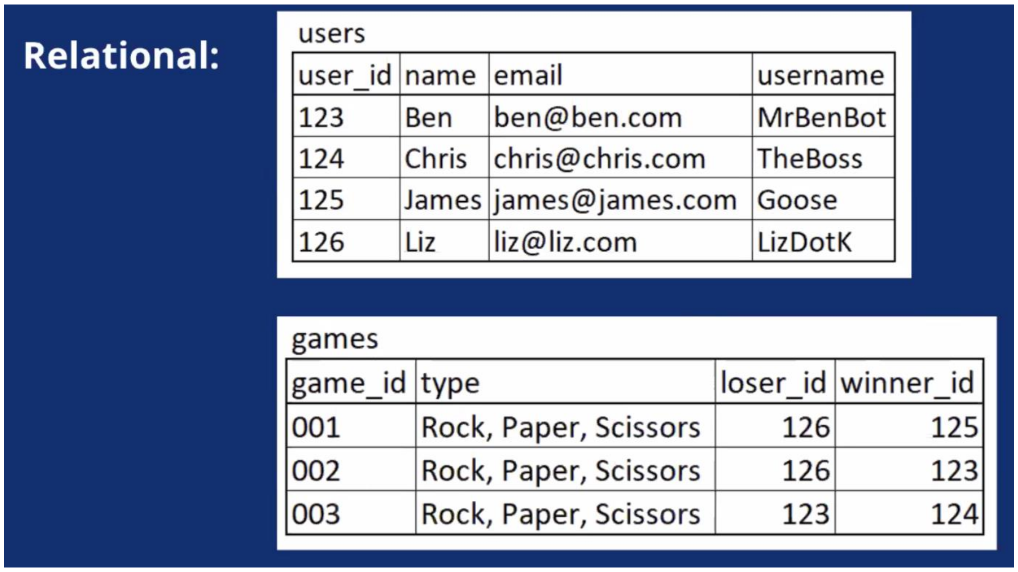
2 main types of databases: Relational vs. Non-relational


Blog Post
Blog Tags
Blog Comments
"field"
A non-relational database does not incorporate the table model. Instead, data can be stored in a single document file.
A relational database table organizes structured data fields into defined columns.
BLOG POST
Comments
Tags
Categories
All other related data

Always trade-offs, neither better than the other! Horses for courses – depends on the data you want to store (and what the company is already using!

Relational – a bit like a spreadsheet, tables and columns and data is linked. Advantages – you know exactly how the data is shaped, you can query it programmatically and you can join each table very easily. Trade-offs – very rigid! You specify each column right down to the data type for that column.

e.g. In the example below you could easily find out who won game 3!



Relational:

users

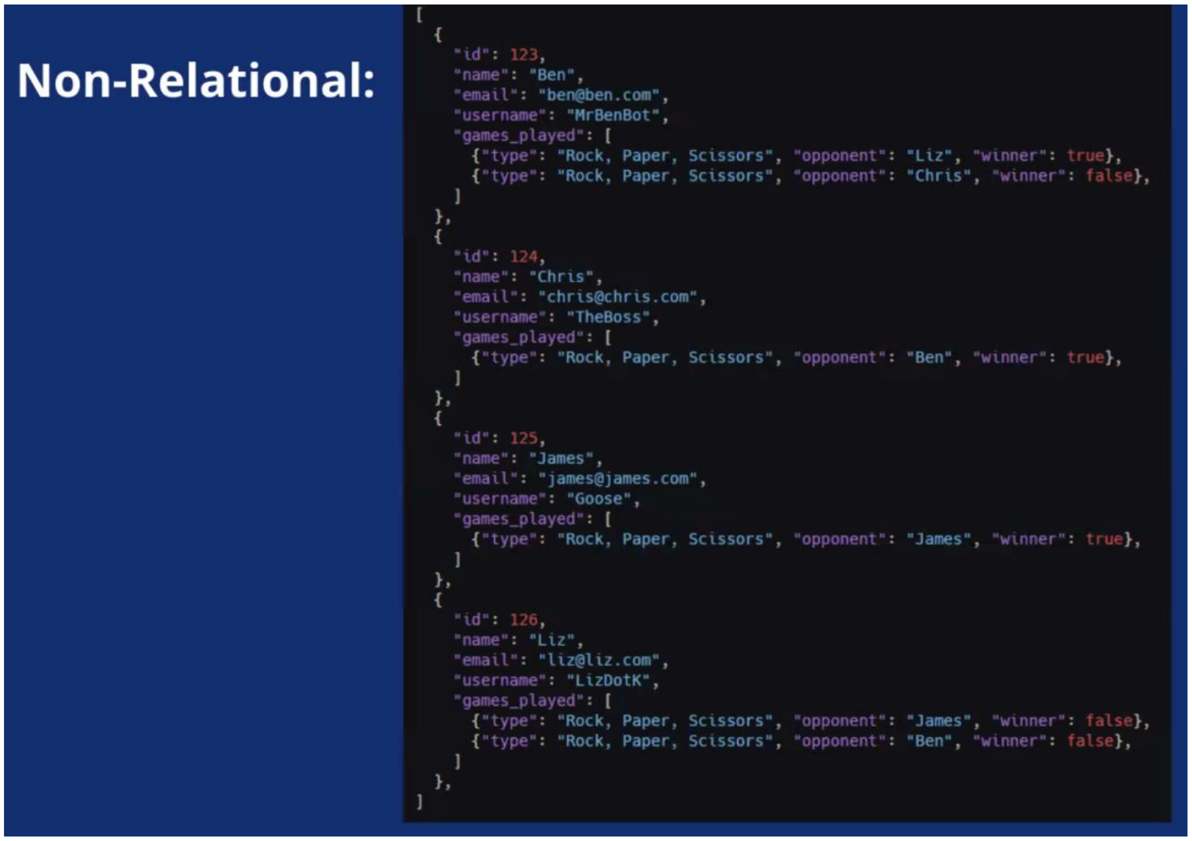
user_id	name	email	username
123	Ben	ben@ben.com	MrBenBot
124	Chris	chris@chris.com	TheBoss
125	James	james@james.com	Goose
126	Liz	liz@liz.com	LizDotK

games

game_id	type	loser_id	winner_id
001	Rock, Paper, Scissors	126	125
002	Rock, Paper, Scissors	126	123
003	Rock, Paper, Scissors	123	124

Non-relational – Data can be stored in documents, or key: value pairs for example. Advantages – you can add more data and data types more freely. Trade-off – without a defined structure, joining different elements and data is difficult and hard to access it with a programmatic query.

e.g. Similar data in this example but it's harder to even tell who won which game, but more flexible as you could add a new key of "Is employed by SOC" and easy to add to all



Non-Relational:

```
[
  {
    "id": 123,
    "name": "Ben",
    "email": "ben@ben.com",
    "username": "MrBenBot",
    "games_played": [
      {"type": "Rock, Paper, Scissors", "opponent": "Liz", "winner": true},
      {"type": "Rock, Paper, Scissors", "opponent": "Chris", "winner": false},
    ]
  },
  {
    "id": 124,
    "name": "Chris",
    "email": "chris@chris.com",
    "username": "TheBoss",
    "games_played": [
      {"type": "Rock, Paper, Scissors", "opponent": "Ben", "winner": true},
    ]
  },
  {
    "id": 125,
    "name": "James",
    "email": "james@james.com",
    "username": "Goose",
    "games_played": [
      {"type": "Rock, Paper, Scissors", "opponent": "James", "winner": true},
    ]
  },
  {
    "id": 126,
    "name": "Liz",
    "email": "liz@liz.com",
    "username": "LizDotK",
    "games_played": [
      {"type": "Rock, Paper, Scissors", "opponent": "James", "winner": false},
      {"type": "Rock, Paper, Scissors", "opponent": "Ben", "winner": false},
    ]
  },
]
```

Lots of different providers of databases:

Relational:
Non-Relational:
MySQL
Amazon RDS
Microsoft SQL Server
ORACLE
mongoDB
Amazon DynamoDB
Amazon DocumentDB
Couchbase
redis

<u>We will be using PostgreSQL</u>

- SQL is <u>relational</u>

- Stands for **Structured Query Language**
- Was invented by IBM in the 70s!
- SQL is like ice cream – it has different flavours! But with similarities between them
- PostgreSQL in one of these flavours invented in 1996
- We will be using it because it's free, open-source and widely used in industry

<u>db-fiddle.com</u>

- Online playground where you can mock up some Database queries
- Make sure top left says PostgreSQL v13
- Schema SQL is how we're setting up the database
- Run your queries in the Query SQL
- Data doesn't persist

SQL query can run in all lowercase, but set the STATEMENT and the LOCATION in caps to make it more readable
YOU MUST FINISH END OF QUERY WITH SEMICOLON; (unlike JS where you can get away with it)

<u>SELECT Statement</u>

This is READ from the CRUD (a.k.a GET)

SELECT * FROM users; ** = everything, users = the table of users in our DB*

SELECT user_id FROM users; *Select just the user_id column from the table*

SELECT user_id, username FROM users; *Selects the user_id and username columns from the table,* ***COMMA is separator***

<u>WHERE Clause</u>

Still READ from the CRUD (a.k.a GET by ID)

Above code is for selecting whole columns. This allows us to select specific rows:

SELECT * FROM users WHERE user_id = 126; *Select all columns, in the row with user_id 126*

SELECT email FROM users WHERE user_id = 126; *Select the email column, in the row with user_id 126*

In SQL "=" is back to equals (not assignment operator like in JS!)

<u>INSERT Statement</u>

CREATE from CRUD (a.k.a POST in http)

INSERT INTO users (name, email, username) VALUES (‘Tao’, ‘tao@tao.com’, ‘BegoniaFan’);

Defines the columns you are populating then passes what those values should be.

But this doesn’t show you the result!

So to ensure users are told if it’s successful, add RETURNING * at the end. This gives you all the columns you’ve just inserted, not ALL the columns that were already there

INSERT INTO users (name, email, username) VALUES (‘Tao’, ‘tao@tao.com’, ‘BegoniaFan’) RETURNING *;

Must use single quotes, not double quotes!
SQL will automatically assign a unique ID to whatever property is set as primary key and “SERIAL” type.

UPDATE Statement

This is UPDATE from CRUD (a.k.a. PUT in http)

UPDATE users SET username = ‘Ta-yoyo’ WHERE user_id = 127;

Updates the users table, setting the username column to Ta-yoyo in Row where the user_id is 127.
If you didn’t specify WHERE – it will update the WHOLE COLUMN!!!!

DELETE Statement

This is DELETE from CRUD (a.k.a. DELETE in http)

DELETE FROM users WHERE user_id = 124;

Deletes any rows where user_id is 124 from the Users table. Does not change any other rows, so user_id 124 will not be reused, even if you INSERT INTO new data.