

Data I/O and Preprocessing with SQL and Python

Module 3: Databases

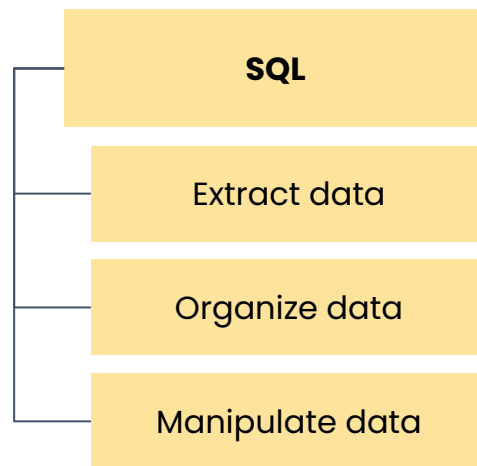
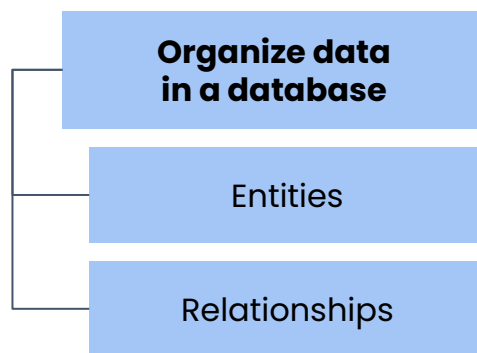
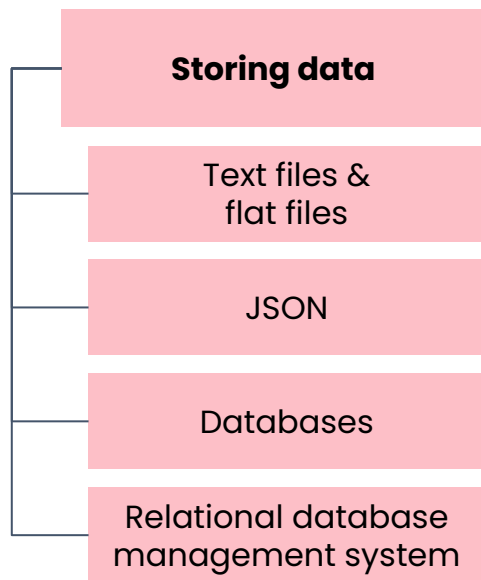




Databases

Module 3 introduction

Module 3 outline

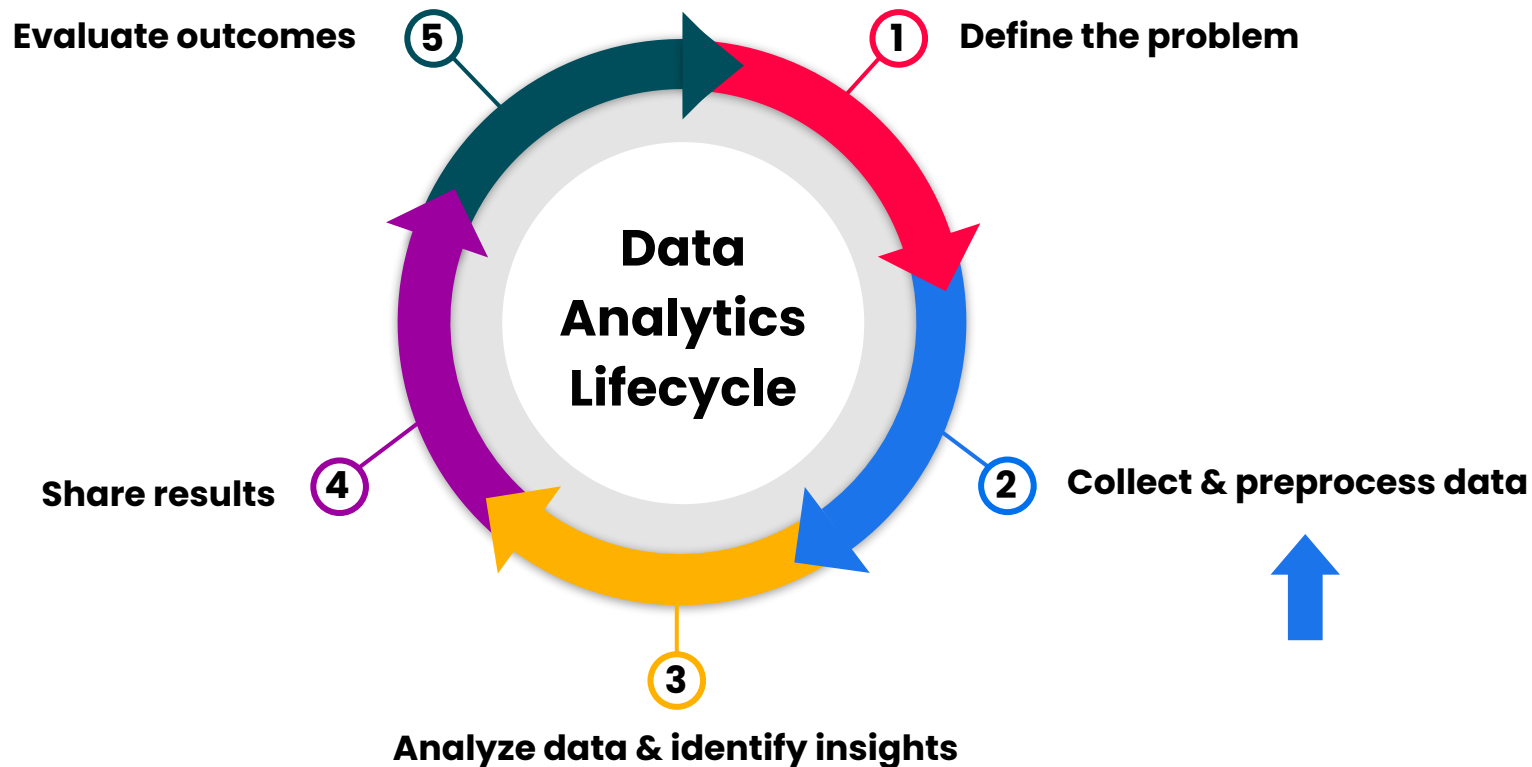




Databases

Data storage systems

Data storage: retaining digital information for later access



Complexity of data storage

- **Text file:**

- Performs basic functions
- Retrieve information later
- Doesn't provide much structure

- **Flat file like CSV or XLSX:**

- Highly useful for structuring data in rows and columns

- **Structured file format like JSON**

- Step up from flat files
- Organize data hierarchically

- **Relational database:**

- Stores data in many tables, each with rows and columns
- Can represent relationship between tables

How data storage needs evolve



Scenario 1: Tracking your grocery list

- Size: Just a few items
- ✓ Storage: Phone's Notes app



Scenario 2: Small delivery business

- Size: Managing lists for multiple people
- ✓ Storage: Spreadsheet program to efficiently track and share access



Solution that worked well for a handful of customers won't work anymore



Scenario 3: Large delivery business

- Size: Tens of thousands of customers
 - Every week
 - Multiple regions
 - Varying pricing tiers
 - Seasonal coupons



Storage: ~~CSV file~~



Will become increasingly slow



Won't update dynamically



Easy to create conflicting versions or overwrite

Your role

Your job isn't to:

- ✗ Decide most appropriate storage system
- ✗ Build it yourself

Your role is to:

- ✓ Understand trade-offs each system offers
- ✓ Advise decision-makers and data engineers
- ✓ Help shape how company stores info
- ✓ Make sure data is ready and available

Apply what you've learned

- You can import data from a CSV file, APIs, or web scraping
- You can perform same preprocessing and analysis on data from a database
- You already have analytical skills needed to work with data!

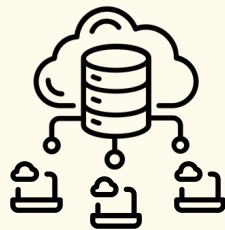


Databases

What is a database?

What is a database?

- A structured system for storing data to handle huge volumes of data efficiently
- **Think of a database as:**
 - 📁 Organized files + ⚙️ Specialized software
- ⚙️ Software handles:
 - ☐ How data is added, removed, or updated
 - ☐ Relationships between files



- Typically housed on a server (or in the cloud)
- Many people can access from their own computer



- Can also store database on your own computer
- Provided you have enough space

Database advantages

- Designed to handle large volumes of data
- Key advantages:

 Efficient at storing and retrieving large volumes of information

- Use optimization techniques to retrieve data quickly

 Designed to handle concurrent users

- Handles conflicts and manages who can interact with information



Database enforce constraints

- Examples:
 - Certain value must be a number
 - All cells in row must be filled



Databases model relationships

- Connect related information across different tables
- Difficult to do in spreadsheets (i.e. using XLOOKUP)

How to interact with database

- Interact with database through **querying**



You: Send a query to database



Database: Send you data in return

- A database query can return:



Rows that meet certain conditions



Summary statistics



Rows of multiple tables joined



Exactly the data you need

- You can query data through



Website provided by company



Using code in Python notebook

Types of databases

Relational databases

- Organize data in well-defined tables



Records → Observation



Fields → Feature

- Each table represents an entity
- Each row represents one instance of entity
- **Example:** Customers that make orders
 - Customer: Name, Contact information
 - Order: Product name, Order date
 - Connect to analyze how entities interact

Non-relational database

- Used to store data that varies in structure and content
- Don't use tables with fixed columns
- Sacrifice strict consistency
- Provide better performance for high volume data with flexible structure

Databases

Database management
systems

Database management systems (DBMS)

- **Relational DBMS:**

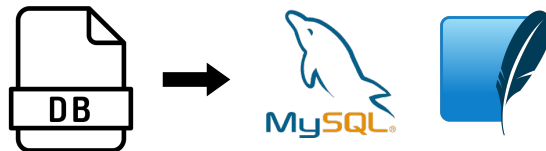
- MySQL
- ★ SQLite
- PostgreSQL (Postgres)
- Oracle
- Presto ← **N**

- **Non-relational DBMS:**

- MongoDB
- Cassandra







- All programs load csv file
- Each has its own features and interface



- Each create and manage database files
- Likely to encounter different ones at different companies

Relational database management system (RDBMS)

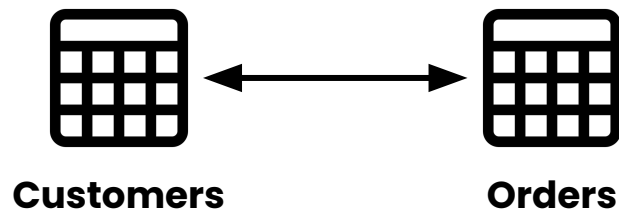
- An interface between analyst (or app) and the database files
- It has multiple responsibilities:
 -  **Processing queries** - Interpret query and return results
 -  **Controlling concurrency** - Makes sure updates to database don't conflict
 -  **Enforcing security** - Control who can access what data through authentication
 -  **Maintaining backup and recovery** - Help prevent data loss

Data integrity and constraints

Strength: Databases enforce constraints (rules)

- RDBMS does the enforcement
 - Helps maintain integrity:
 - Accuracy
 - Completeness
 - Consistency
 - Maintains rules that prevent invalid data from being stored in table

Example: Every row has a unique identifier



The RDBMS might:

- Enforce that an order cannot reference non-existent customer
- When someone violates a constraint, reject operations and provide error

Databases

Tidy data

What is tidy data?

- Tidy data is a technical term
- Data that meets three conditions:
 - ☐ Each feature is a column.
 - ☐ Each observation is a row.
 - ☐ Each entity has its own table.
- Introduced in 2014 by data scientist Hadley Wickham
 - To define how to maintain data that is easy to analyze

1. Introduction

It is often said that 80% of data analysis is spent on the process of cleaning and preparing the data (Dasu and Johnson 2003). Data preparation is not just a first step, but must be repeated many over the course of analysis as new problems come to light or new data is collected. Despite the amount of time it takes, there has been surprisingly little research on how to clean data well. Part of the challenge is the breadth of activities it encompasses: from outlier checking, to date parsing, to missing value imputation. To get a handle on the problem, this paper focusses on a small, but important, aspect of data cleaning that I call data **tidying**: structuring datasets to facilitate analysis.

The principles of tidy data provide a standard way to organise data values within a dataset. A standard makes initial data cleaning easier because you don't need to start from scratch and reinvent the wheel every time. The tidy data standard has been designed to facilitate initial exploration and analysis of the data, and to simplify the development of data analysis tools that work well together. Current tools often require translation. You have to spend time

Benefits of tidy data

- ✓ Data cleaning can be **more standardized**
 - Develop tools that solve similar problems
 - Many tools (like pandas) expect tidy data
- ✓ Minimizes errors by **reducing complexity**
 - Summarization, visualization, and modeling can be approached in a standard way
 - Allowing you to focus on domain questions
- ✓ Tidy data is **more efficient**
 - Its structure pairs neatly with vectorized operations often used with pandas

- You can use Pandas and Python to tidy up untidy data
- **BUT!** It's better to address these issues earlier in the workflow
- Data should be stored in a database in a tidy manner



Databases

Entities and attributes

Structure of data

- The way data is structured has a impact on how useful it is for analysis
- Tidy data organizes:
 - Observations → Rows
 - Features → Columns
- Each table represents a distinct entity
- Key to building relational databases that are structured and efficient

Terminology differences

- Databases use different terminology than statistics and programming
- **Example:**
 - Statistics/Programming:
Observations and features
 - Databases:
Entities and attributes

Entities

- Represent a distinct thing in the world



People



Events



Products



Processes



Iguanas

- Represented as tables
- Entities have two key constraints:
 - Entity must be uniquely identifiable
 - Table must have only one entity

Customers

CustomerID	Name	Birth Date
001	John Smith	03-12
002	John Smith	03-12

✗ Mixed Customers & Orders

Name	Email	Product Name	Quantity
John Smith	js@mail.com	Avocado	2
John Smith	sj@mail.com	Mango	4

Primary keys

- Unique identifier in a database table
- Allows you to distinguish between two customers with identical information
- Primary keys are **non-nullable**
 - Every row must have a primary key
 - Constraint enforced by your RDBMS
 - Would reject attempts to add row with null primary key
- Primary keys are **stable**
 - Value should ideally never change

Customers

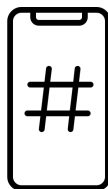
CustomerID	Name	Number of Orders
001	John Smith	10
002	John Smith	10



Attributes

- The characteristics of that entity
- Have well-defined data types
- Data type options:
 - **Text:** names, addresses, descriptions
 - **Integers:** age, quantity
 - **Floating-point number:** prices
 - **Boolean:** whether customer is active
 - **Dates/Times:** order dates
- Primary key is typically an integer

Customers	
Attribute	Data type
Name	Text
Email	Text
Phone #	Text



Standardized format:

✓ **Text:** "0123346789"

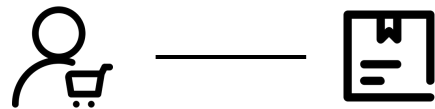
✗ **Integer:** 123346789

Databases

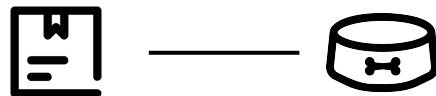
Relationships

Relationships

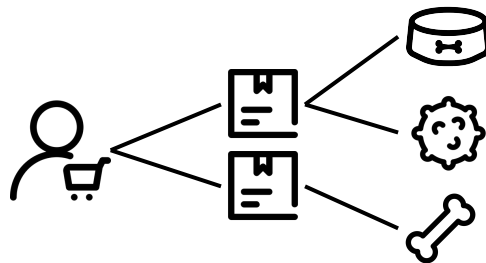
- In a business, you typically have data about many interconnected entities.
- **Example:**
 - Customers
 - Orders
 - Products
- Entities don't exist in a vacuum
- They're deeply interconnected



Customers place **Orders**



Orders include **Products**



A **Customer** can have multiple **Orders**
and an **Order** can have multiple **Products**

Primary & foreign keys

- Give each entity a way to reference another table:

 **Primary key**

 **Foreign key**

- Reference in one table to primary key in another table
- This setup allows:
 - Different types of data to remain connected
 - Maintaining principles of tidy data.

Customers	
CustomerID	Integer
Name	Text
Email	Text

Orders	
OrderID	Integer
OrderDate	Date
Products	Text
Quantity	Integer
CustomerID	Integer

Why use foreign keys?

Example: Store all data in one table

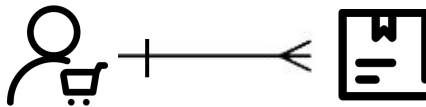
- For every order:
 - Repeat customer details
 - Repeat product details
 - This duplication:
 - ✗ Wastes storage
 - ✗ Makes updates error-prone

Example: Customer updates email

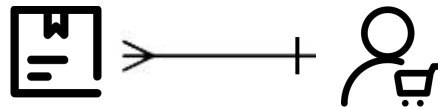
- ✓ Change once in Customers table
- ✓ Updated information instantly reflected wherever referenced

One-to-many

- One entity is associated with multiple instances of another



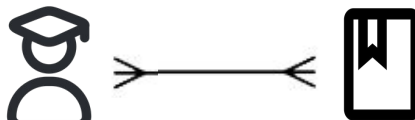
A customer has **many** orders



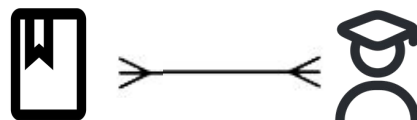
Each order has **one** customer

Many-to-many

- Each entity can be related to multiple instances of another



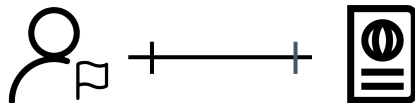
A student has **many** classes



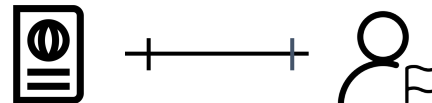
A class has **many** students

One-to-one

- Each instance of one entity is associated with exactly one instance of another



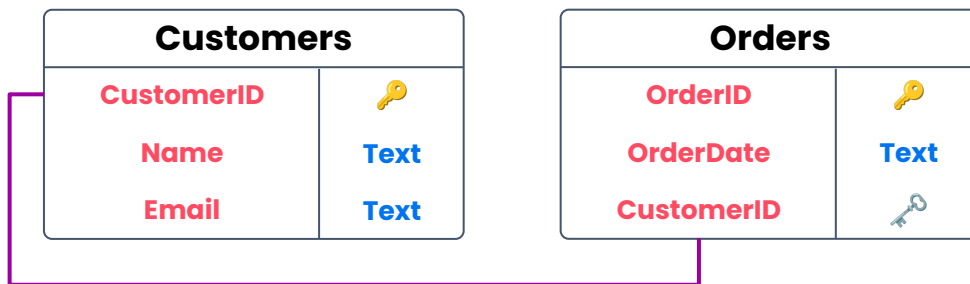
A citizen has **one** passport



Each passport has **one** citizen

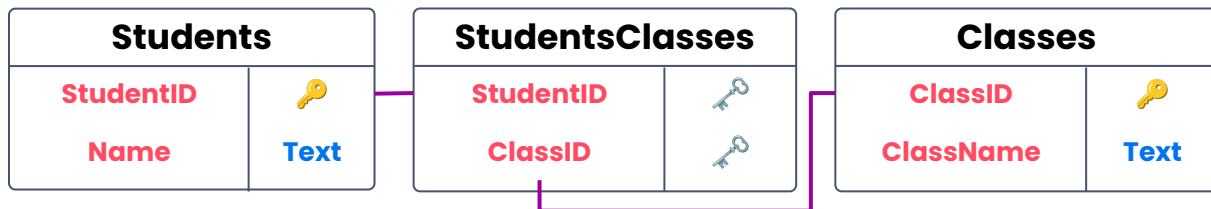
One-to-many

- Represented with the foreign key on the "many" side



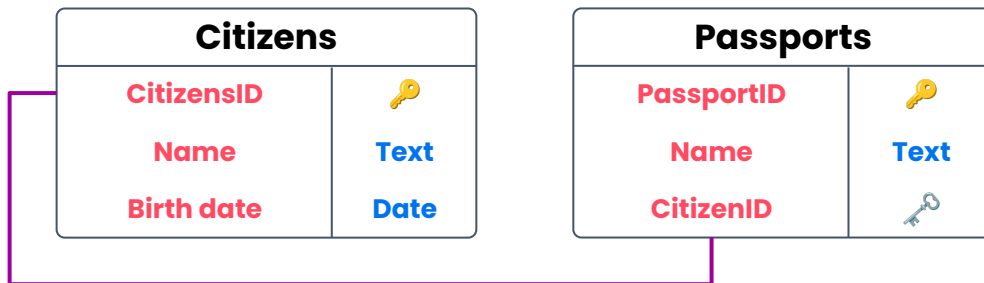
Many-to-many

- Use a separate table with foreign keys for both entities



One-to-one

- Put the foreign key in the more "optional" table





Databases

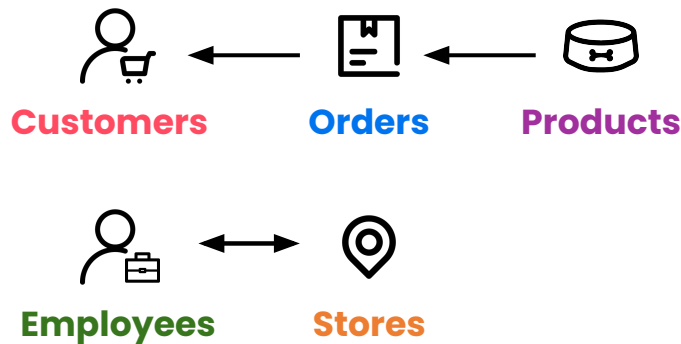
Data models
and data schemas

What is a data model?

- A sketch of a database, including all entities and relationships
- Framework for organizing and structuring data
- Design a structure that supports:
 - ⚡ Efficient data storage
 - 🔍 Easy analysis
- As a data analyst, you:
 - ✗ Won't design from scratch
 - ✓ Should be familiar with development

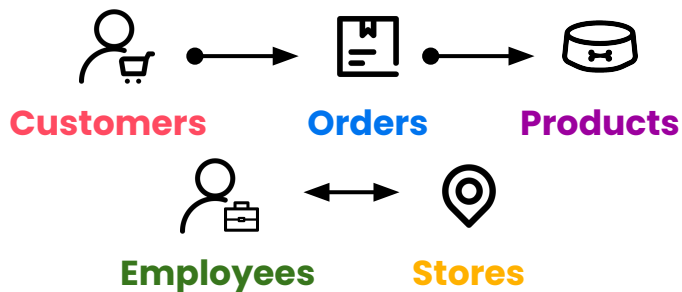
Example: Pet supplies chain


1. Outline the entities involved
2. Define how these entities are connected







What is a schema?

- Specific implementation of model:
 - Tables
 - Attributes and data types
 - How tables link together using keys
- Enforces rules that ensure data is consistent
 - 🔒 Example: Order must reference valid Customer
- **Example:** Pet supplies chain



Customers	
 CustomerID	Int
Name	String
Address	String
Age	Int

StoresEmployees	
 EmployeeID	Int
 StoreID	Int

Orders	
 OrderID	Int
OrderDate	DateTime
 CustomerID	Int

Designing a data model

1. Understand the purpose

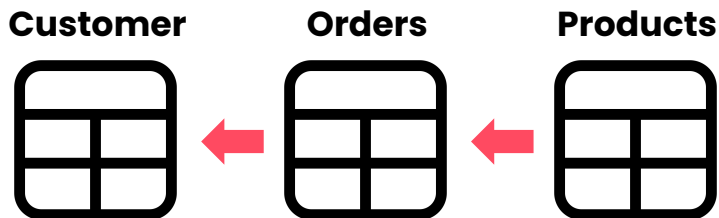
- Questions you need to answer

2. Identify key entities and relationships

- Main things you're analyzing
- How they relate

3. Plan the schema

- What tables you need
- What attributes belong to each table
- How they should be connected using foreign keys



Generative AI

Generative AI tools can help:

 Brainstorm possible data structures

 Clarify questions about how to:



- Organize complex datasets
- Model complex relationships between entities



Databases

Type of tables

Dimension tables

- Stores descriptive information about entities
- Examples:
 -  Table of customers
 -  Table of diamonds
- Model entities in your database
- Providing the:
 - ☐ **Who** the customer is
 - ☐ **What** their occupation is
 - ☐ **Where** they're from
 - ☐ **When** their birthday is

Products

ProductID	Name	Category	Price
101	Chew Toy	Pet Toys	5.99
102	Dog Leash	Accessory	12.99
103	Cat Food	Food	19.99
104	Fish Tank	Equipment	29.99



- Describe products but not transactional data
- Example question: *“What are the names and prices of the products we sell?”*

Relational tables

- Used to connect entities together
- Contain foreign keys that link to primary keys in other tables
 - Create the relationships between entities
 - Make it possible to join and analyze across tables
 - Without foreign keys, the connections wouldn't exist
- Example question: *"Which employees work at each store?"*



Fact table

- Captures events or transactions
- Examples:
 -  Sales
 -  Orders
- Often used to track key metrics
- Include foreign keys to connect to dimension tables
- Difference is what they store:
 - **Dimension table:**
Customer names, product categories, store locations
 - **Fact table:**
Transactions

Orders

OrderID	OrderDate	CustomerID	TotalAmount
301	2024-01-01	201	18.98
302	2024-01-02	245	12.99
303	2024-01-02	517	49.99

- Each row represents an individual transaction.
- Example questions:
 - *"How much revenue did we generate from a specific order?"*
 - *"How many units of a product were sold?"*

Using tables together

Orders

Transactional data

OrderID	OrderDate	CustomerID	TotalAmount
301	2024-01-01	201	18.98

Customers

Who made the purchase

CustomerID	Name	Phone	Address
201	Jane Doe	'123456789'	123 House Rd

OrderDetails

OrderID	ProductID
301	101
301	102

Products

What was sold

ProductID	Name	Category	Price
101	Chew Toy	Pet Toys	5.99
102	Dog Leash	Accessory	12.99



Databases

Introduction to SQL

What is SQL?

- Databases support queries
- For a relational database, use structured query language (SQL) to write queries
- SQL is:
 - A programming language
 - Designed for interacting with relational databases
 - Highly efficient for interacting with structured data
 - Only used to interact with databases
- With SQL:
 - ✓ Retrieve, update, create, delete records
 - ✗ Won't run linear regression model

Commands in SQL

Queries

- Command that retrieves records
- Essentially asking database a question
 - **Example:** "What were the sales figures last month?"
- Essential part of working with SQL
- Allow you to **extract** precise information



Your primary focus will be on retrieving data through queries

Altering databases

- SQL includes commands for:
 - Creating data
 - Modifying data
 - Deleting data



Altering database is handled by database administrators or data engineers

SQL query

- Selects two columns: order_id, total_amount
- From the Orders table
- Order in descending order by total_amount
- Limited to 10 results

```
SELECT order_id, total_amount  
FROM orders  
ORDER BY total_amount DESC  
LIMIT 10;
```



In general, you can read a lot of SQL queries like a sentence.

Why use SQL?

- Standard language for interacting with relational databases
- Relational databases manage structured data efficiently
- SQL queries
 - Handle huge datasets efficiently
 - Return results faster than many operations in Pandas
 - Speed especially true at scale
 - For millions of records: returns results in seconds

Pandas or SQL?

- Simpler operations:
 - Filtering or counting
 - More efficient with SQL
- More complex or custom calculations:
 - More appropriate for Python notebook

Why learn SQL

- ✓ Will likely be a central part of your daily work
- ✓ It's the primary tool for querying structured data in relational databases
- ✓ You'll encounter SQL questions in job interviews
- ✓ Valuable in many industries

 This course teaches:

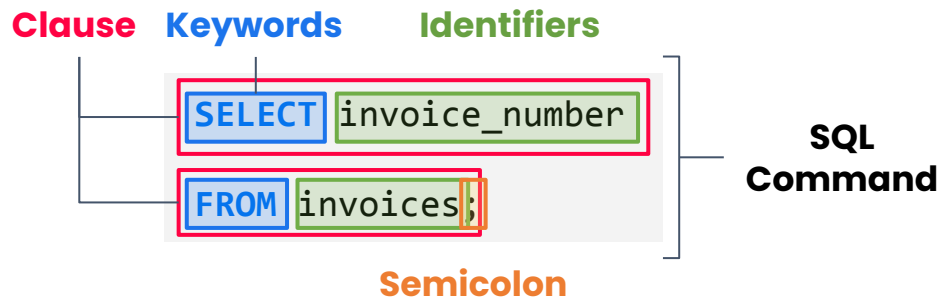
- ☐ **SQL** - to use it effectively in real-world scenarios
- ☐ **SQLite** - a lightweight, file-based database for small projects

Databases

SQL code

Basic SQL query

- SQL helps extract data from a database using commands
- Most foundational command is **SELECT** which retrieves specific columns
- All queries you'll write will include:
 - **SELECT** - what data you want
 - **FROM** - which table it's from
- Two optional conventions (habits):
 1. **Keywords** are written in all caps
 - Not case sensitive
 2. **Clauses** typical start on a new line
 - Not whitespace sensitive



SQL code varies based on RDBMS

Semicolons

- Some require a semicolon
- Others leave off semicolon for individual queries
- Using semicolon is a good habit to maintain!
 - Ensures code will work across different database systems
 - Avoids errors where multiple queries are executed together

Capitalization

- Keywords can be written in all caps, lowercase, or mix
- Column and table names may be case-sensitive
- SQLite is not case-sensitive

Example:

- ✓ `INVOICE_NUMBER`
- ✓ `Invoice_Number`
- ✓ `invoice_number`
- Others have nuanced rules

Naming

- Example: employee name
- In SQLite, use double quotes
- Others may use different rules
- In older systems, column names may not be formatted ideally

Comments

- Single line comments are written with two dashes --
- Help you and others understand what query is doing

Databases

Selecting

Scenario



You
Data Analyst



Goal: Integrate music catalogue into a subscription service



Task: Perform exploratory data analysis on acquired SQLite database to better understand scope of catalog



Look at database schema to see what data is available



Data: Database containing dimension tables of:

Music-related entities:

- Artists
- Tracks
- Playlists

Business-related entities:

- Customers
- Employees
- Invoices

Recap: Selecting

To retrieve specific columns from a specific table:

```
SELECT name, milliseconds, unitprice  
FROM tracks
```

To control how many results were returned:

```
SELECT name, milliseconds, unitprice  
FROM tracks  
LIMIT 10;
```

Selects all columns from a table:

```
SELECT *  
FROM tracks  
LIMIT 10;
```

- Columns separated by commas
- SQLite: not case sensitive but are sensitive to extra whitespace
- Good way to test query before applying it to the full scope of data

Silent errors in SQL

- Just because your code runs doesn't mean it did the right thing
- Errors will stop your code from running
- Logical mistakes quietly give wrong answers
- Have an expectation of what query results should look like in advance:
 - ☐ How many rows
 - ☐ How many columns
 - ☐ What data types



Databases

Ordering results

Scenario



You
Data Analyst



Goal: To better understand catalog of music recently acquired after purchasing a competitor



Task: Perform exploratory data analysis to better understand types of music in the catalog



Understand the length and storage space required for different tracks



Allocate appropriate database resources

Recap: Ordering results



Use **ORDER BY** to sort results

```
SELECT name, milliseconds, bytes  
FROM tracks  
ORDER BY bytes;
```



Order by multiple columns

```
SELECT name, milliseconds, bytes, genreid  
FROM tracks  
ORDER BY genreid, bytes DESC;
```



To give column temporary nickname

```
SELECT name AS track_title  
FROM tracks  
ORDER BY genreid, bytes DESC;
```

- By default, sorting is ascending
- To sort in reverse: **ORDER BY bytes DESC;**
- Sorting order matters
- First column is sorted first

Databases

LLMs for databases

Scenario



You
Data Analyst




Data: Music database



Goal: Perform exploratory analysis to understand catalog



This demo uses  **ChatGPT o1** offered by OpenAI

- Reasoning model
- Available for both free and paid accounts
- Other options for reasoning:
 -  **Claude 3.7 Sonnet** Extended Thinking Mode
 - You may have a more up to date model available



Databases

SQL in Python

Recap: SQL in Python

1. Import sqlite3 into your notebook:

```
import sqlite3
```
2. Open the connection:

```
connection = sqlite3.connect("music.db")
```
3. Create queries using strings:
 - Typical:

```
query = "SELECT * FROM artists;"
```
 - Multiline:

```
query = """SELECT name, milliseconds, bytes,  
genreid  
FROM tracks  
ORDER BY genreid, bytes DESC;"""
```
4. Execute query:

```
results = pd.read_sql_query(query, connection)
```