Database

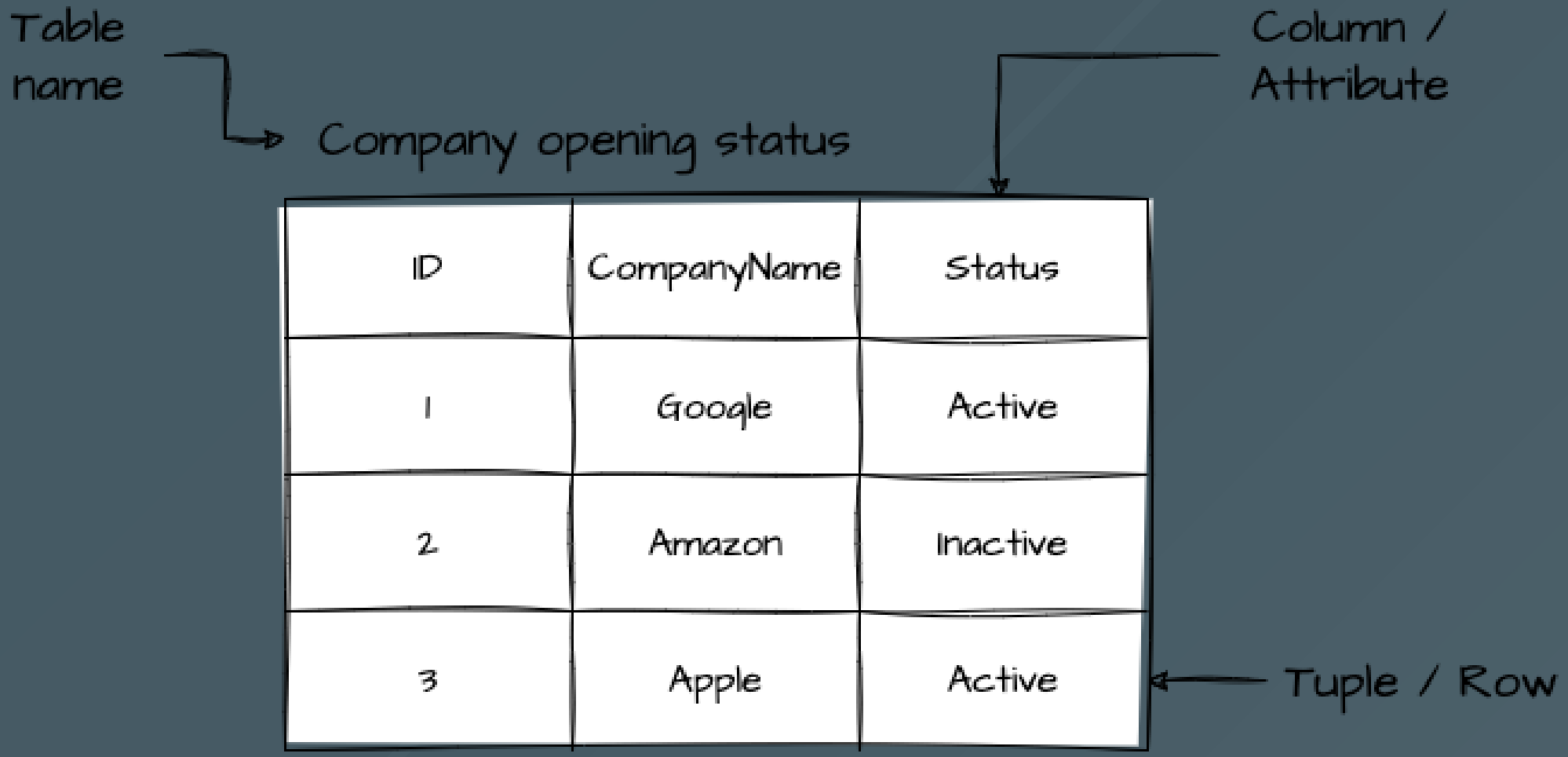
- Database
- Database Management System
- Relational Database
- NoSQL Database

Relational vs NoSQL

- PostgreSQL, SQLite, Oracle, ...
- Data Model
  - Tables
  - Relationships
- Operations
  - CRUD
- Access Control
  - GUI
  - API, ODBC, JDBC, ...

- MongoDB, Redis, Cassandra, ...
- Data Model
  - Documents
  - Collections
- Operations
  - CRUD
- Access Control
  - GUI
  - API, mongoose for JS

Relational Data Model



Company opening status

ID	CompanyName	Status
1	Google	Inactive
2	Walmart	Active
3	Apple	Active

Interviewee

ID	Name	Company
1	Aaron	Apple
2	Alex	Walmart
3	John	Apple

company_id	interviewee_id	Status
3	1	offered
2	2	offered
3	3	pending

DB Design Principles - ACID

ACID
Strong consistency
Pessimistic (lock early)
Vertical scaling
Traditional RDBMS

ACID: Atomicity

> "All or nothing" — transactions complete fully or not at all

```
// Bank transfer example
await client.query('BEGIN');

await client.query('UPDATE accounts SET balance = balance - 100 WHERE id = 1');
await client.query('UPDATE accounts SET balance = balance + 100 WHERE id = 2');
// ⚡ Power failure here?

await client.query('COMMIT'); // Only now are both changes saved
```

ACID: Consistency

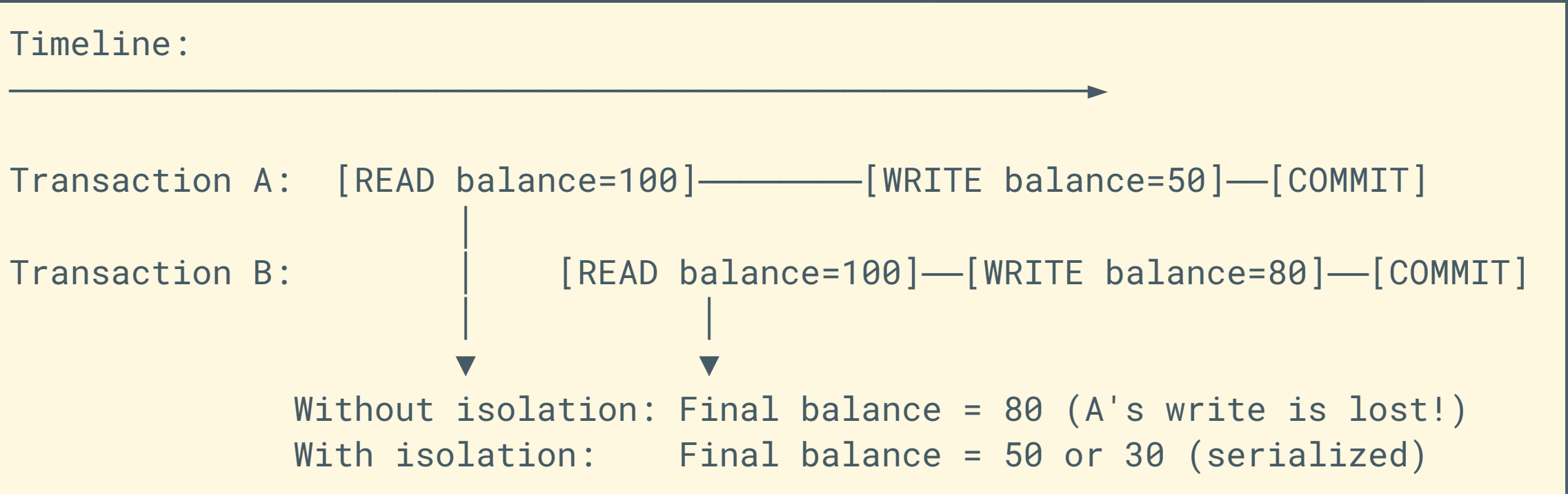
> Data always moves from one valid state to another valid state

```
-- Constraints enforce consistency
CREATE TABLE accounts (
  id SERIAL PRIMARY KEY,
  balance DECIMAL(10,2) CHECK (balance >= 0),   -- No negative balances!
  user_id INTEGER REFERENCES users(id)           -- Must reference valid user
);

-- This will FAIL (constraint violation)
UPDATE accounts SET balance = -50 WHERE id = 1;
-- ERROR: violates check constraint "accounts_balance_check"
```

ACID: Isolation

> Concurrent transactions don't interfere with each other

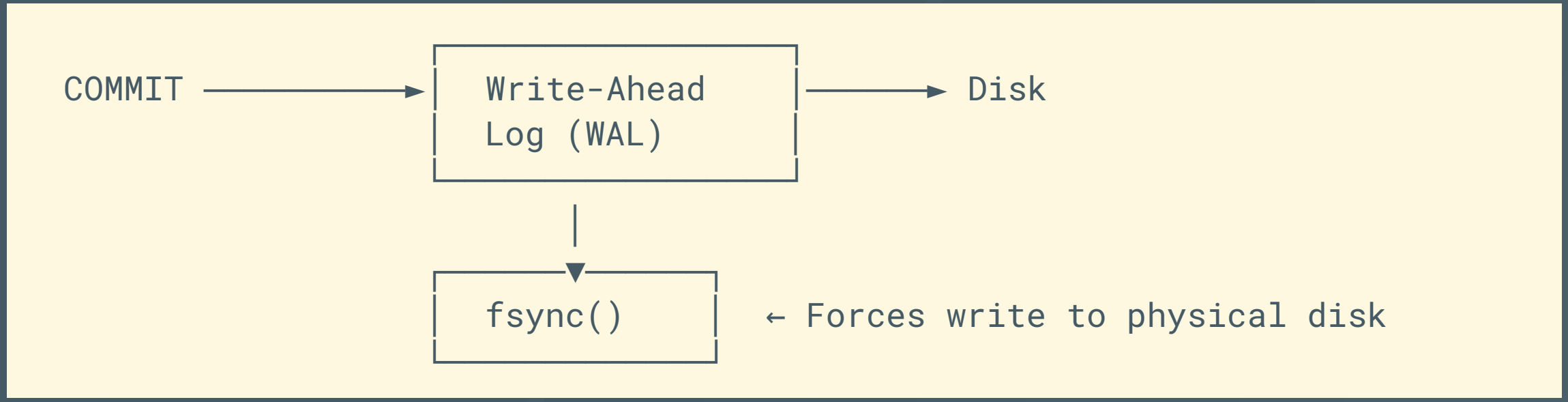


Read Uncommitted → Read Committed → Repeatable Read → **Serializable**

Trade-off: Stronger isolation = lower concurrency

ACID: Durability

> Committed transactions survive system failures



After COMMIT returns:

- Data is on disk (not just in memory) - Survives power loss, crashes, restarts - Can be recovered from transaction log

Database Access

- GUI
  - pgAdmin
  - PostgreSQL for VS Code
- API
  - ODBC
  - JDBC
  - ORM - Object Relational Mapping
    - Hibernate, SQLAlchemy, Sequelize, ...

SQL

- SQL
- SQL Syntax
- SQL Tutorial

```sql
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

```sql
CREATE TABLE table_name (
  column1 datatype,
  column2 datatype,
  column3 datatype,
  ...
);
```

PostgreSQL

- Local Setup
  - PostgreSQL
  - MacOS: Homebrew - `brew install postgresql@15`
  - Docker
    - PostgreSQL
- `psql -U postgres`

PostgreSQL connection with Node.js

sql/index.js

- pg (node-postgres)
- sequelize

NoSQL Database

- Document
  - JSON
- Collection
  - Table
- Document Database
  - MongoDB
  - CouchDB

Document Data Model

https://api.github.com/users/mojombo/repos

Evolution of MongoDB

- Initial NoSQL databases supported read-only operations
- Not good at high volume transactional operations
  - MongoDB 4.0 supports multi-document ACID transactions
- MongoDB has evolved for
  - Foreign key in addition to embedded documents
  - Schema validation
  - Aggregation pipeline
  - Indexes
  - ...

Comparison between MongoDB and PostgreSQL

MongoDB	PostgreSQL
Collection	Table
Document	Row
Field	Column
Index	Index
Embedded Document	Join
Reference	Foreign Key

MongoDB

- Local Setup
  - MongoDB Community Server
  - Docker: MongoDB
- Cloud
  - MongoDB Atlas
- Access to MongoDB
  - GUI: MongoDB Compass
  - API: mongoose

Shared access to MongoDB

You can definitely setup your own MongoDB server either in local or in cloud, and have your own connection access. Or, you can use our shared access to sample MongoDB server I created for this class. I will share the username and password in the Slack channel.

```
mongodb+srv://<username>:<password>@fullstack-
training.gw3nkbl.mongodb.net/<database>
```

Defining a Schema and Model

- Schema

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const userSchema = new Schema({
  name: String,
  email: String,
  created: { type: Date, default: Date.now }
  ...
});

const User = mongoose.model('User', userSchema);
```

MongoDB CRUD

CRUD Operations

- Read
  - `db.collection.find()`
  - `db.collection.findOne()`
- Create
  - `db.collection.insertOne()`
  - `db.collection.insertMany()`
- Update
  - `db.collection.updateOne()`
  - `db.collection.updateMany()`
- Delete
  - `db.collection.deleteOne()`
  - `db.collection.deleteMany()`

MongoDB connection with Node.js

mongo/index.js

- mongoose
- mongoose document

Http Server with MongoDB

http-server-mongodb/index.js

- Setup MongoDB connection
- Setup HTTP server
- Map HTTP requests to CRUD operations
  - get all
  - get one
  - create
  - update
  - delete

How to test the server

- Termial tools
  - curl
  - httpie
- GUI tools
  - Postman
  - Thunder Client for VS Code
  - Talend API Tester