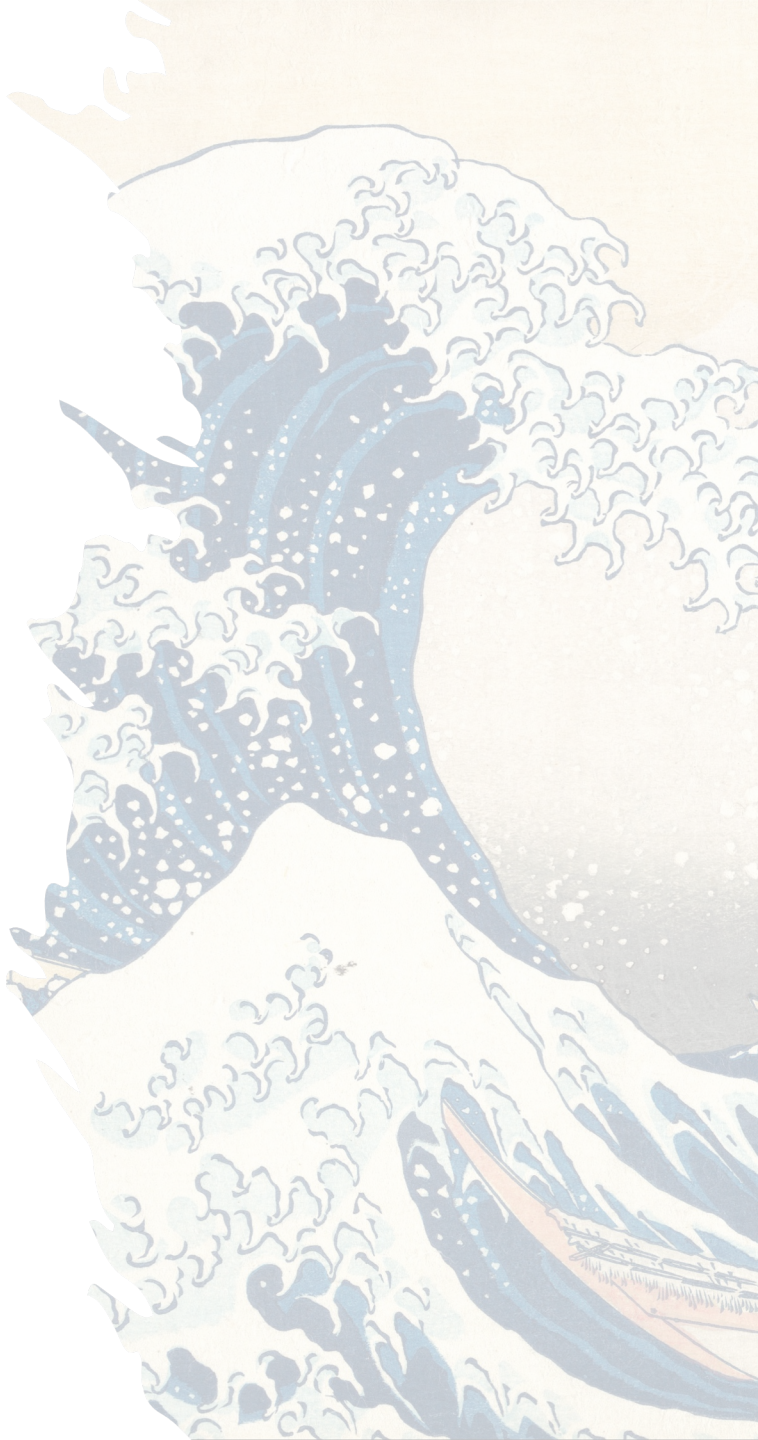
Stelios Sotiriadis

6. Advanced SQL

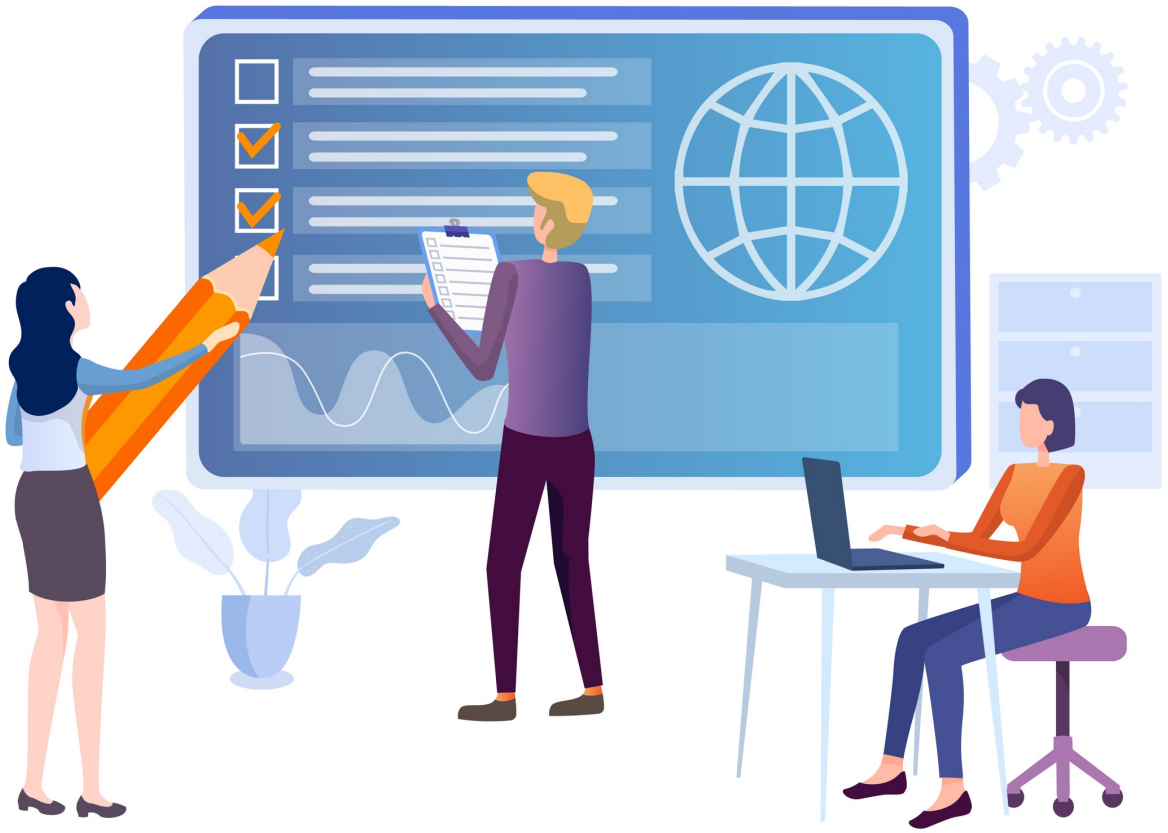
Agenda

- ER modelling
- Introduction to SQL
  - Relational algebra
  - SQL commands
- Lab 6 – MySQL race lab & tutorial



Quiz of the day

Get ready!



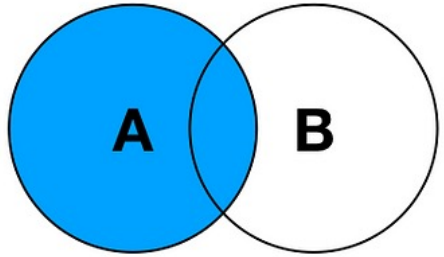
Back to school! Union, intersection, and difference of sets!

- 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}

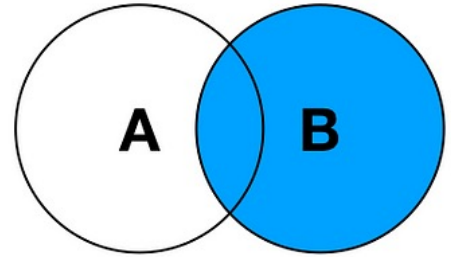
Relational algebra (RA) operations

- 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

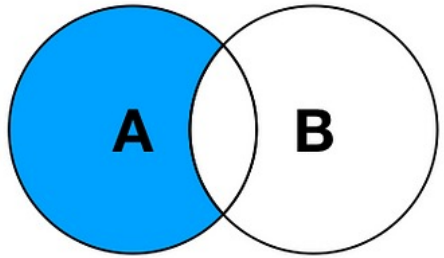
SQL JOINS



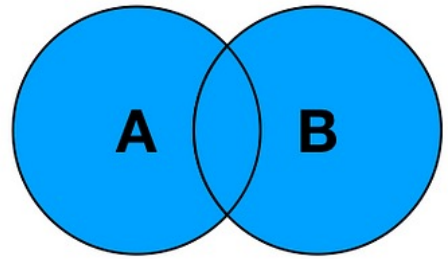
LEFT JOIN



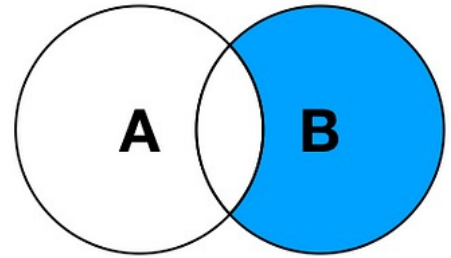
RIGHT JOIN



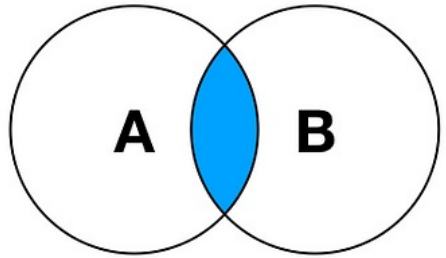
LEFT JOIN EXCLUDING
INNER JOIN



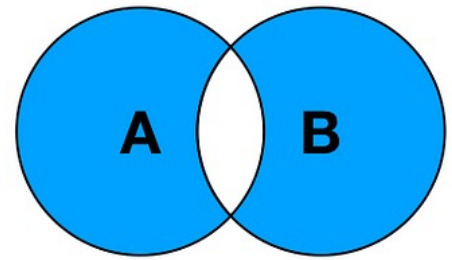
FULL OUTER JOIN



RIGHT JOIN EXCLUDING
INNER JOIN

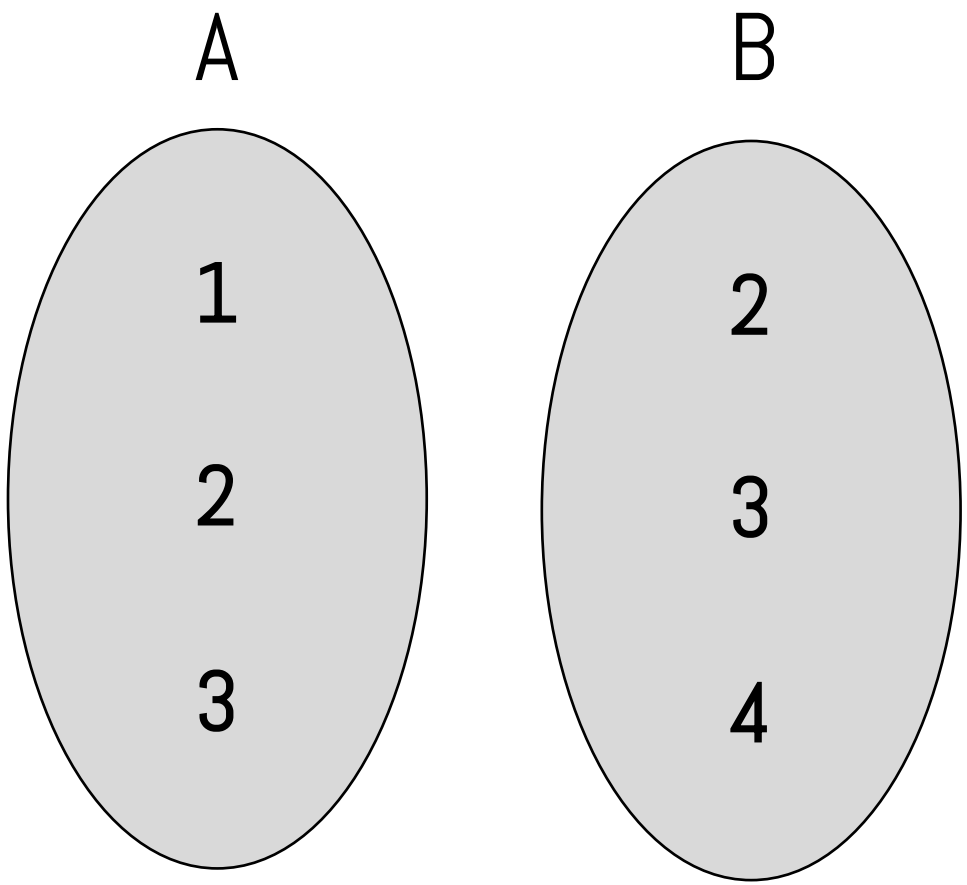


INNER JOIN



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

True or False?

- The cartesian product combines rows from two tables in all possible ways. — False
- The cartesian product shows the meaningful relationship between the two tables. — True
- The inner join results show the meaningful relationship between the two tables. — True
- A left join keyword returns all records from a right table, and the matched records from a left table. — False

What is the SQL query?

- What is each actor's city? Show only the actor's name and the associated city.

actors

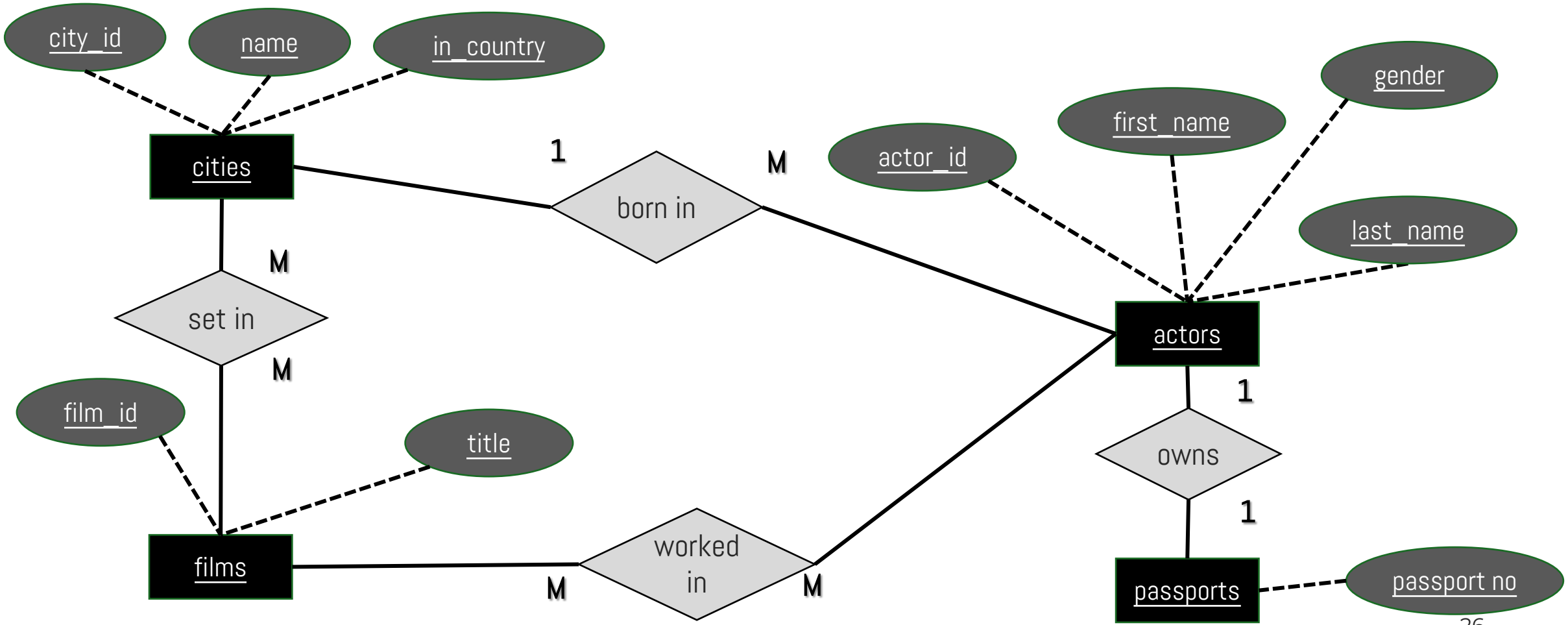
actor_id	actor_name	actor_age	actor_gender	city_id
A1	Emilia Clarke	34	F	C1
A2	Jennifer Aniston	51	F	C2
A3	Jim Carrey	58	M	C3
A4	Sarah Paulson	46	F	C4

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

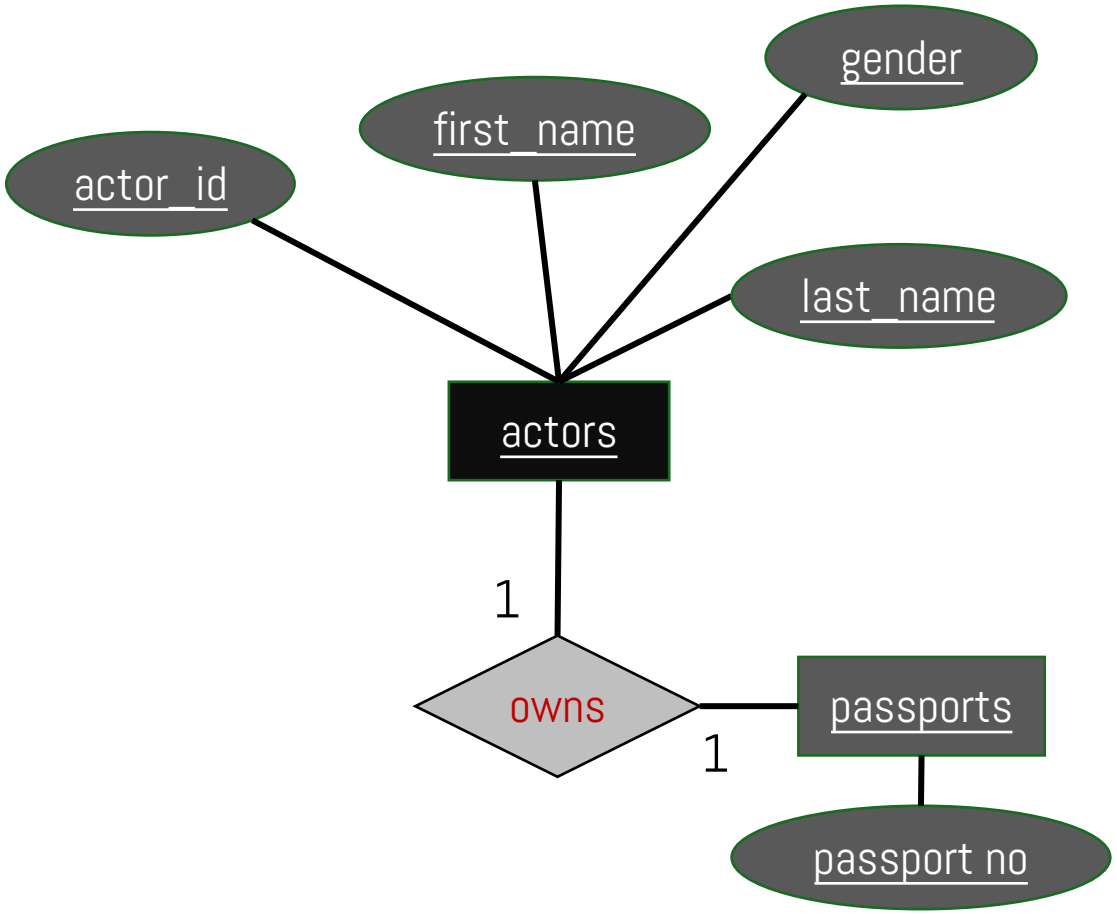
```
SELECT actors.actor_name, cities.city_name
FROM actors, cities
WHERE actors.city_id = cities.city_id;
```

Film DB: ER relationships



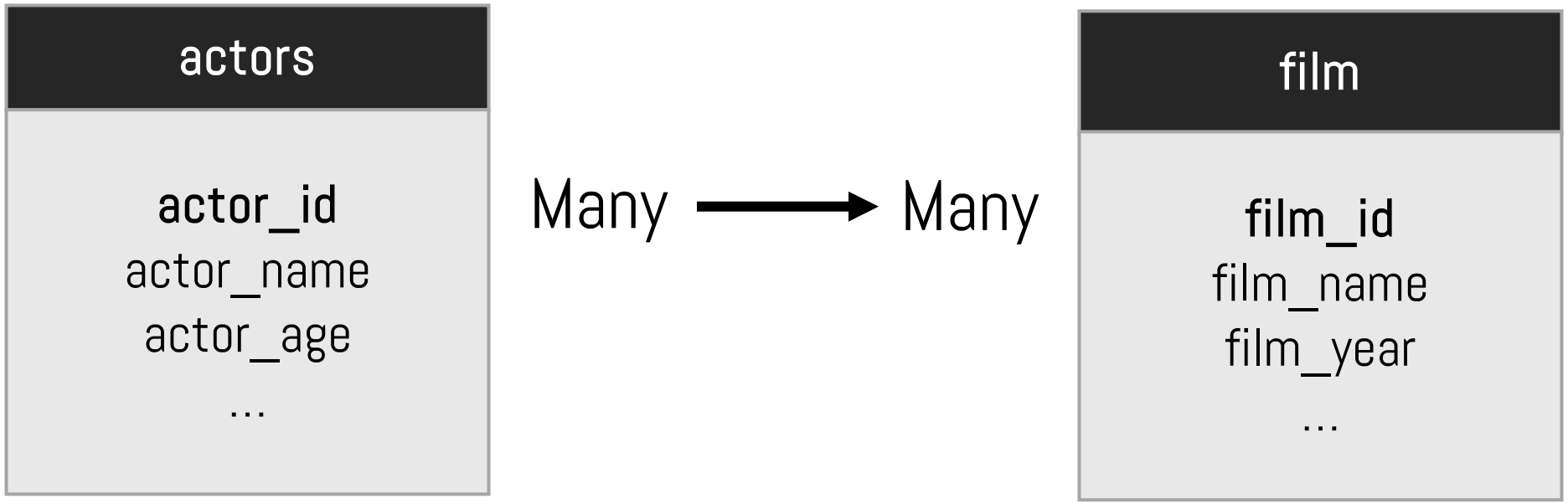
1-to-1 relationship

- A one to one (1:1) relationship is the relationship of one entity to only one other entity, and vice versa.
- It should be rare in any relational database design.
  - In fact, it could indicate that two entities actually belong in the same table!



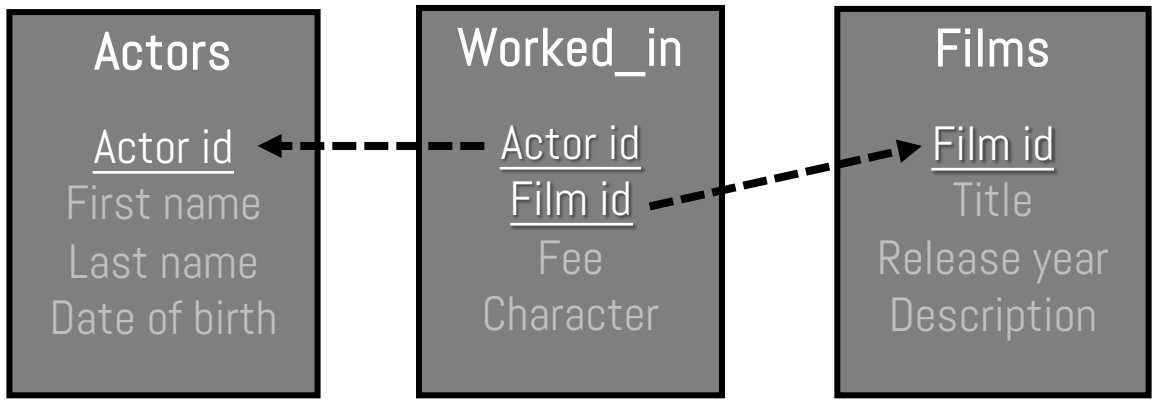
Many-to-many relationships

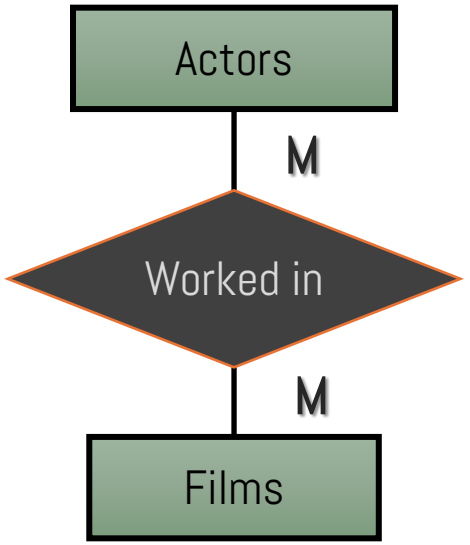
- How do you link **actors to the movies they worked in,** in a DB with these two tables?



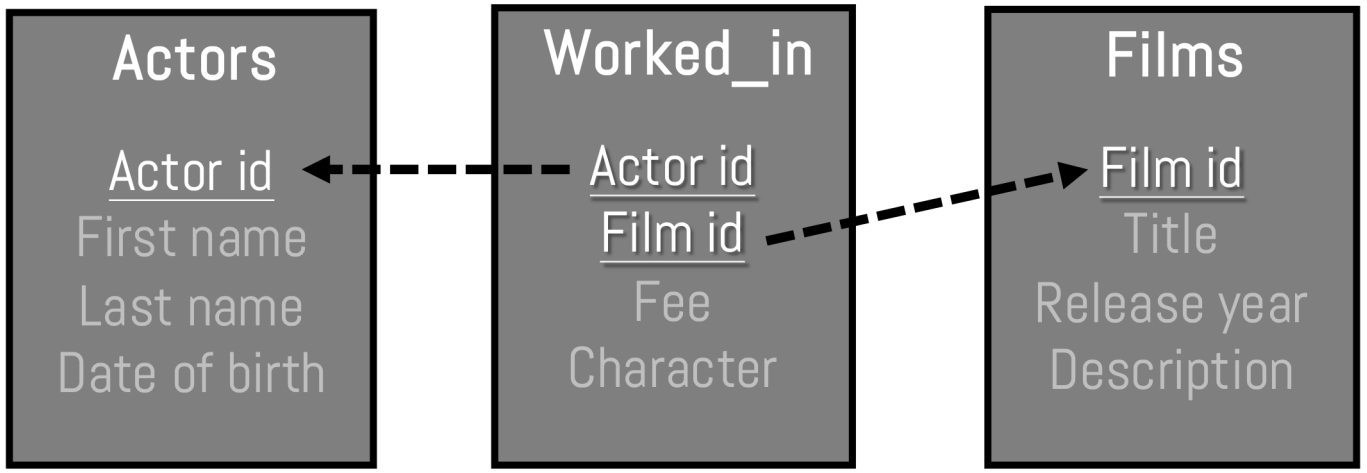
How to build a Many-to-Many relationship?

- We need to create a new **additional table!**
- The new table's primary key (PK) is a **composite key** of the two foreign keys (FK).
  - This means that the combination of **actor_id** and **film_id** is unique and not null.





Many-to-many relationships



- Constraint that is automatically enforced in this scenario:
  - The same actor cannot work more than once in the same film.
- pair **<actor_id, film_id>** is unique.

Entity Relationship Model

- A model based on three basic concepts:
  - Entities that become table names
  - Attributes that become table fields (columns)
  - Relationships

How to Create an Entity Relationship diagram (ER)

- Steps to create an ER diagram
  1. Entity identification
  2. Relationship Identification
  3. Cardinality identification
  4. Attribute identification
  5. Create the ER diagram

Let’s explore a use case

- In a university, a Student enrols in Courses. A student must be assigned to at least one or more Courses. Each course is taught by a single Professor. A Professor can deliver more than one courses.

Step 1: Entity identification

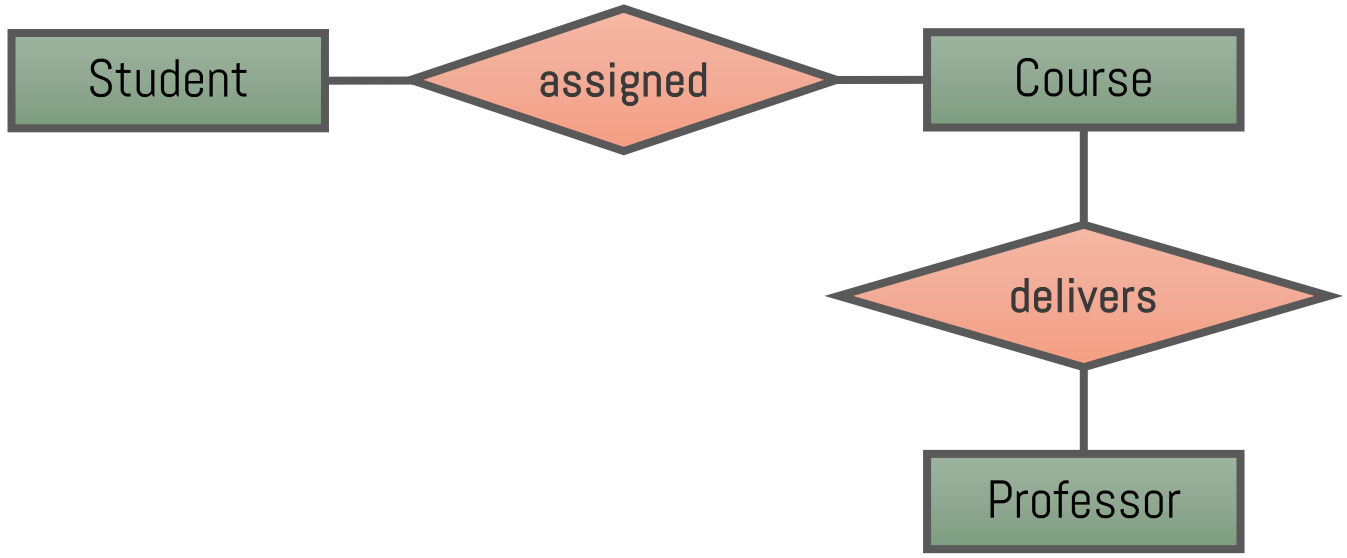
- In a university, a Student enrols in Courses. A student must be assigned to at least one or more Courses. Each course is taught by a single Professor. A Professor can deliver more than one courses.

- We have three entities (tables)
  - Student
  - Course
  - Professor

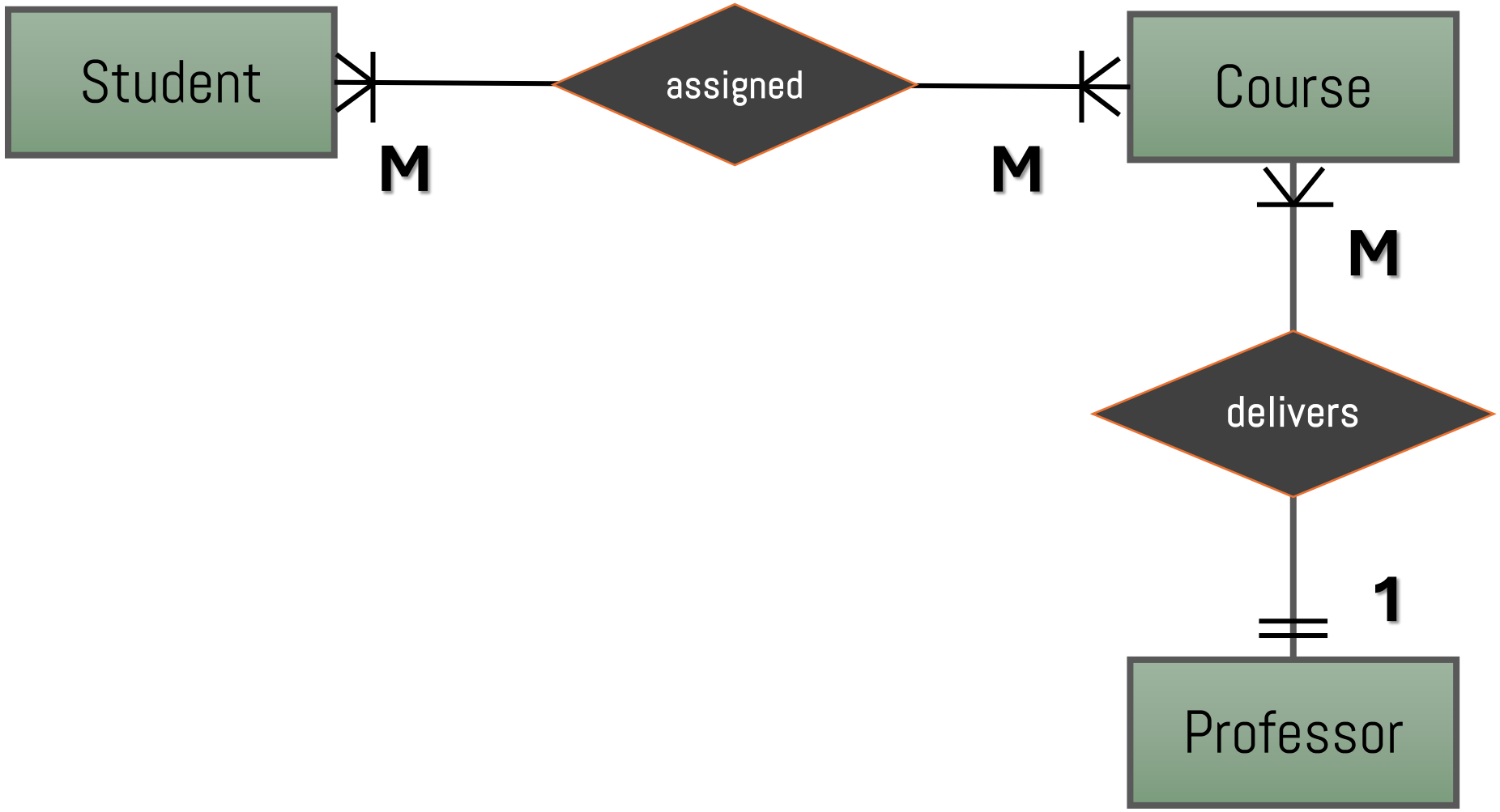
Step 2: Relationship Identification

- In a university, a Student enrols in Courses. A student must be assigned to at least one or more Courses. Each course is taught by a single Professor. A Professor can deliver more than one courses.

- We have the following two relationships
  - The Student is assigned to a course
  - A Professor delivers a course



Step 3: Cardinality Identification



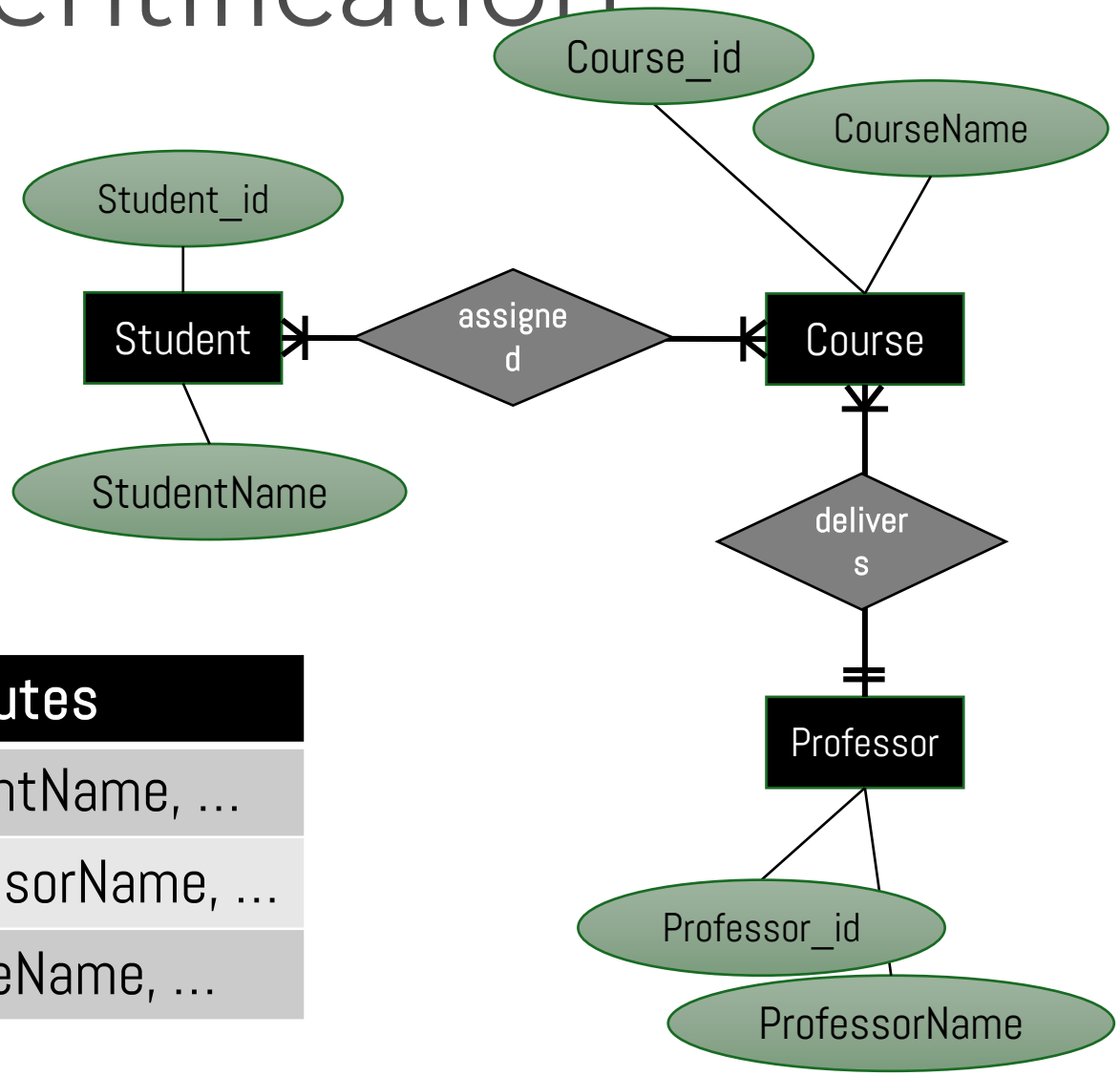
Step 4: Attribute Identification

- You need to explore files, forms, reports and data to identify attributes.
- Once, you have a list of Attributes, you need to map them to the identified entities.
- Ensure an attribute is to be paired with exactly one entity.
- Once the mapping is done, identify the primary Keys.
- If a unique key is not readily available, create one.

Entity	Primary Key	Attributes
Student	Student_ID	StudentName, …
Professor	Employee_ID	ProfessorName, …
Course	Course_ID	CourseName, …

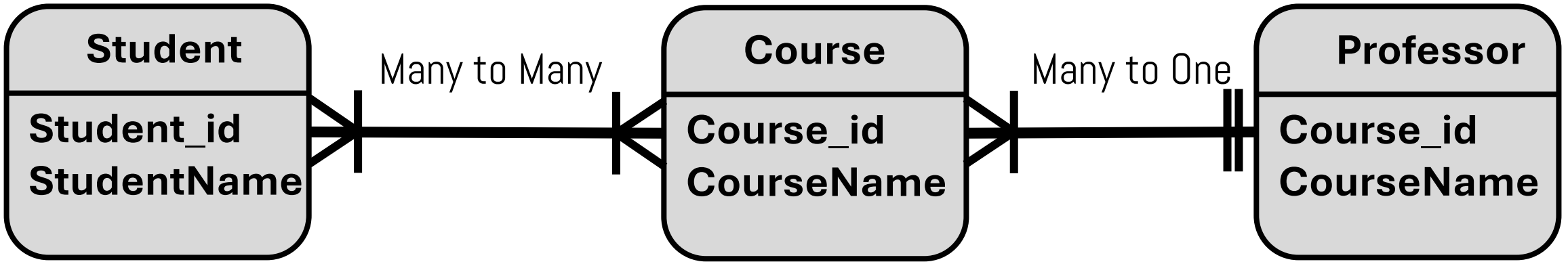
Step 4: Attribute Identification

- Once the mapping is done, identify the primary Keys.
- If a unique key is not readily available, create one.



Entity	Primary Key	Attributes
Student	Student_id	StudentName, …
Professor	Professor_id	ProfessorName, …
Course	Course_id	CourseName, …

Step 5. Create the ERD



Use case example

Design an ER diagram for the following scenario

- In a university, a student enrols in courses. A student must be assigned to at least one or more courses. Each course is taught by a single professor. Each course has one or more tutors, and tutors can be assigned to one or more courses. Each module is taught in the same University classroom, and a classroom can host more than one modules.

- **Can you design an ER diagram?**

Design an ER diagram for the following scenario

- In a university, a student enrols in courses. A student must be assigned to at least one or more courses. Each course is taught by a single professor. Each course has one or more tutors, and tutors can be assigned to one or more courses. Each module is taught in the same University classroom, and a classroom can host more than one modules.

- Steps to create an ER diagram
    1. Entity identification
    2. Relationship Identification
    3. Cardinality identification
    4. Attribute identification
    5. Create the ER diagram

Let’s solve it!

- In a university, a student enrols in courses. A student must be assigned to at least one or more courses. Each course is taught by a single professor. Each course has one or more tutors, and tutors can be assigned to one or more courses. Each module course is taught in the same University classroom, and a classroom can host more than one module course.

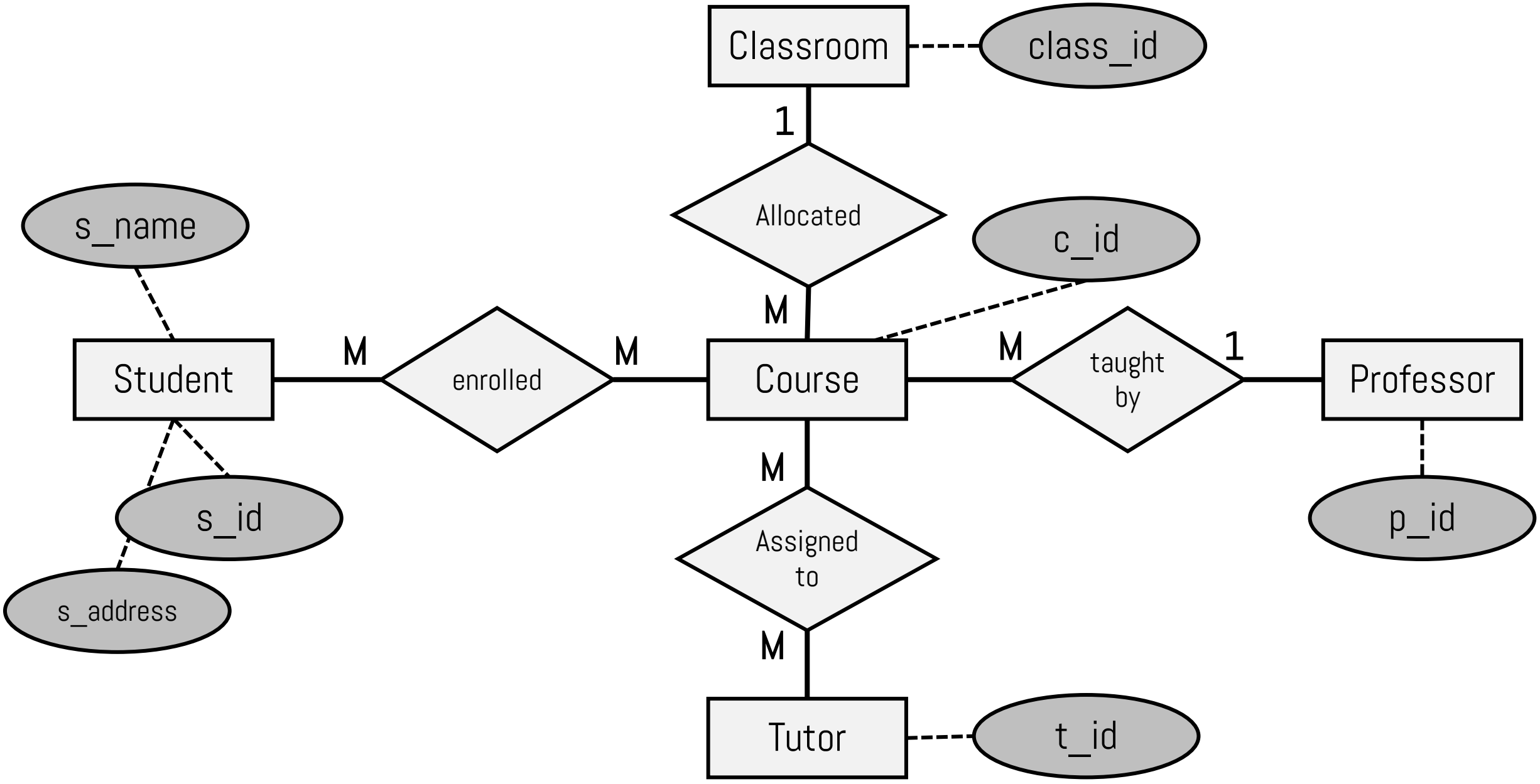
Student

Course

Professor

Tutor

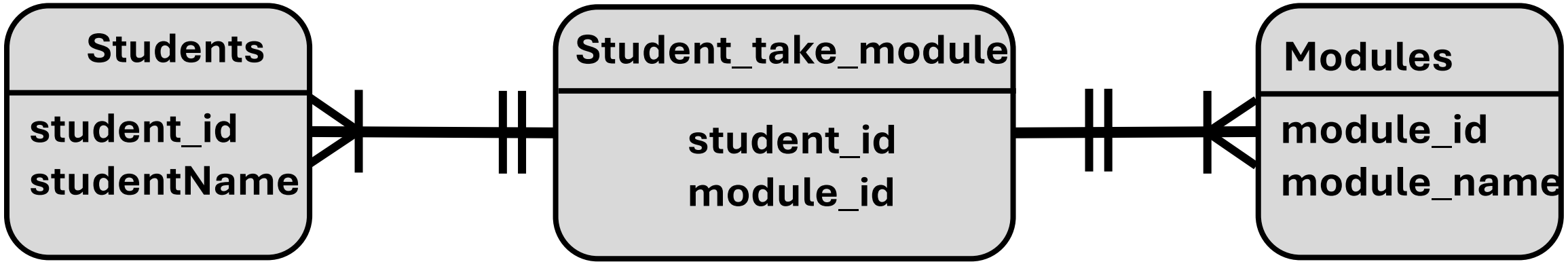
Classroom


Classroom
class_id
1
Allocated
s_name
c_id
M
Student
M
enrolled
M
Course
M
taught
by
1
Professor
s_id
M
Assigned
to
p_id
s_address
M
Tutor
t_id

Developing the SQL queries

A use case:

- In a University a **Student** can take many **Modules** and a **Module** can be taken by many **Students**.



Let's examine the tables

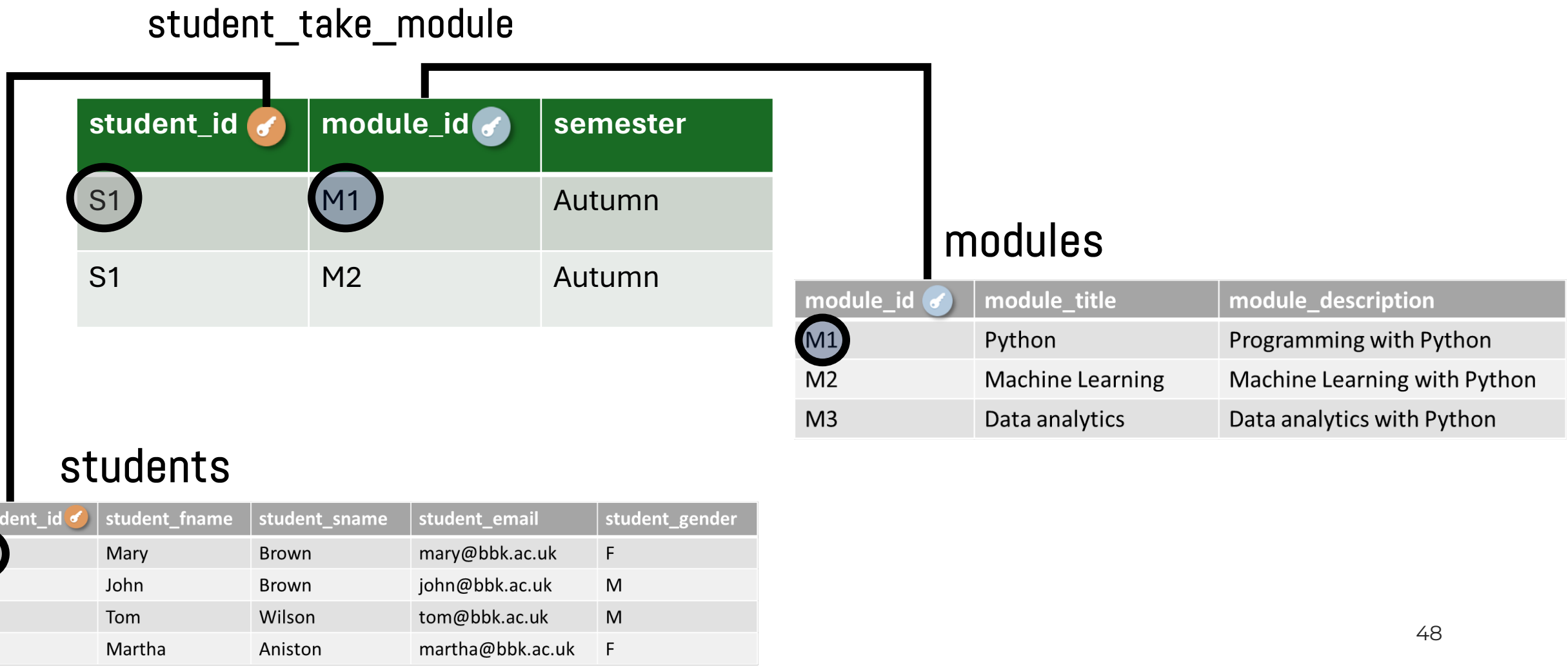
students

student_id 🔑	student_fname	student_sname	student_email	student_gender
S1	Mary	Brown	mary@bbk.ac.uk	F
S2	John	Brown	john@bbk.ac.uk	M
S3	Tom	Wilson	tom@bbk.ac.uk	M
S4	Martha	Aniston	martha@bbk.ac.uk	F

modules

module_id 🔑	module_title	module_description
M1	Python	Programming with Python
M2	Machine Learning	Machine Learning with Python
M3	Data analytics	Data analytics with Python

Create a new table in the middle



Create a new Table in the middle

```
CREATE TABLE student_take_module(

student_id VARCHAR (20),

module_id VARCHAR (20),

semester VARCHAR (20),
```

```
PRIMARY KEY(student_id,module_id),
```

Create a composite key

```
CONSTRAINT FK1 FOREIGN KEY (student_id)

REFERENCES students(student_id),
```

Create a CONSTRAINT (called FK1)
of a FOREIGN KEY
(A student that does not exist in the student table cannot take a module)

```
CONSTRAINT FK2 FOREIGN KEY (module_id)

REFERENCES modules(module_id)

);
```

Create a CONSTRAINT (called FK2)
of a FOREIGN KEY
(A module that does not exist in the module table cannot be taken by a student)

Some common errors…

student_id	module_id	semester
S1	M1	Autumn
S1	M2	Autumn

- Let's say we accidentally run the following:
  - A student (S1) retakes a module (M1)
  - `# ERROR 1062 (23000): Duplicate entry 'S1-M1' for key 'PRIMARY'`

```
INSERT
INTOstudent_take_module
VALUES
('S1','M1','Autumn');
```

- Insert data:
  - Student (S2) takes a module (M22) that does not exist in the modules table
  - `# ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint fails`

```
INSERT INTO
student_take_module
VALUES
('S4','M22','Autumn');
```

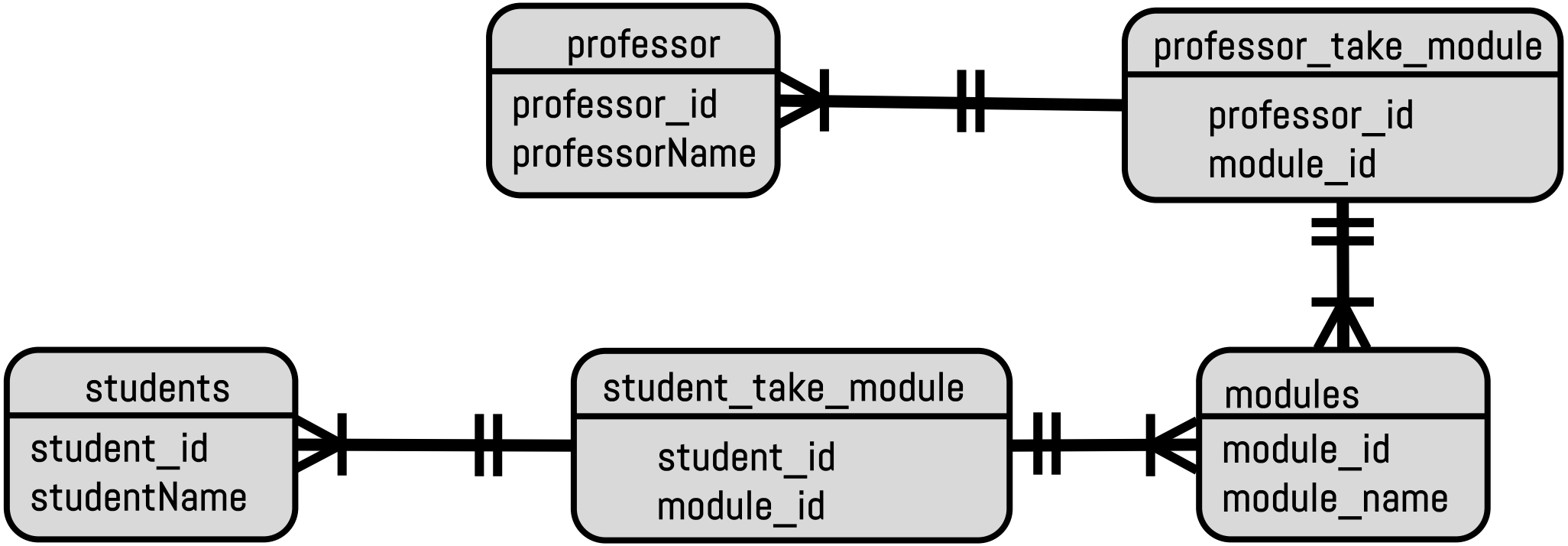
Let’s create another table

professors

professor_id	professor_fname	professor_sname	professor_email	professor_gender
P1	Alex	Rory	alex@bbk.ac.uk	F
P2	Olivia	Harrison	olivia@bbk.ac.uk	F
P3	George	Max	george@bbk.ac.uk	M
P4	Lucas	Logon	lucas@bbk.ac.uk	F

Our database by now...

- To make it more complex, let's adjust the use case:
  - **Let's assume that a Professor can Deliver multiple modules and that a module can be taught by many Professors**



Create table with CONSTRAINTS

```
CREATE TABLE professor_take_module(
professor_id VARCHAR (20),
module_id VARCHAR (20),
semester VARCHAR (20),
PRIMARY KEY(professor_id,module_id),

CONSTRAINT FK3 FOREIGN KEY (professor_id)
REFERENCES professors(professor_id),

CONSTRAINT FK4 FOREIGN KEY (module_id)
REFERENCES modules(module_id)
);
```

Create a composite key

Create a CONSTRAINT (called FK3)
of a FOREIGN KEY
(A professor that does not exist in the professor table cannot take a module)

Create a CONSTRAINT (called FK4)
of a FOREIGN KEY
(A module that does not exist in the module table cannot be taken by a professor)

Let’s insert data

```
INSERT INTO

professor_take_module

VALUES

('P2','M2','Autumn'),

('P2','M3','Autumn');
```

Professor with ID P2 takes two modules in the Autumn term, M2 and M3

professor_take_module

professor_id	module_id	semester
P2	M2	Autumn
P2	M3	Autumn

Can you link the tables?

students

student_id	student_fname	student_sname	student_email	student_gender
S1	Mary	Brown	mary@bbk.ac.uk	F
S2	John	Brown	john@bbk.ac.uk	M
S3	Tom	Wilson	tom@bbk.ac.uk	M
S4	Martha	Aniston	martha@bbk.ac.uk	F

student_take_module

student_id	module_id	semester
S1	M1	Autumn
S1	M2	Autumn

modules

module_id	module_title	module_description
M1	Python	Programming with Python
M2	Machine Learning	Machine Learning with Python
M3	Data analytics	Data analytics with Python

professors

professor_id	professor_fname	professor_sname	professor_email	professor_gender
P1	Alex	Rory	alex@bbk.ac.uk	F
P2	Olivia	Harrison	olivia@bbk.ac.uk	F
P3	George	Max	george@bbk.ac.uk	M
P4	Lucas	Logon	lucas@bbk.ac.uk	F

professor_take_module

professor_id	module_id	semester
P2	M2	Autumn
P2	M3	Autumn

An example of a query from all tables

- Extract which **student** is taught by which **professor** in which **module**!

```
SELECT
students.student_fname,students.student_sname,
modules.module_title,
professors.professor_fname,professor_sname

FROM
students,modules,student_take_module,
professors,professor_take_module

WHERE
students.student_id = student_take_module.student_id
AND
modules.module_id = student_take_module.module_id
AND
professors.professor_id =
professor_take_module.professor_id
AND
modules.module_id = professor_take_module.module_id;
```

You need to include all the tables here!
(tables used in the query)

Link all the tables using the appropriate keys!

Race lab

Big Data Analytics

Visit the following link

- https://www.programiz.com/sql/online-compiler/

Display the first and last names of customers from the US.

Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT first_name, last_name FROM Customers WHERE country='USA';
```

first_name	last_name
John	Doe
Robert	Luna

Create table **Countries**

- Write an SQL query to create a new table called Countries with the following four columns: country_id, country_name, continent, and population.

```
CREATE TABLE Countries (
    country_id INT PRIMARY KEY,
    country_name VARCHAR(100),
    continent VARCHAR(50),
    population INT
);
```

Create table

- Write an SQL query to create a new table called **Cities** that includes a foreign key referencing the country_id from the Countries table.
  - Columns: city_id, city_name, population, and country_id.

```
CREATE TABLE Cities (
    city_id INT PRIMARY KEY,
    city_name VARCHAR(100),
    population INT,
    country_id INT,
    FOREIGN KEY (country_id) REFERENCES Countries(country_id)
);
```

Delete both tables

INSERT – UPDATE - DELETE

- Write an SQL query to insert a new record into the `Customers` table with the following details: customer ID of 6, first name "Alice", last name "Smith", age 30, and country "Canada".

```
INSERT INTO Customers VALUES (6, 'Alice', 'Smith', 30, 'Canada');
```

- Update customer with id 6 age to 32

```
UPDATE Customers
SET age = 32
WHERE customer_id = 6;
```

- Delete Alice

```
DELETE FROM Customers WHERE customer_id = 3;
```

Display all information for customers aged between 27 and 31, inclusive (31) but not 27.

Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT * FROM Customers WHERE age > 27 AND age <= 31;
```

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
5	Betty	Doe	28	UAE

```
SELECT * FROM Customers WHERE age BETWEEN 26 and 31;
```

Show all customer data for those from USA or UK

Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT * FROM Customers WHERE country="USA" or country="UK";
```

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK

Show customers whose last name starts with the letters Do (like Doe)

Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

A* → Starts with "A"
*s → Finish with "s"
the → Includes "the"

```
SELECT * FROM Customers WHERE last_name LIKE 'Do%';
```

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
5	Betty	Doe	28	UAE

What is the average age of your customers?

Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT AVG(age) AS "Average" FROM Customers;
```

Average
25.6

Show customers who are over 22 but not 31 (age)

Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT * FROM customers WHERE age > 22 and age <> 31;
```

customer_id	first_name	last_name	age	country
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

How many customers are per country?

Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT country, count(country)FROM Customers GROUP BY(country);
```

country	count(country)
UAE	1
UK	2
USA	2

Write an SQL query to retrieve the first names and last names of customers along with the items they have ordered.

Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

```
SELECT Customers.first_name, Customers.last_name, Orders.item
FROM Customers, Orders
WHERE Customers.customer_id=Orders.customer_id;
```

first_name	last_name	item
John	Reinhardt	Keyboard
John	Reinhardt	Mouse
David	Robinson	Monitor
John	Doe	Keyboard
Robert	Luna	Mousepad

Write an SQL query to retrieve all columns from the Orders table and sort the results by the amount column in ascending order.

Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

```
SELECT * FROM Orders ORDER BY amount ASC;
```

order_id	item	amount	customer_id
5	Mousepad	250	2
2	Mouse	300	4
1	Keyboard	400	4
4	Keyboard	400	1
3	Monitor	12000	3

```
SELECT * FROM Orders ORDER BY amount ASC LIMIT 3;
```

Write an SQL query to retrieve customers' first names and last names, along with the amounts and items from their orders. Include only those customers who are older than 25 years.

Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

```
SELECT customers.first_name, customers.last_name, orders.amount, orders.item
FROM customers,orders
WHERE customers.customer_id = orders.customer_id
AND customers.age > 25;
```

first_name	last_name	amount	item
John	Doe	400	Keyboard

Retrieve the items ordered by a customer named John Reinhardt. Show first name, last name and item.

Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

```
SELECT O.item
FROM Orders AS O INNER JOIN Customers AS C
ON C.customer_id=O.order_id
WHERE C.first_name='John' AND C.last_name='Reinhardt';
```

first_name	last_name	item
John	Reinhardt	Keyboard

Who is your youngest customer?

Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

```
SELECT Min(Age), last_name, first_name FROM Customers
```

Min(Age)	last_name	first_name
22	Luna	Robert

```
SELECT c.last_name, c.first_name
FROM customers AS c
ORDER BY age ASC LIMIT 1
```

Write an SQL query to retrieve the first names and last names of customers along with the items they have ordered.

Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

```
SELECT Customers.first_name, Customers.last_name, Orders.item
FROM Customers, Orders
WHERE Customers.customer_id=Orders.customer_id;
```

first_name	last_name	item
John	Reinhardt	Keyboard
John	Reinhardt	Mouse
David	Robinson	Monitor
John	Doe	Keyboard
Robert	Luna	Mousepad

Write an SQL query to retrieve the last names of customers along with the items they have ordered and their shipping status.

```
SELECT O.item, C.last_name, S.status
FROM Orders AS O, Customers AS C, Shippings AS S
WHERE C.Customer_id = S.Customer AND
C.customer_id=O.customer_id;
```

item	last_name	status
Keyboard	Reinhardt	Pending
Mouse	Reinhardt	Pending
Monitor	Robinson	Delivered
Keyboard	Doe	Delivered
Mousepad	Luna	Pending

Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

Shippings

shipping_id	status	customer
1	Pending	2
2	Pending	4
3	Delivered	3
4	Pending	5
5	Delivered	1

Write an SQL query to calculate the total sum of the amounts for delivered orders.

```
SELECT sum(O.amount) AS "Sum of amount - Delivered
orders”

FROM Orders AS O, Customers AS C, Shippings AS S
WHERE C.Customer_id = S.Customer AND
C.customer_id=O.customer_id
AND S.status = "Delivered"
```

Sum of amount - Delivered orders
12400

Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

Shippings

shipping_id	status	customer
1	Pending	2
2	Pending	4
3	Delivered	3
4	Pending	5
5	Delivered	1

Write an SQL query to retrieve the customer IDs, items ordered, and shipping status for orders where the amount exceeds 500.

```
SELECT C.customer_id,item,status
FROM Orders AS O, Customers AS C, Shippings AS S
WHERE C.Customer_id = S.Customer AND
C.customer_id=O.customer_id
AND O.amount > 500;
```

customer_id	item	status
3	Monitor	Delivered

Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

Shippings

shipping_id	status	customer
1	Pending	2
2	Pending	4
3	Delivered	3
4	Pending	5
5	Delivered	1

Who is the customer that ordered the most expensive product?

Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

```
SELECT * FROM Customers AS C, Orders AS O
WHERE C.customer_id = O.customer_id
AND O.amount =
  (SELECT Max(amount) FROM Orders);
```

customer_id	first_name	last_name	age	country	order_id	item	amount
3	David	Robinson	22	UK	3	Monitor	12000

Who is/are the youngest customer(s)?

Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

```
SELECT * FROM Orders
WHERE customer_id = (
  SELECT customer_id FROM Customers
  WHERE age = (SELECT MIN(age) FROM Customers)
);
```

order_id	item	amount	customer_id
5	Mousepad	250	2

What is the average of the orders?
For this query, you cannot use the AVG function.

Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

```
SELECT SUM(amount)/COUNT(amount)
FROM Orders;
```

Average of orders
2670

Write an SQL query to retrieve the countries and the count of customers from each country, but only include countries with at least 2 customers. Use the Customers table and group the results by country, applying the HAVING clause to filter the groups with a customer count of 2 or more.

```
SELECT country, COUNT(customer_id)
FROM Customers
GROUP BY country
HAVING COUNT(customer_id)>=2;
```

country	COUNT(customer_id)
UK	2
USA	2

Retrieve the customers with their associated order items – including those who did not order.

Customers

customer_id	first_name	last_name	age	country
1	John	Doe	31	USA
2	Robert	Luna	22	USA
3	David	Robinson	22	UK
4	John	Reinhardt	25	UK
5	Betty	Doe	28	UAE

Orders

order_id	item	amount	customer_id
1	Keyboard	400	4
2	Mouse	300	4
3	Monitor	12000	3
4	Keyboard	400	1
5	Mousepad	250	2

```
SELECT Customers.first_name, Customers.last_name, Orders.item
FROM Customers
LEFT JOIN Orders ON Customers.customer_id = Orders.customer_id
ORDER BY Customers.country;
```

first_name	last_name	item
Betty	Doe	
David	Robinson	Monitor
John	Reinhardt	Keyboard
John	Reinhardt	Mouse
John	Doe	Keyboard
Robert	Luna	Mousepad

Classify the amounts to there categories that are less, equal or greater than 400.

```
SELECT order_id, amount,
CASE
      WHEN amount > 400 THEN 'The amount is greater than 300’
      WHEN amount = 400 THEN 'The quantity is 400’
      ELSE 'The quantity is under 400'
END
AS "Amount class"
FROM Orders;
```

order_id	amount	Amount class
1	400	The quantity is 400
2	300	The quantity is under 400
3	12000	The amount is greater than 300
4	400	The quantity is 400
5	250	The quantity is under 400

Lab 6

Big Data Analytics

Lab activities

- Complete lab 6 activities and exercises.
- Use the GCP to run exercises in MySQL.