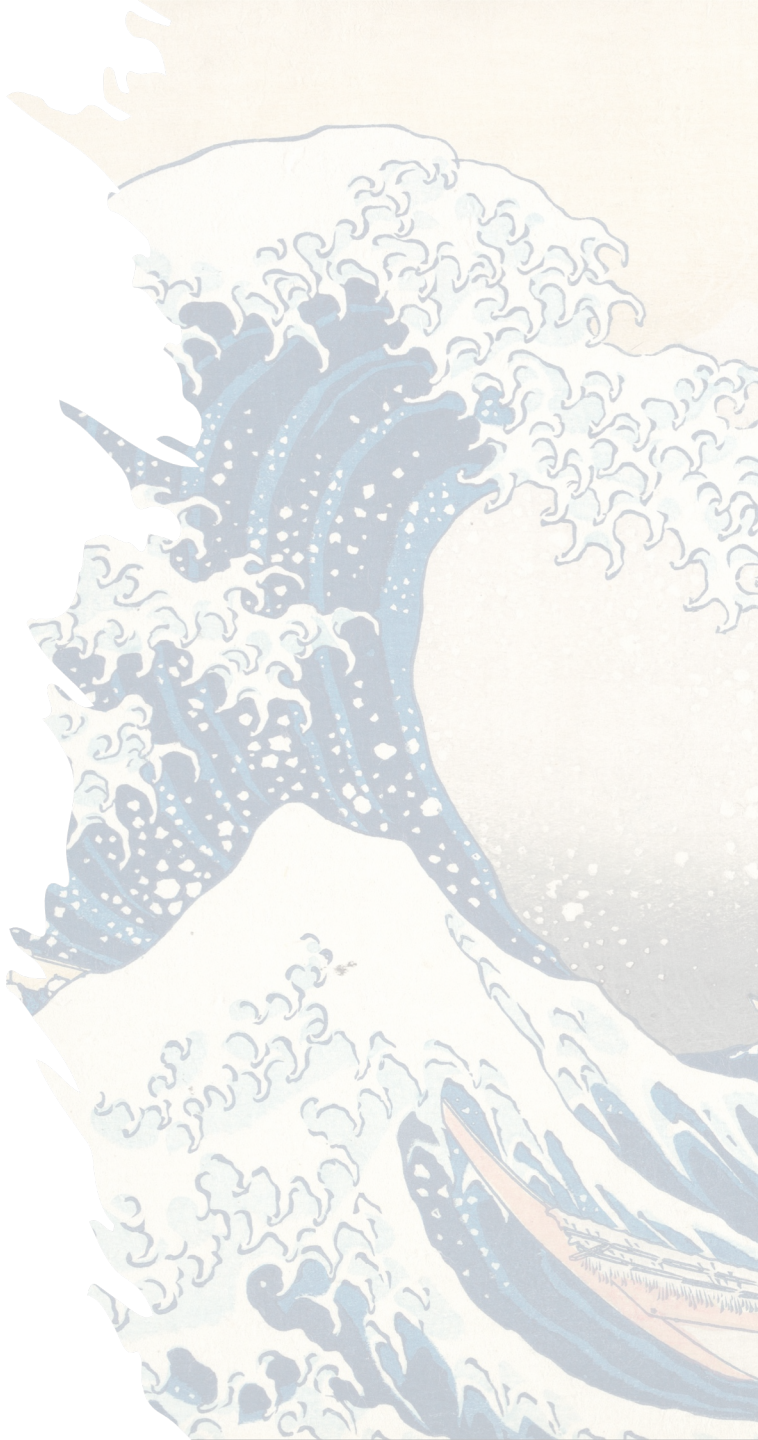
Stelios Sotiriadis

5. Introduction to data modelling

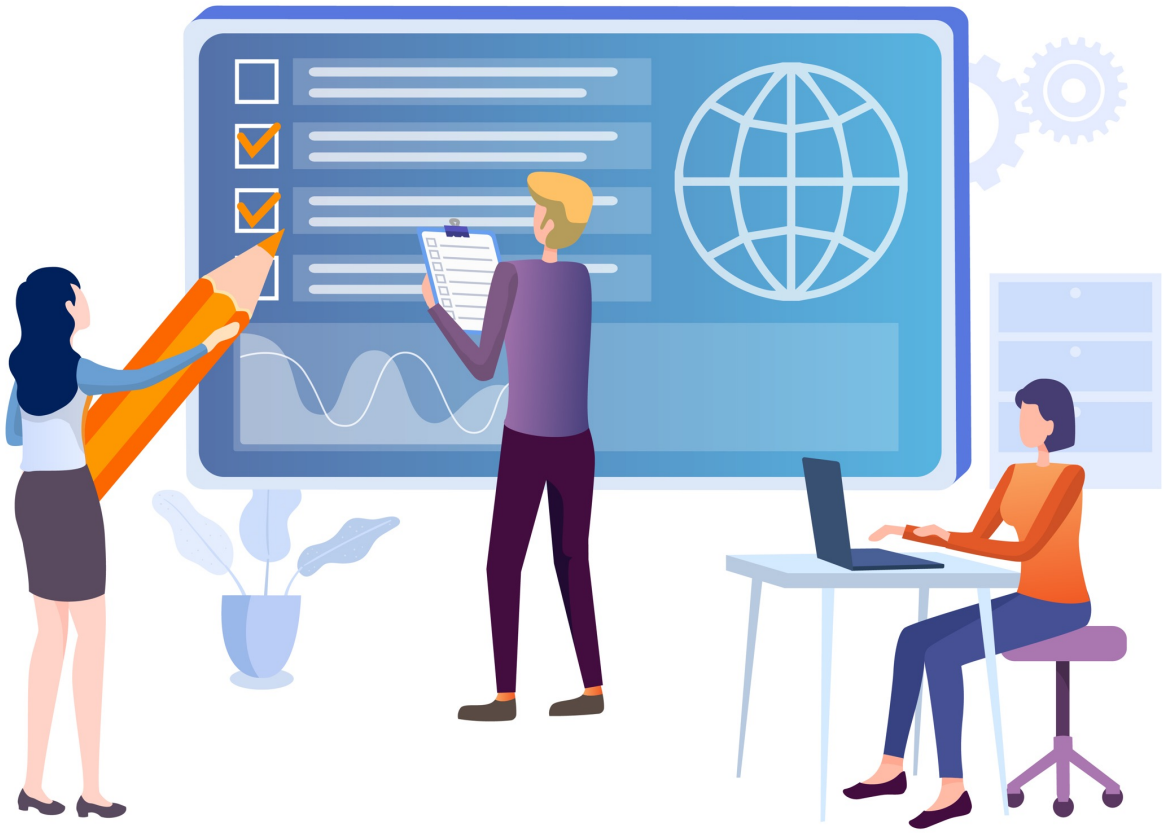
Agenda

- Introduction to data modelling
  - Relational vs non-relational
- Introduction to SQL
  - **C**reate – **R**ead – **U**pdate – **D**elete
  - Relational algebra
  - SQL commands
- Lab 5 – MySQL for data modelling
  - **Complete MySQL – next week→SQL race-lab**



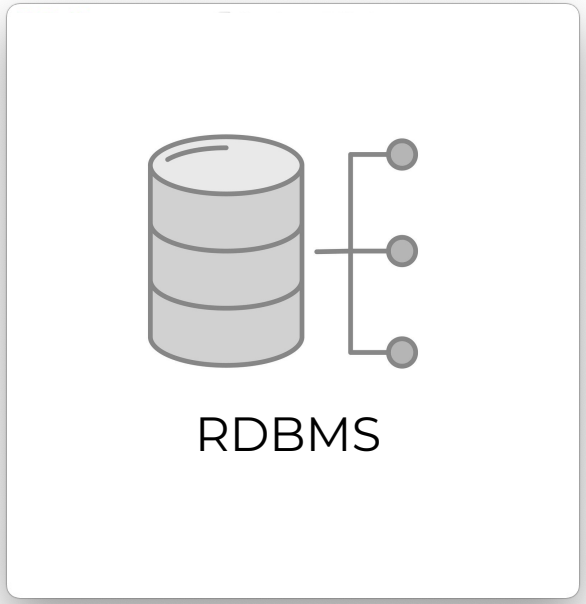
Quiz of the day

Get ready!



Structured data

- **Relational Database Management System (RDBMS)**
  - A data model (or schema) defines how data should be stored
    - **`Age`** is an integer, salary is a float etc.
    - The **`date`** is in **`01/01/2023`** format
  - The data model defines rules and constraints
    - A salary should be between **`12500`** and **`150000`**
    - The **`city`** cannot be **`null`**

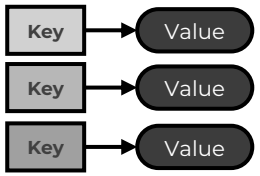


Unstructured data

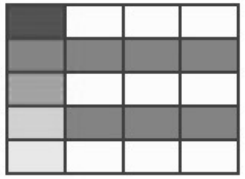
- The internal structure is not pre-defined
- There is **no data model**, rules and constraints
  - Data can be in any format, including text, audio and video
- Usually stored in NoSQL databases

NoSQL

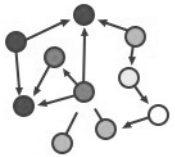
Key-Value



Column family

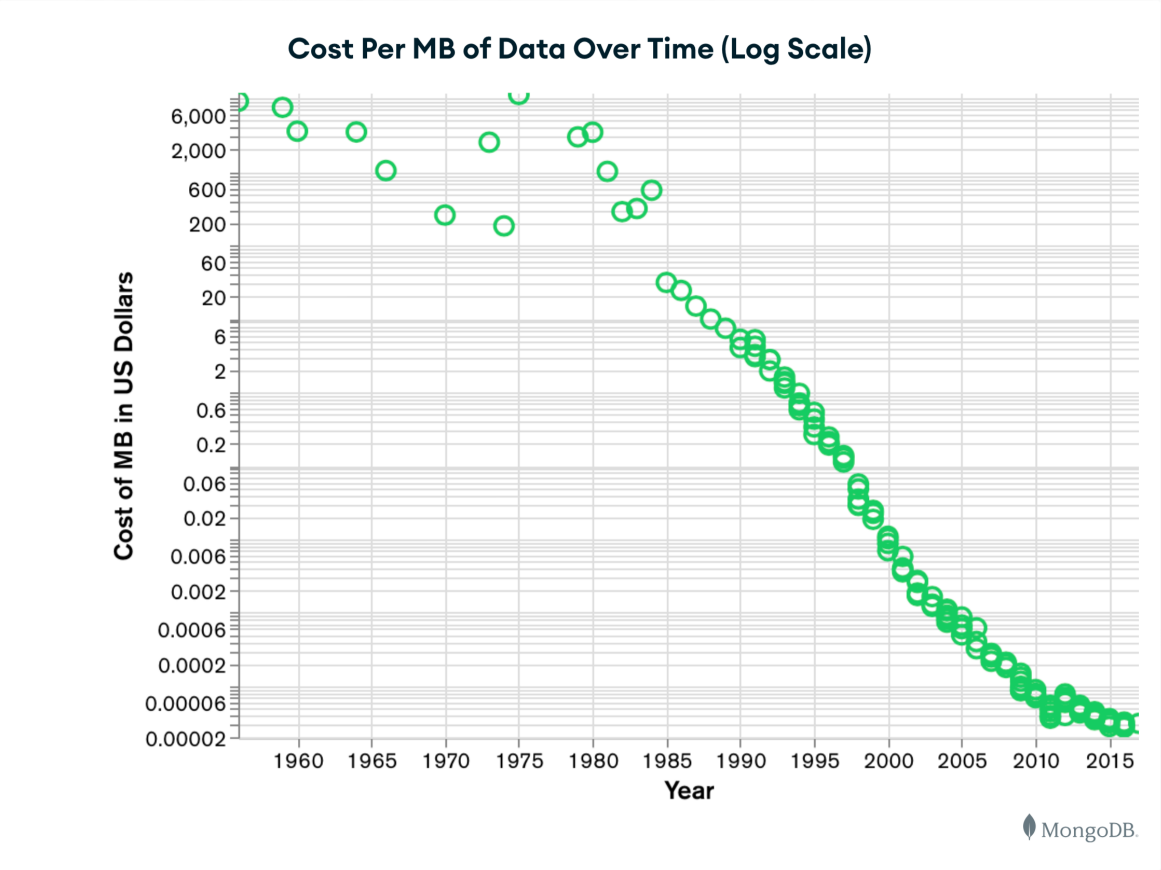


Graph



Document

What is a NoSQL database?



- Non-relational database.
  - It emerged in the late 2000s as storage costs dramatically decreased.
- **NoSQL offers efficient storage of big and unstructured data.**
  - It supports flexible schemas and fast queries.

* We will explore more on NoSQL soon...

Structured vs Unstructured data

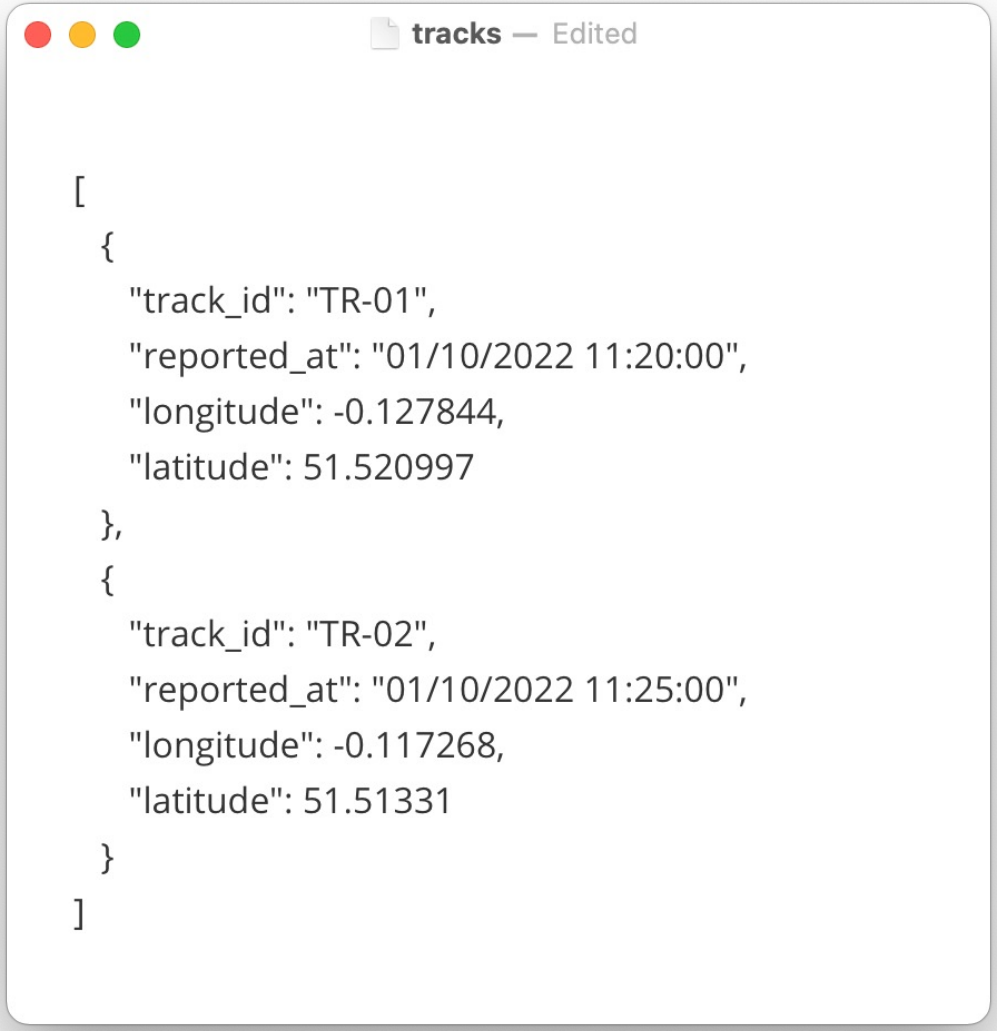
- Structured data resides in relational databases:
  - A database made of relations between stored items of data
- Unstructured data is everything else!
  - Text files, emails, websites, media, social media etc.
- As data scientists, we need to deal with both!

Semi-structure data

- **Semi-structured data** contains tags or other markers to separate semantic elements and enforce hierarchies of records and fields within the data.
- Examples:
  - Key-Value data: JSON
- NoSQL systems are used to store semi-structured data

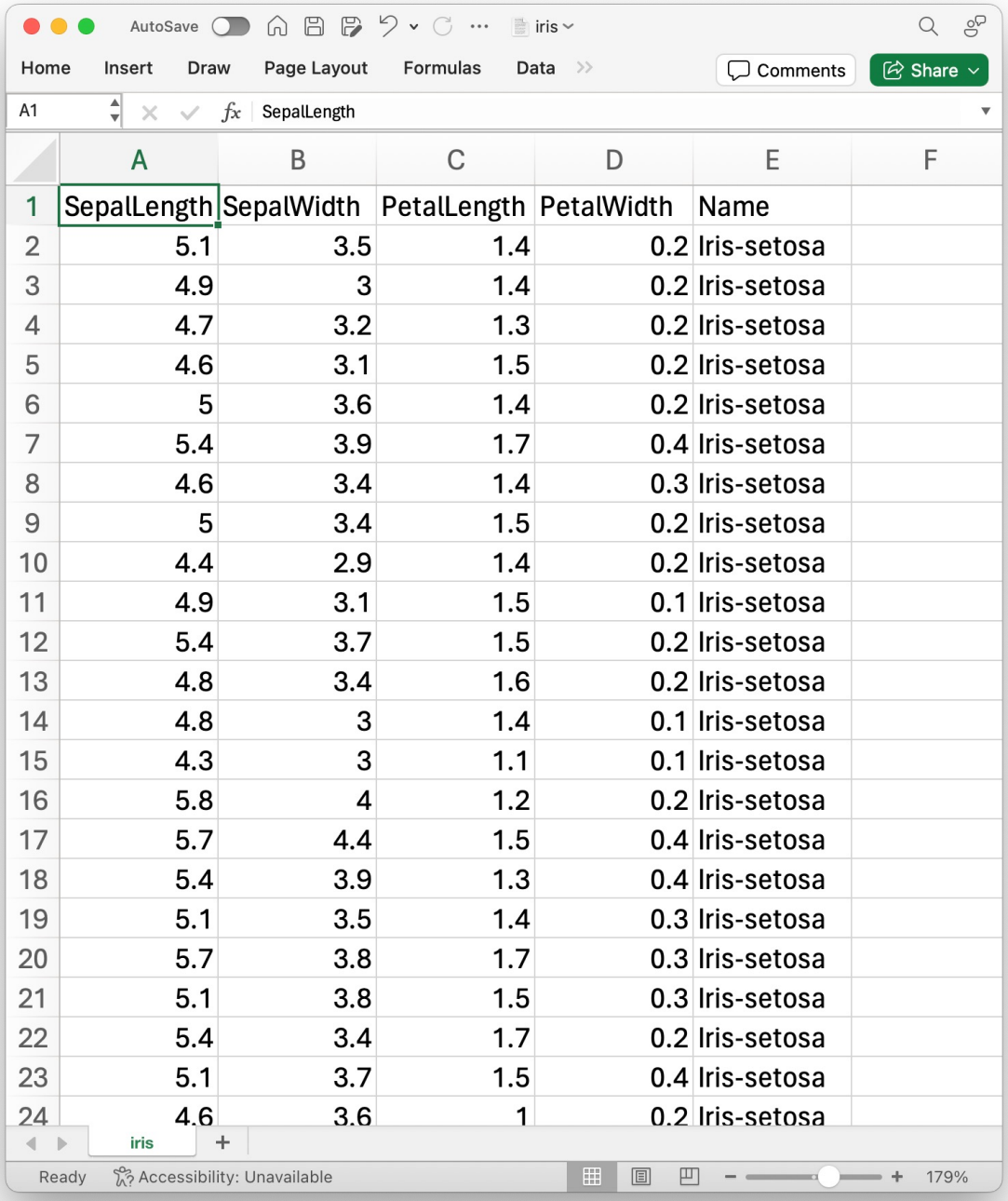
What is a JSON file?

- A JSON file (**mydata.json**), it stores simple data structures and objects in JavaScript Object Notation (JSON) format.
- JSON files are lightweight, text-based, and human-readable.



What is a CSV file?

- A comma-separated values file, allows data to be saved in a tabular format.
- CSV files can be used with most any spreadsheet program, such as Microsoft Excel or Google Spreadsheets.



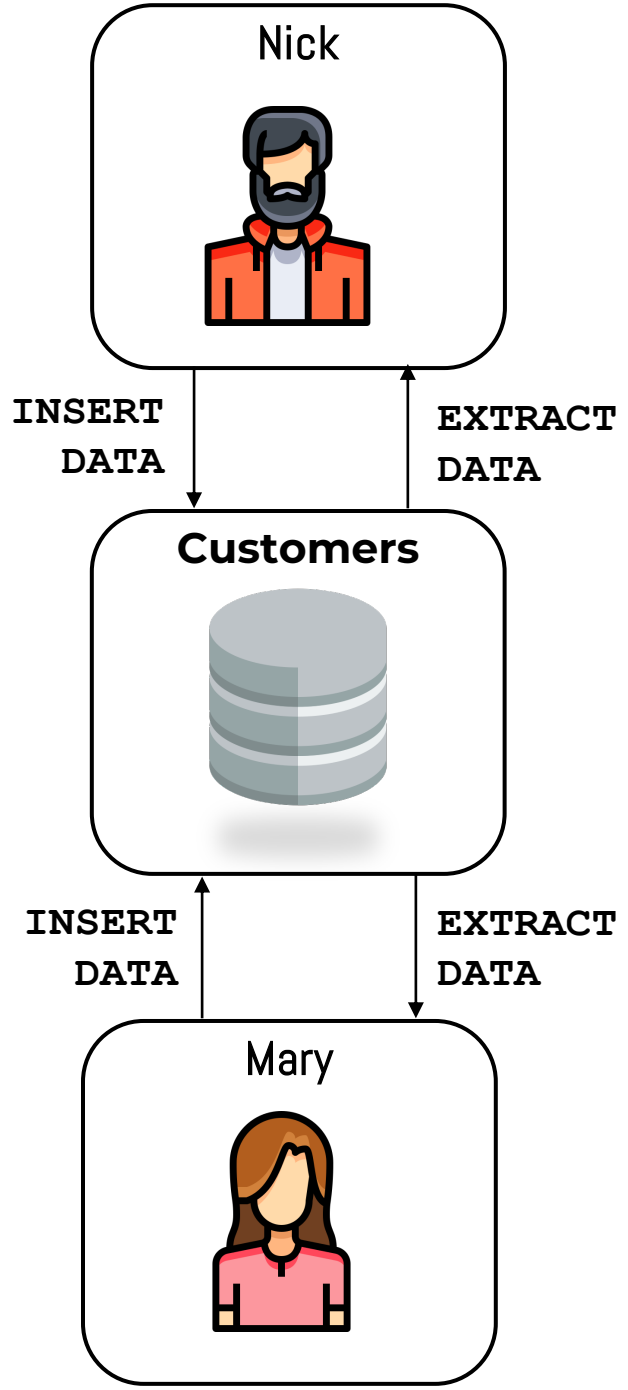
	A	B	C	D	E	F
1	SepalLength	SepalWidth	PetalLength	PetalWidth	Name	
2	5.1	3.5	1.4	0.2	Iris-setosa	
3	4.9	3	1.4	0.2	Iris-setosa	
4	4.7	3.2	1.3	0.2	Iris-setosa	
5	4.6	3.1	1.5	0.2	Iris-setosa	
6	5	3.6	1.4	0.2	Iris-setosa	
7	5.4	3.9	1.7	0.4	Iris-setosa	
8	4.6	3.4	1.4	0.3	Iris-setosa	
9	5	3.4	1.5	0.2	Iris-setosa	
10	4.4	2.9	1.4	0.2	Iris-setosa	
11	4.9	3.1	1.5	0.1	Iris-setosa	
12	5.4	3.7	1.5	0.2	Iris-setosa	
13	4.8	3.4	1.6	0.2	Iris-setosa	
14	4.8	3	1.4	0.1	Iris-setosa	
15	4.3	3	1.1	0.1	Iris-setosa	
16	5.8	4	1.2	0.2	Iris-setosa	
17	5.7	4.4	1.5	0.4	Iris-setosa	
18	5.4	3.9	1.3	0.4	Iris-setosa	
19	5.1	3.5	1.4	0.3	Iris-setosa	
20	5.7	3.8	1.7	0.3	Iris-setosa	
21	5.1	3.8	1.5	0.3	Iris-setosa	
22	5.4	3.4	1.7	0.2	Iris-setosa	
23	5.1	3.7	1.5	0.4	Iris-setosa	
24	4.6	3.6	1	0.2	Iris-setosa	

Quiz: Is it True or False?

- A CSV file is an example of a structured dataset. — False
- A JSON file is an example of a semi-structured dataset. — True
- A relational database is an example of a structured dataset. — True
- You can enforce rules and constraints on data input in a relational database. — True

Relational databases

- Traditional databases are stored on the filesystem of one single machine.
  - **Centralized data storage!**
- Users can insert/query data from the machine directly, and they always extract the same data.

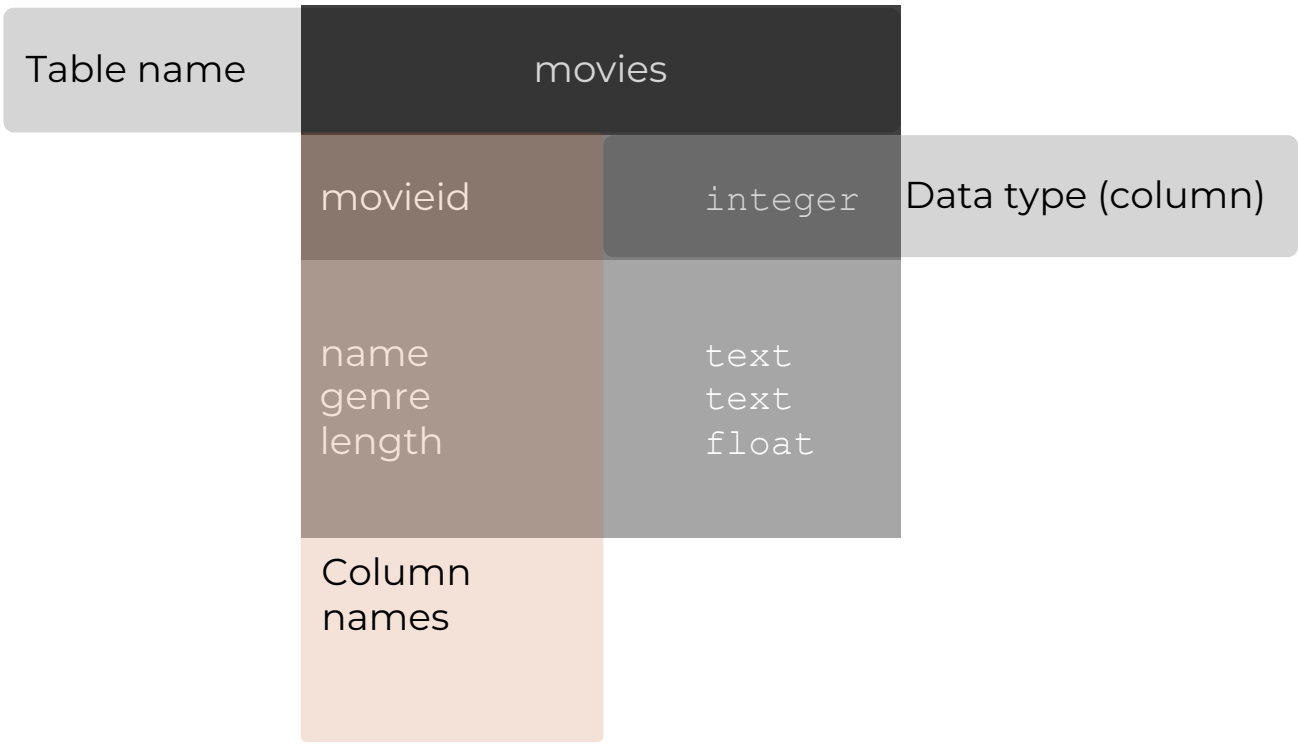


Today we focus on RDBMS ☺

- A type of database that stores and provides access to data points related to one another.
- A logical view of the dataset, where data is stored in tables.
- A table is a 2-D data structure with rows (records) and columns (labels).

RDBMS Tables

- Data is structured in tables
  - Data is well organized
  - Data cleaning is not needed
  - It is much easier to insert, update or delete
  - Much easier to search

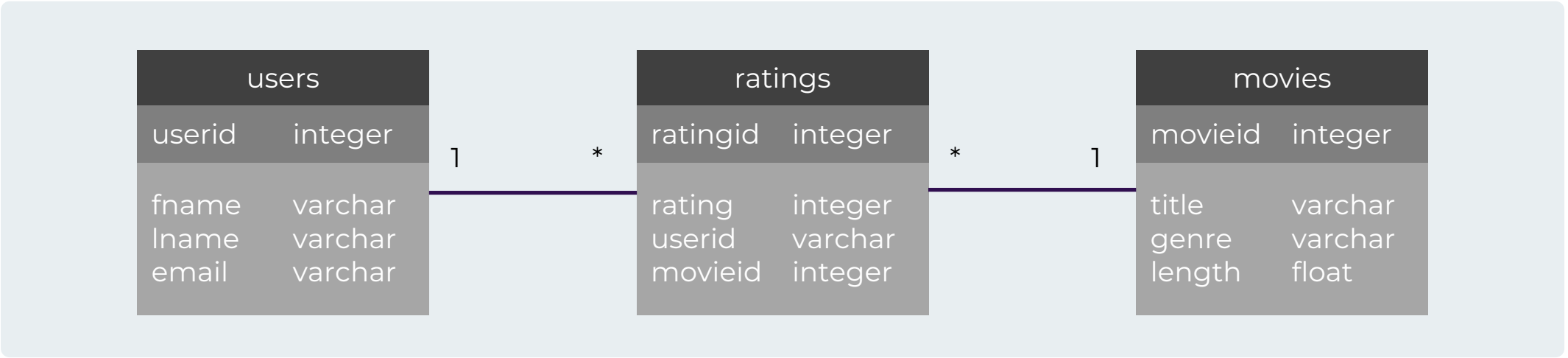


What is a key?

- A primary key is an attribute that uniquely identifies any given entity
    - Examples?
        - Passport ID, NINO, NHS Number
- Primary Key characteristics:
    - It cannot be empty for a record (Not Null)
    - It has to be unique

What is a data model (schema)?

- A data model is an abstract model that organizes elements of data and standardizes how they relate to one another and to the properties of real-world entities.



What is wrong with the following data?

Book_ID	Book_Title	Book_Author	Author_Address	Authors_Phone_Number
1	Hobbit	Tolkien	Bournemouth	01202 345234
2	LOTR 1	Tolkien	Bournemouth	01202 345234
3	LOTR 2	Tolkien	Bournemouth	01202 345234
4	LOTR 3	Tolkien	Bournemouth	01202 345234
5	The Silmarillion	Tolkien	Bournemouth	01202 345234

This dataset has repeated data ☹

* **5 rows * 5 columns = 25 data points**

Why is that bad?

Update is a problem...

Quiz: Can you redesign it?

- Produce a better design.
  - Define "better"...

Book_ID	Book_Title	Book_Author	Author_Address	Authors_Phone_Number
1	Hobbit	Tolkien	Bournemouth	01202 345234
2	LOTR 1	Tolkien	Bournemouth	01202 345234
3	LOTR 2	Tolkien	Bournemouth	01202 345234
4	LOTR 3	Tolkien	Bournemouth	01202 345234
5	The Silmarillion	Tolkien	Bournemouth	01202 345234

Designing a data model

Author_ID	Author_Name	Author_Address	Authors_Phone_Number
A1	Tolkien	Bournemouth	01202 345234

Book_ID	Book_Title	Author_ID
1	Hobbit	A1
2	LOTR 1	A1
3	LOTR 2	A1
4	LOTR 3	A1
5	The Silmarillion	A1

- Better means less data to manage.
- Now we have **19 data points.**
- We now have two tables, but RDBMS can still help to organise them...

Advantages of RDBMS

Management

- ✓ Easier to manage & maintain
- ✓ Easy to insert/update data
- ✓ Great support for relational models

Redundancy

- ✓ Focus on removing duplicated data
- ✓ Reduces the need for high storage
- ✓ Easy to backup & restore

Integrity

- ✓ Enforces referential integrity
- ✓ Ideal for consistency of data
- ✓ Support structural dependencies

What is referential integrity?

- A data property stating that all table relationships and references are valid.
- Examples:
  - You cannot order on Amazon if you don’t have items in the cart.
  - You cannot delete an eBay customer and keep her/his orders.
  - A student who does not exist in the Birkbeck registry cannot take a module.

Let’s talk about databases: MySQL

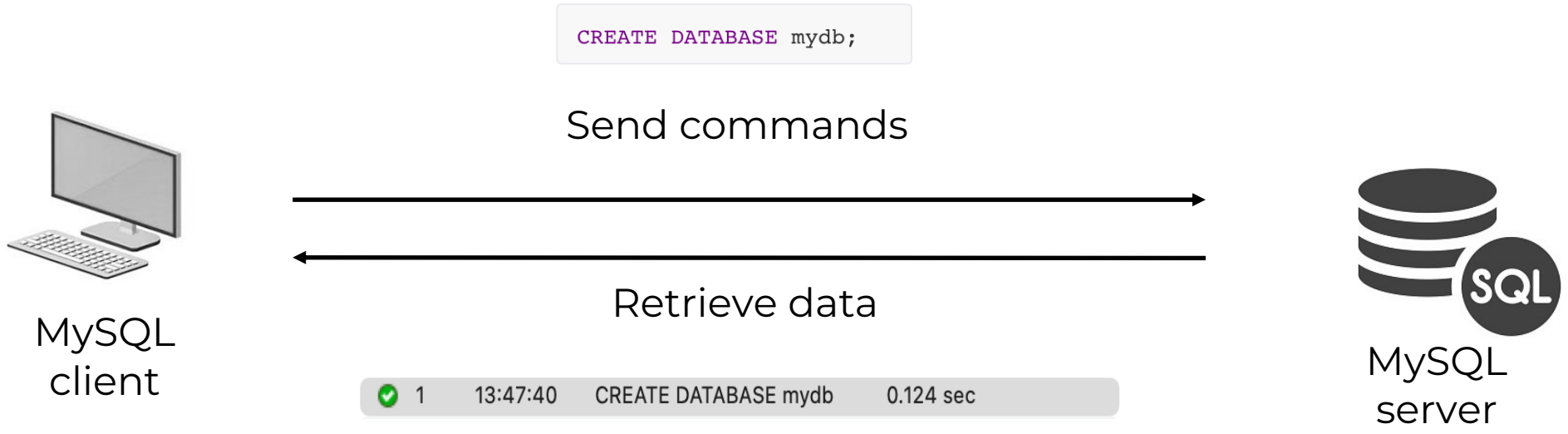
- MySQL is an open-source relational database management system.
- A MySQL RDBMS is available for free ☺
  - We will deploy (install) MySQL on the Google Cloud platform
- MySQL is one of the de-facto solutions for web application development
- Other RDBMSs:
  - PostgreSQL
  - Oracle SQL server
  - Microsoft SQL Server

SQL

- SQL stands for Structured Query Language
- SQL is used to communicate with a database and send commands
  - The SQL commands help you interact with a database system
- SQL is the standard language for relational database management systems

Database (DB) setup: Client and Server

- A **DB server** is responsible for storing the data
  - Usually, we store data in a Cloud environment, not in our laptops!
- A **Client** connects and extracts data



SQL Datatypes

- `VARCHAR(size)`
  - Can contain letters, numbers, and special characters.
  - The size parameter specifies the maximum column length in characters
  - Can be from 0 to 65535
- `TEXT`
  - A field with a maximum length of 65535 characters
- `INTEGER(M)`
  - M is optional and denotes the display length
  - The signed range is from -2147483648 to 2147483647
  - The unsigned range is from 0 to 4294967295
- `FLOAT(M,D)`
  - A floating-point number (signed)
  - You can define the display length (M) and the number of decimals (D)

What is the data type?

* Up to 50 characters

VARCHAR(50)

* Up to 10 numbered digits

Integer(10)

author_id	author_name	author_salary	author_phone_number	author_description
A1	Tolkien	2555.34	01202345234	He was the author of …

VARCHAR(25)

* Up to 25 characters

Float(10,2)

* Up to 10 digits and 2 decimal points

Text

* By default, up to 65535 characters

Create a new table

Table Name: cities

city_id	name	country
1	London	GB
2	New York City	US

```
CREATE TABLE cities (
    city_id INTEGER PRIMARY KEY,
    name VARCHAR(24) NOT NULL,
    country VARCHAR(2)
);
```

INTEGER	→ Integer ☺
PRIMARY KEY	→ Unique and cannot be empty
VARCHAR(24)	→ Set of up to 24 characters
NOT NULL	→ Cannot be empty

What are the SQL constraints?

- SQL constraints are used to specify rules for the data in a table.
  - **`NOT NULL`** → Ensures that a column cannot have a `NULL` value
  - **`UNIQUE`** → Ensures that all values in a column are different
  - **`PRIMARY KEY`** → Uniquely identifies each row in a table, and it is always required
  - **`FOREIGN KEY`** → Prevents actions that would destroy links between tables
  - **`CHECK`** → Ensures that the values in a column satisfy a specific condition
  - **`DEFAULT`** → Sets a default value for a column if no value is specified
  - **`CREATE INDEX`** → Used for fast creating and data retrieval

SQL DDL vs DML

Data Definition Language (DDL)

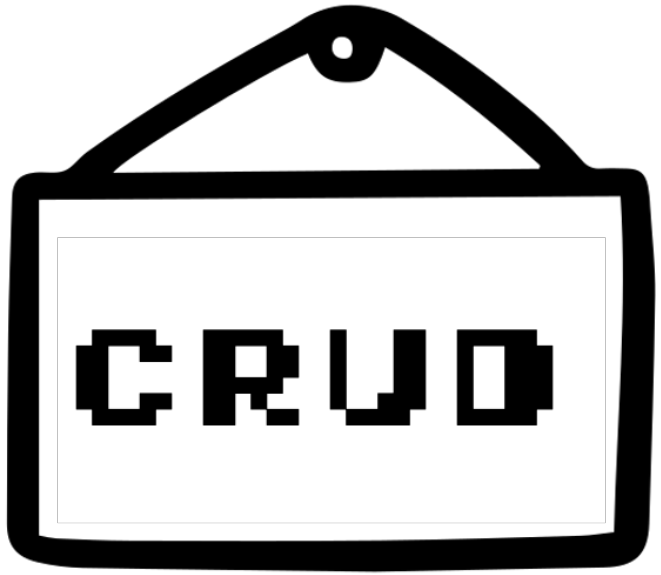
- ✓ Create/Alter database schema and constraints
- ✓ Create attributes & data types
- ✓ Commands include CREATE, DROP, RENAME, ALTER etc.

Data Manipulation Language (DML)

- ✓ Add, retrieve or update the data
- ✓ Commands include: SELECT, INSERT, UPDATE etc.
- ✓ Uses the WHERE clause to retrieve or manipulate data

Basic DML operations on data

- **C**reate
  - Insert data to one or more tables
- **R**ead
  - Query for specific columns and rows
- **U**pdate
  - Update a record or a set of records
- **D**elete
  - Delete a record or a set of records



Create (Insert) data

- Insert data to a table:

```
INSERT INTO cities (city_id, name, country)
VALUES (1,'London','GB’);
```

- Insert multiple data to a table:

```
INSERT INTO cities (city_id, name, country) VALUES
(1,'London','GB’),
(2,’New York City','US');
```

Read (Select) data from a table

- The SELECT statement is used to select data from a database
- We can select all the columns (*) from a table using the next command:

```
SELECT * FROM cities;
```

- Or we can select specific columns e.g., city name and country:

```
SELECT name, country FROM cities;
```

- We can also query for specific rows e.g., data from GB:

```
SELECT name, country FROM cities WHERE country="GB";
```

Update data

- You can update the existing data using the UPDATE command
- You need to specify the SET and WHERE conditions
- For example, update the country name to UK using city with id 1:

```
UPDATE cities SET country='UK' WHERE city_id=1;
```

Delete data

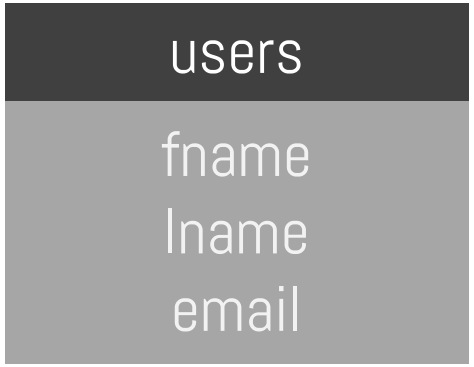
- You can delete the existing data using the DELETE command
- You need to specify the WHERE condition(s)
- For example, delete all the data from Great Britain:

```
DELETE FROM cities WHERE country = 'GB';
```

Part 2: Building data models

Entity Relationship Model (ER)

- ER model is used to show the conceptual design of a database.
- It is a diagram to represent tables, fields and their relationships



Entity: distinguishable “thing” in the real world

Attributes: properties of the “thing”

Let’s design a database!

- Consider the following scenario:

  A friend asked you to design and develop a database for her company.

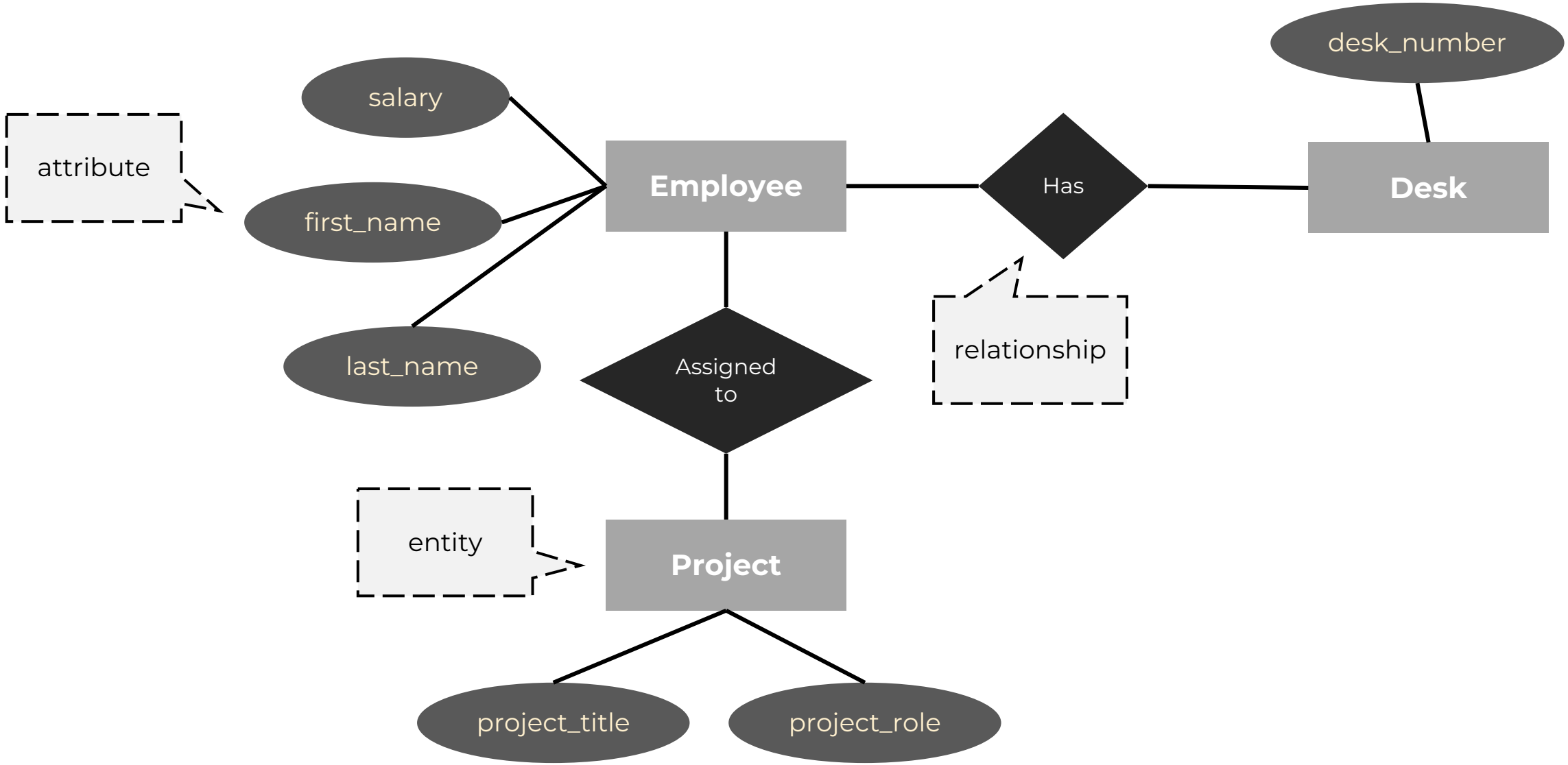
  - My employees are always working on one project.
  - In a project, many employees might be assigned to work.
  - Each employee is always assigned to a specific desk.
  - The employees do not share desks.

Let’s create a database diagram!

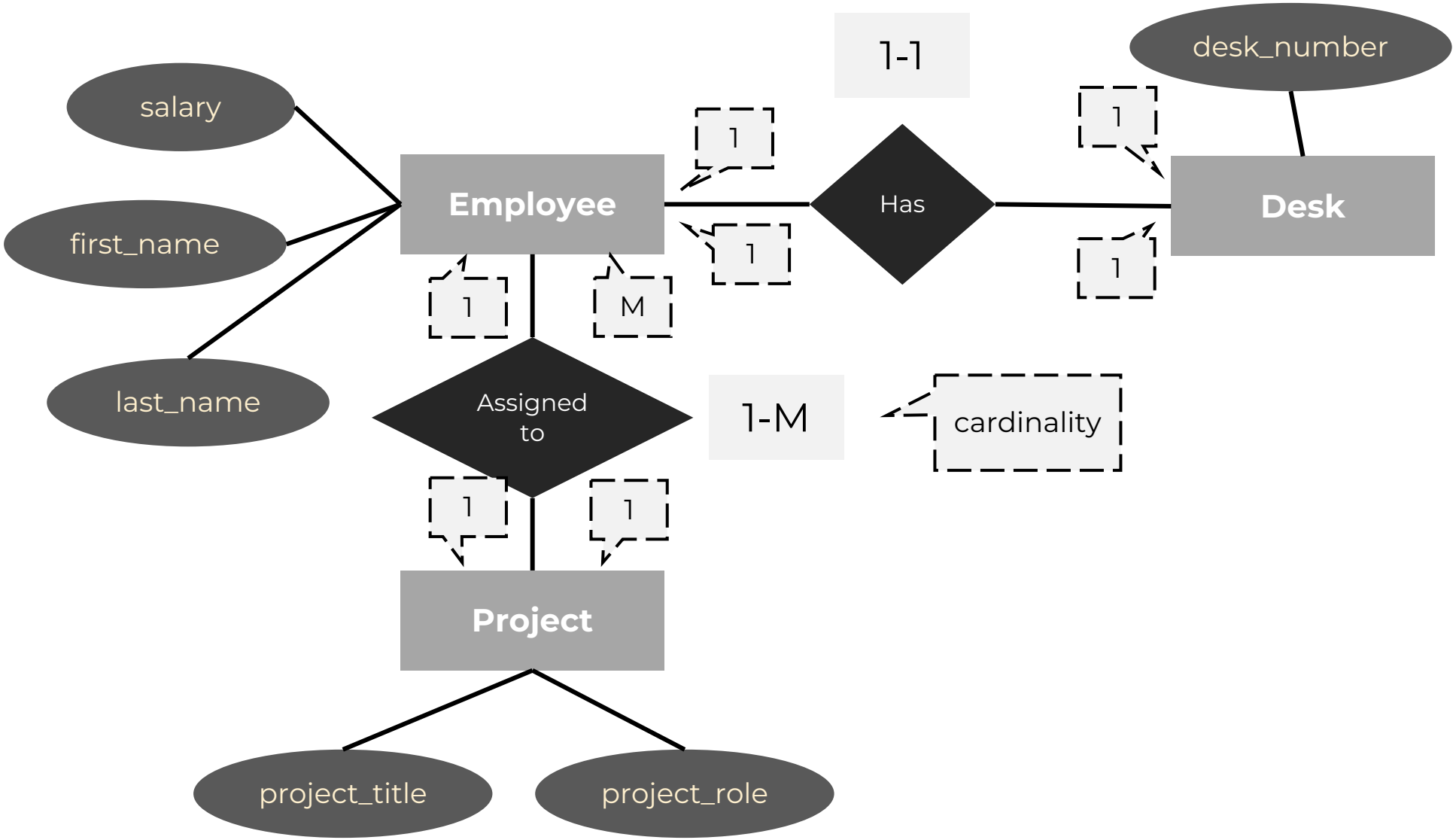
What are the entities?

- My employees are always working on one project.
- In a project, many employees might be assigned to work.
- Each employee is always assigned to a specific desk.
- The employees do not share desks.

ER Model example



ER Model example

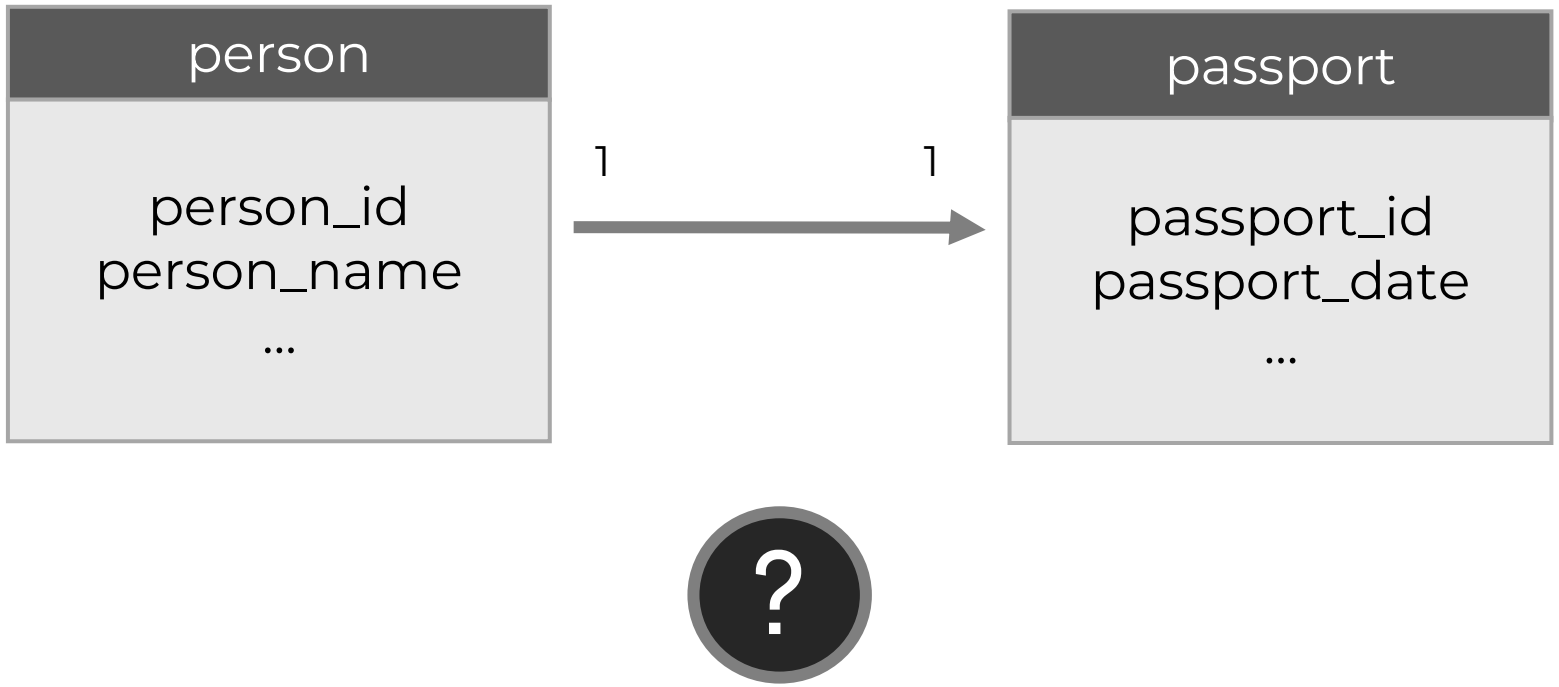


ER Cardinality

- The number of times an entity of an entity set participates in a relationship.
  - **One to one** – When each entity in each entity set can take part only once in the relationship, the cardinality is one to one.
  - **Many to one** – When entities in one entity set can take part only once in the relationship set and entities in other entity set can take part more than once in the relationship set, cardinality is many to one.
  - **Many to many** – When entities in all entity sets can take part more than once in the relationship cardinality is many to many.

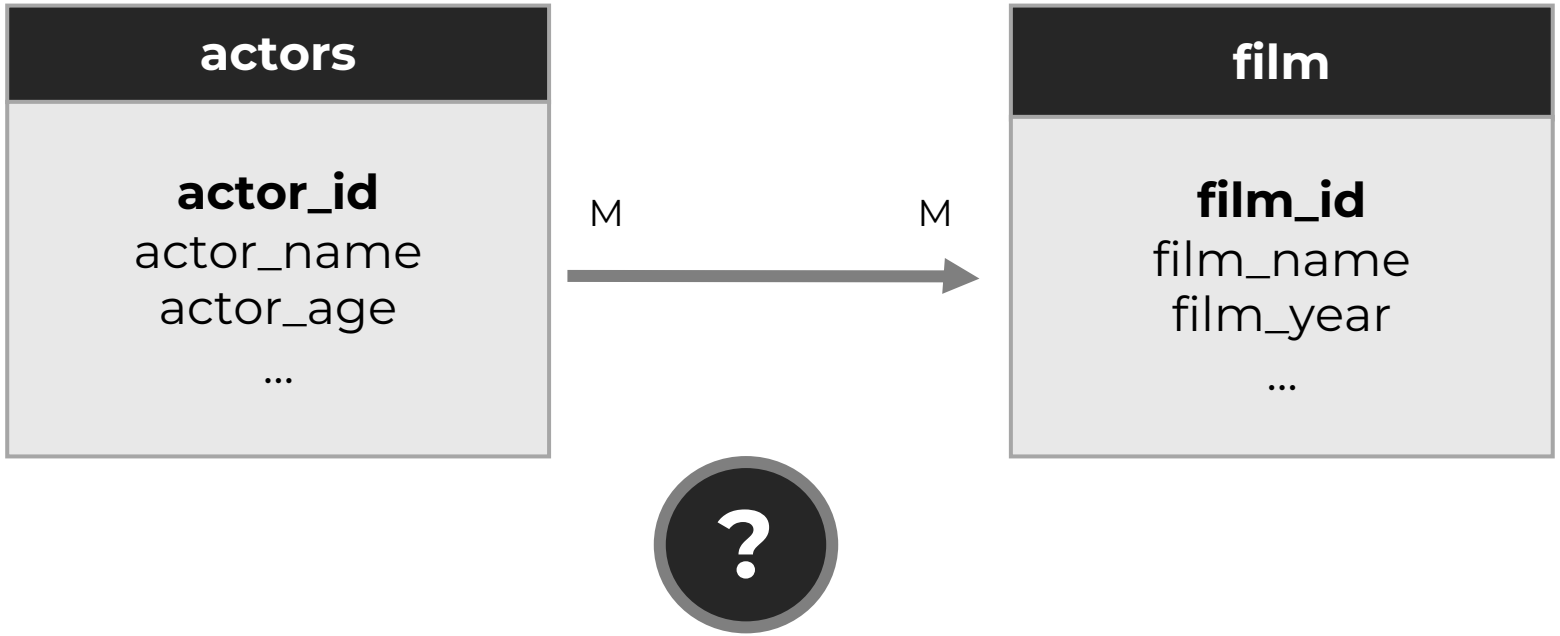
What is the relationship?

- How can you link the next tables?



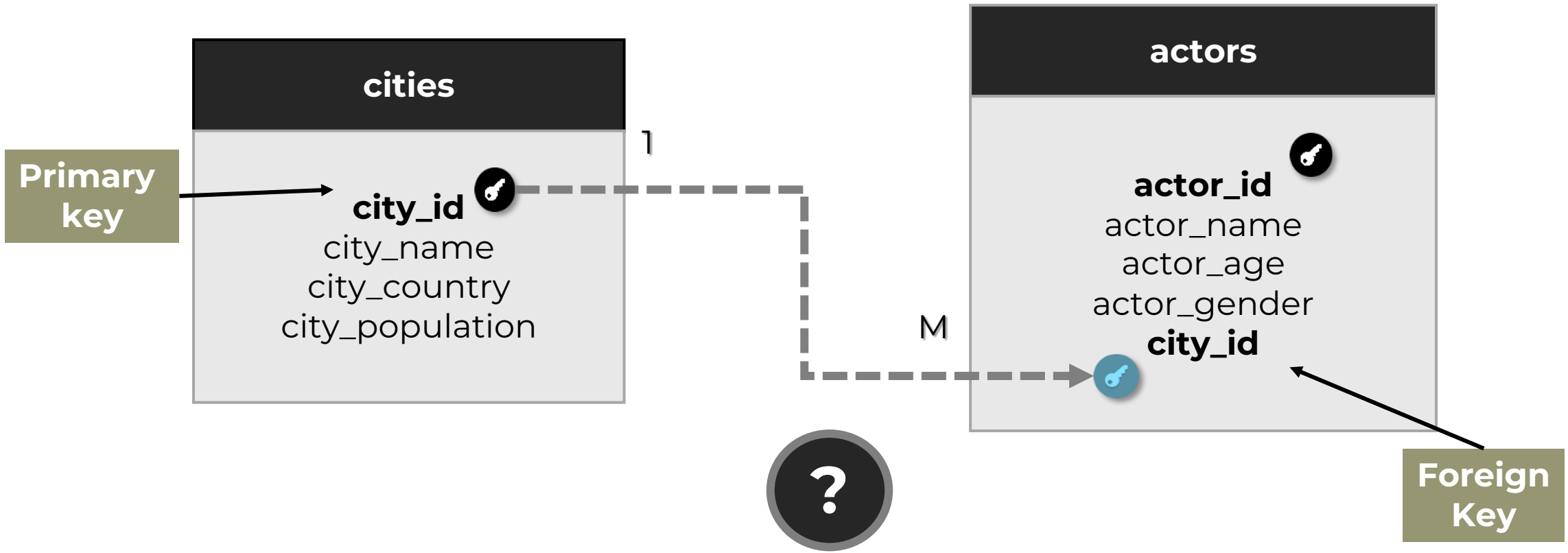
What is the relationship (cont.)?

- How do you link actors to movies they **worked in** (or vice-versa) in a DB with these two tables?



What is the relationship?

- We need to move the **`city_id`** into the **`actors`** table



Let us create two new tables...

cities

city_id	city_name	city_country	city_population
C1	London	England	8.982.000
C2	Athens	Greece	3.154.000
C3	Toronto	Canada	6.197.000

actors

actor_id	actor_name	actor_age	actor_gender	city_id
A1	Emilia Clarke	34	F	C1
A2	Jennifer Aniston	51	F	C2
A3	Jim Carrey	58	M	C3

MySQL scripts

- Let's create the tables using Primary and Foreign keys in SQL

```
CREATE TABLE cities (
city_id VARCHAR(30),
city_name VARCHAR(30) NOT NULL,
city_country VARCHAR(30) NOT NULL,
city_population INT(8),
PRIMARY KEY (city_id)
);
```

```
CREATE TABLE actors (
actor_id VARCHAR(30),
actor_name VARCHAR(30) NOT NULL,
actor_age INT(8),
actor_gender VARCHAR(30) NOT NULL,
city_id VARCHAR(30),
PRIMARY KEY (actor_id),
FOREIGN KEY (city_id) REFERENCES cities(city_id)
);
```

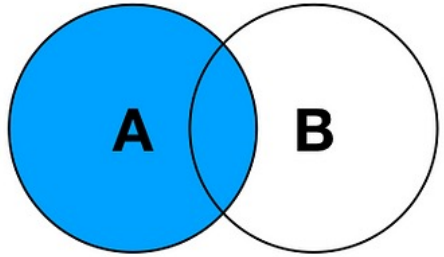
Relational algebra (RA) operations

- Operations on two sets (tables):
  - Cartesian product (x): combine rows from two tables in all possible ways
  - Joins: combine rows, applying different strategies
  - Union (∪) and intersection (∩)
- These operations are combined to extract data from DBs

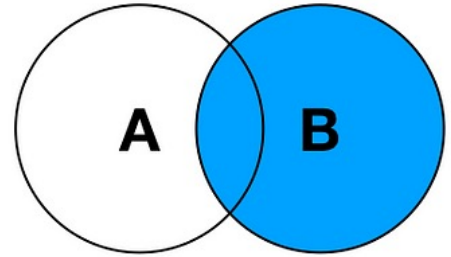
Back to school! Union, intersection, and difference of sets!

- Given two sets:
  - A={1,2,3,4}
  - B={3,4,5,6}
- What is $A \cup B$? {1,2,3,4,5,6}
- What is $A \cap B$? {3,4}
- What is $B - A$? {5,6}

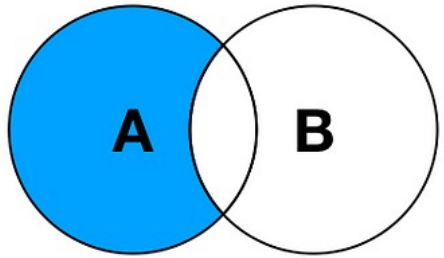
SQL JOINS



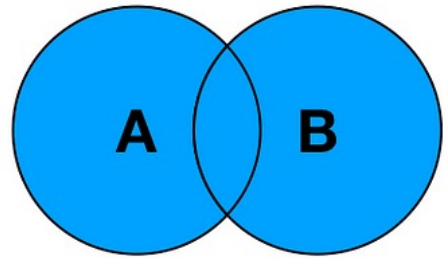
LEFT JOIN



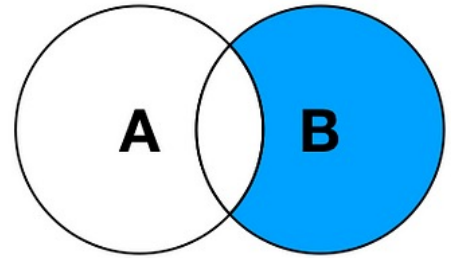
RIGHT JOIN



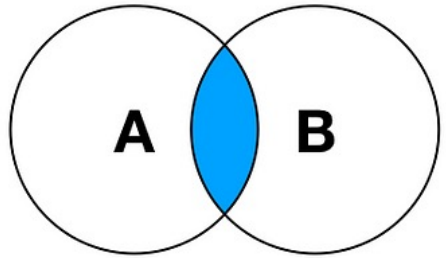
LEFT JOIN EXCLUDING
INNER JOIN



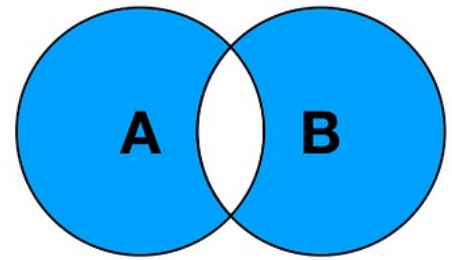
FULL OUTER JOIN



RIGHT JOIN EXCLUDING
INNER JOIN

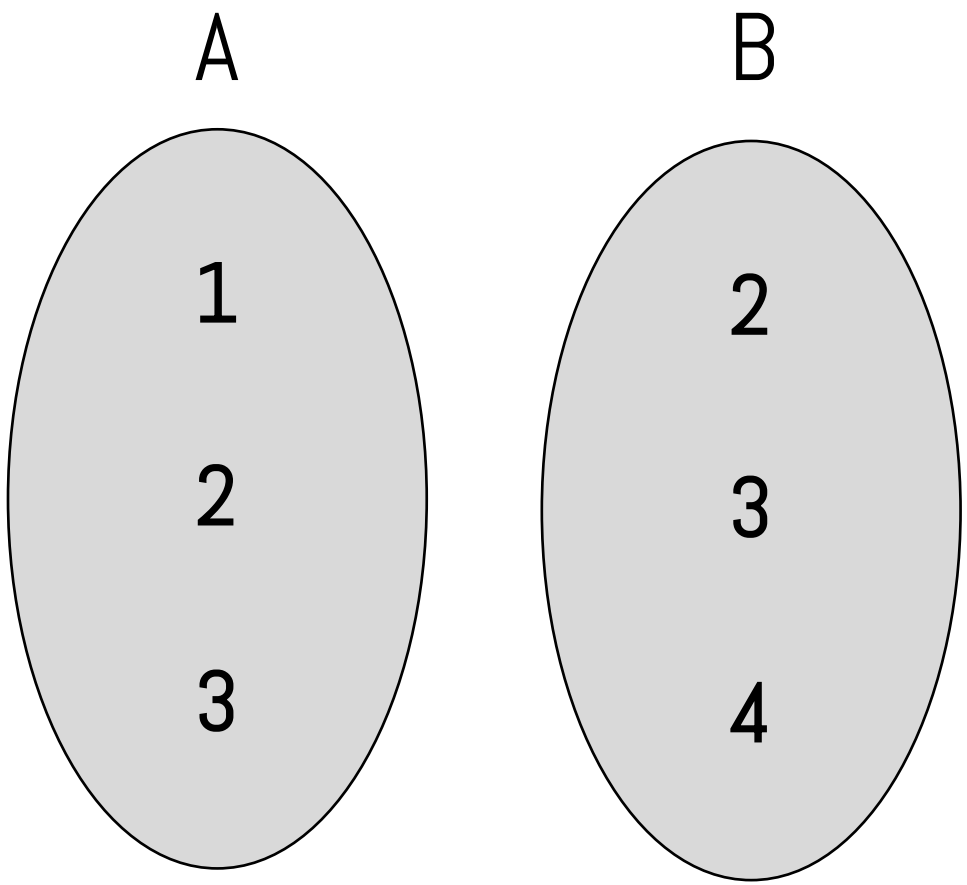


INNER JOIN



FULL OUTER JOIN EXCLUDING
INNER JOIN

Example of Joins



- A inner join B
  - {2,3}
- A left join B
  - {1,2,3}
- A right join B
  - {2,3,4}
- A full outer join B
  - {1,2,3,4}

Cartesian product (or “cross join”)

- Generate a **new table** with all possible rows from `table1` and `table2`
- **Columns** of the new table are from both `table1` and `table2`

```
SELECT columns FROM table1,table2;
```

What is the output of this script?

actors

actor_id	actor_name	actor_age	actor_gender	city_id
A1	Emilia Clarke	34	F	C1
A2	Jennifer Aniston	51	F	C2
A3	Jim Carrey	58	M	C3
A4	Sarah Paulson	46	F	C4

cities

city_id	city_name	city_country	city_population
C1	London	England	8982000
C2	Athens	Greece	3154000
C3	Toronto	Canada	6197000
C4	Tampa	United States	392890

```
SELECT * FROM actors, cities;
```

Cartesian product (or “cross join”)

```
SELECT * FROM actors, cities;
```

actor_id	actor_name	actor_age	actor_gender	city_id	city_id	city_name	city_country	city_population
A4	Sarah Paulson	46	F	C4	C1	London	England	8982000
A3	Jim Carrey	58	M	C3	C1	London	England	8982000
A2	Jennifer Aniston	51	F	C2	C1	London	England	8982000
A1	Emilia Clarke	34	F	C1	C1	London	England	8982000
A4	Sarah Paulson	46	F	C4	C2	Athens	Greece	3154000
A3	Jim Carrey	58	M	C3	C2	Athens	Greece	3154000
A2	Jennifer Aniston	51	F	C2	C2	Athens	Greece	3154000
A1	Emilia Clarke	34	F	C1	C2	Athens	Greece	3154000
A4	Sarah Paulson	46	F	C4	C3	Toronto	Canada	6197000
A3	Jim Carrey	58	M	C3	C3	Toronto	Canada	6197000
A2	Jennifer Aniston	51	F	C2	C3	Toronto	Canada	6197000
A1	Emilia Clarke	34	F	C1	C3	Toronto	Canada	6197000
A4	Sarah Paulson	46	F	C4	C4	Tampa	United Sta…	392890
A3	Jim Carrey	58	M	C3	C4	Tampa	United Sta…	392890
A2	Jennifer Aniston	51	F	C2	C4	Tampa	United Sta…	392890
A1	Emilia Clarke	34	F	C1	C4	Tampa	United Sta…	392890

- A new table with 9 rows that include all possible combinations which is meaningless!

Equi-join (or “inner join”)

- Two (or more) attributes must be equal in each row
- The results show a meaningful relationship between the two tables

```
SELECT columns
FROM table1,table2
WHERE table1.primary_key = table2.foreing_key;
```

Equi-join (or “inner join”)

actors

actor_id	actor_name	actor_age	actor_gender	city_id
A1	Emilia Clarke	34	F	C1
A2	Jennifer Aniston	51	F	C2
A3	Jim Carrey	58	M	C3
A4	Sarah Paulson	46	F	C4

cities

city_id	city_name	city_country	city_population
C1	London	England	8982000
C2	Athens	Greece	3154000
C3	Toronto	Canada	6197000
C4	Tampa	United States	392890

```
SELECT *
FROM actors,cities
WHERE actors.city_id=cities.city_id;
```

Equi-join (or “inner join”)

```
SELECT * FROM actors, cities
WHERE actors.city_id=cities.city_id;
```

actor_id	actor_name	actor_age	actor_gender	city_id	city_id	city_name	city_country	city_population
A1	Emilia Clarke	34	F	C1 ⟷	C1	London	England	8982000
A2	Jennifer Aniston	51	F	C2 ⟷	C2	Athens	Greece	3154000
A3	Jim Carrey	58	M	C3 ⟷	C3	Toronto	Canada	6197000
A4	Sarah Paulson	46	F	C4 ⟷	C4	Tampa	United States	392890

Equi-join with selection

```
SELECT * FROM actors, cities
WHERE actors.city_id=cities.city_id
AND actor_name = 'Jim Carrey';
```

actor_id	actor_name	actor_age	actor_gender	city_id	city_id	city_name	city_country	city_population
A3	Jim Carrey	58	M	C3 ⟷	C3	Toronto	Canada	6197000

- You can extract the actor **Jim Carrey**

Left/right join

- The equi-join only keeps rows where both tables have matching values
- What if we want to enrich the records from table A with data from Table B, while keeping all records from table A?
  - Actors with birth city information (even with actors with birth_city = NULL)
  - Actors and their first films, also showing actors who never worked in a film

```
SELECT *
FROM actors
RIGHT JOIN cities ON
actors.city_id=cities.city_id;
```

Right join: Select actors and all available cities

actors

actor_id	actor_name	actor_age	actor_gender	city_id
A1	Emilia Clarke	34	F	C1
A2	Jennifer Aniston	51	F	C2
A3	Jim Carrey	58	M	C3
A4	Sarah Paulson	46	F	C4

cities

city_id	city_name	city_country	city_population
C1	London	England	8982000
C2	Athens	Greece	3154000
C3	Toronto	Canada	6197000
C4	Tampa	United States	392890
C5	Tokyo	Japan	9273000

New data

```
SELECT * FROM actors
RIGHT JOIN cities
ON actors.city_id=cities.city_id;
```

Right join: Select actors and all available cities

actors

actor_id	actor_name	actor_age	actor_gender	city_id
A1	Emilia Clarke	34	F	C1
A2	Jennifer Aniston	51	F	C2
A3	Jim Carrey	58	M	C3
A4	Sarah Paulson	46	F	C4

cities

city_id	city_name	city_country	city_population
C1	London	England	8982000
C2	Athens	Greece	3154000
C3	Toronto	Canada	6197000
C4	Tampa	United States	392890
C5	Tokyo	Japan	9273000

New data

actor_id	actor_name	actor_age	actor_gender	city_id	city_id	city_name	city_country	city_population
A1	Emilia Clarke	34	F	C1	C1	London	England	8982000
A2	Jennifer Aniston	51	F	C2	C2	Athens	Greece	3154000
A3	Jim Carrey	58	M	C3	C3	Toronto	Canada	6197000
A4	Sarah Paulson	46	F	C4	C4	Tampa	United States	392890
NULL	NULL	NULL	NULL	NULL	C5	Tokyo	Japan	9273000

Aggregating records

- In data science, we often need to get aggregate results, and not individual records (e.g., mean, count, etc.).
  - E.g., How many actors were born in each city?
- SQL can group records into blocks, and then aggregate the results with operators (**GROUP BY**)

Aggregating records

- **AVG()** → Returns the average value
- **COUNT()** → Returns the number of rows
- **MAX()** → Returns the largest value
- **MIN()** → Returns the smallest value
- **SUM()** → Returns the sum

Aggregating records

- Operators like **COUNT, SUM, AVG etc.** take multiple values and return a single one

```
SELECT actor_gender,count(actor_gender)
FROM actors,cities
WHERE actors.city_id=cities.city_id
GROUP BY actor_gender;
```

actor_gender	count(actor_gender)
F	3
M	1

Using an Alias (AS)

- Using AS as an alias, it is ideally for renaming our columns

```
SELECT actor_gender AS "Actor Gender",
count(actor_gender) AS "Count Gender"
FROM actors, cities
WHERE actors.city_id=cities.city_id
GROUP BY actor_gender;
```

Actor Gender	Count Gender
F	3
M	1

Lab 5

Big Data Analytics

What is the Google Cloud Platform (GCP)?

- We will explore more on the GCP more as we move forward.
- GCP is a set of services that help you run your code, store your data, and create applications.
  - There is a variety of databases available to use.
  - Cloud theory is out of the scope of this class.

Introduction to SQL

- SQL is very easy or very difficult!
  - Commands can grow into very complex queries...
- Essential skill for every data scientist!
- Any DS interview will always include writing SQL
- No need to memorise commands, but keep a logbook
- Take your time, no need to complete the lab in one go, it is important to understand the commands

Introduction to the Google Cloud Platform

Task 1:

- Redeem your $50 coupon
- Access the SQL server

Task 2:

1. Tutorial: Intro to SQL (part-1/README.md)
2. Exercises (part-1/Exercises.md)
3. Tutorial: Advanced SQL queries (part-2/README.md)

Lab activities

- Complete lab 5 activities and exercises.
- Use the GCP to run exercises in MySQL.