Modeling Our World With Data

CSC301

Announcements

1. D1 & team signups on Quercus
2. Assignment 2 will be released soon

Context for This Lecture

- How will you work with data in when building your software?
- What tools do you have available to you?
- How will you model the world?
- Can you manipulate data without learning a database management system?

David, I don’t know anything about databases! I haven’t taken 343. I’m worried I won’t get this

Agenda

- Data Modeling
  - Conceptual, Logical, and Physical Modeling
- Serialization & Persistence
  - Binary
  - Text
  - XML
  - JSON
- Data and Code
  - Data Access Object (DAO) Pattern
  - Object-Relational Management (ORM) Guidance
- RDBMS and NoSQL
  - Brief intro on SQL
  - Examine use cases for each

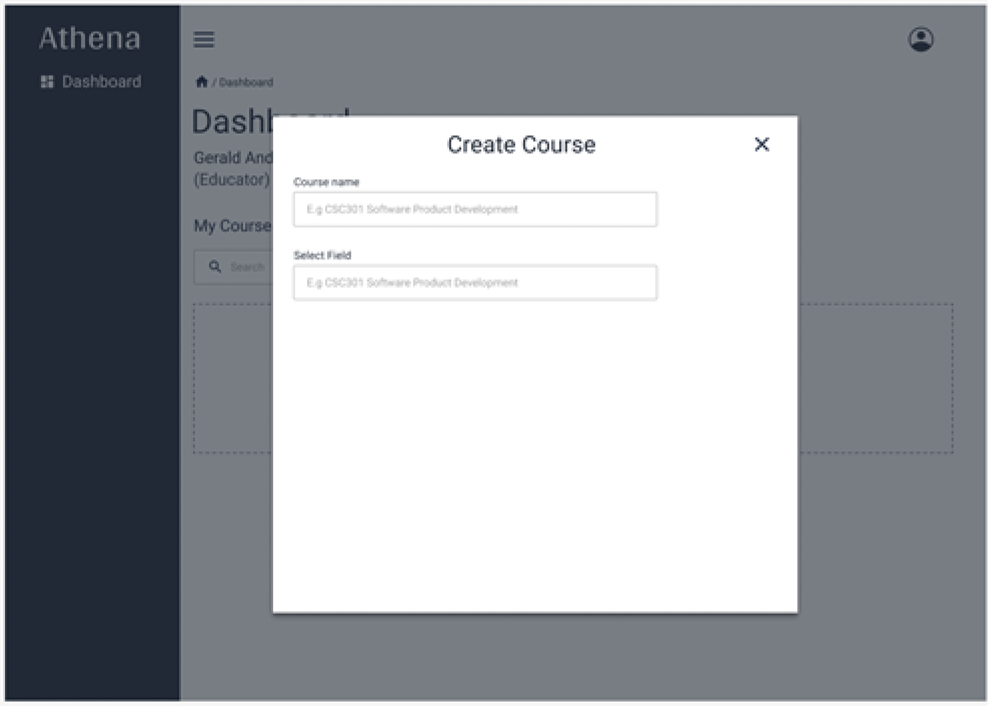
Heavy focus on Relational Databases here, but most guidance is equally applicable to NoSQL

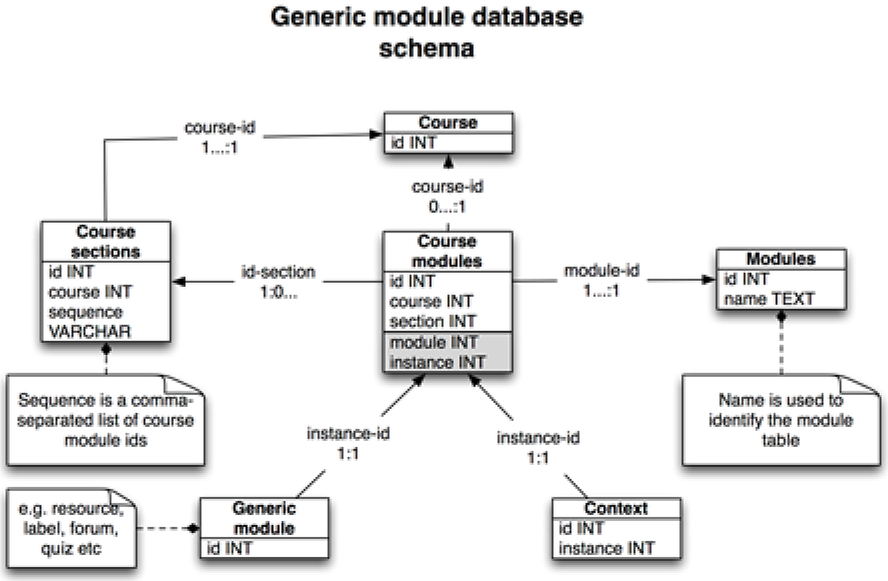
Why Now?

- Now that you know what problem you are solving, you should know how to capture it and build the software for it.

How do you go from user stories to building software?

Two Parallel Paths





Data Models

Conceptual Data Model

- The Conceptual Data Model (CDM) of a system defines
  - The entities whose state must be persisted
  - Key attributes of entities (optional)
  - Relationships between entities (1:1, 1:many, many:many)
- CDM is a business-level artifact
  - if a businessperson can't understand your CDM, it's not a CDM
- How do you construct a CDM?
  - Stakeholder interviews – their business language will form the basis of your CDM
  - User stories – **nouns** in stories
    - As an educator, I want to review **proposals** so that I can select great proposals for students

Objects in your data model

Student

- First name
- Last Name
- Birth Date
- UTORID

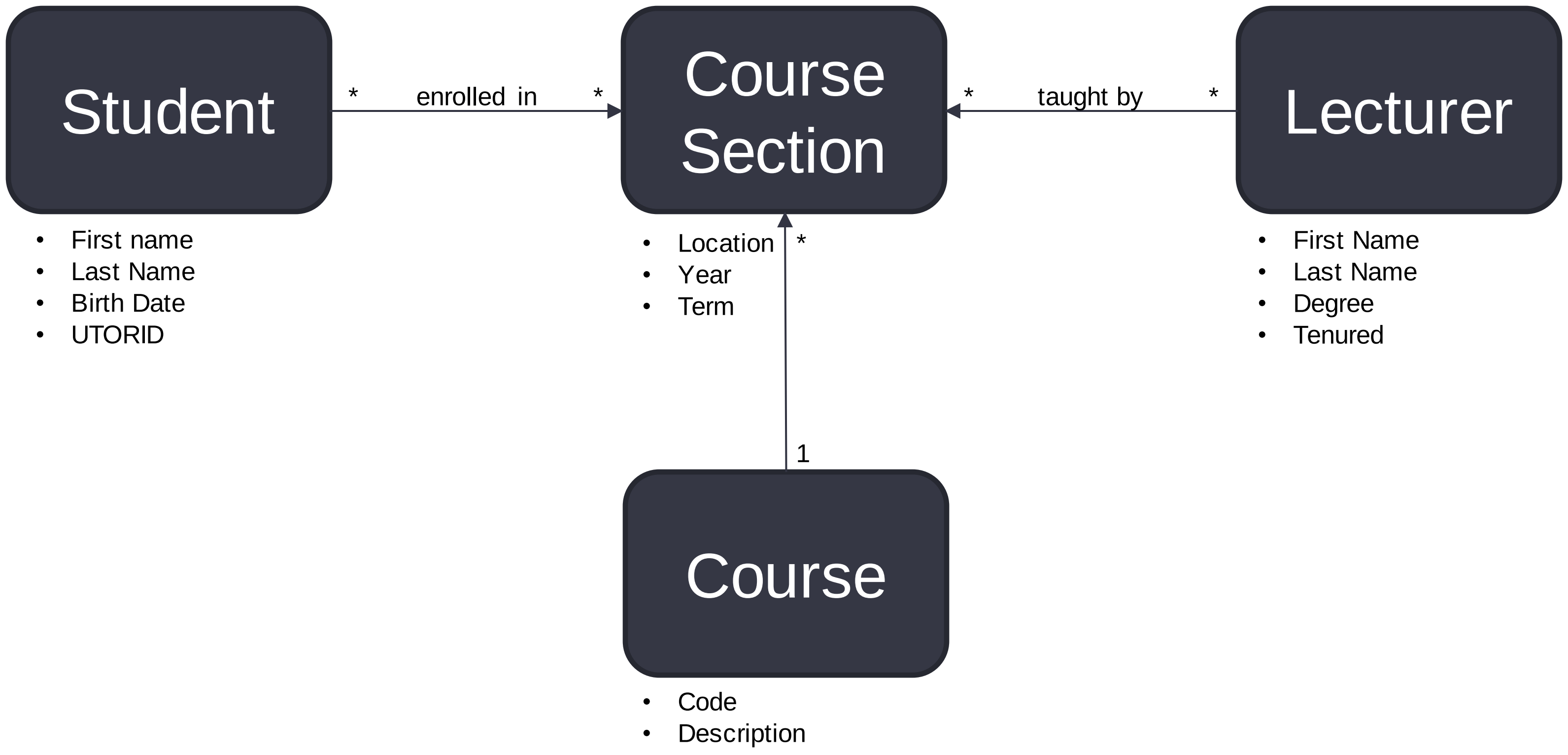
Course

- Code
- Description

Lecturer

- First Name
- Last Name
- Degree
- Tenured

CDM Example


Student
*
enrolled in
*
Course Section
*
taught by
*
Lecturer
*
1
Course
• First name
• Last Name
• Birth Date
• UTORID
• Location
• Year
• Term
• First Name
• Last Name
• Degree
• Tenured
• Code
• Description

Exercise 1

Take 5 minutes to identify the objects for your project based on looking at your user stories

Logical Data Model

- The Logical Data Model (LDM) is a platform independent, ***normalized*** data model
- LDM comprised of tables
  - 1-1 mapping between CDM entities and LDM tables
    - There will be other tables!
  - Tables store information about instances of an entity (like Objects are instances of Classes)
  - Tables have a primary key (set of one or more columns that uniquely identify an entity instance)
  - Tables can have foreign keys (PK of related table)
- Normalized?
  - We want to ensure that each fact is represented in the data model exactly once. Bad things happen when this is violated

Practical Normalization

- Exists to ensure we do not encounter anomalies in persisted data
    - For example, your name stored separately in multiple places within the system
- Can get incredibly complicated! We'll focus on what you need to know
- 1st Normal Form (1NF)
    - Make sure your tables have a unique ID. Use a synthetic ID for entity tables.
    - No repeating groups within or across columns. Introduce a related table instead
- 2nd Normal Form (2NF)
    - All attributes must depend on all columns of the primary key. You won't see this a lot given our guidance for 1NF
- More normal forms…

Let’s learn more about normalization

Practical Normalization

Not 1NF

Student	
ID	Enrollments
1	CSC301-LEC0501 CSC302-LEC0301 CSC303-LEC0101

Not atomic! Multiple data points in one field

1NF

Student
ID
1

Enrollments	
Student ID	Course Section
1	CSC301…
1	CSC302…
1	CSC303…

Not 2NF

Enrollments		
Student ID	Course Section	Location
1	2	Bahen

Location has nothing to do with student. Could wind up with same section in two locations?

2NF

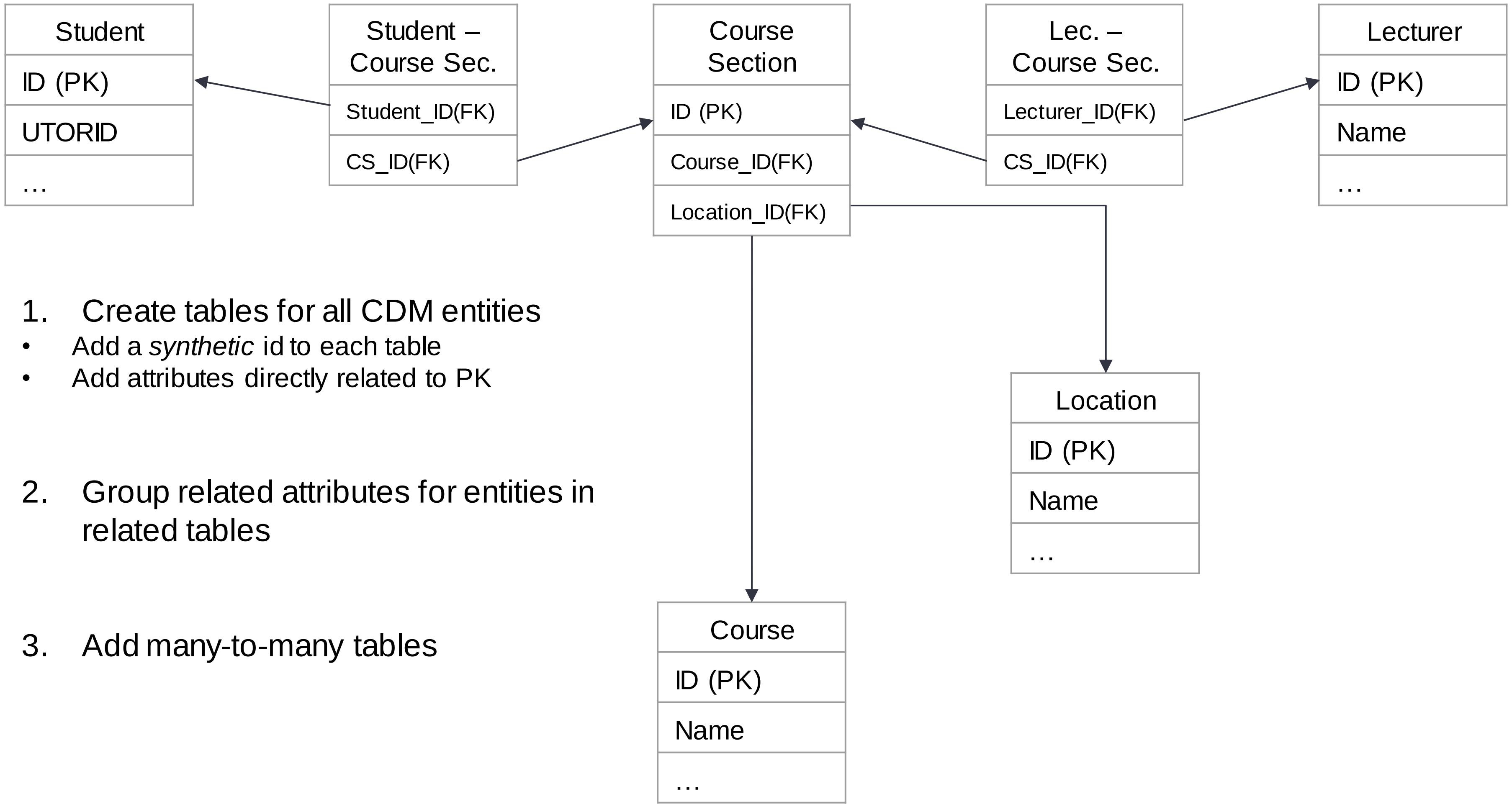
Enrollments	
Student ID	Course Section
1	2

Section Location	
Course Section	Location Name
2	Bahen

Exercise 2

Take another 5 minutes to start normalizing your database

LDM Example



1. Create tables for all CDM entities
   - Add a *synthetic* id to each table
   - Add attributes directly related to PK

2. Group related attributes for entities in related tables

3. Add many-to-many tables

How Do We Store And Move Data?

Serialization

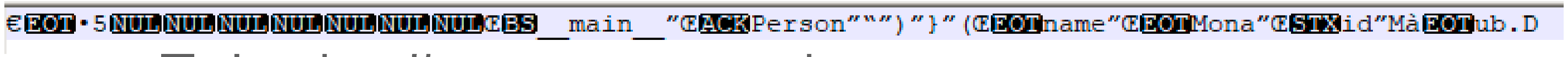
- *Serialize*
  - Convert in-memory objects into data so that it can be:
    - Written/saved somewhere
    - Streamed across a communication link
  - In this context, *data* (usually) means a string or an array of bytes
- *Deserialize*
  - Convert (serialized) data into in-memory objects

Serialize To Text

- How can we overcome the limitations of Java’s built-in serialization?
- One option is to write objects to text
  - Convenient debugging - Text is human-readable
  - Flexibility - Easily define custom serialization
  - Convenient deserialization - Read text, then determine the type of the object
  - Cross-language - Every programming language is capable of text processing
  - Comma Separated Values (CSV) common

Common Methods

- Language specific (e.g., Java built-in)
  - Binary
    - Pickle in Python



```
€EOT•5NULNULNULNULNULNULNULŒBS__main__”ŒACKPerson”“”)”}”(ŒEOTname”ŒEOTMona”ŒSTXid”MàEOTub.D
```

    - Faster & more compact
- Text-based
  - Text file
  - CSV
  - XML
  - JSON
- Databases

eXtensible Markup Language (XML) Example

```xml
<?xml version="1.0"?>
<catalog>
    <product category="books">
        <title>Clean Code</title>
    </product>
    <product category="books">
        <title>GoF - Design Patterns</title>
    </product>
    <product category="music">
        <title>Abbey Road</title>
        <artist>The Beatles</artist>
        <year>1969</year>
    </product>
</catalog>
```

- Tag names are user defined
  (That's the eXtensible part of XML)
- White spaces are ignored
  Can be added for pretty-printing

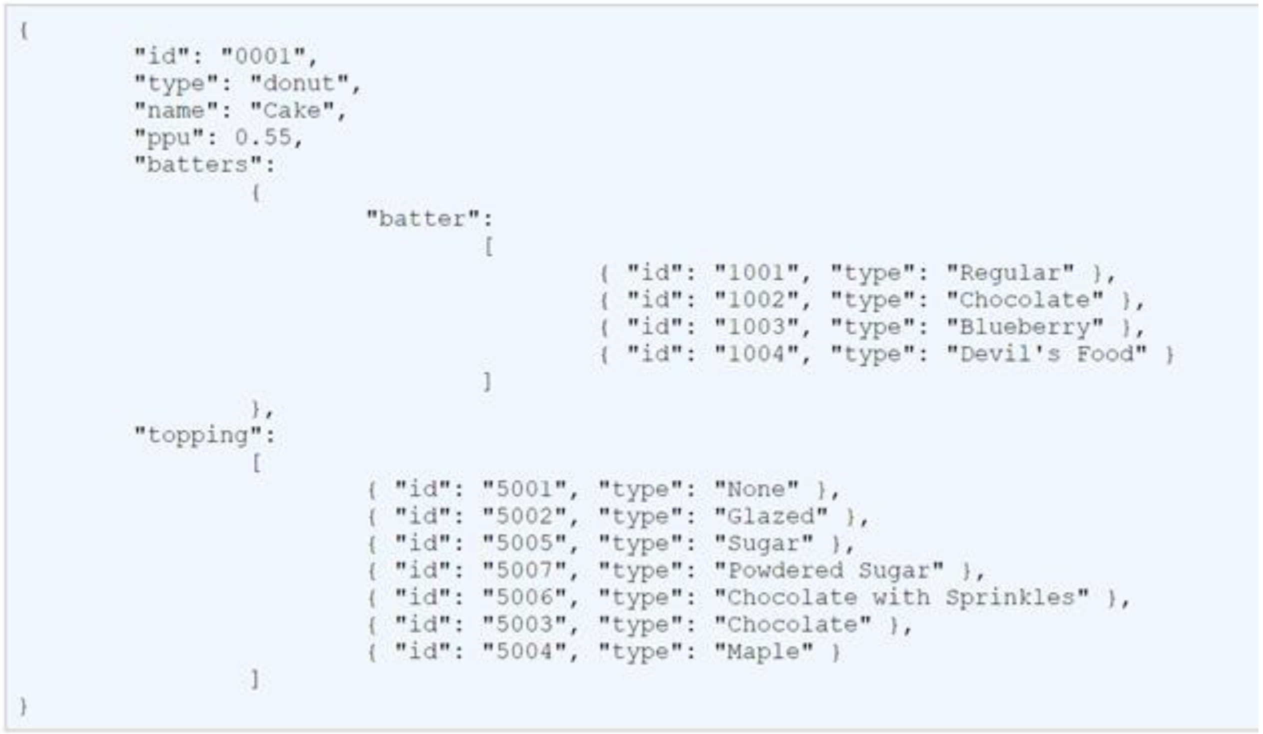
JSON

- **JSON** (JavaScript Object Notation)
  - **lightweight data-interchange format**.
  - easy for **humans** to read and write & easy for **machines** to parse and generate
- completely language independent
- Light, cross-language text format

Example: {"name": "GoF", "category": “books”}

https://www.json.org/

JSON Example



```
{
        "id": "0001",
        "type": "donut",
        "name": "Cake",
        "ppu": 0.55,
        "batters":
                {
                        "batter":
                                [
                                        { "id": "1001", "type": "Regular" },
                                        { "id": "1002", "type": "Chocolate" },
                                        { "id": "1003", "type": "Blueberry" },
                                        { "id": "1004", "type": "Devil's Food" }
                                ]
                },
        "topping":
                [
                        { "id": "5001", "type": "None" },
                        { "id": "5002", "type": "Glazed" },
                        { "id": "5005", "type": "Sugar" },
                        { "id": "5007", "type": "Powdered Sugar" },
                        { "id": "5006", "type": "Chocolate with Sprinkles" },
                        { "id": "5003", "type": "Chocolate" },
                        { "id": "5004", "type": "Maple" }
                ]
}
```

Why JSON

- A great light-weight solution to standardization and integration
  - We need to simple solution maintain data structure
    - CSVs are too limiting
    - Because XML sucks
- Straight Through Processing
  - JSON - Angular -> Express -> Node -> MongoDB (not a recommendation, just saying it's possible)
  - AKA MEAN architecture

JSON Structure

JSON top-level structures are:

- Object – unordered set of name value pairs
- Array – ordered collection of values

Values in JSON are:

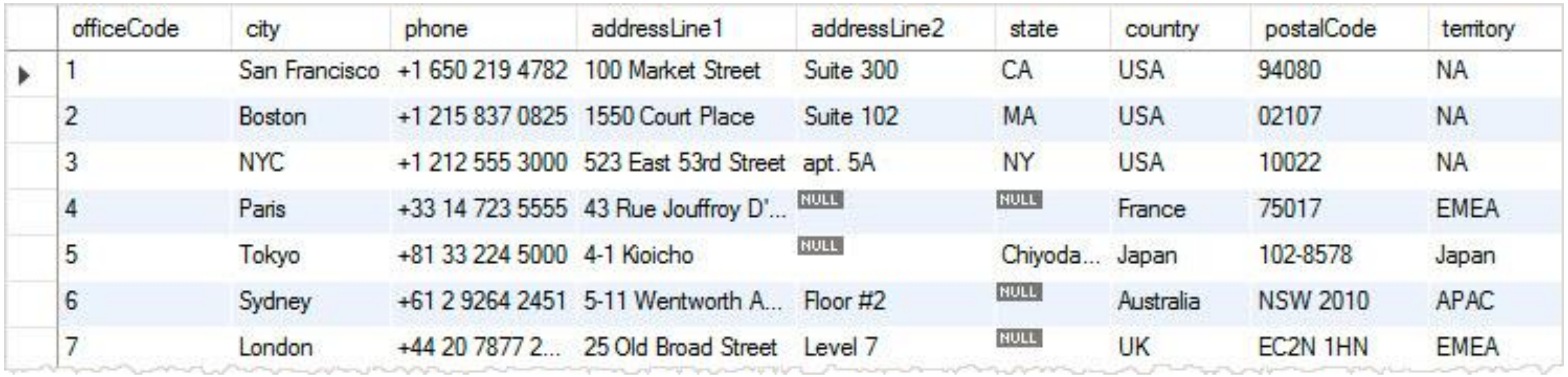
- String
- Boolean
- Object (aka maps, dictionaries, hashes, etc.)
- Array
- Number
- Null

More JSON Examples

- Basic W3 School Examples
- Google Maps API JSON Example (Simple)
- Twitter JSON Example (Complex)

Databases (SQL and NoSQL)

Relational Table Example



	officeCode	city	phone	addressLine1	addressLine2	state	country	postalCode	territory
▶	1	San Francisco	+1 650 219 4782	100 Market Street	Suite 300	CA	USA	94080	NA
	2	Boston	+1 215 837 0825	1550 Court Place	Suite 102	MA	USA	02107	NA
	3	NYC	+1 212 555 3000	523 East 53rd Street	apt. 5A	NY	USA	10022	NA
	4	Paris	+33 14 723 5555	43 Rue Jouffroy D'...	NULL	NULL	France	75017	EMEA
	5	Tokyo	+81 33 224 5000	4-1 Kioicho	NULL	Chiyoda...	Japan	102-8578	Japan
	6	Sydney	+61 2 9264 2451	5-11 Wentworth A...	Floor #2	NULL	Australia	NSW 2010	APAC
	7	London	+44 20 7877 2...	25 Old Broad Street	Level 7	NULL	UK	EC2N 1HN	EMEA

What if you need fields from multiple tables?

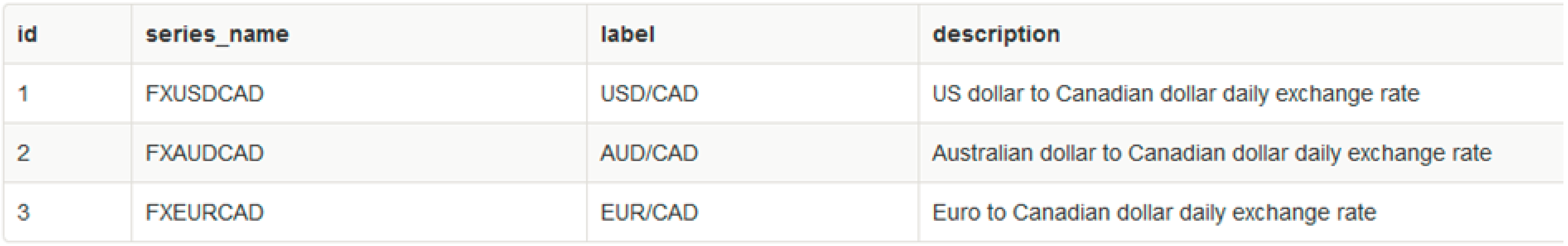
SQL Resources

1. SQL Bolt: Great starting point for those with minimal SQL background
2. SQLZoo: Good refresher for those with basic knowledge and needing a review
3. SQLFiddle with them For those who want to fiddle with some tables
4. W3 Schools: More technical and syntax focused learning
5. Select * SQL: Absolute beginner’s intro with real examples and stories
6. CodeAcademy: 8-hour course on SQL
7. For those with CS background and interest in databases
    a. Leetcode:
    b. HackerRank

Exercise 3

Take 5 minutes to explore the resources above and identify one that can help you

Tables vs. Documents



id	series_name	label	description
1	FXUSDCAD	USD/CAD	US dollar to Canadian dollar daily exchange rate
2	FXAUDCAD	AUD/CAD	Australian dollar to Canadian dollar daily exchange rate
3	FXEURCAD	EUR/CAD	Euro to Canadian dollar daily exchange rate

```
"seriesDetail": {
        "FXUSDCAD": {
            "label":
"USD/CAD",
            "description":
"US dollar to Canadian dollar
daily exchange rate"
        }
    }
```

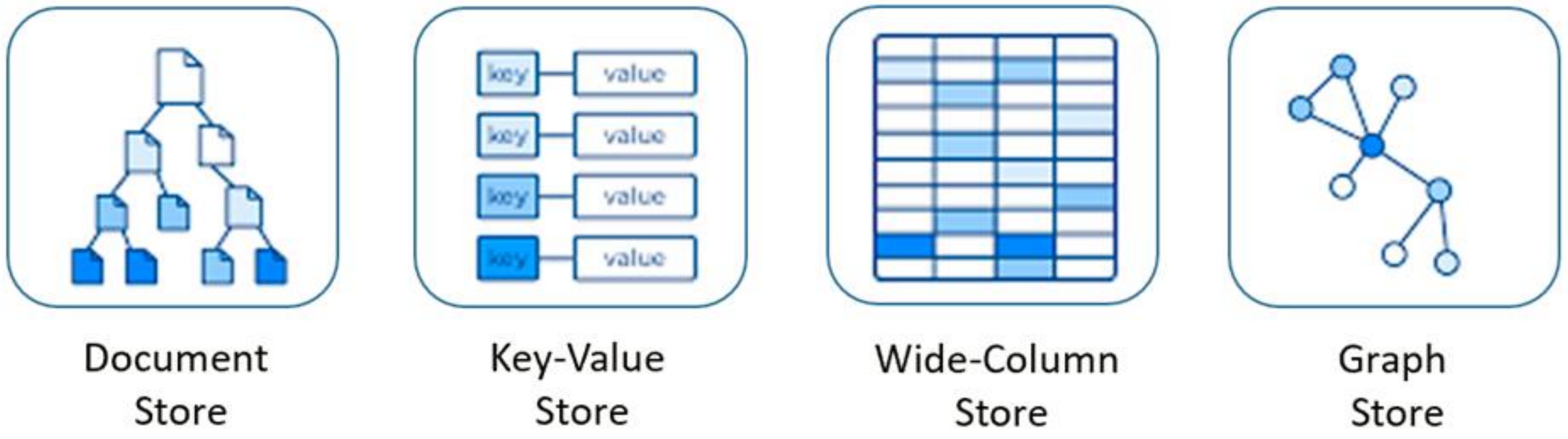
```
"seriesDetail": {
        "FXAUDCAD": {
            "label":
"AUD/CAD",
            "description":
"Australian dollar to
Canadian dollar daily
exchange rate"
        }
    }
```

```
"seriesDetail": {
        "FXUSDCAD": {
            "label":
"ERU/CAD",
             "description":
"EUR to Canadian dollar daily
exchange rate",
            "dimension": {
                "key": "d",
                "name":
"date"
            }
        }
    }
```

RDBMS and NoSQL

	SQL	**NoSQL**
Type	Relational	Non-relational
Data	Structured, tables	Unstructured documents (JSON files)
Schema	Static/rigid	Dynamic (different documents can have different properties)
Joins	Complex queries	No joins
Language	*Structured* Query Language	Un-structured
Benefits	Fast writes to tables, complex relational queries	Fast reads of objects directly - no need to join tables together first Easier to scale

Types of NoSQL Databases



Document: dynamic scalable data
Key-value: simple key-value structure
Wide-column: analytics
Graph: where connections and relationships matter

A more in-depth breakdown

Choosing Relational or NoSQL

- RDBMS is a good default choice
    - Proven, easy to hire for, easy to program, SQL is the *lingua franca* language of data (hence why nearly every NoSQL system has SQL now)
    - Enterprise options: Microsoft SQL Server, Oracle
    - Open Source: PostgreSQL, MySQL, MariaDB
- Choose NoSQL only in presence of specific, observed architectural requirements
    - Scale is usually what drives engineering decisions here
    - Query performance on VERY LIMITED set of queries is another factor

Can we delay the decision?

Can we live without SQL?

You want to build an application but

- Don’t know which database to use
- Or it may change later
- Or you don’t want to learn SQL!

How can you move forward with building your application?

Working with Data In Code

- So far, we've learned how to model the conceptual, logical, and physical state of our data
- Now we're going to focus on how to work with data in code. We'll examine
    - How to properly isolate data access code from the rest of your application
    - How to implement data access code
    - How to optimize for performance

The *Data Access Object* Design Pattern

- *Data Access Object*
  - Define data-access methods in an interface
  - Implement the DAO interface for different data stores
  - Code is written against the interface, without knowing/caring about the underlying data store.
- DAO is a design pattern for abstracting the details of an underlying data store
  - Allows us to write database-agnostic applications

Data Access Object

We don't care where the data from `loadTweets()` comes from

```
public static void example(ITweetEngine dao){
        Iterator<ITweet> itr = dao.loadTweets();
        while(itr.hasNext()){
                System.out.println(itr.next());
        }
}
```

- Application code is not responsible for persisting data
- Application code does not depend on a specific data store
- DAO methods use domain language (e.g., loadTweets)

DAO Example(1)

```
public class Employee {
    private int id;
    private String name;

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

DAO Example(2)

```
interface EmployeeDAO {

    List<Employee> findAll();
    List<Employee> findById(int Id);
    List<Employee> findByName(String name);
    boolean insertEmployee(Employee employee);
    boolean updateEmployee(Employee employee);
    boolean deleteEmployee(Employee employee);

}
```

DAO Example(3)

```
SQLEmployeeDAO implements EmployeeDAO {
    @override
    List<Employee> findAll(){
        ...
    }
...
}
```

OR

```
MongoEmployeeDAO implements EmployeeDAO {
    @override
    List<Employee> findAll(){
        ...
    }
}
```

DAO Example(4)

```
public static voic main(String args[]){
...
EmployeeDAO searchDAO = new MongoEmployeeDAO();
List<Employee> employees = searchDAO.findByName(names);
...
}
```

Data Access Object

- Can be extremely useful when starting to work on a project:
    - When you start working on a project, you might want to avoid
        - Choosing and integrating to a data store/DB
    - You can define a DAO, and create a simple in-memory implementation
        - Start quickly, change to a realistic solution when you are ready
    - You can leverage this for testing as well
        - DAO returns your test data instead of retrieving from the database to reduce test run time

DAO In Projects

- DAO can help you cleanly separate DB work from applications
    - You can decide on the interfaces and methods in your projects
    - One person can work on the implementation while another can work on using the data in the application
    - Update each other when things change

DAO Tips and Tricks

- We can transparently add *decorators* to change the DAO's behavior
  - E.g. timing decorator that logs timing of DAO invocation
  - Exception handling decorator
  - Can map between domain object and value object (more on this in ORM)
- Keep your DAO interfaces platform neutral
  - A key capability of the DAO pattern is to change implementations. Don't let implementation details leak into the interface

Imagine you want to work with databases but don't want to learn/deal with a new language (e.g., SQL). What can you do?

Object Relational Management (ORM)

- Data is manipulated in object graphs but stored in tables
- ORM is a category of software library that reconciles the differences between object graphs and tables
- You may be surprised at how much of an application's code is devoted to mapping object graphs onto tables

Popular ORMs

- Hibernate was one of the first popular Java ORM solutions
- Hibernate (www.hibernate.org) consists of
    - Mechanisms to map object graphs onto databases
    - Code to manage the persistent state of object graphs
    - Supporting code to manage database connections, transactions, etc.
- Python has multiple ORMs
- JS has multiple ORMs too

Data Modeling in Document Databases

- Untyped collections replace strongly typed tables
  - Different “types” have a type attribute
- Often highly denormalized
- No enforced schema
  - Good because schemas change over time, bad because you still need to explain the schema at any moment in time

Takeaways

- Data modeling is important to get right early - migrations are a nightmare
- Picking a relational database is a safe default option for most applications
- A mix of ORM and SQL is the norm
  - ORM for object graphs
  - SQL for queries that require heavy optimization
- If you do choose a NoSQL system, be aware of the design trade-offs

Further Reading

- Crash course on database and MySQL
- Relational vs NoSQL

Free resources for you!

Next weeks

- Project progress
  - Choose your database
  - Design your data model
  - Divide the work and start working on your first working prototype