

DAVID LIN

CSC309 Week 4

Databases, Prisma and Memes

[https://utoronto.zoom.us/j/8](https://utoronto.zoom.us/j/87457857764)

**7457857764 (Passcode:
655949)**

**Meeting Code:
87457857764**

Password:655949

Breakdown

Welcome to 4th tutorial of CSC309!

Databases

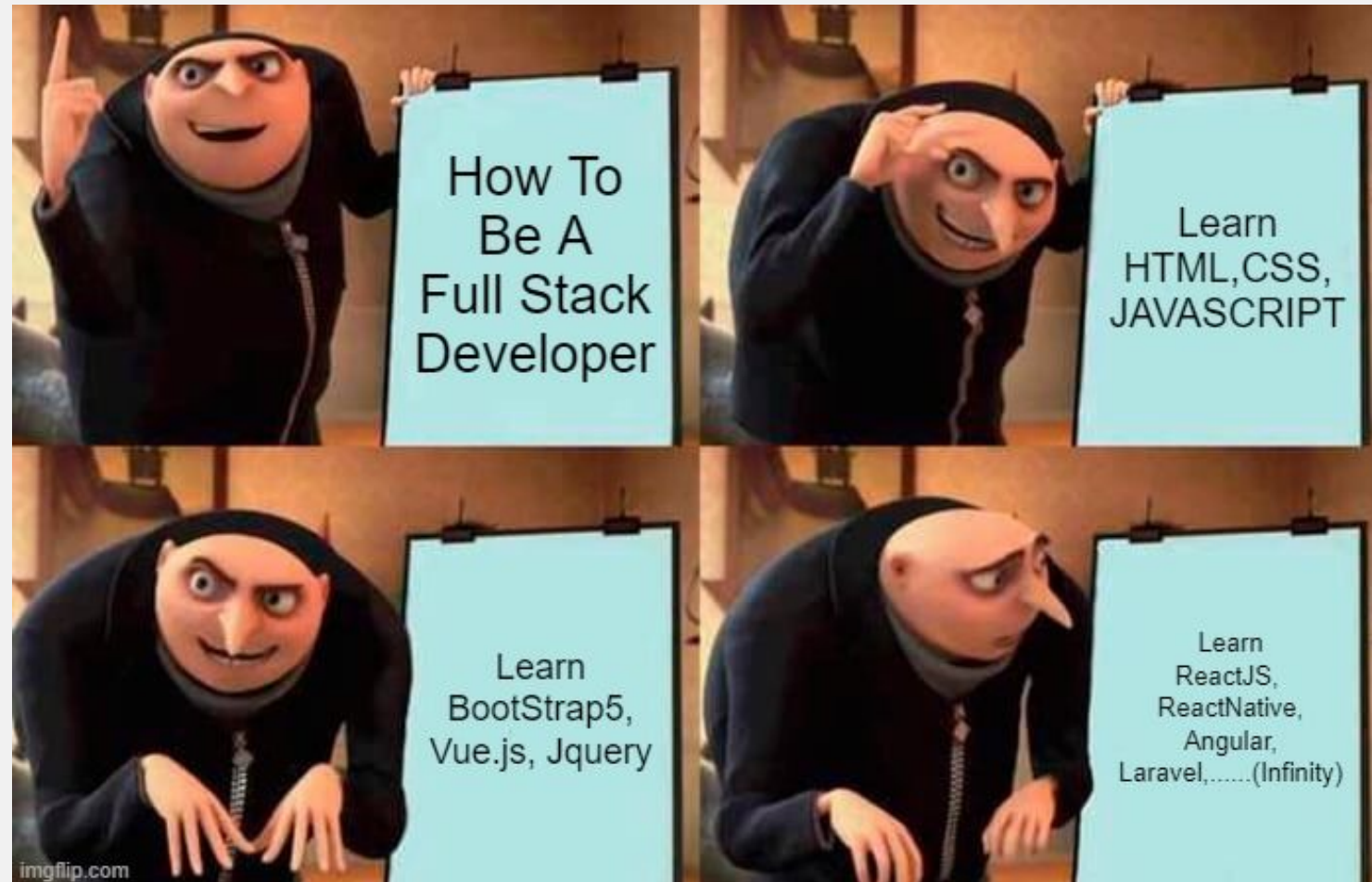
ORMs

Design

Full Stack Before



Full Stack Now



Databases!

- One of the most important factor in full-stack development
- Know how to use databases correctly and the right database to use



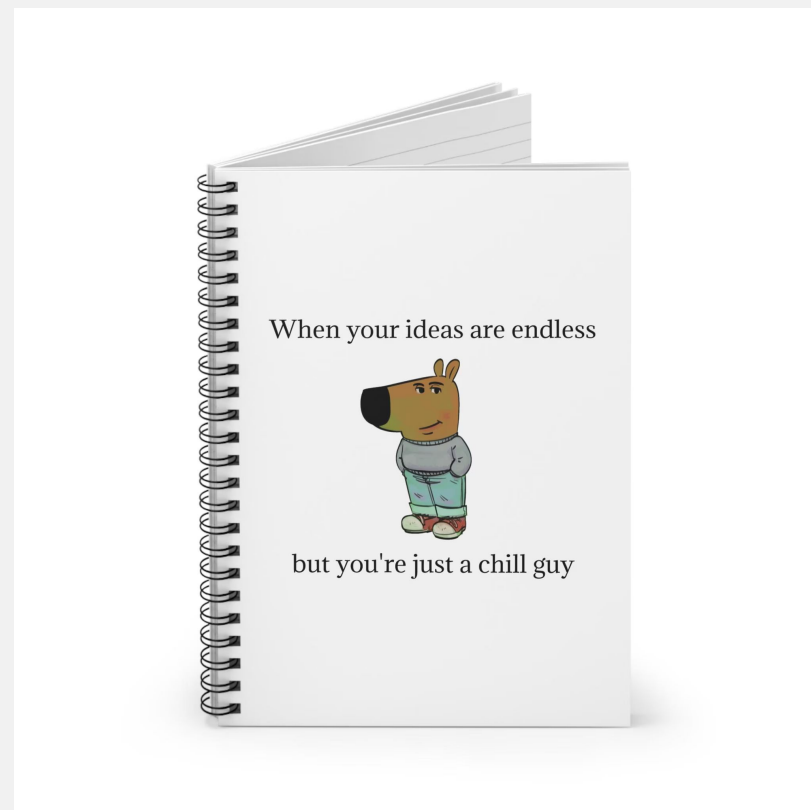
I PREFER A REAL DATABASE



PERFECTION

What is a Database

- A database is an organized collection of data stored electronically.
- Purpose: Enables data storage, retrieval, and management.

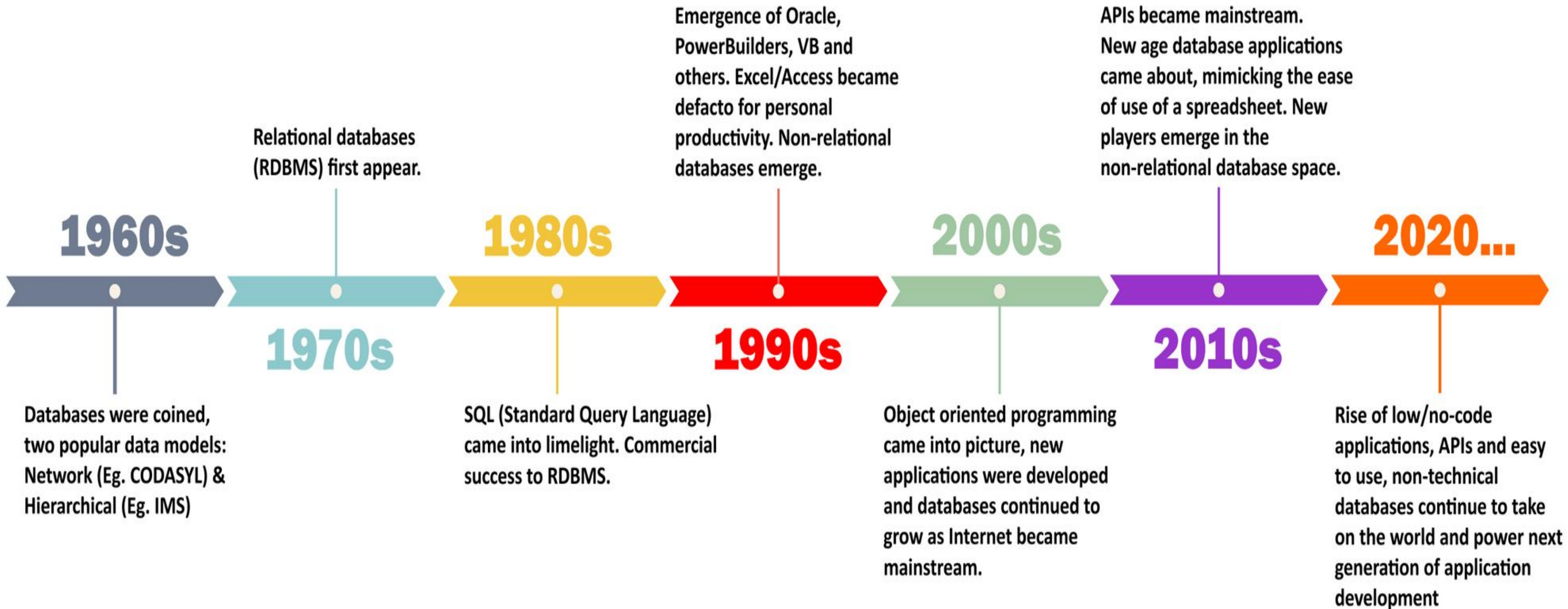


What is a DBMS

- Database Management System
- Provides an interface for querying and modifying data
- Ensures data integrity and security
- Manages concurrent access by multiple users
- Handles backups, recovery, and logging

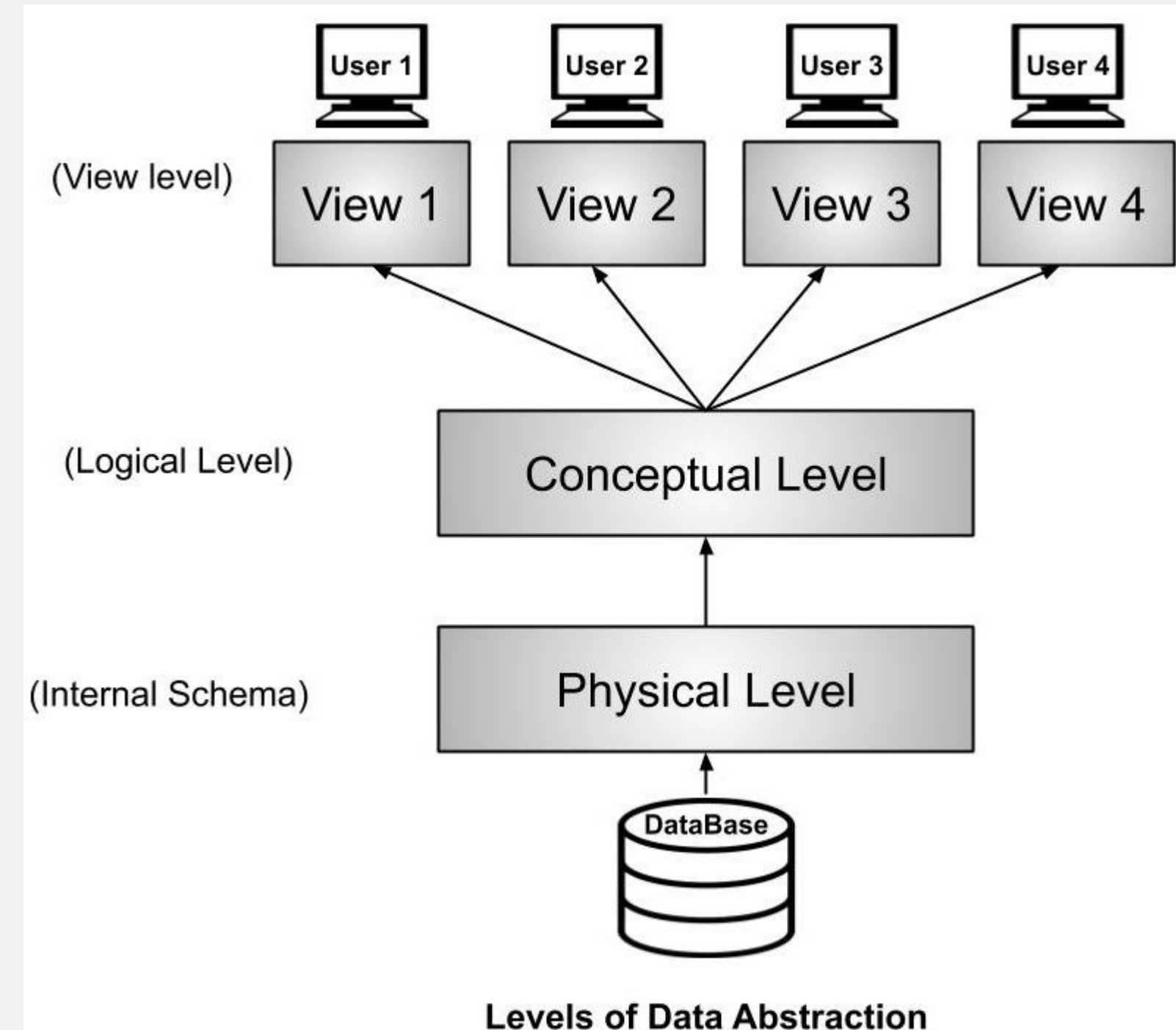


History of Databases (1960-2020)



Independence

- **Physical independence:** DBMS can change how it stores things like compressions, but logical schema remains unchanged
- **Logical independence:** Allowing changes to the logical schema without needing to change the application layer above it



Relational Model

- **Relations:** tables
- **Tuples:** rows
- **Attributes:** columns
- Achieves both data and physical independence

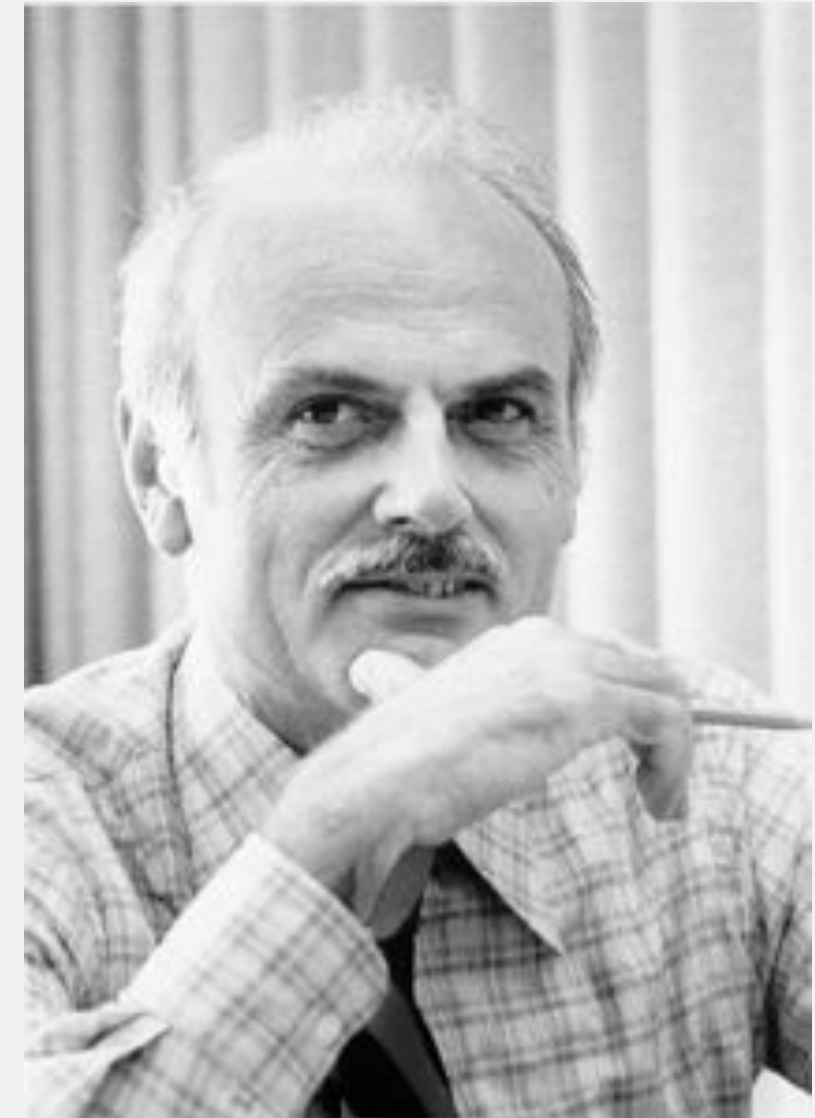
Relational Model

Activity Code	Activity Name
23	Patching
24	Overlay
25	Crack Sealing

Key = 24

Activity Code	Date	Route No.
24	01/12/01	I-95
24	02/08/01	I-66

Date	Activity Code	Route No.
01/12/01	24	I-95
01/15/01	23	I-495
02/08/01	24	I-66



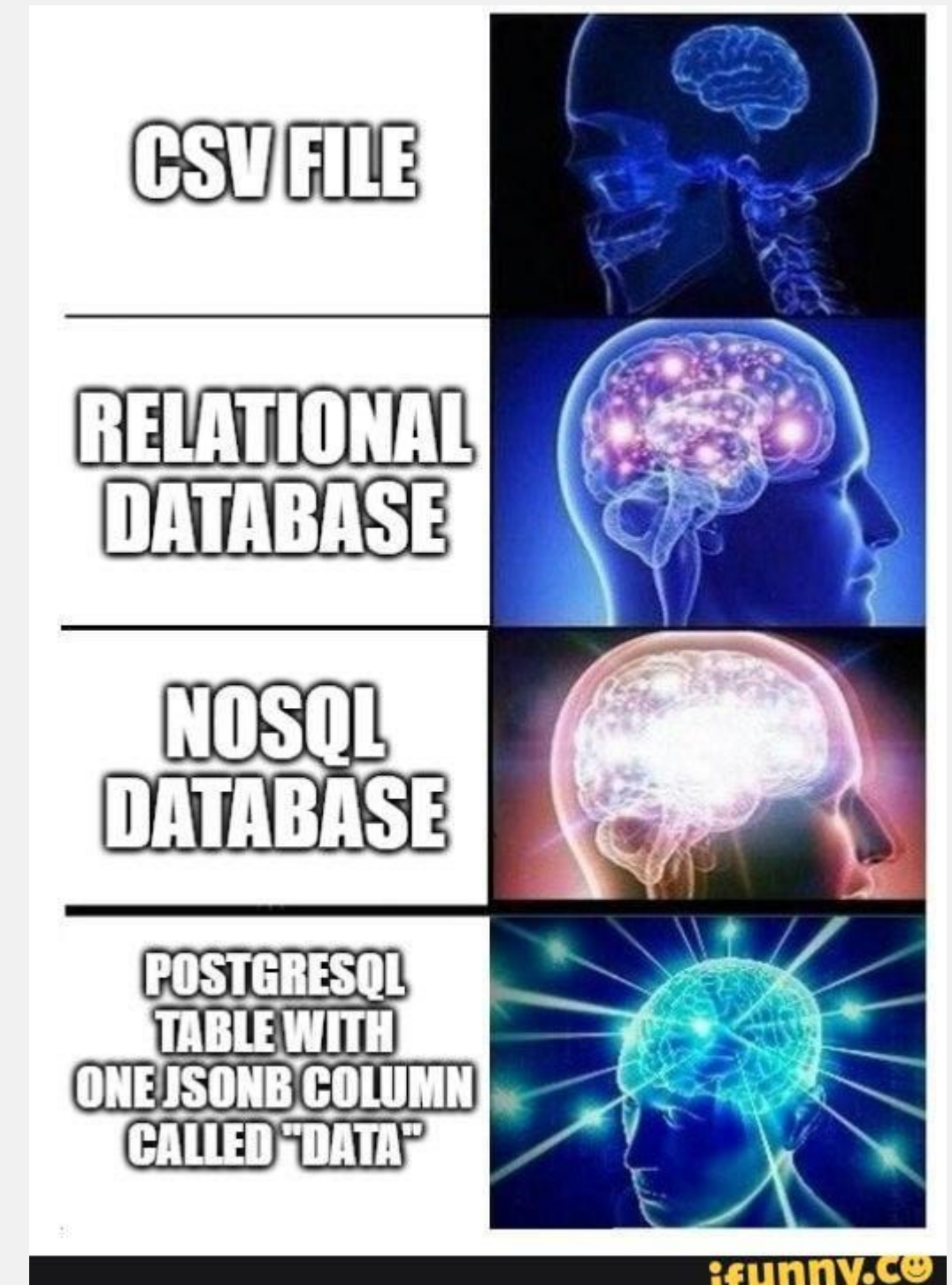
SQL vs. NoSQL

SQL

- Structured, relational schema (e.g., MySQL, PostgreSQL)
- ACID compliance, strong consistency
- Use cases: Financial systems, inventory

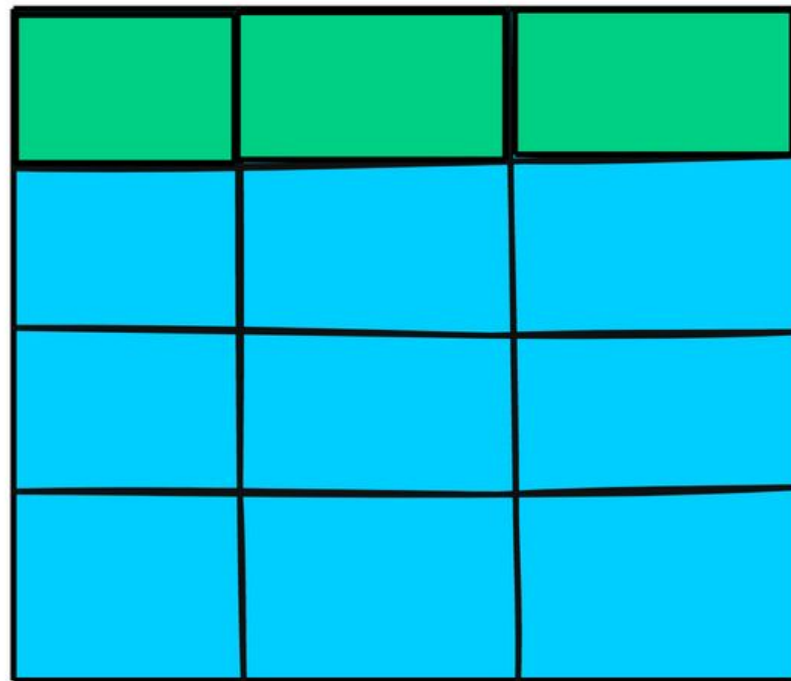
NoSQL

- Not only SQL
- Flexible, schema-less (e.g., MongoDB, Cassandra)
- Horizontal scalability, eventual consistency
- Use cases: Real-time analytics, IoT



SQL vs. NoSQL

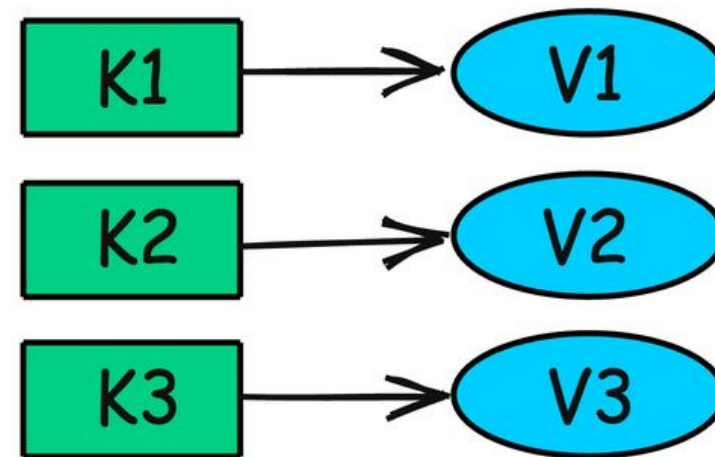
SQL



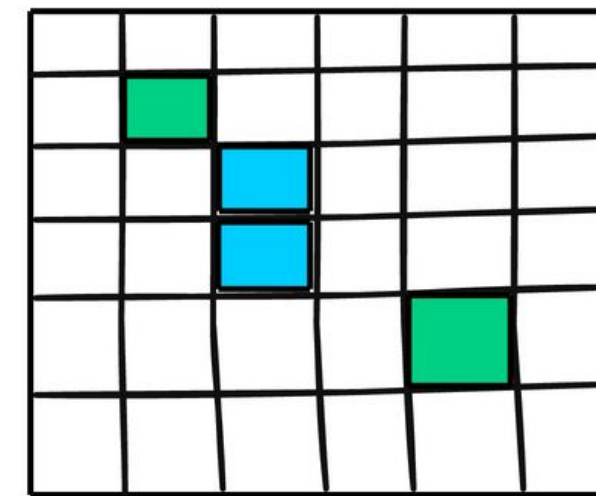
Relational

blog.algomaster.io

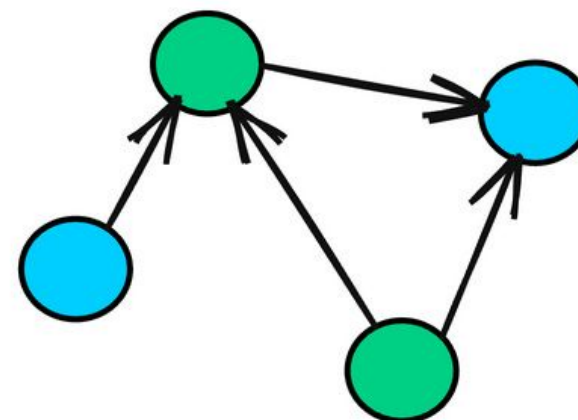
NoSQL



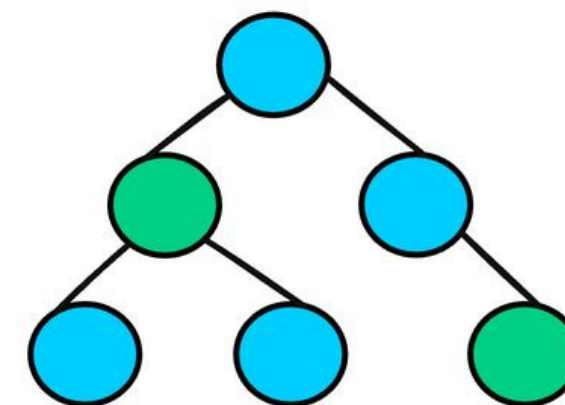
Key-Value



Column Store



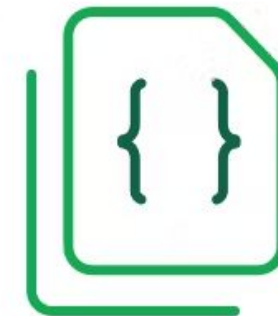
Graph



Document

MongoDB

```
1  {  
2    _id: "5cf0029caff5056591b0ce7d",  
3    firstname: 'Jane',  
4    lastname: 'Wu',  
5    address: {  
6      street: '1 Circle Rd',  
7      city: 'Los Angeles',  
8      state: 'CA',  
9      zip: '90404'  
10  }  
11 }
```



SQL Database Scalability

Vertical Scaling

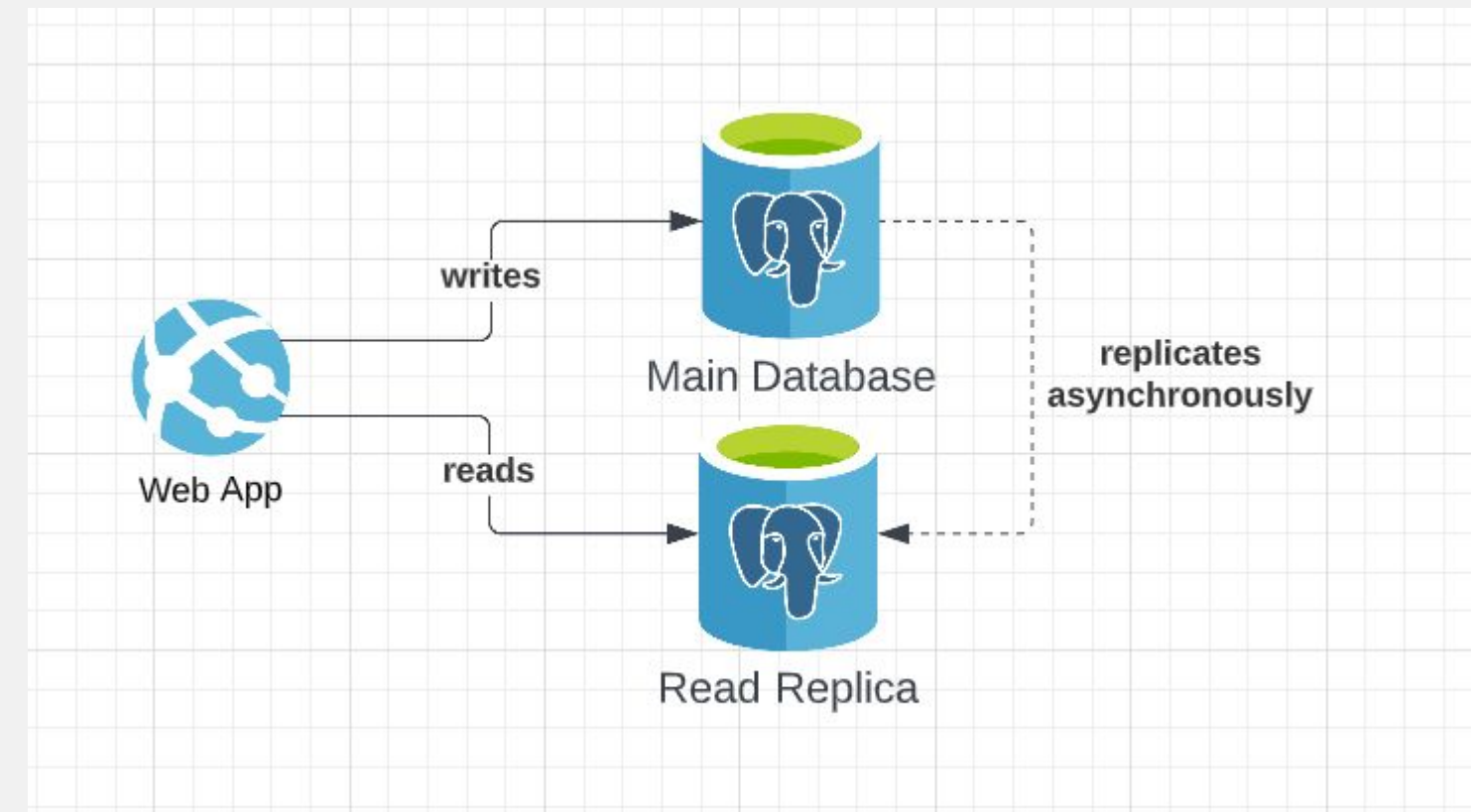
- Increasing the power of a single server (e.g., more CPU, RAM)
- Limitations: Hardware limits, cost

Horizontal Scaling

- Adding more servers to distribute the load
- Benefits: Better for large-scale applications, fault tolerance

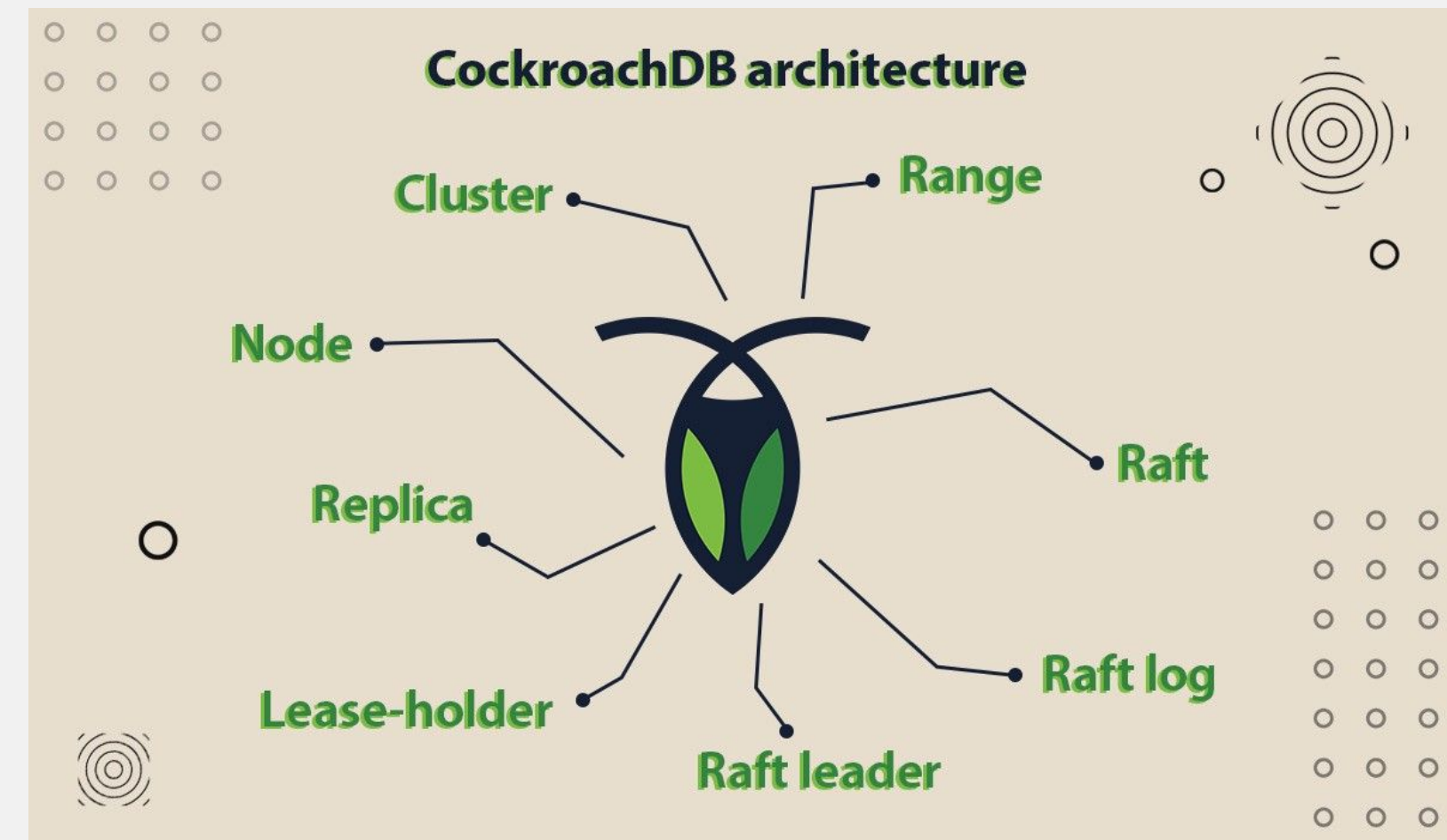
Scalability Challenges

- Consistency vs Availability (CAP Theorem)



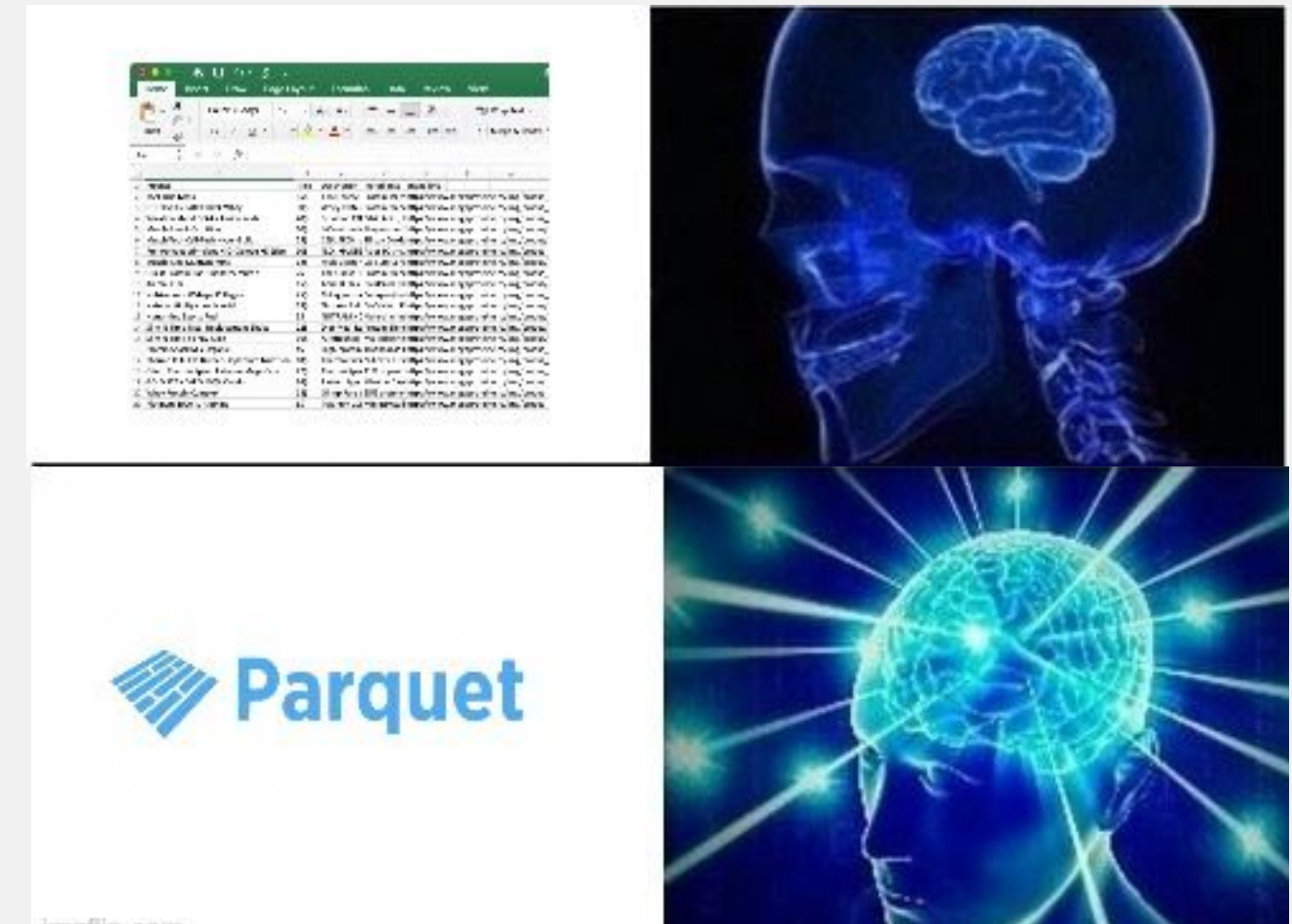
NewSQL

- <https://www.youtube.com/watch?v=PheGC Ru48EU>
- People are trying to combine the scalability of NoSQL with the consistency of traditional databases
- Ex: CockroachDB, a distributed SQL database but it's a KV store inside



Distributed SQL Query Engine

- Big data processing
- Use SQL to query large datasets stored in distributed file systems

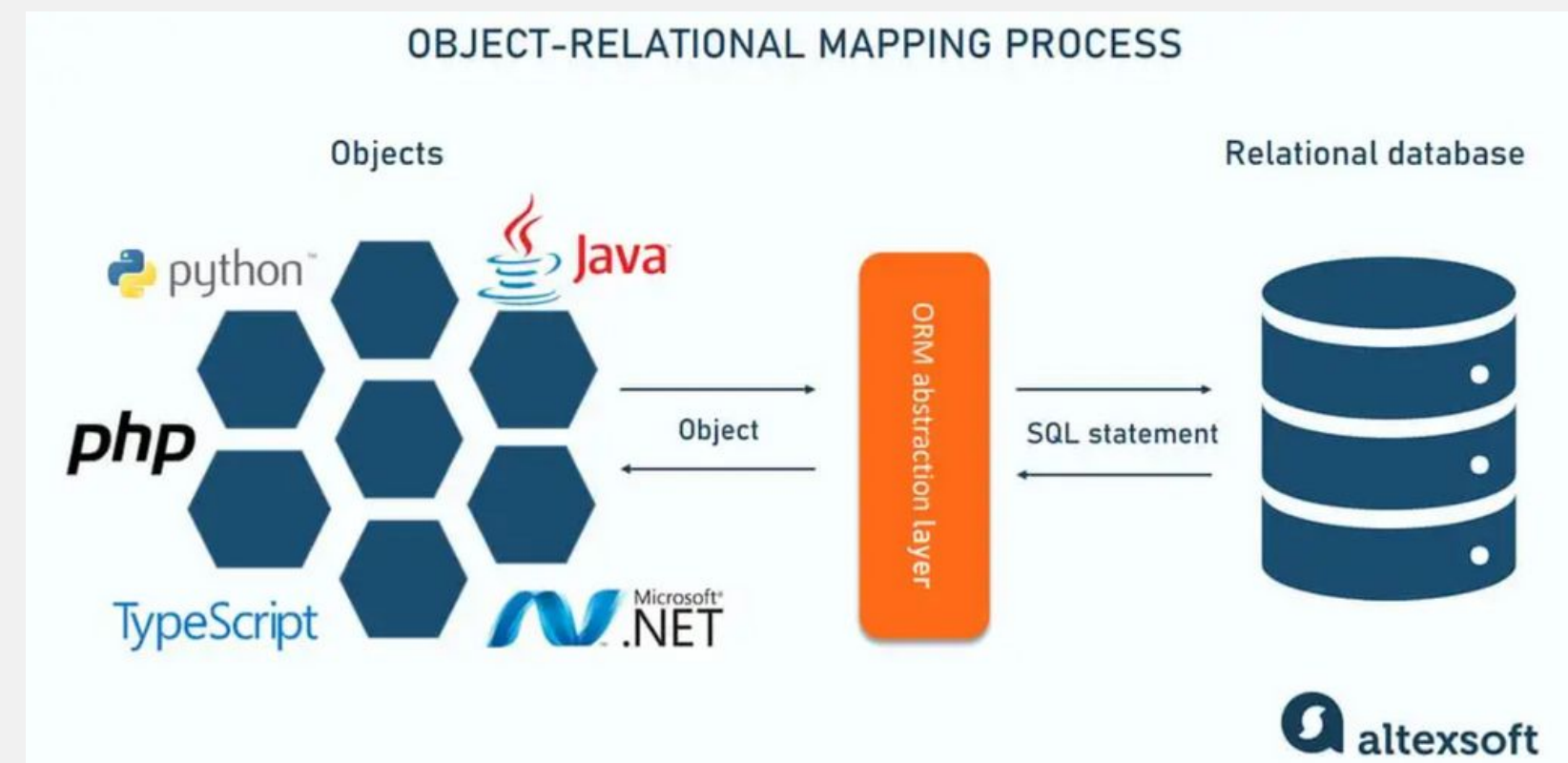


ORMs



Recap: ORMs

- Object–relational mapping
- Provide an abstraction layer over the database, allowing developers to interact with the database using objects instead of raw SQL queries



ORM Benefit

- Simplifies database interactions
- Reduces SQL injection risks
- Enables object-oriented programming with databases
- Clean architecture!!



ORM Drawbacks

- Performance overhead for complex queries
 - Sometimes when you have a lot of joins, it's terrible
- Learning curve for ORM-specific syntax
- Missing functionalities
 - Prisma has trouble with polymorphism



ORM in Node.js

- Prisma
- Sequelize
- TypeORM
- Prisma is not only for Next.js!

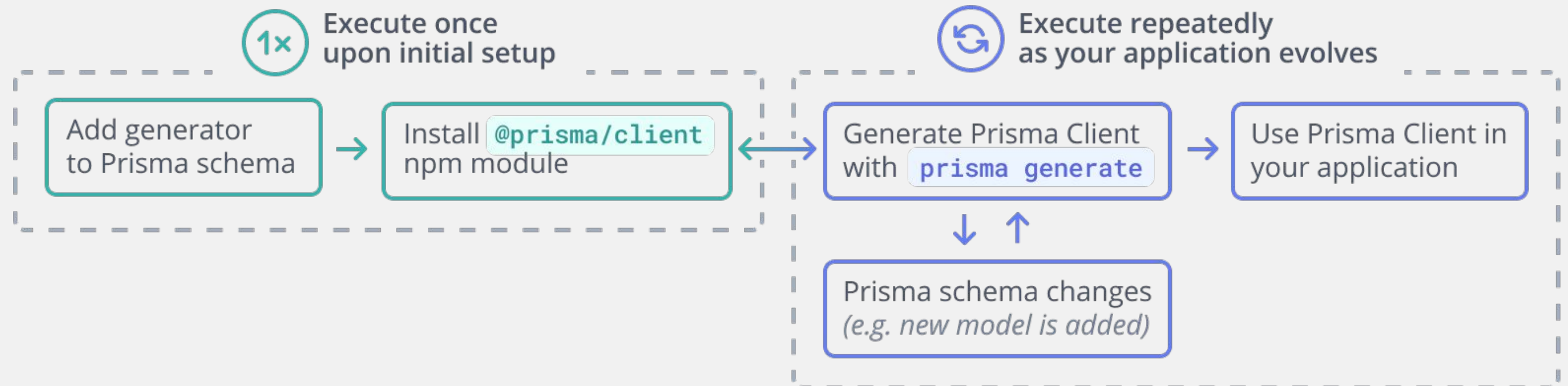


Prisma

- Prisma Client
 - Auto-generated database client that's tailored to the database schema
- Prisma Schema
- Prisma Migrate
- Prisma Studio

```
Generate
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "sqlite"
  url      = env("DATABASE_URL")
}
```



Prisma Client

PRISMA CLIENT

Explore the Prisma Client API

From simple reads to complex nested writes, the Prisma Client supports a wide range of operations to help you make the most of your data.

 JavaScript



Get Started

Reading Data



FIND RECORDS

TRAVERSE RELATIONS

ORDER BY, LIMITS & CURSORS

AGGREGATES & GROUP BY

```
1 // Find all posts
2 const allPosts = await prisma.post.findMany()
```



```
1 // Find a user by ID
2 const userById = await prisma.user.findUnique({
3   where: {
4     id: 2
5   }
6 })
```



```
1 // Find a user by email
2 const userByEmail = await prisma.user.findUnique({
3   where: {
4     email: 'ada@prisma.io'
5   },
6 })
```

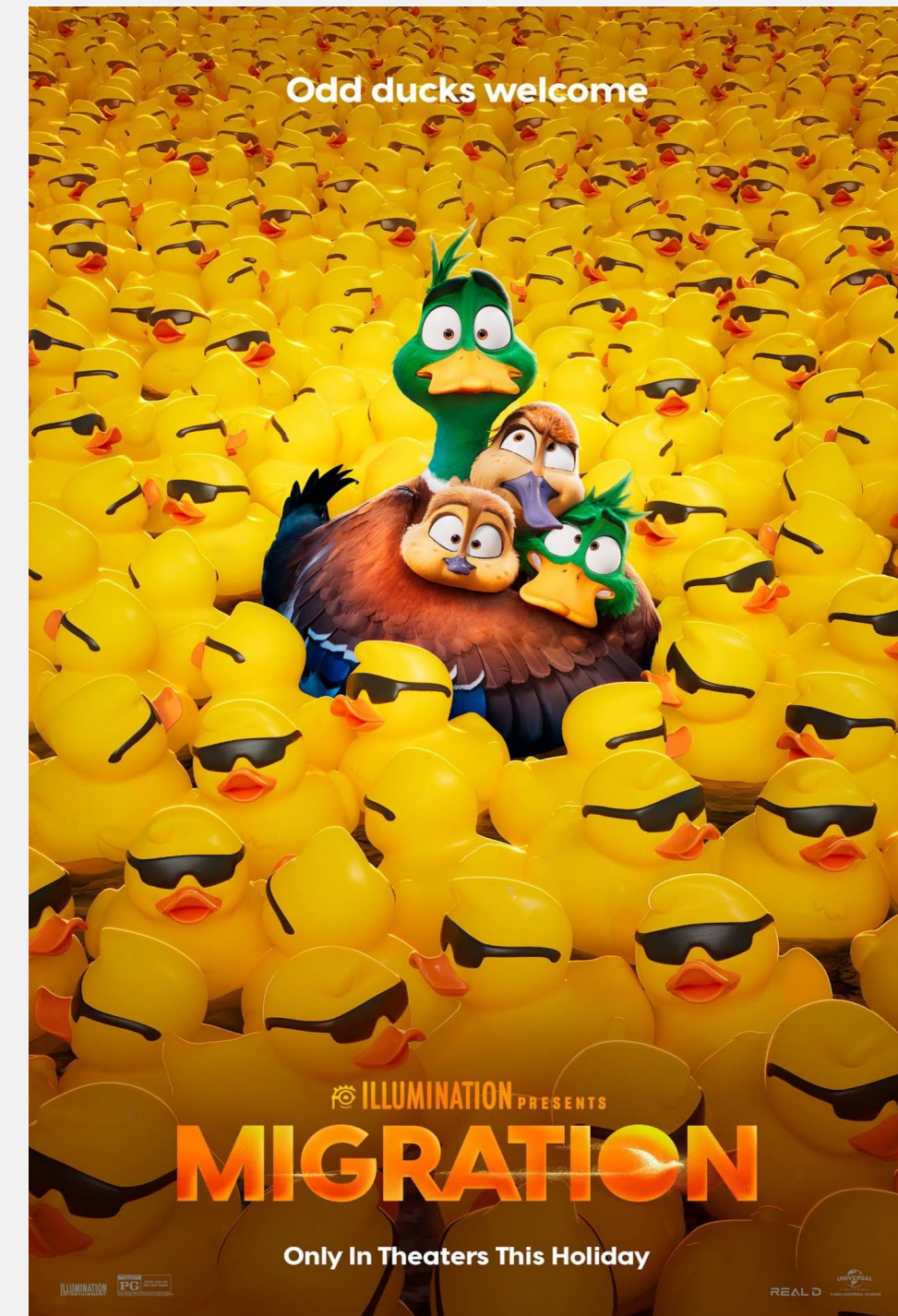


```
1 // Find the first user that starts with Ada
2 const userByName = await prisma.user.findFirst({
3   where: {
4     name: {
5       startsWith: 'Ada'
6     }
7 })
```



Recap: Migration

- Keeps schema changes synchronized across development, staging, and production environments
- Allows reverting changes if something goes wrong
- **NEVER EDIT THE MIGRATION FILES**



Migration Problems IRL

- Migration might be easy for small amount of data
- But when you have millions of rows, ALTER TABLE is hard
- NULLS!
- Interesting Technique: Migration in one PR, code change in the following PR
- <https://www.rainforestqa.com/blog/2014-06-27-zero-downtime-database-migrations>



Prisma Migration

- `npx prisma migrate dev`
- `npx prisma migrate deploy`
 - Only applies existing migrations
 - Doesn't generate new ones
- Example Migration Workflow
 - Update `schema.prisma`
 - Run `prisma migrate dev`
 - Apply changes to the database

Dev vs. Deploy

- In production, schema changes should be pre-tested and reviewed before deployment
- Migrate dev relies on an interactive and iterative workflow to guide the migration process
 - Not suitable for pipeline environment
- Inconsistencies



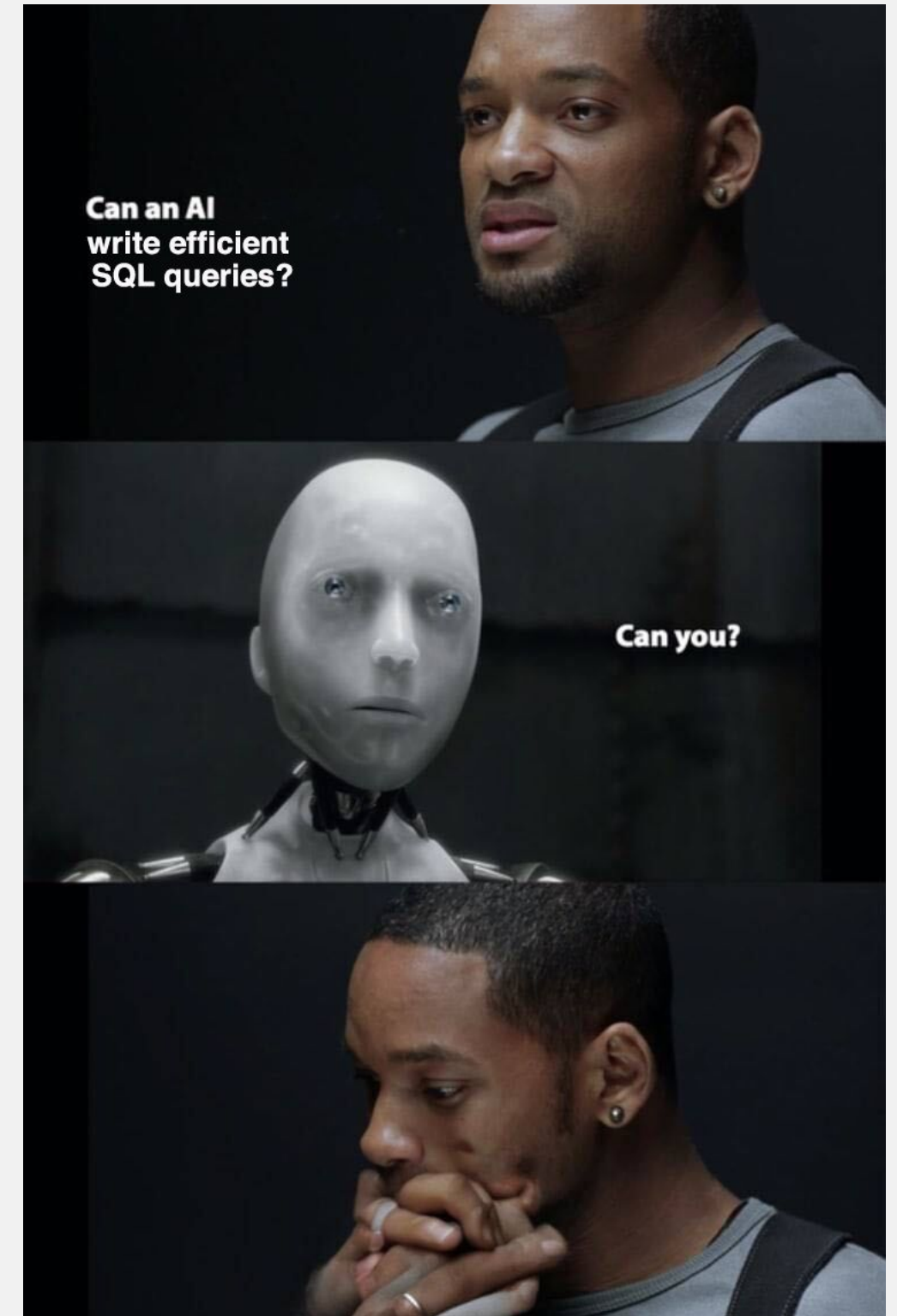
Prisma Studio

- Interface for examining local data
- Confirm if the application is working correctly
- `npx prisma studio`

	Album	Genre	InvoiceLine	Track	+		
Filters None Fields All Showing 100 of 3503 Add record							
	AlbumId #?	Bytes #?	Composer A?	GenreId #?	MediaTypeId #	Milliseconds #	Name A
	1	11170334	Angus Young, Malcolm ...	1	1	343719	For Those About To Ro...
	2	5510424	null	1	2	342562	Balls to the Wall
	3	3990994	F. Baltes, S. Kaufman...	1	2	230619	Fast As a Shark
	3	4331779	F. Baltes, R.A. Smith...	1	2	252051	Restless and Wild
	3	6290521	Deaffy & R.A. Smith-D...	1	2	375418	Princess of the Dawn
	1	6713451	Angus Young, Malcolm ...	1	1	205662	Put The Finger On You
	1	7636561	Angus Young, Malcolm ...	1	1	233926	Let's Get It Up
	1	6852860	Angus Young, Malcolm ...	1	1	210834	Inject The Venom
	1	6599424	Angus Young, Malcolm ...	1	1	203102	Snowballed
	1	8611245	Angus Young, Malcolm ...	1	1	263497	Evil Walks
	1	6566314	Angus Young, Malcolm ...	1	1	199836	C.O.D.
	1	8596840	Angus Young, Malcolm ...	1	1	263288	Breaking The Rules
	1	6706347	Angus Young, Malcolm ...	1	1	205688	Night Of The Long Kni...
	1	8817038	Angus Young, Malcolm ...	1	1	270863	Spellbound
	4	10847611	AC/DC	1	1	331100	Go Down
	4	7032162	AC/DC	1	1	215196	Dog Eat Dog

Embrace Beauty of SQL

- You can run raw SQL in Prisma
- `const result = await prisma.$queryRaw`SELECT * FROM "User" WHERE id = ${userId}`;`



Prisma Extensions

- New feature for Prisma
- Add functionality to your models, result objects, and queries, or to add client-level methods
- Use cases
 - Computed fields
 - Soft delete
 - Limiting result batch size

You can use the `result` extension component to add fields to query results. These fields are computed at runtime and are type-safe.

In the following example, we add a new virtual field called `fullName` to the `user` model.

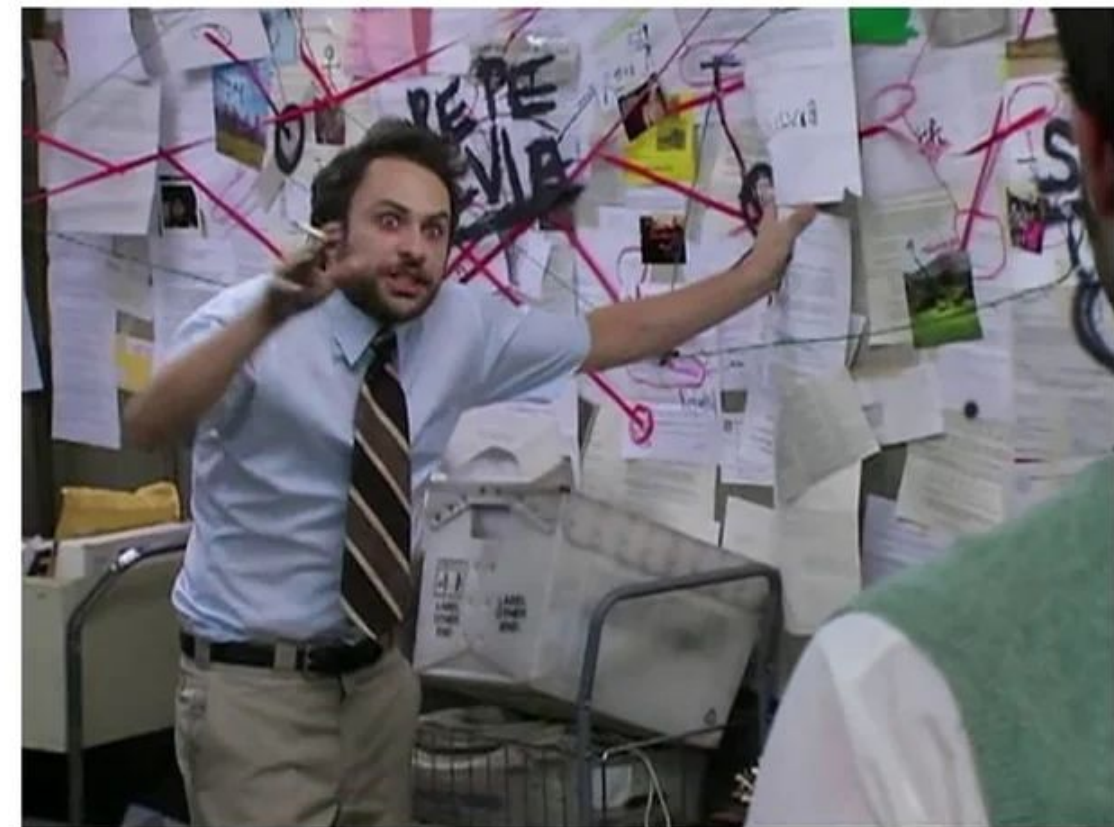
```
const prisma = new PrismaClient().$extends({
  result: {
    user: {
      fullName: {
        // the dependencies
        needs: { firstName: true, lastName: true },
        compute(user) {
          // the computation logic
          return `${user.firstName} ${user.lastName}`
        },
      },
    },
  },
})

const user = await prisma.user.findFirst()

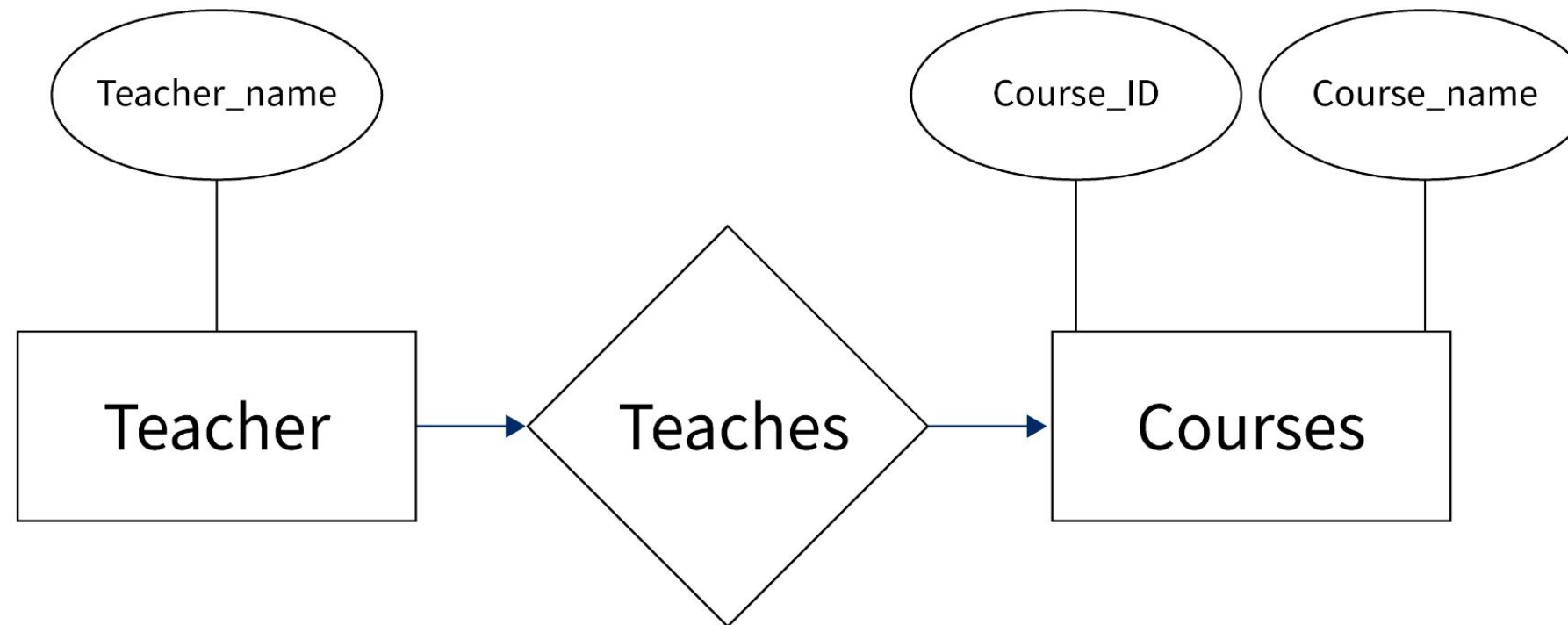
// return the user's full name, such as "John Doe"
console.log(user.fullName)
```

Designing

When people ask me to explain my database design



ER Diagram



One-to-One

- Example: One person has one resume (ideally), and the resume can only be used by one person

One-to-one (1-1) relations refer to relations where at most **one** record can be connected on both sides of the relation. In the example below, there is a one-to-one relation between `User` and `Profile`:

Relational databases

MongoDB

```
model User {
  id      Int      @id @default(autoincrement())
  profile Profile?
}

model Profile {
  id      Int      @id @default(autoincrement())
  user    User     @relation(fields: [userId], references: [id])
  userId  Int      @unique // relation scalar field (used in the '@relation' attribute above)
}
```

One-to-Many

- Example: One student can have multiple internships

One-to-many (1-n) relations refer to relations where one record on one side of the relation can be connected to zero or more records on the other side. In the following example, there is one one-to-many relation between the `User` and `Post` models:

Relational databases

MongoDB

```
model User {
  id      Int      @id @default(autoincrement())
  posts   Post[]
}

model Post {
  id      Int      @id @default(autoincrement())
  author   User    @relation(fields: [authorId], references: [id])
  authorId Int
}
```

Many-to-many

- Example: One intern can work on multiple projects, and a project can have multiple interns

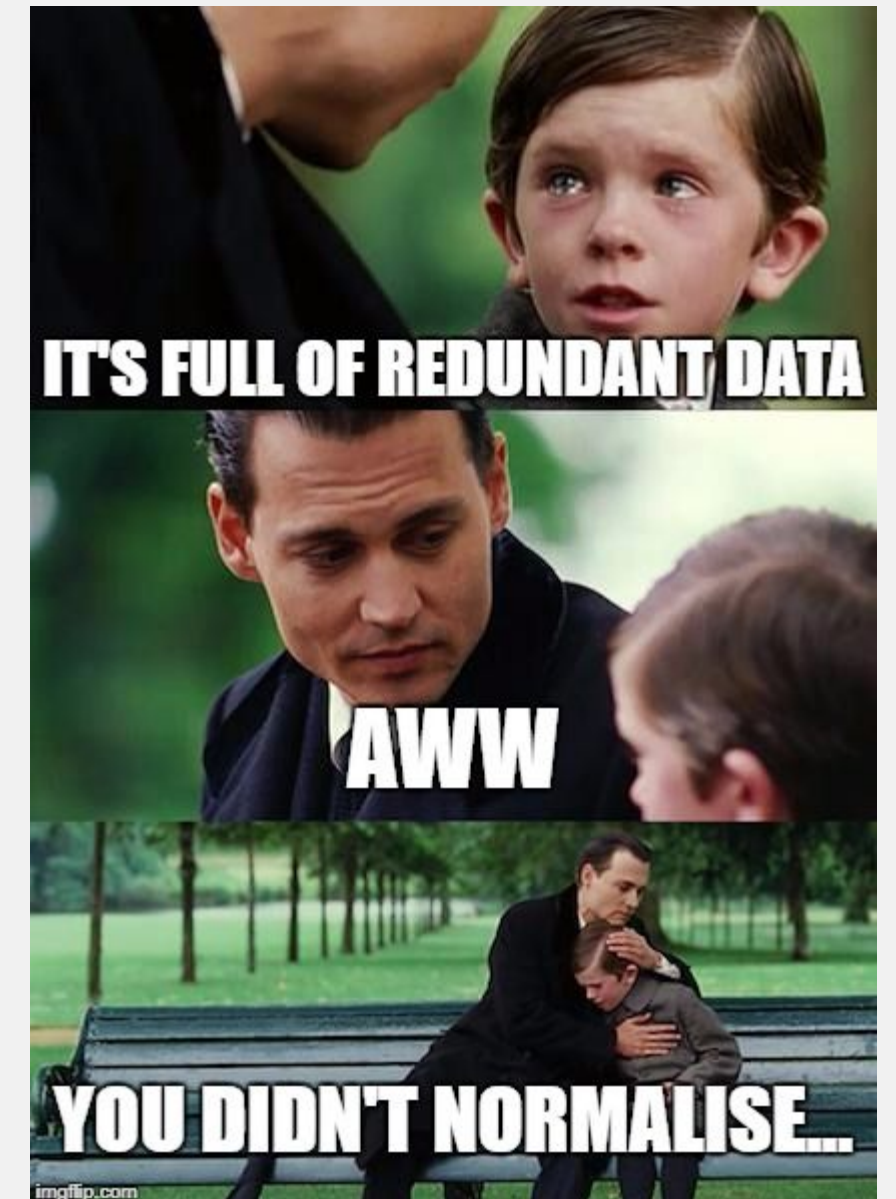
```
Relational databases MongoDB

model Post {
  id      Int      @id @default(autoincrement())
  title   String
  categories Category[]
}

model Category {
  id      Int      @id @default(autoincrement())
  name    String
  posts   Post[]
}
```

Normalizations

- Organizing database tables and relationships to minimize redundancy
- Improve data integrity
- Example
 - Like count for a post
 - Denormalized: Save as integer
 - Normalized: COUNT like table



Indexing

- An **index** is a data structure (B-tree, hash maps) that improves the speed of data retrieval operations on a database table
- Similar to the index in a book
- Reduces the time required to locate rows
- Drawbacks
 - Storage Overhead: Indexes require additional space
 - Inserts, updates, and deletes can be slower because indexes must be updated
- Index only frequently used columns



Example

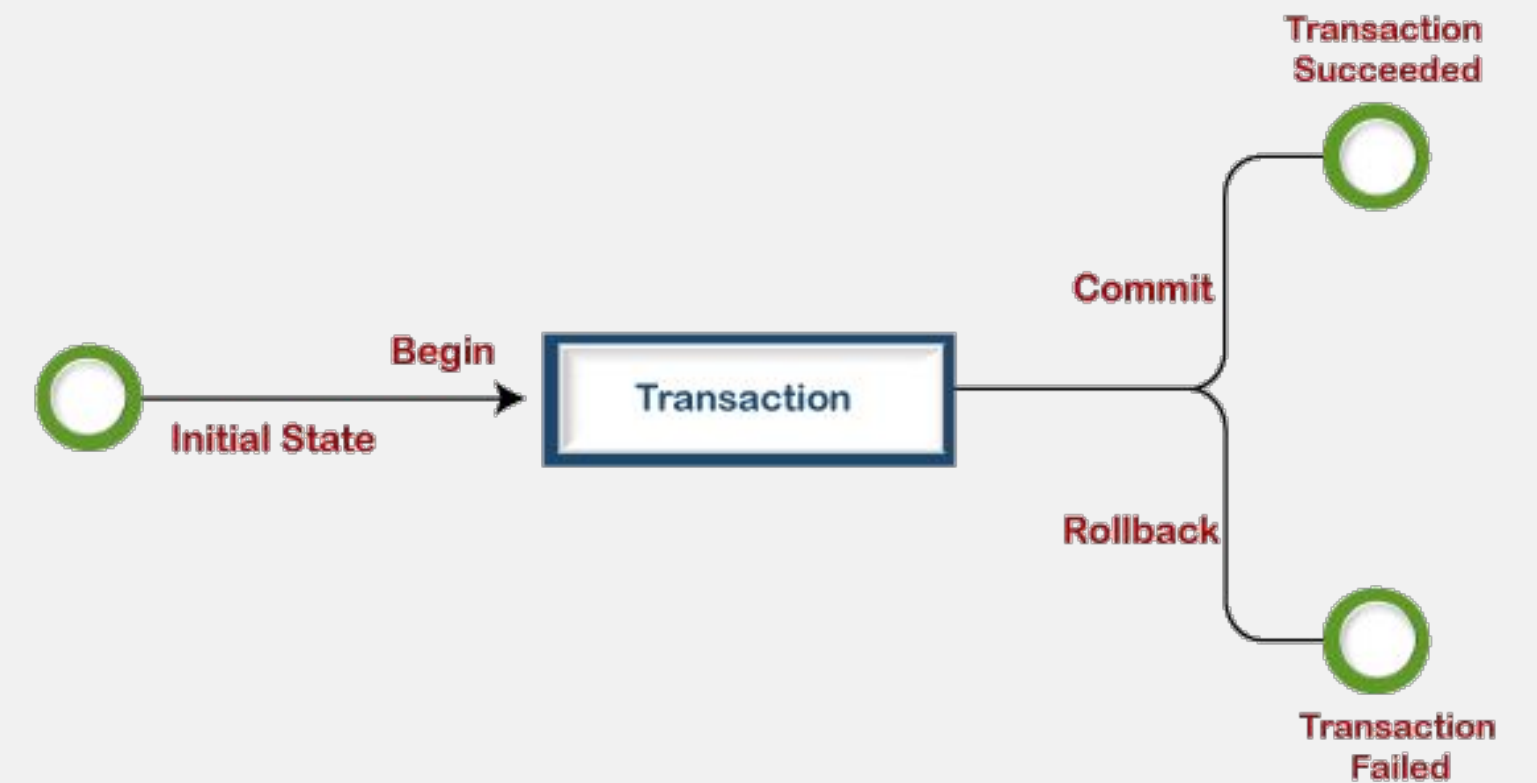
- <https://www.prisma.io/docs/orm/prisma-schema/data-model/indexes>
- Automatically creates unique indexes for fields marked with the `@unique` attribute or the `@id` attribute
- You can create a composite index using the `@@index` or `@@unique` attribute in the model block.
- Full Text Index: full-text search in databases like MySQL or PostgreSQL
`@@fulltext`

schema.prisma

```
1 model Post {
2   title      String   @db.VarChar(300)
3   abstract   String   @db.VarChar(3000)
4   slug       String   @unique(sort: Desc, length: 42) @db.VarChar(3000)
5   author     String
6   created_at DateTime
7
8   @@id([title(length: 100, sort: Desc), abstract(length: 10)])
9   @@index([author, created_at(sort: Desc)])
10 }
```

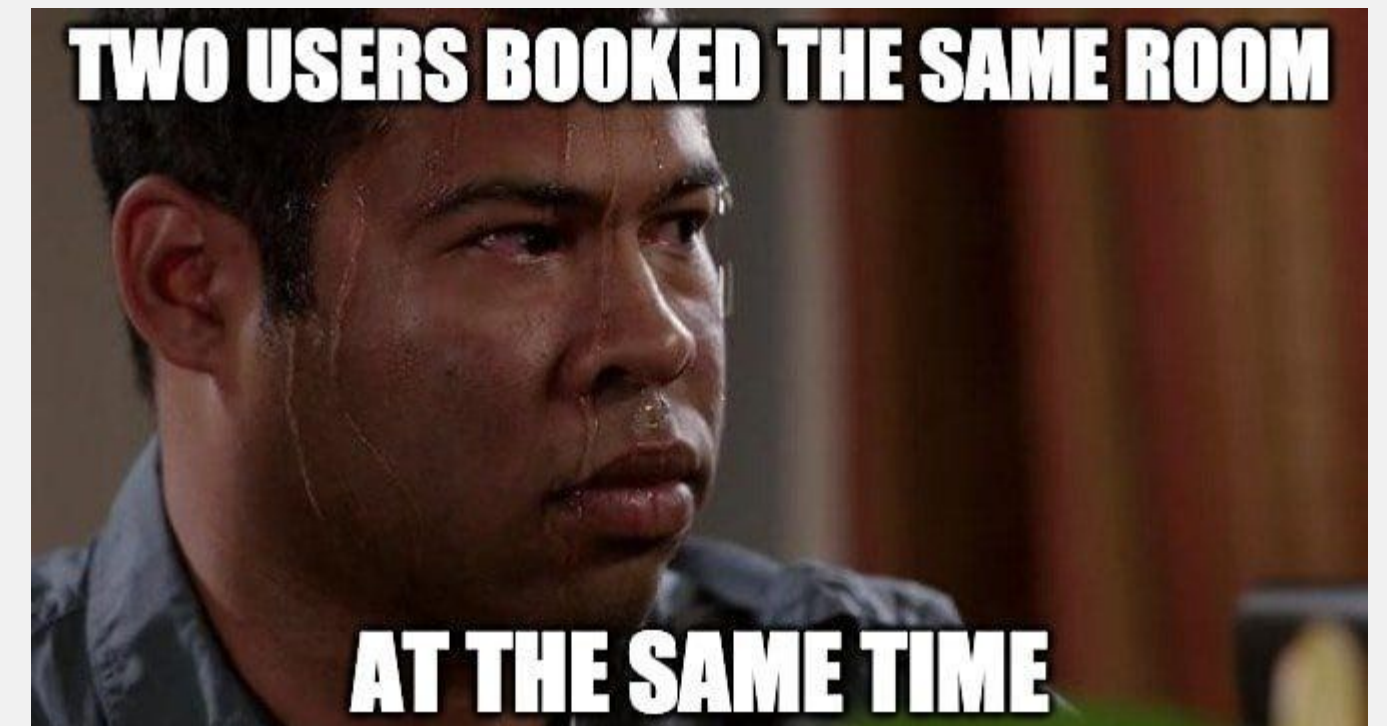
Transactions

- Interview Question
- Atomicity
- Database transaction refers to a sequence of read/write operations that are guaranteed to either succeed or fail as a whole
- If any step fails, transaction is rolled back



Example

- Placing an order in an online store
 - Deduct item quantity from inventory
 - Create an order record
 - Record payment details
- If one operation fails
 - Inventory is reduced
 - No order or payment records are created
- Database is inconsistent



Prisma Transactions

The following query returns all posts that match the provided filter as well as a count of all posts:

```
const [posts, totalPosts] = await prisma.$transaction([
  prisma.post.findMany({ where: { title: { contains: 'prisma' } } }),
  prisma.post.count(),
])
```

You can also use raw queries inside of a `$transaction`:

Relational databases

MongoDB

```
import { selectUserTitles, updateUserName } from '@prisma/client/sql'

const [userList, updateUser] = await prisma.$transaction([
  prisma.$queryRawTyped(selectUserTitles()),
  prisma.$queryRawTyped(updateUserName(2)),
])
```

DAVID LIN

Thank You

CSC309 Week 4

29 JANUARY, 2025