Database Modeling

Chapter 3

Michael Tsiang

Stats 167: Introduction to Databases

UCLA

The Structure of Data

Structured and Unstructured Data

Recall: Data can be classified into three types, based on whether it has a consistent and/or organized format:

- **Structured** data follows a consistent and organized format. This structure makes the data easily organized and searchable. Most structured data is stored in relational databases (i.e., data tables).
- **Semi-structured** data has some consistent organization but is not necessarily stored in tabular form, e.g., JSON or XML data.
- **Unstructured** data does not always follow a well organized format, e.g., PDFs, images, or audio/video files.

Semi-structured Data Formats

Semi-structured data is often stored in **JSON** (JavaScript Object Notation) or **XML** (Extensible Markup Language) format.

Some examples of JSON and XML files:

- https://en.wikipedia.org/wiki/JSON
- https://www.w3schools.com/js/js_json_xml.asp
- https://json.org/example.html

For comparisons between CSV, JSON, and XML formats:

- https://sonra.io/csv-vs-json-vs-xml/

Structured Versus Semi-Structured Formats

Questions to consider:

- Can structured data be stored in a semi-structured format?
- Can semi-structured data be stored in a structured format?
- What are some pros and cons between CSV and JSON/XML?

Structured to Semi-Structured

Consider the following data table:

Name	Height	Weight	Income
Leslie	62	115	4000
Ron	71	201	NA
April	66	119	2000

What might this data look like in a JSON format?

Semi-Structured to Structured

Consider the following JSON data[1]:


```
{
    "orders": [
        {
            "orderno": "748745375",
            "date": "June 30, 2088 1:54:23 AM",
            "trackingno": "TN0039291",
            "custid": "11045",
            "customer": [
                {
                    "custid": "11045",
                    "fname": "Sue",
                    "lname": "Hatfield",
                    "address": "1409 Silver Street",
                    "city": "Ashland",
                    "state": "NE",
                    "zip": "68003"
                }
            ]
        }
    ]
}
```


How might this data translate into a CSV or tabular format?

[1] Source: https://www.goanywhere.com/resources/training/how-to-read-json-and-insert-into-database

Structured, Semi-Structured, or Unstructured?

For each of the following scenarios, we want to classify the data into its most suitable type:

- Structured
- Semi-structured
- Unstructured

Structured, Semi-Structured, or Unstructured?

1. A spreadsheet with customer ID, name, age, and purchase amount

2. A folder of `.txt` files with product reviews in varied formats

3. JSON files from an API containing user profiles and settings

4. A database of MRI images used in medical research

5. Timestamps and sensor readings from a weather station

Structured, Semi-Structured, or Unstructured?

6. XML logs of system events from multiple servers

7. Raw video footage from security cameras

8. A CSV file with school enrollment data by year and region

9. Email exports including headers, body, and attachments

10. A collection of tweets, including hashtags, text, and embedded media

Designing a Data Model

Data Modeling

A **data model** is an abstract or high-level representation of a system's data and how elements of the data are organized and related to each other.

By considering data at a structural level, we can better understand how to access and query the data we need for analysis in later applications.

We will consider a couple scenarios to develop a logical thought process for basic data modeling.

Scenario: Social Media Platform

Consider the following scenario:

Suppose we are building a backend system to store data from a social media platform where users can share text updates, tag other users, include images or videos, and share their location. The data will be used later for analytics, moderation, and content discovery.

We want to think about what and how to store the data for our platform. Some guiding questions:

- What data might be important to record/collect?
- How would we store the data?
  - What components are structured, semi-structured, or unstructured?
- How would we identify pieces of data together?

Scenario: Online Retail Orders

Consider the following scenario:

Suppose you are managing the backend for an online retail store. Customers place orders, which may include multiple items. You want to design a system that allows you to track what was ordered, when, and for how much. The data will be used for historical analysis and future reporting/projections.

- What data might be important to record/collect?
- How would we store the data?
  - What components are structured, semi-structured, or unstructured?
- How would we identify pieces of data together?

Entity-Relationship Modeling

Entities and Relationships

A common way to formalize a data model is through an **entity-relationship** (or **ER**) model.

- An **entity** is an object or concept we want to store data about.
- An **attribute** is a characteristic of an entity we want to record.
- A **relationship** describes how entities are connected.

A basic entity-relationship model describes the entities of the data and how they're related to each other.

Note: Entity-relationship models were originally developed for relational data, but they can also be applied to some non-relational (e.g., semi-structured or graph) data with a few modifications and/or additions.

Scenario: Online Retail Orders

Revisiting the online retail store scenario:

Suppose you are managing the backend for an online retail store. Customers place orders, which may include multiple items. You want to design a system that allows you to track what was ordered, when, and for how much. The data will be used for historical analysis and future reporting/projections.

- What are the entities in this context?
- What are the attributes for each entity?
- How are the entities related to each other?

Online Retail Orders: Entities and Attributes

- Customers
  - Customer ID (Primary Key)
  - Name
  - Email
  - Address
- Orders
  - Order ID (Primary Key)
  - Customer ID
  - Order Date
  - Shipping Address
- Products
  - Product ID (Primary Key)
  - Name
  - Category
  - Price

An attribute (or set of attributes) that uniquely identifies an individual entity is called a **primary key**. A **foreign key** is an attribute that is a primary key of a different entity.

Entity Types and Entity Sets

Is an individual customer an entity? Or are all customers together an entity?

The **entity type** (sometimes just called **entity**) is the category or class of objects being stored or modeled.

An **entity instance** (also sometimes called **entity**) is a specific example of an entity type.

An **entity set** is the collection of all instances of an entity type.

For example, in our scenario:

- Customer is an entity type (or just entity).
- An individual customer is an entity instance.
- All customers in our database is an entity set.

How are the entities (entity types) related to each other?

Defining Relationships and Cardinality

Relationships between entities can be thought of as verbs that link two or more nouns (entities).

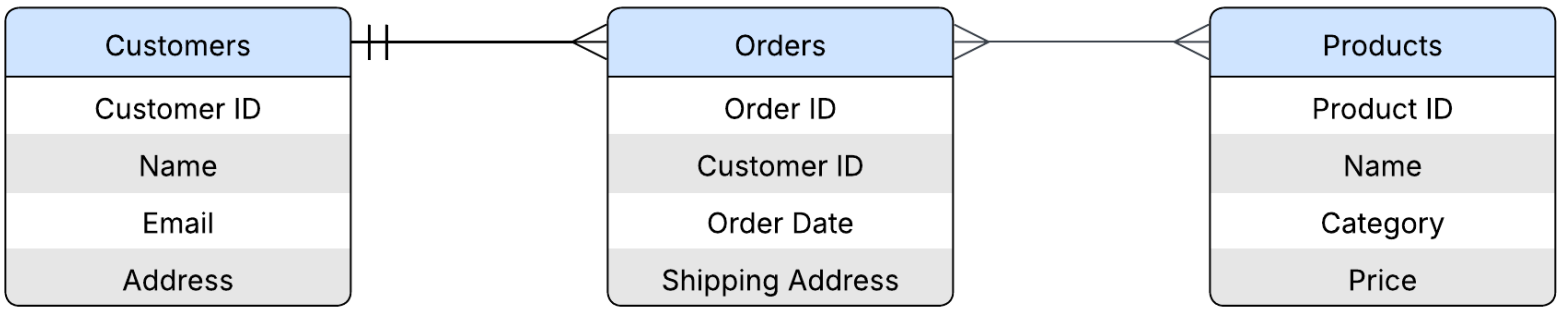
- Customers place orders.
- Orders contain products.

The **cardinality** is the numerical relationship between entities. The most common cardinalities are **one-to-one**, **one-to-many**, and **many-to-many**.

- A customer can place many orders, while each order is placed from only one customer, so the cardinality between customer and order is one-to-many.
- An order can contain many products, and a product can be in many orders, so the cardinality between orders and products is many-to-many.

Entity-Relationship Diagram

An **entity-relationship diagram** (or **ERD**) is a visualization of the entity-relationship model. The basic ERD below shows the entities and relationships for the online retail orders scenario.



The relationships use **crow's foot** notation to represent the one-to-many and many-to-many relationships.

- The two parallel lines denote "one and only one."
- The "crow's foot" denotes "many."

Source: Made in Lucidchart (https://www.lucidchart.com/)

Associative Entities

The many-to-many relationship between Orders and Products is problematic, because we would need to add redundant observations in our data to clarify the connection between individual orders and individual products.

More specifically, the relationship between Orders and Products has *relationship-specific* attributes we should record. For example, for an order:

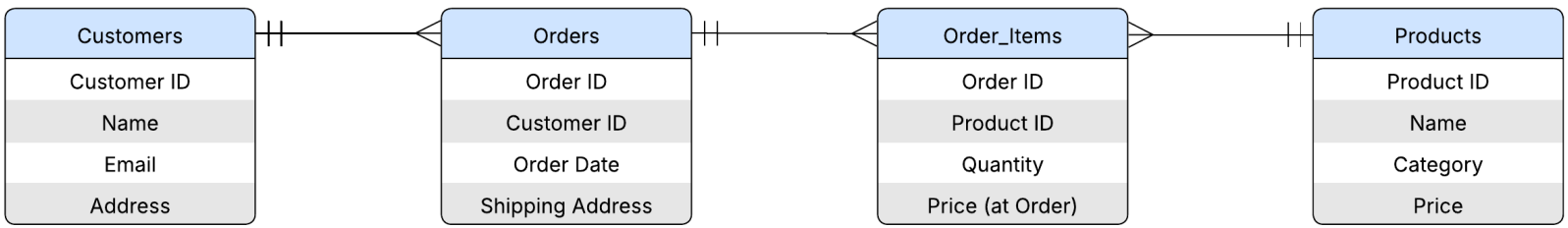
- What is the quantity of each product ordered?
- What was the price of the product at the time of order?

When a relationship needs attributes, we can define an **associative entity** that serves as a bridge between entities with a many-to-many relationship.

In this scenario, an Order_Items entity with attributes quantity and price at order would "resolve" the many-to-many relationship into two one-to-many relationships.

Refined Entity-Relationship Diagram

An updated ERD for the online retail orders scenario shows the associative entity Order_Items that clarifies the relationship between Orders and Products.



Source: Made in Lucidchart (https://www.lucidchart.com/)

Further Information on Entity-Relationship Modeling

Additional resources on entity-relationship models and ERDs:

- https://www.lucidchart.com/pages/er-diagrams
- https://www.smartdraw.com/entity-relationship-diagram/

Both Lucidchart and Smartdraw have diagram visualization tools, but only Lucidchart has a free tier and a free educational account.

- How to sign up for a free educational account on Lucid

More on many-to-many relationships:

- https://vertabelo.com/blog/many-to-many-relationship/