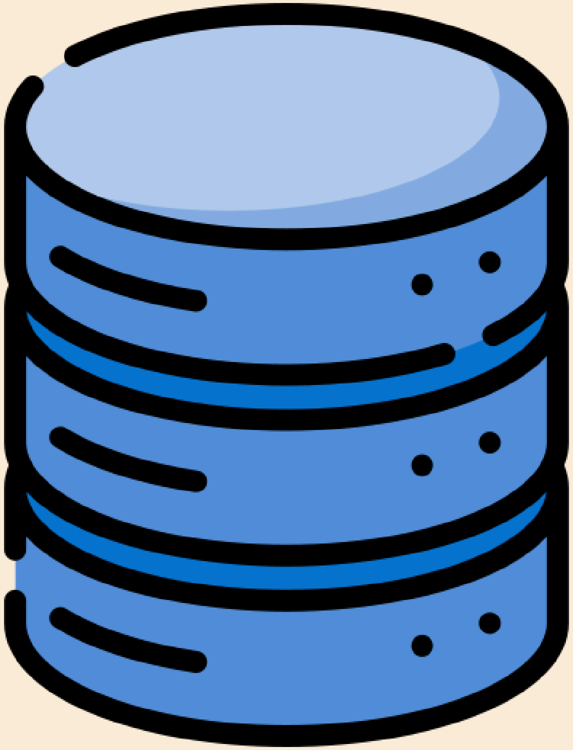
Databases



Homelab Club at UMD Meeting 2025-04-14

Prerequisites

Throughout this talk, we are going to be talk about ways to store data. For this, we need some *example data*:

Employee Number	First Name	Last Name	Department	Phone Number	Date of Hire	Hourly Salary
001	Eugene	Krabs	Management	(555) 123-0003	1984-05-15	9999.99
002	SpongeBob	SquarePants	Fry Cook	(555) 123-0001	1999-07-17	12.50
003	Squidward	Tentacles	Cashier	(555) 123-0002	1999-07-18	11.00

What is a database?

"A database is an organized collection of structured information, or data, typically stored electronically in a computer system."[0]

Databases allow us to store information such as employee or user data/accounts, product lists, etc.

[0] https://www.oracle.com/database/what-is-database/

What is a Database Management System (DBMS)?

Software program to serve as an interface between the database and its end users or programs. Allows for the management (e.g. retrievals, updates, etc.) and administration of databases.

Presentation Overview

1. Storing Data in Files
2. Relational Databases
3. Non-relational Databases

Storing Data in Files

Let's start with what our first *primitive solution* might be to this problem of storing data.

We *could*, create a new *file* and put the data straight in there. We would likely want to keep it organized, and for that there are a few common formats such as JSON, CSV, XML, YAML, etc.

Note: Some text based formats are also frequently used for configuration files.

Storing Data in Files (cont.)

Pros:

- *Very low spin-up cost*: other than copying the syntax of the type you choose, you do not need to know or learn how to create and manage a database
- Can store without having to install *additional software*
- Can use commands you might *already know* such as `cat` and `grep` to look at or search through the data.

Cons:

- Easy to *misconfigure* (e.g. accidentally deleting a property, or even missing a singular comma)
- No *management or data manipulation features* (must be hand edited and hand edited *correctly*, or code/scripts written to complete the desired change)
- No concurrent connections, *data integrity*
- *Slower* data retrieval and potentially *more storage space* used as opposed to an optimized database

Storing Data in Files (cont.)

While text files are great for personal use or small projects, they *do not scale well* and are not appropriate for more *persistent* or larger scale projects.

Example:

- If you were doing a *quick audit* of the Krusty Krab's employees and looking at some data about them, a CSV or a spreadsheet might work for that purpose.
- But to store employee data and records for the Krusty Krab's *long term operational needs*, you would want something more stable, robust, and with more features (a database).

Storing Data in Files: JSON

JavaScript Object Notation

File extension(s): .json

```
[
  {
    "Employee Number": "001",
    "First Name": "Eugene",
    "Last Name": "Krabs",
    "Department": "Management",
    "Phone Number": "(555) 123-0003",
    "Date of Hire": "1984-05-15",
    "Hourly Salary": 9999.99
  },
  {
    "Employee Number": "002",
    "First Name": "SpongeBob",
    "Last Name": "SquarePants",
    "Department": "Fry Cook",
    "Phone Number": "(555) 123-0001",
    "Date of Hire": "1999-07-17",
    "Hourly Salary": 12.50
  },
  {
    "Employee Number": "003",
    "First Name": "Squidward",
    "Last Name": "Tentacles",
    "Department": "Cashier",
    "Phone Number": "(555) 123-0002",
    "Date of Hire": "1999-07-18",
    "Hourly Salary": 11.00
  }
]
```

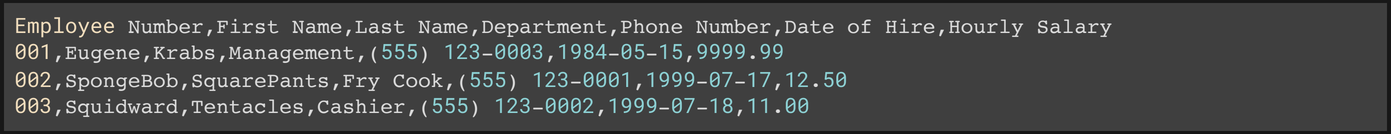
See also: https://www.json.org/json-en.html

Storing Data in Files: CSV

Comma Separated Values

File extension(s): `.csv`

Can be read by spreadsheet applications (e.g. Excel, Google Sheets)

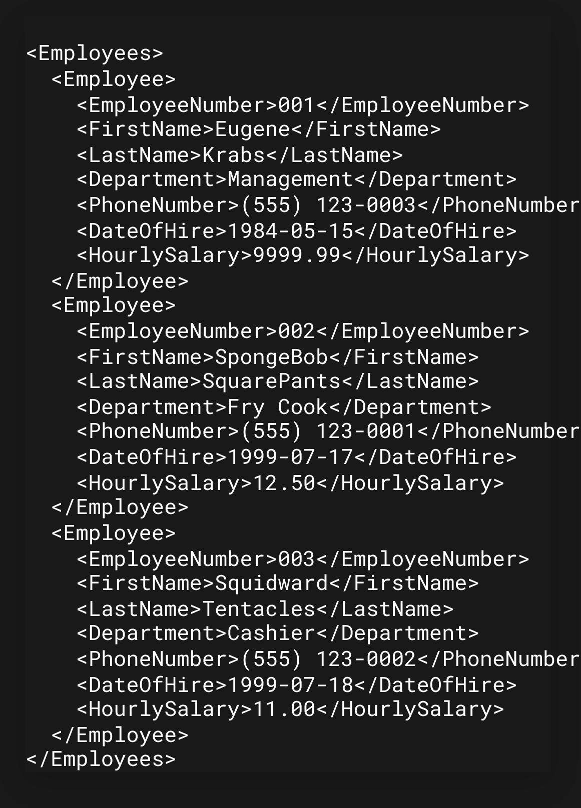


```
Employee Number,First Name,Last Name,Department,Phone Number,Date of Hire,Hourly Salary
001,Eugene,Krabs,Management,(555) 123-0003,1984-05-15,9999.99
002,SpongeBob,SquarePants,Fry Cook,(555) 123-0001,1999-07-17,12.50
003,Squidward,Tentacles,Cashier,(555) 123-0002,1999-07-18,11.00
```

Storing Data in Files: XML

*Extensible **Markup** Language*

File extension(s): .xml

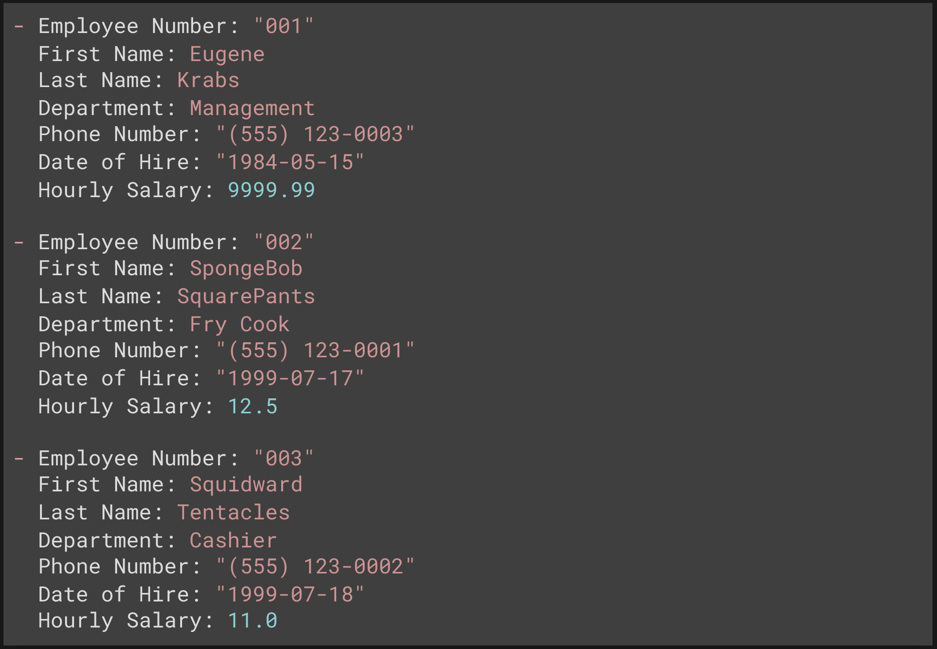


```xml
<Employees>
  <Employee>
    <EmployeeNumber>001</EmployeeNumber>
    <FirstName>Eugene</FirstName>
    <LastName>Krabs</LastName>
    <Department>Management</Department>
    <PhoneNumber>(555) 123-0003</PhoneNumber>
    <DateOfHire>1984-05-15</DateOfHire>
    <HourlySalary>9999.99</HourlySalary>
  </Employee>
  <Employee>
    <EmployeeNumber>002</EmployeeNumber>
    <FirstName>SpongeBob</FirstName>
    <LastName>SquarePants</LastName>
    <Department>Fry Cook</Department>
    <PhoneNumber>(555) 123-0001</PhoneNumber>
    <DateOfHire>1999-07-17</DateOfHire>
    <HourlySalary>12.50</HourlySalary>
  </Employee>
  <Employee>
    <EmployeeNumber>003</EmployeeNumber>
    <FirstName>Squidward</FirstName>
    <LastName>Tentacles</LastName>
    <Department>Cashier</Department>
    <PhoneNumber>(555) 123-0002</PhoneNumber>
    <DateOfHire>1999-07-18</DateOfHire>
    <HourlySalary>11.00</HourlySalary>
  </Employee>
</Employees>
```

Storing Data in Files: YAML

YAML Ain't Markup Language
(was originally: **Y**et **A**nother **M**arkup **L**anguage)

File extension(s): .yaml, .yml



```yaml
- Employee Number: "001"
  First Name: Eugene
  Last Name: Krabs
  Department: Management
  Phone Number: "(555) 123-0003"
  Date of Hire: "1984-05-15"
  Hourly Salary: 9999.99

- Employee Number: "002"
  First Name: SpongeBob
  Last Name: SquarePants
  Department: Fry Cook
  Phone Number: "(555) 123-0001"
  Date of Hire: "1999-07-17"
  Hourly Salary: 12.5

- Employee Number: "003"
  First Name: Squidward
  Last Name: Tentacles
  Department: Cashier
  Phone Number: "(555) 123-0002"
  Date of Hire: "1999-07-18"
  Hourly Salary: 11.0
```

See also: https://yaml.org/

Types of Databases

Relational Databases

Store data in *tables* (rows, columns)

Establish *relationships* between tables (unique ID for each **row** in a table)

Structured schema: you define the structure (what kind of data goes into which column) up front

Compliance: ACID (Atomicity, Consistency, Isolation, Durability) - rules to ensure data stays accurate, even if something goes wrong

SQL (Structured Query Language): Used to interact with (search, update, delete, etc.) the database

Also known as: *SQL Databases*

Relational Database Management Systems (RDBMS)

Examples:

MySQL: open-source (repository)

PostgreSQL: open-source (repository)

SQLite: open-source (repository)

Oracle Database / Oracle DBMS: proprietary

Microsoft SQL Server: proprietary

Relational Databases: Use Cases

You have *consistently structured data*

You need *consistency* and *accuracy*

e.g. financial data / banking systems, HR & payroll, customer records, inventory / e-commerce product listings and orders, school or university course registration, etc.

Relational Databases: MySQL

Create a database:

```
CREATE DATABASE KrustyKrab;
```

Define a table:

```
CREATE TABLE KrustyKrab.Employees (
  employee_number VARCHAR(3) PRIMARY KEY,
  first_name VARCHAR(50),
  last_name VARCHAR(50),
  department VARCHAR(50),
  phone_number VARCHAR(20),
  date_of_hire DATE,
  hourly_salary DECIMAL(10, 2)
);
```

Insert some data:

```
INSERT INTO KrustyKrab.Employees VALUES
('001', 'Eugene', 'Krabs', 'Management', '(555) 123-0003', '1984-05-15', 9999.99),
('002', 'SpongeBob', 'SquarePants', 'Fry Cook', '(555) 123-0001', '1999-07-17', 12.50),
('003', 'Squidward', 'Tentacles', 'Cashier', '(555) 123-0002', '1999-07-18', 11.00);
```

Query some data:

```
mysql> SELECT first_name, last_name FROM KrustyKrab.Employees WHERE hourly_salary > 12.00;
+------------+-------------+
| first_name | last_name   |
+------------+-------------+
| Eugene     | Krabs       |
| SpongeBob  | SquarePants |
+------------+-------------+
2 rows in set (0.00 sec)

mysql>
```

Non-relational Databases

Designed to handle more *flexible* or large-scale data needs

No rigid table structure

Schema-less or *dynamic schema*

Sub-types:

- *Document*
- *Key-Value*
- *Column*
- *Graph*

Also known as: *NoSQL Databases*

Non-relational Databases: Document Databases

Data stored in document formats like *JSON* or *BSON*.

Each document is a self-contained unit.

Example:

```
{
    "Employee Number": "001",
    "First Name": "Eugene",
    "Last Name": "Krabs",
    "Department": "Management",
    "Phone Number": "(555) 123-0003",
    "Date of Hire": "1984-05-15",
    "Hourly Salary": 9999.99
}
```

Non-relational Databases: Document Databases - Examples

Examples:

MongoDB: source-available (repository)

Couchbase: source-available (repository)

Apache CouchDB: open-source (repository)

Amazon DocumentDB: proprietary

Non-relational Databases: Document Databases - Use Cases

e.g. content management systems, e-commerce catalogs, mobile/web apps, etc.

Non-relational Databases: Document Databases - MongoDB

Create or switch to a database:

```
use KrustyKrab
```

Insert some data:

```
db.employees.insertMany([
  {
    employee_number: "001",
    first_name: "Eugene",
    last_name: "Krabs",
    department: "Management",
    phone_number: "(555) 123-0003",
    date_of_hire: ISODate("1984-05-15T00:00:00Z"),
    hourly_salary: 9999.99
  },
  {
    employee_number: "002",
    first_name: "SpongeBob",
    last_name: "SquarePants",
    department: "Fry Cook",
    phone_number: "(555) 123-0001",
    date_of_hire: ISODate("1999-07-17T00:00:00Z"),
    hourly_salary: 12.50
  },
  {
    employee_number: "003",
    first_name: "Squidward",
    last_name: "Tentacles",
    department: "Cashier",
    phone_number: "(555) 123-0002",
    date_of_hire: ISODate("1999-07-18T00:00:00Z"),
    hourly_salary: 11.00
  }
]);
```

Query some data:

```
KrustyKrab> db.employees.find({ hourly_salary: { $gt: 12.00 } }, {first_name: 1, last_name: 1, _id: 0});
[
  { first_name: 'Eugene', last_name: 'Krabs' },
  { first_name: 'SpongeBob', last_name: 'SquarePants' }
]
```

Non-relational Databases: Key-Value Stores

Data stored as *key-value pairs*

Non-relational Databases: Key-Value Stores - Examples

Examples:

Redis: source-available (repository)

Memcached: open-source (repository)

Amazon DynamoDB: proprietary

Non-relational Databases: Key-Value Stores - Use Cases

e.g. caching, session storage, user preferences, etc.

Non-relational Databases: Wide-Column Store

Data stored in *tables* (rows, columns), but in contrast to *relational databases*, the names and formats of the columns can *change/vary from row to row*.

Also known as: *Extensible Record Store*

Non-relational Databases: Wide-Column Store - Examples

Examples:

Apache Cassandra: open-source (repository)

Apache HBase: open-source (repository)

Google Cloud Bigtable: proprietary

Non-relational Databases: Wide-Column Store - Use Cases

e.g. time-series data, analytics platforms, IoT (Internet of Things), etc.

Non-relational Databases: Graph Databases

Data stored as a *graph* (nodes, edges)

Node: data
Edge: relationship between data

Non-relational Databases: Graph Databases - Examples

Examples:

Neo4j: open-source (repository) (also has proprietary version)

ArangoDB: source-available (repository)

Amazon Neptune: proprietary

Non-relational Databases: Graph Databases - Use Cases

You have heavily inter-connected data

e.g. social networks, recommendation engines, fraud detection, etc.

Other Databases: Time-Series Databases

Data stored as *times* and *values*

Optimized for tracking data over time

Other Databases: Time-Series Databases - Examples

Examples:

TimescaleDB: open-source (repository)

InfluxDB: proprietary

Prometheus: open-source (repository) (software used for event monitoring and alerting, has its own implementation of a time-series database)

Other Databases: Time-Series Databases - Use Cases

e.g. temperature readings, stock prices, app metrics, etc.

Quick Comparison

Database Type	Use Cases
Relational	Structured data, strong consistency
Document	Flexible data, hierarchical structures
Key-Value	Speed, simple data pairs
Column-Family	Large-scale analytics, time-series
Graph	Complex relationships
Time-Series	Tracking changes over time
Cloud/Serverless	Scalable web/mobile apps