Relational and Non-Relational Databases

Dylan Vidal

Slides adapted from Nick LaMontagna

Overview

- Why do we use Databases
- Comparing NoSQL vs SQL
- What is a Non-Relational Database
- What is a Relational Database
- SQL: CRUD & Conditionals
- SQL: Primary and Foreign Keys
- SQL: Joins

Why do we use Databases

Why do we use Databases?

What if we just used one big json…

Why do we use Databases

- Data Persistence
- Data Synchronization between Clients
- Large amounts of Data
- Optimized Data Operations (CRUD)

Why do we use Databases

Let's Design a Minecraft Server!

Why do we use Databases

Ground Rules

- You are playing a Minecraft server that has 10,000 concurrent players
- Each month that server will have 1,000,000 monthly active users (MAU)
- Each player has progression via mods that they need to store, and mods for the server (plugins) need to save data.
- You want to store some things
    - In-game currency
    - Unlocked perks (bonuses you get from playing the game)
    - Friends list (alert you when your friends are online)
    - Factions
        - Who is in what faction
        - Where is that faction
        - Who owns the faction

What is the best way to store all of the information above…

Why do we use Databases

```
{
    "type": "message",
    "channel": "C2147483705",
    "user": "U2147483697",
    "text": "Hello world",
    "ts": "1355517523.000005",
    "is_starred": true,
    "pinned_to": ["C024BE7LT", ...],
    "reactions": [
        {
            "name": "astonished",
            "count": 3,
            "users": [ "U1", "U2", "U3" ]
        },
        {
            "name": "facepalm",
            "count": 1034,
            "users": [ "U1", "U2", "U3", "U4", "U5"
]
        }
```

… Nah

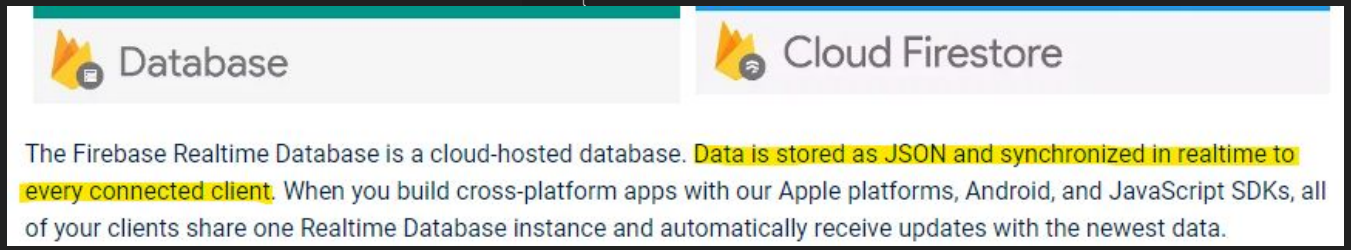
Why do we use Databases: NoSQL

Actually…

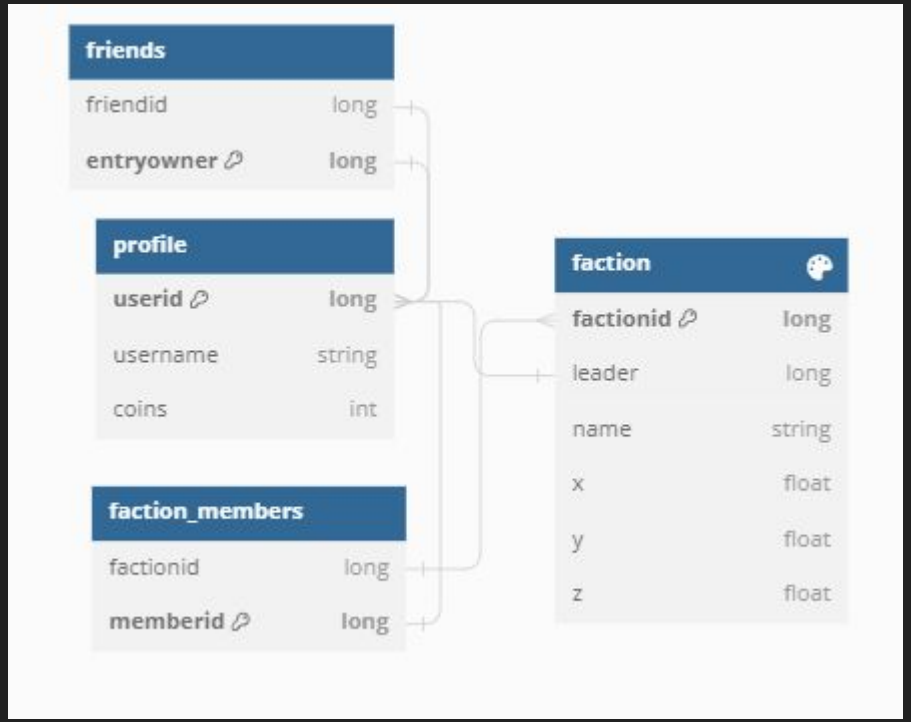
```
{
    "userid" : 1234,
    "username" : "nickrocky213",
    "coins" : 1000,
    "friends" : {
        "username" : "knightro",
        "username" : "floatbob",
        "username" : "mirage.tsx"
    },
    "faction" : "knighthacks"
}
```

```
{
    "factionid" : 22334455,
    "leader" : "nickrocky213",
    "name" : "knighthacks",
    "x" : 1,
    "y" : 2,
    "z" : 3,
    "members" : {
        "member" : "floatbob",
        "member" : "altmed",
        "member" : "mbucket",
        "member" : "mirage.tsx",
        ... x 300 more people
```

While it may seem like this is a poor idea, that should never be approached, it really depends on the use case.



Why do we use Databases: SQL



- We can ensure that all of our user data can be stored in this schema
- This storage solution should also scale fairly well for this use case

Which do we use?

It realistically depends, in this case the SQL solution is a better way to solve this problem.

Comparing NoSQL vs SQL

So what is faster?

Comparing NoSQL vs SQL

They largely are equal, both having a ton of pros and cons performance wise.

Comparing NoSQL vs SQL:

SQL

Pros

- ACID (Atomicity, Consistency, Isolation, Durability)
  - *Basically things that ensure the data stays valid in the event of errors, power failure, etc*
- Simple Keyword Queries
- Large long standing community for support
- Easier to self host
- High data integrity and security

Cons

- Scales vertically not horizontally (most of the time)
- Slows down as a DB gets bigger (billions-trillions of rows)
- Requires more planning

NoSQL

Pros

- About as fast as a SQL query for small data sets (1 billion and less), but faster than SQL for large data sets
- Stable and distributed, no single point of failure
- Horizontally scales rather than vertical (cheaper)
- Can store unstructured data

Cons

- No standardized Query language (different for each NoSQL solution)
- Queries are generally less efficient than SQL (especially when looking for unstructured data)
- Distributed data storage causes issues with data privacy and integrity for certain business use cases (DoD/Gov work)

Note: You CAN self host NoSQL solutions, in fact for my company we do, but you need to seriously spend the time to set it up correctly, as most of the pros come from the “distributed” part of how they are hosted.

Horizontal Scaling vs. Vertical Scaling

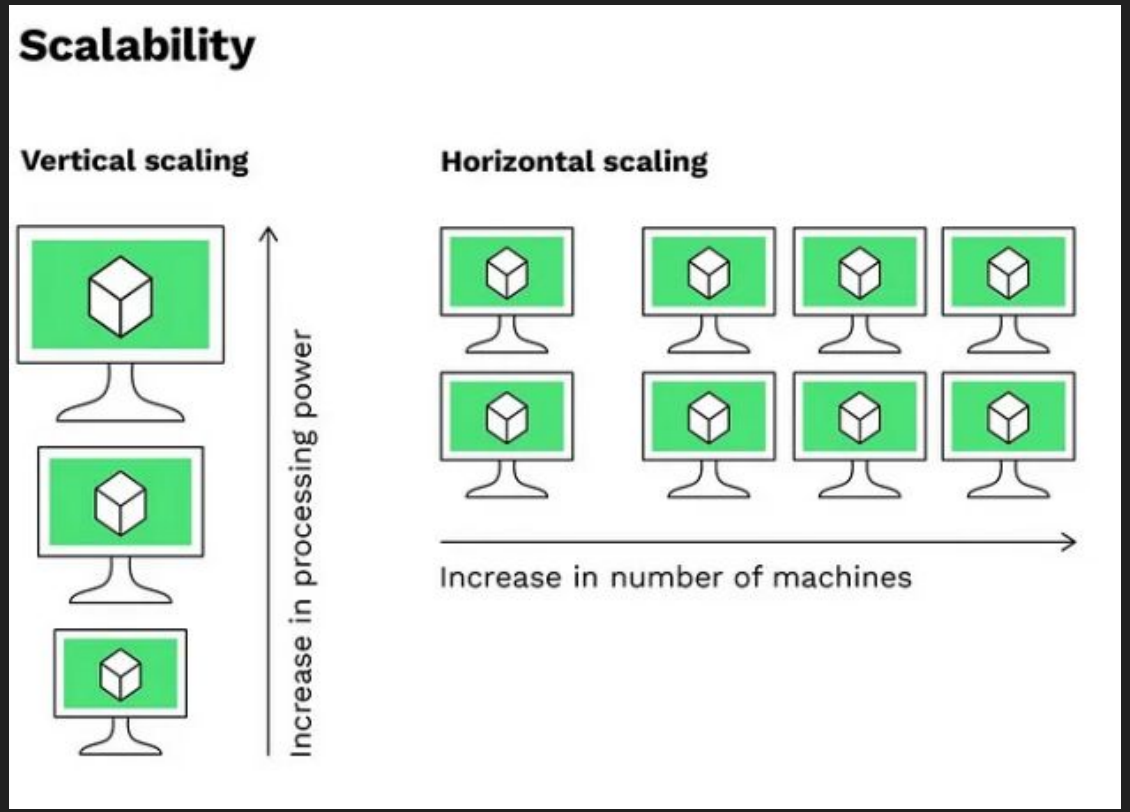
Horizontal Scaling

- When resources run low, just get another system.
- Synchronization between the systems.
- Ideal for independent systems (low dependency)

Vertical Scaling

- When resources run low, just get better specifications (upgrading GPU/CPU).
- One system, all the infrastructure is contained in one place.
- Costly

Horizontal Scaling vs. Vertical Scaling



What is a Non-Relational Database

Ok, so what actually is a Non-Relational Database (NoSQL)

What is a Non-Relational Database

***NoSQL** is effectively an approach to storing data that doesn't involve the typical **table-based** approach we use with **SQL** and instead focuses on a **Model** based approach.*

What is a Non-Relational Database

```
{
    "userid" : 1234,
    "username" : "nickrocky213",
    "coins" : 1000,
    "friends" : {
        "username" : "knightro",
        "username" : "floatbob",
        "username" : "mirage.tsx"
    },
    "faction" : "knighthacks"
}
```

```
public struct Player {
  public long userid;
  public string username;
  public int coins;
  public List<string> friends;
  public long faction;
}
```

- The main goal here is to make programming structures that act as models that you can then write to a database
- These models are used to create documents that actually act to store the information in the database
- The top example is a document, the bottom example is a model that would create the above document

What is a Non-Relational Database

We call sets of these documents collections, so a NoSQL database is made of documents in collections

What is a Relational Database

With SQL databases things take a bit of a turn, we instead create rows and store them in tables.

What is a Relational Database

The major difference is that we additionally specify the relationships between parts of the data.

We use keys to specify relationships between tables.

What is a Relational Database

SQL has many types of keys but the two most commonly used types of keys are Primary Keys and Foreign Keys

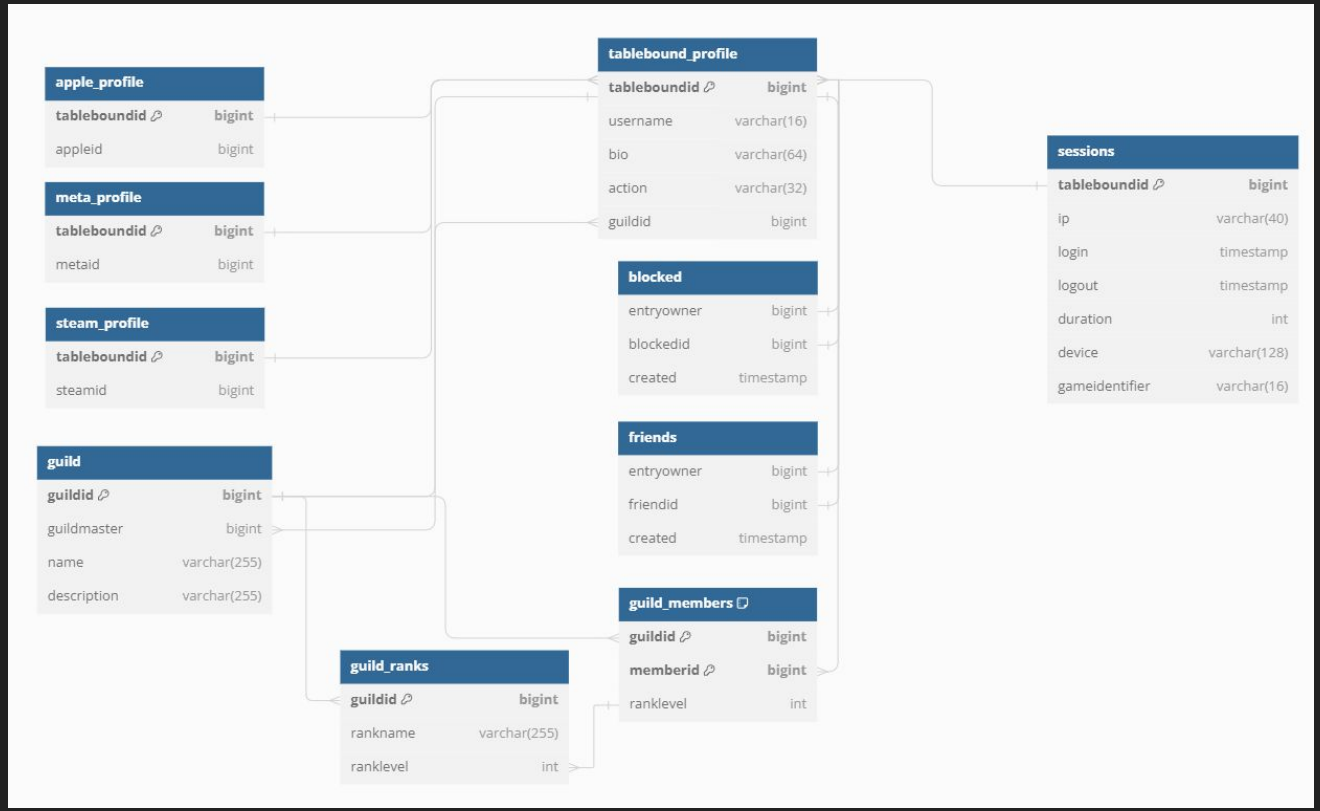
A primary key is a value that uniquely identifies a given row in a table

A foreign key is a value that relates to a key in another table

What is a Relational Database

Let's take a quick look at a real SQL database

What is a Relational Database



What is a Relational Database

	tableboundid	username	bio	action
1	102	nickrocky213	""	""
2	103	altmed	""	""
3	104	knightro	""	""

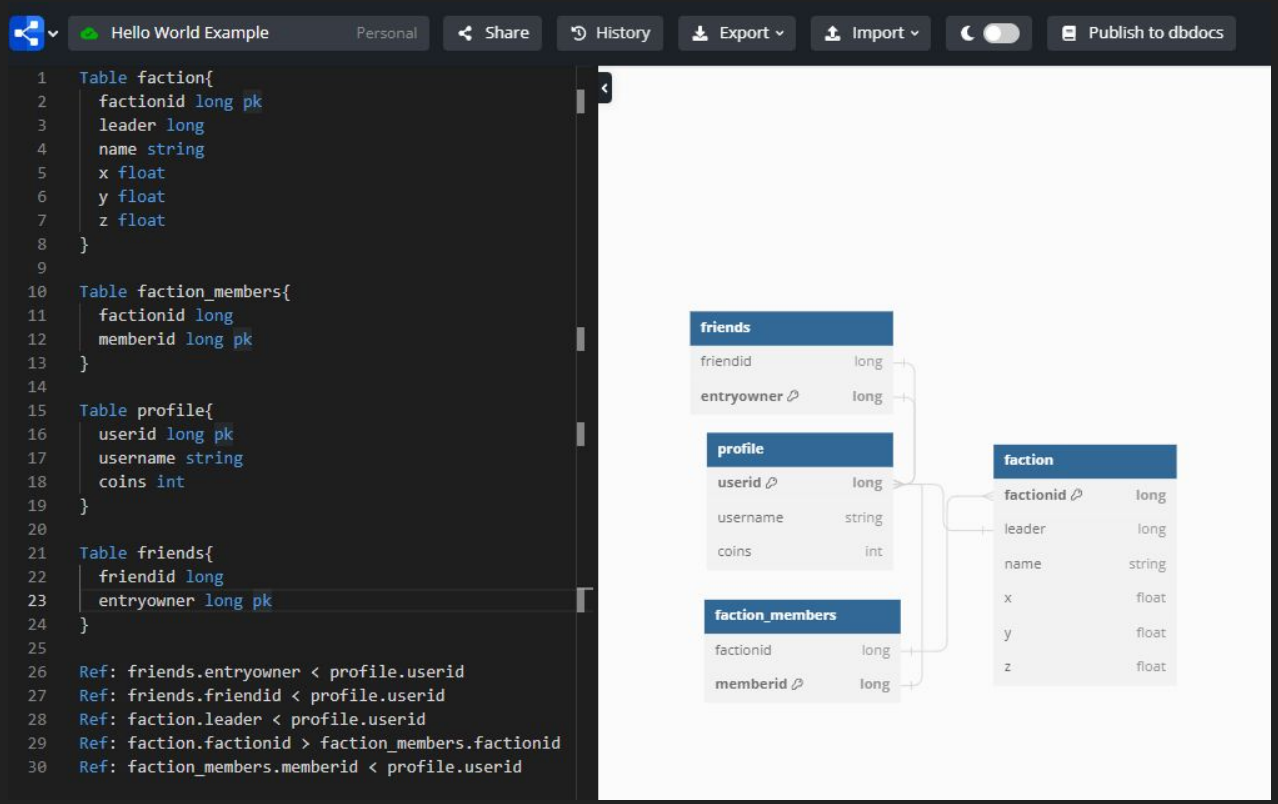
For the sake of Hello World I took some snippets from a dev table and put in some temporary data

What is a Relational Database

So in essence, columns act as our types we are storing, and rows are the actual data we want to store.

Hence why we call them tables

Tool Break!



DBDiagram is a neat tool for laying out databases.

Entity-Relationship Diagrams (ERDs) are really nice and easy to make with this tool

SQL: CRUD (Create Read Update Delete)

When creating any service four main operations are needed to have the bare minimum data storage wise.

- Create
  - Makes a new entry
- Read
  - Reads a previously created entry
- Update
  - Updates a previously created entry
- Delete
  - Deletes an entry

SQL: CRUD (Create Read Update Delete)

Before you can begin you will need to make a new database, for the sake of this workshop I will walk everyone through the commands but realistically this information is online and fairly easy to get a hold of.

We will be using the PostgreSQL dialect of SQL

SQL: Type Cheatsheet

Data Types in SQL are slightly different from the usual suspects you would see in a coding language

This is a direct link to all of the types you can store in PostgreSQL, however, for the sake of this workshop

string > VARCHAR(size)

long > bigint

VARCHAR is “Variable Characters” and the size is just the length of the string

SQL: CRUD (Create Read Update Delete)

With that out of the way let's get into it!

SQL: CRUD (Create Read Update Delete)

```
CREATE DATABASE knighthacks;
```

Create database creates a new database to store all of our tables in

```
USE knighthacks;
```

Sets our database to be knighthacks

```
CREATE TABLE profiles (firstname VARCHAR(255), lastname VARCHAR(255));
```

Creates a new table to store profiles

SQL: CRUD (Create Read Update Delete)

To create an entry in this table we need to create what's called a statement

Statements are effectively commands that we are issuing a database

SQL: CRUD (Create Read Update Delete)

Insertion statements (CRUD) require the table, the columns you want to insert into, and the values to put into the new row.

SQL Reserved words are indicated in this color

INSERT INTO profiles (firstname, lastname) VALUES (‘dylan’, ‘vidal’);

SQL: CRUD (Create Read Update Delete)

Query statements (CRUD) require the table and the columns you want to query for

SELECT firstname, lastname FROM profiles;

Alternatively if you want ALL columns in a table you can use *

SELECT * FROM profiles;

SQL: Conditionals

In SQL often times you're looking for a particular row not just all rows, to achieve this we use the WHERE keyword.

The following conditional keywords can be used in WHERE statements: AND, OR, NOT

*Ex. SELECT * FROM profiles WHERE firstname = 'dylan' AND lastname = 'vidal';*

SQL: CRUD (Create Read Update Delete)

Update statements (CRUD) require the table and the columns you want to set the data you want to set it to and a WHERE statement to limit the changes

```
UPDATE profiles SET firstname = 'MR', lastname = 'PRESIDENT'
WHERE firstname = 'dylan' AND lastname = 'vidal';
```

So I've changed my row in the profiles table from saying my name to MR PRESIDENT!

SQL: CRUD (Create Read Update Delete)

Delete statements (CRUD) require the table and a WHERE statement to limit the changes

DELETE FROM profiles WHERE firstname = 'MR' AND lastname = 'PRESIDENT';

So now we have deleted Knightro from the table entirely!

SQL: Primary and Foreign Keys

To specify a particular column as a key we need to use something called a CONSTRAINT.

Note: Yes this is confusing, but the naming convention is effectively saying “when i am talking about X constrain your view to Y”

SQL: Primary and Foreign Keys

Lets do primary keys first since they are often the simplest!

CREATE TABLE faction (factionid BIGINT PRIMARY KEY, name VARCHAR(255));

That's it! Just specify that when you are making the table!

Note: There is also a way to specify a primary key after the table is created using the keyword ALTER TABLE.

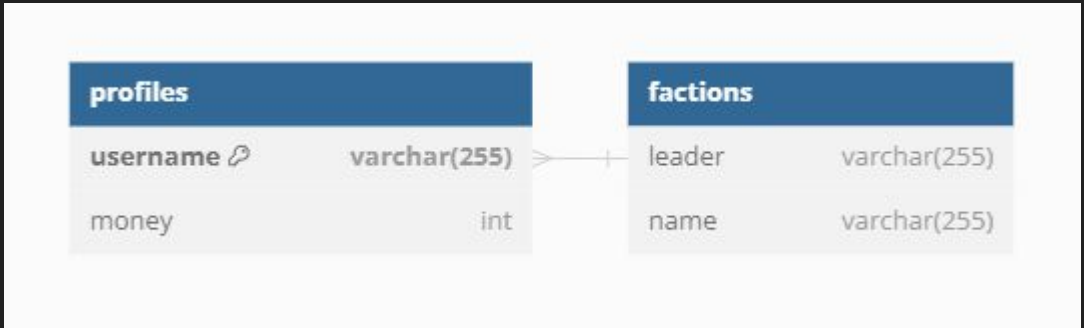
SQL: Primary and Foreign Keys

***Foreign keys** are more complicated, but also are the thing that is actually specifying the **relationship** between the **tables**.*

SQL: Primary and Foreign Keys

```
CREATE TABLE users (username VARCHAR(255) PRIMARY KEY, money int);

CREATE TABLE factions (leader VARCHAR(255), name VARCHAR(255),
CONSTRAINT FK_LEADER FOREIGN KEY (leader) REFERENCES users (username));
```



SQL: Joins

Joins are the primary reason why so many people get frustrated with SQL, it is simultaneously the best and worst part of working with SQL Databases.

SQL: Joins

To start we are going to learn about the types of joins, then move into writing a single join.

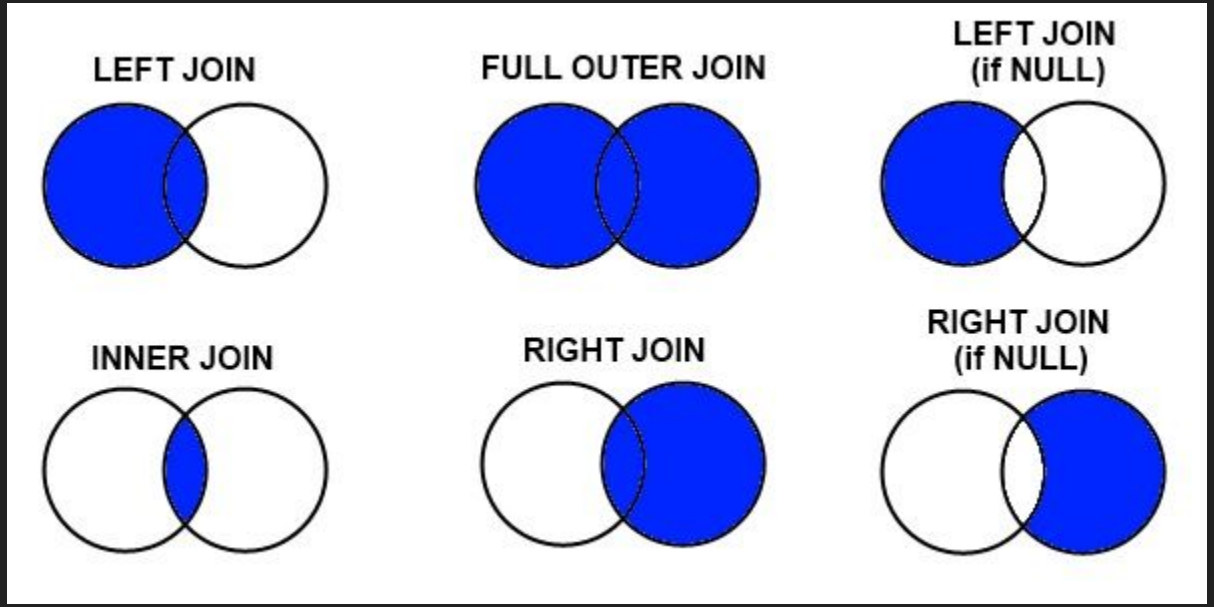
This will take time to get the hang of, don’t be discouraged!

SQL: Joins

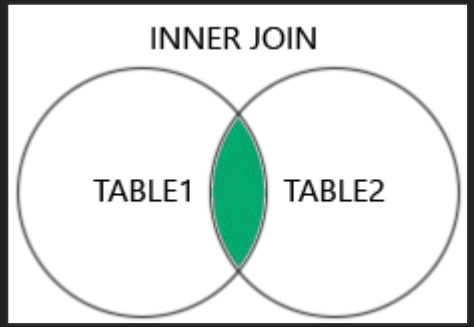
To start we are going to learn about the types of joins, then move into writing a single join.

This will take time to get the hang of, don’t be discouraged!

SQL: Joins


LEFT JOIN
FULL OUTER JOIN
LEFT JOIN
(if NULL)
INNER JOIN
RIGHT JOIN
RIGHT JOIN
(if NULL)

SQL: Joins - Inner Join



Username	Money
knightro	100000
nickrocky	10

Leader	Name
nickrocky	knighthacks
caleb	devteam

```
SELECT profiles.username, factions.name, profiles.money FROM profiles
INNER JOIN factions ON profiles.username = factions.leader
```

Result!

Username	Name	Money
nickrocky	knighthacks	10

Using Databases in Applications?

1. Pick a Database Paradigm and dialect (NoSQL/SQL)
2. Find a host (Local machine, VPS, or provider like PlanetScale, Neon, or Turso)
3. Research clients for your framework (language specific)
4. Create connections using the client
5. Perform CRUD operations as needed

Step 4.1 (Optional) - ORMs

Object Relational Mappings

- Allows for treating database tables more like objects
- Abstracts some of the more complicated SQL statements