Table Relations

Database Design and Rules

SoftUni Team

Technical Trainers

Software University

https://softuni.bg

Questions

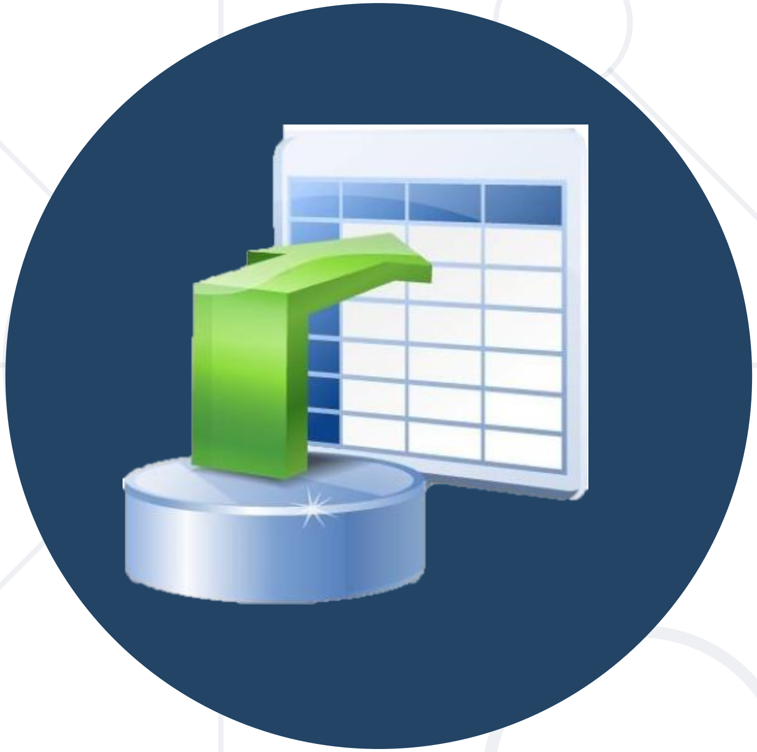
Software University

sli.do

#python-db

Table of Contents

1. Database Design
2. Table **Relations**
3. **JOINs**
4. **Cascade Operations**
5. **E/R** Diagrams



Database Design

Fundamental Concepts

Steps in Database Design

1 Identification of the entities

2 Defining table columns

3 Defining primary keys

4 Modeling relationships

5 Defining constraints

6 Filling test data

Identification of Entities

- Entity tables represent objects from the real world
  - Most often they are nouns in the specification
  - For example:

```
We need to develop a system that stores
information about students, which are trained
in various courses. The courses are held in
different towns. When registering a new
student, the following information is entered:
name, faculty number, photo, and date.
```

  - Entities: **Student**, **Course**, **Town**

Identification of the Columns

- Columns are the attributes of entities, defined in the specification's text, for example:

```
We need to develop a system that stores
information about students, which are trained
in various courses. The courses are held in
different towns. When registering a new
student, the following information is entered:
name, faculty number, photo, and date.
```

- Students have the following characteristics (attributes):
  - Name, faculty number, photo, date of enrolling, and a list of courses they visit

How to Choose a Primary Key?

- Always define an additional column for the primary key
  - Don't use an existing column (for example "name")
  - Can be an integer number
  - Must be declared as a `PRIMARY KEY`
  - Put the primary key in a first column
- Exceptions
  - Entities that have well-known ID, e.g., countries (BG, DE, US) and currencies (USD, EUR, BGN)

Identification of Relationships

- Relationships are dependencies between entities:

> We need to develop a system that stores information about students, which are trained in various courses. The courses are held in different towns. When registering a new student, the following information is entered: name, faculty number, photo, and date.

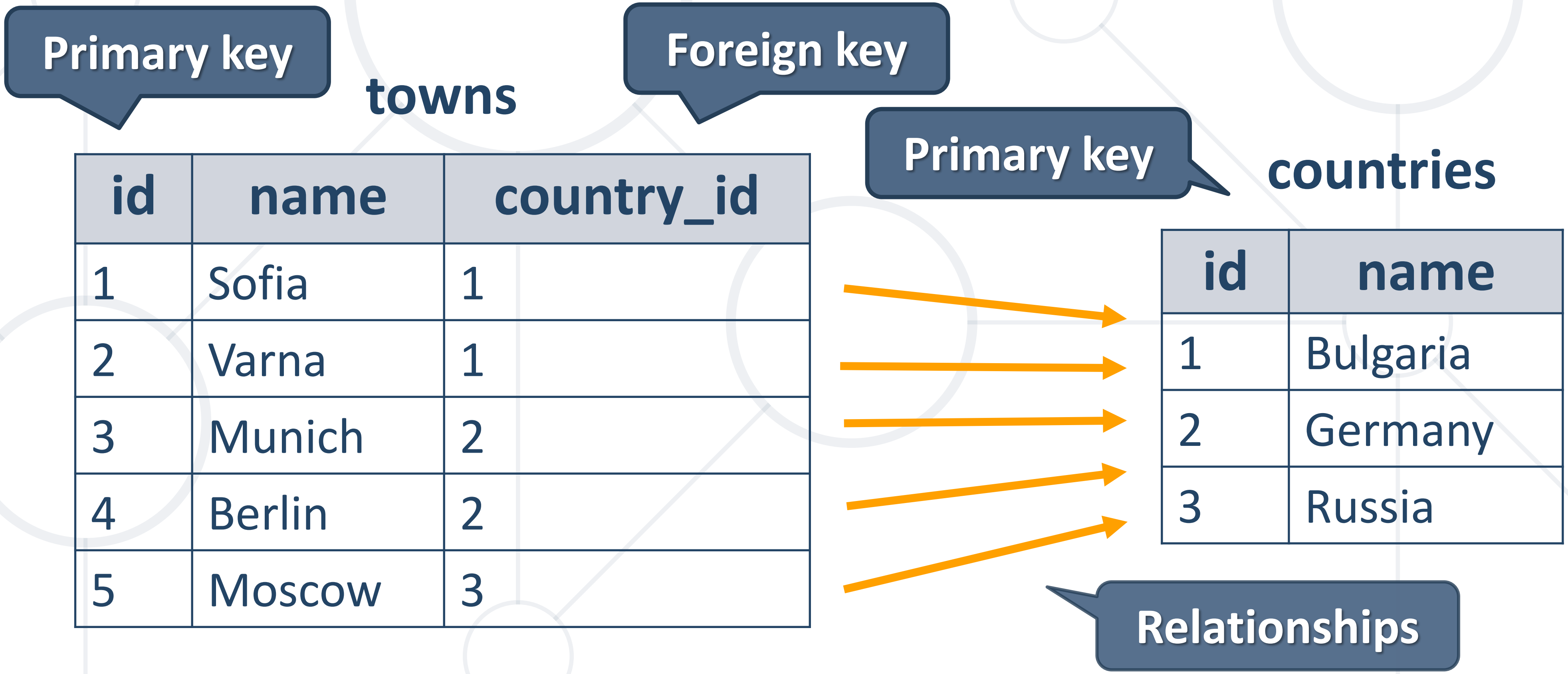
  - "Students are trained in courses" – **many-to-many** relationship
  - "Courses are held in towns" – **many-to-one** (or many-to-many) relationship

Table Relations

Relational Database Model in Action

Relationships

- **Relationships** between tables are based on interconnections: `PRIMARY KEY` / `FOREIGN KEY`



Primary key

Foreign key

towns

id	name	country_id
1	Sofia	1
2	Varna	1
3	Munich	2
4	Berlin	2
5	Moscow	3

Primary key

countries

id	name
1	Bulgaria
2	Germany
3	Russia

Relationships

Primary Key Implementation

- Primary Key

```
id INT PRIMARY KEY
```

- Auto-increment

```
id SERIAL PRIMARY KEY
```

- Auto-increment always (cannot provide an explicit value)

```
id INT GENERATED ALWAYS AS IDENTITY
```

- Auto-increment (cannot guarantee uniqueness)

```
id INT GENERATED BY DEFAULT AS IDENTITY
```

Relationships

- The **foreign key** is an **identifier** of a record located in another table (usually its primary key)
- By using relationships, we avoid repeating data in the database
- Relationships have multiplicity:
  - **One-to-many** – e.g., mountain / peaks
  - **Many-to-many** – e.g., students / courses
  - **One-to-one** – e.g., country / capital

Foreign Key Implementation

- Define a foreign key

```
CREATE TABLE clients (
    id SERIAL PRIMARY KEY,
    name VARCHAR(30)
);
CREATE TABLE orders (
    id SERIAL PRIMARY KEY,
    client_id INT REFERENCES clients
);
```

Reference Table

Foreign Key

Foreign Key Implementation

- Define a foreign key using the foreign key constraint

```
CREATE TABLE orders (
    id INT PRIMARY KEY,
    client_id INT,
    CONSTRAINT fk_client
        FOREIGN KEY(client_id)
            REFERENCES clients(id)
);
```

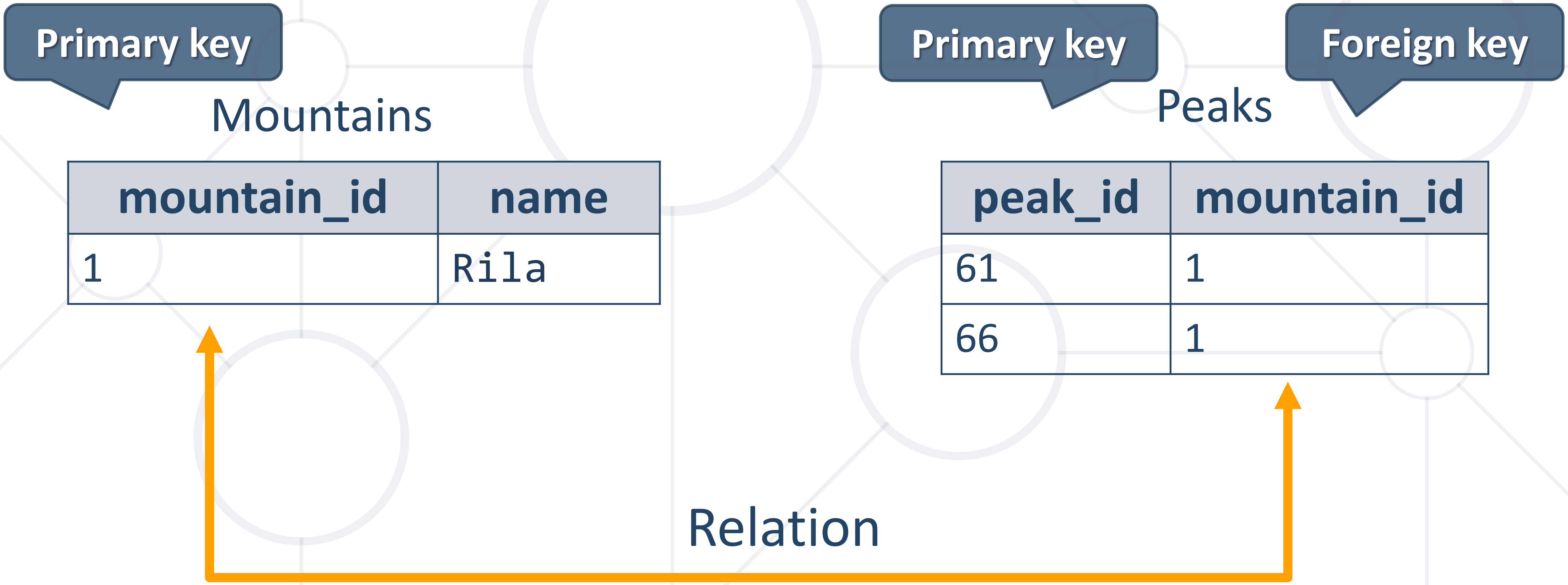
Constraint Name

Foreign Key

Reference Table

Primary Key

One-to-Many/Many-to-One



Primary key

Mountains

mountain_id	name
1	Rila

Primary key

Foreign key

Peaks

peak_id	mountain_id
61	1
66	1

Relation

One-to-Many: Setup

```
CREATE TABLE mountains(
  id SERIAL PRIMARY KEY,
  name VARCHAR(50)
);

CREATE TABLE peaks(
  id SERIAL PRIMARY KEY,
  name VARCHAR(50),
  mountain_id INT REFERENCES mountains
);
```

Primary key

Foreign Key

Problem: Mountains and Peaks

- Create two tables – **`mountains`** and **`peaks`**
- **Link** their fields properly by setting a **`Foreign Key Constraint`** (**`fk_peaks_mountains`**)
  - `Mountains:`
    - `id, name`
  - `Peaks:`
    - `id, name, mountain_id`

Solution: Mountains and Peaks

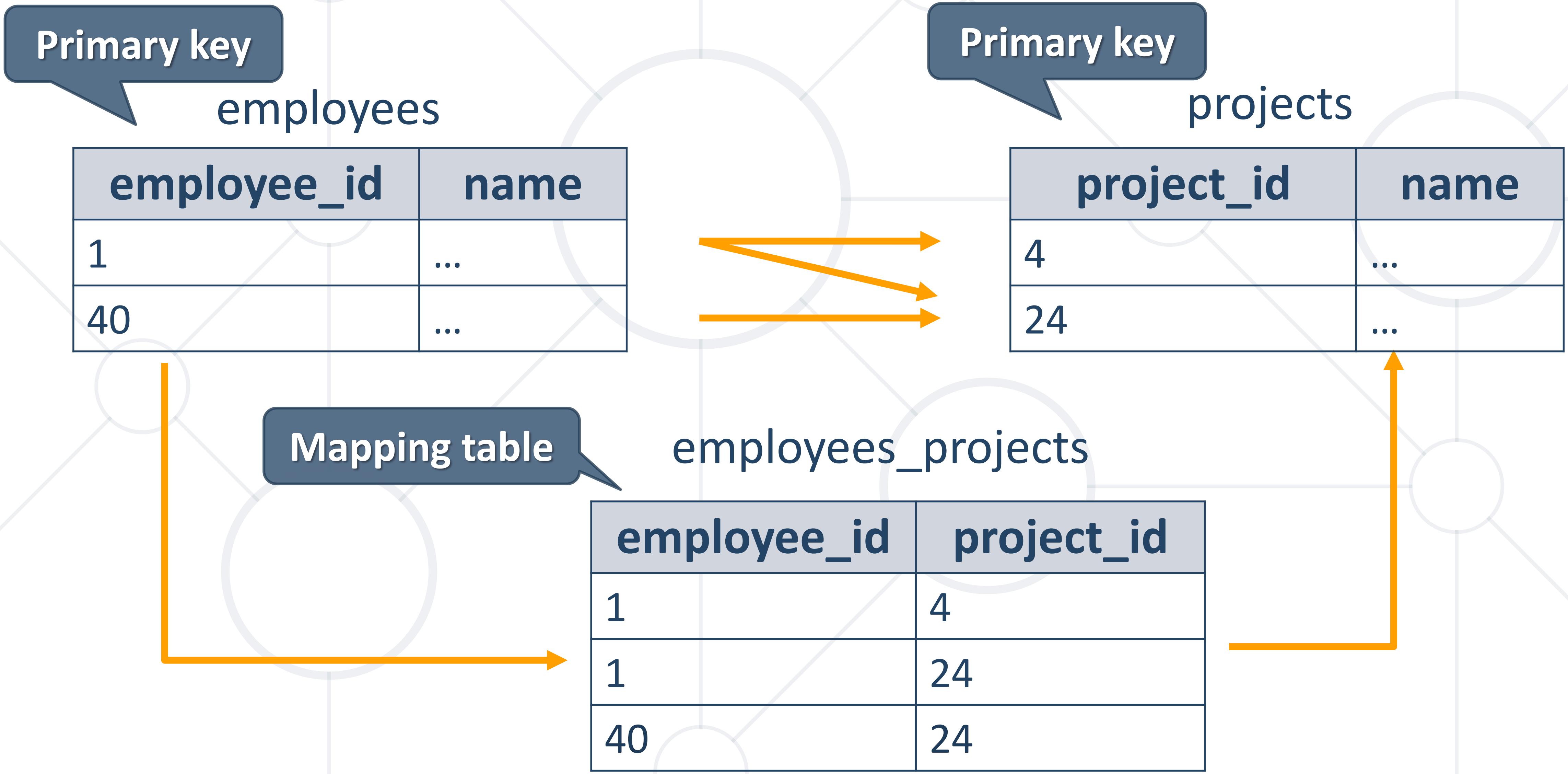
```
CREATE TABLE mountains(
     id INT GENERATED ALWAYS AS IDENTITY UNIQUE,
     name VARCHAR(50) NOT NULL
);
CREATE TABLE peaks(
  id INT GENERATED ALWAYS AS IDENTITY UNIQUE,
  name VARCHAR(50) NOT NULL,
  mountain_id INT,
  CONSTRAINT fk_peaks_mountains
    FOREIGN KEY (mountain_id)
       REFERENCES mountains(id)
);
```

Primary key

Table Peaks

Foreign Key

Many-to-Many



Primary key

employees

employee_id	name
1	…
40	…

Primary key

projects

project_id	name
4	…
24	…

Mapping table

employees_projects

employee_id	project_id
1	4
1	24
40	24

Many-to-Many: Setup

```
CREATE TABLE employees(
  id SERIAL PRIMARY KEY,
  employee_name VARCHAR(50)
);
```

Table Employees

```
CREATE TABLE projects(
  id SERIAL PRIMARY KEY,
  project_name VARCHAR(50)
);
```

Table Projects

Many-to-Many: Setup

```sql
CREATE TABLE employees_projects(
  employee_id INT,
  project_id INT,
  CONSTRAINT pk_employees_projects
  PRIMARY KEY(employee_id, project_id),
  CONSTRAINT fk_employees_projects_employees
  FOREIGN KEY(employee_id)
  REFERENCES employees(id),
  CONSTRAINT fk_employees_projects_projects
  FOREIGN KEY(project_id)
  REFERENCES projects(id)
);
```

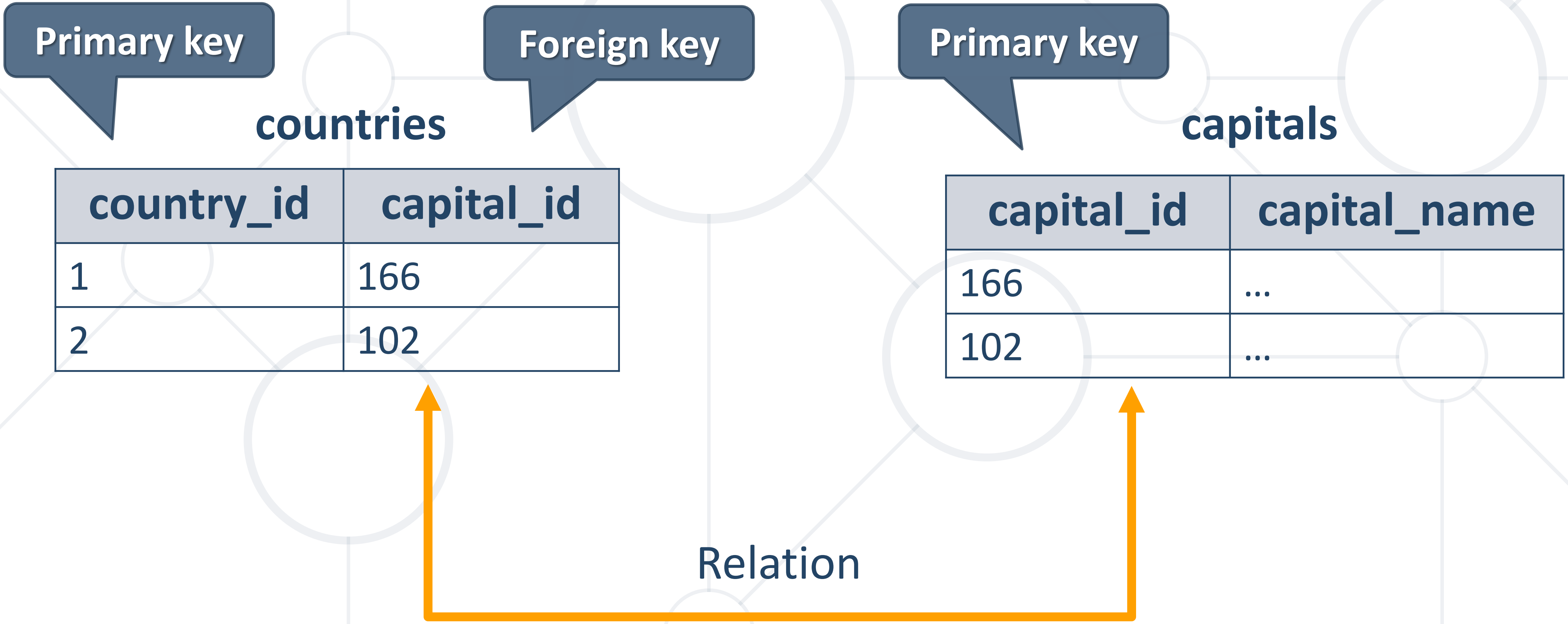
Mapping Table

Primary Key

Foreign Key

Foreign Key

One-to-One



Primary key

Foreign key

countries

country_id	capital_id
1	166
2	102

Primary key

capitals

capital_id	capital_name
166	…
102	…

Relation

One-to-One: Setup

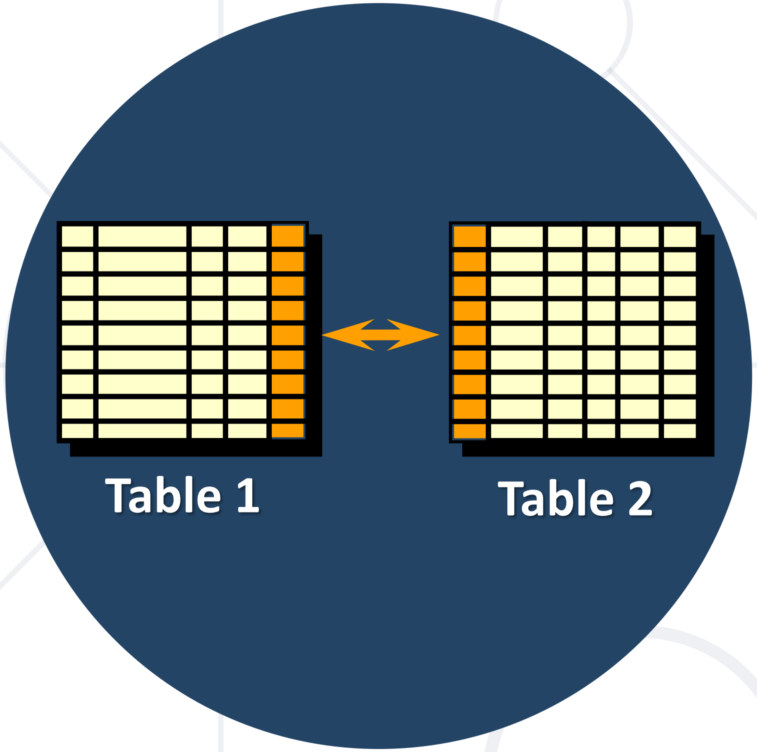
```
CREATE TABLE capitals(
  capital_id SERIAL PRIMARY KEY,
  capital_name VARCHAR(50)
);

CREATE TABLE countries(
  country_id SERIAL PRIMARY KEY,
  capital_id INT UNIQUE,
  CONSTRAINT fk_countries_capitals FOREIGN KEY
  (capital_id) REFERENCES capitals(capital_id)
);
```

Primary Key

One Capital per Country

Foreign Key



JOINs

Using Simple JOIN Statements

Joins

- Table relations are useful when combined with JOINS
- With JOINS we can get data from two tables **simultaneously**
  - By pointing a "**join condition**"
  - Example:

Select from Tables

```
SELECT * FROM table_a
  JOIN table_b ON
    table_b.common_column = table_a.common_column
```

Join Condition

Problem: Trip Organization

- Write a query to retrieve information about **SoftUni camp's transportation organization**
  - Get information about people who are **drivers**(full name and id) and their **vehicle type**
    - **`driver_id`**
    - **`vehicle_type`**
    - **`driver_name`**
  - Use database "**`camp`**"

Solution: Trip Organization

Cross Table Selection

```
SELECT driver_id, vehicle_type,
  CONCAT(first_name, ' ', last_name) AS driver_name
FROM vehicles AS v
JOIN campers AS c
    ON v.driver_id = c.id;
```

Join Condition

Problem: SoftUni Hiking

- Write a query to retrieve information about **hiking routes** and their **leaders**
  - Get information about **leaders** (full name and id) and **hiking routes** (start point, end point)
    - **`start_point`**
    - **`end_point`**
    - **`leader_id`**
    - **`leader_name`**
  - Use database "**`camp`**"

Solution: SoftUni Hiking

Cross Table Selection

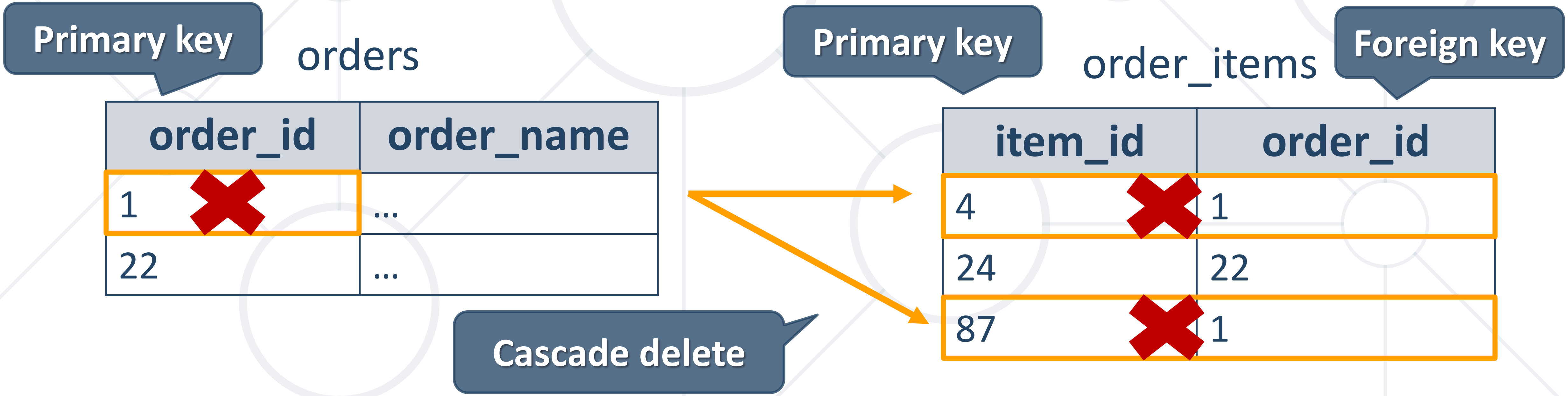
```
SELECT start_point, end_point, leader_id,
  CONCAT(c.first_name, ' ', c.last_name) AS
    leader_name
FROM routes AS r
JOIN campers AS c
    ON r.leader_id = c.id;
```

Join Condition

Cascade Operations

Definition

- **CASCADE** allows changes made to a specific entity to be applied to all related entities



Primary key

orders

order_id	order_name
1	…
22	…

Primary key

order_items

Foreign key

item_id	order_id
4	1
24	22
87	1

Cascade delete

CASCADE DELETE

- **CASCADE** can be either **DELETE** or **UPDATE**
- Use **CASCADE DELETE** when:
  - The related entities are **meaningless** without the "main" one
- Do **not** use **CASCADE DELETE** when:
  - You make "**logical delete**"
  - You preserve **history**
- Keep in mind that in more complicated relations it won't work with **circular** references

Foreign Key with Cascade Delete

```
CREATE TABLE drivers(
  driver_id INT PRIMARY KEY,
  driver_name VARCHAR(50)
);

CREATE TABLE cars(
  car_id INT PRIMARY KEY,
  driver_id INT,
  CONSTRAINT fk_car_driver FOREIGN KEY(driver_id)
  REFERENCES drivers(driver_id) ON DELETE CASCADE
);
```

Table Drivers

Table Cars

Foreign Key

Problem: Delete Mountains

- Write a query to create a **one-to-many** relationship between tables **`mountains`** and **`peaks`**.
  - Set a **`Foreign Key Constraint (fk_mountain_id)`**
  - `Mountains:`
    - **`id, name`**
  - `Peaks:`
    - **`id, name, mountain_id`**
  - When a **mountain** gets **removed** from the database, **all** of its **peaks** are **deleted** too

Solution: Delete Mountains

```
CREATE TABLE mountains(
    id SERIAL PRIMARY KEY,
    name VARCHAR(50) NOT NULL
);
```

Solution: Delete Mountains

```
CREATE TABLE peaks(
   id SERIAL PRIMARY KEY,
   name VARCHAR(50) NOT NULL,
   mountain_id INT,
   CONSTRAINT fk_mountain_id
   FOREIGN KEY(mountain_id)
   REFERENCES mountains(id)
   ON DELETE CASCADE
);
```

CASCADE UPDATE

- Use when:
  - The primary key is **NOT auto-incremented** and therefore it **can** be changed
  - Best used with the **UNIQUE** constraint
- Do **not** use when:
  - The primary key is auto-incremented
- Avoid cascading updates by using triggers or procedures

Foreign Key Update Cascade

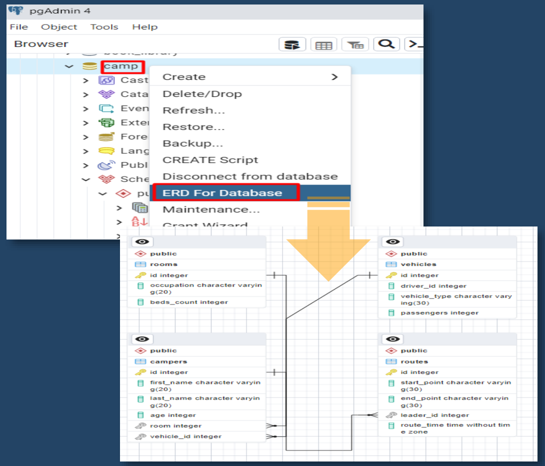
```
CREATE TABLE drivers(
  driver_id INT PRIMARY KEY,
  driver_name VARCHAR(50)
);

CREATE TABLE cars(
  car_id INT PRIMARY KEY,
  driver_id INT,
  CONSTRAINT fk_car_driver FOREIGN KEY(driver_id)
  REFERENCES drivers(driver_id) ON UPDATE CASCADE
);
```

Table Drivers

Table Cars

Foreign Key



E/R Diagrams

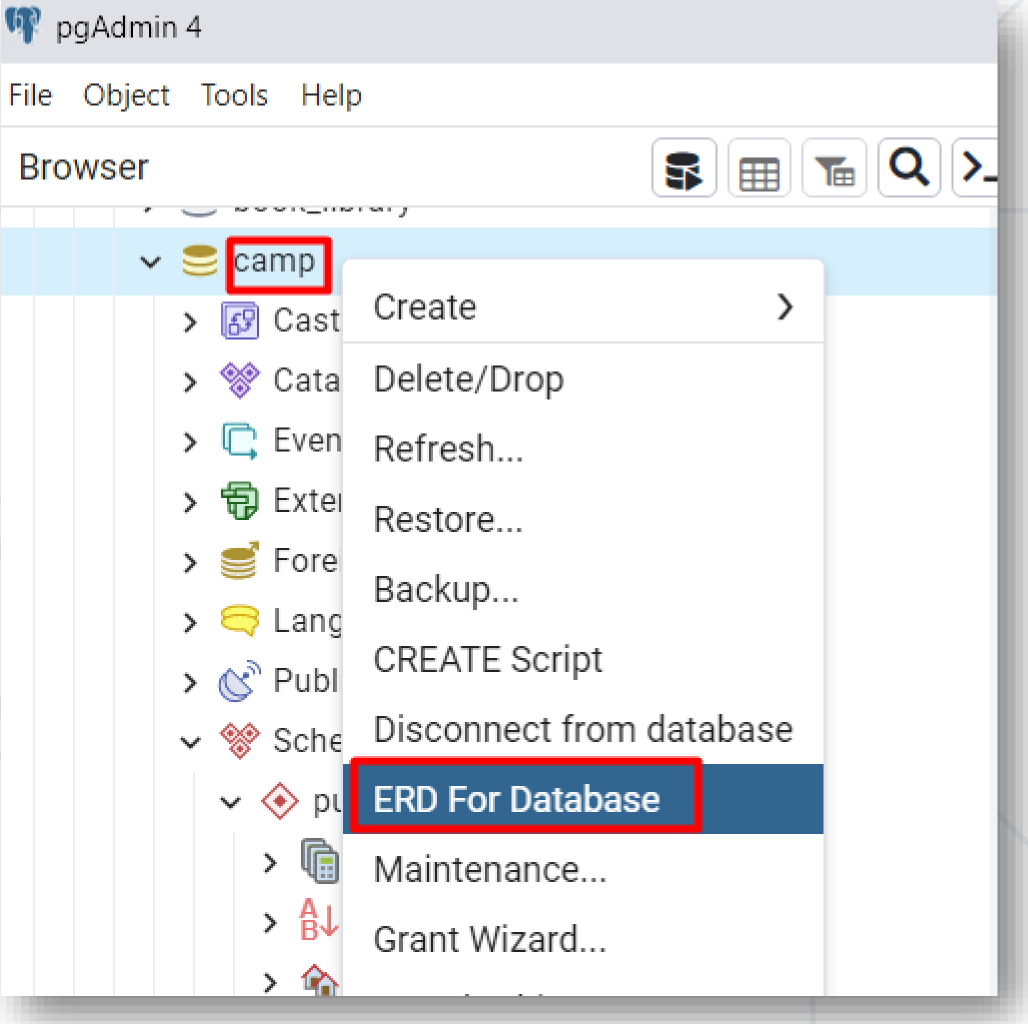
Entity / Relationship Diagrams

Relational Schema

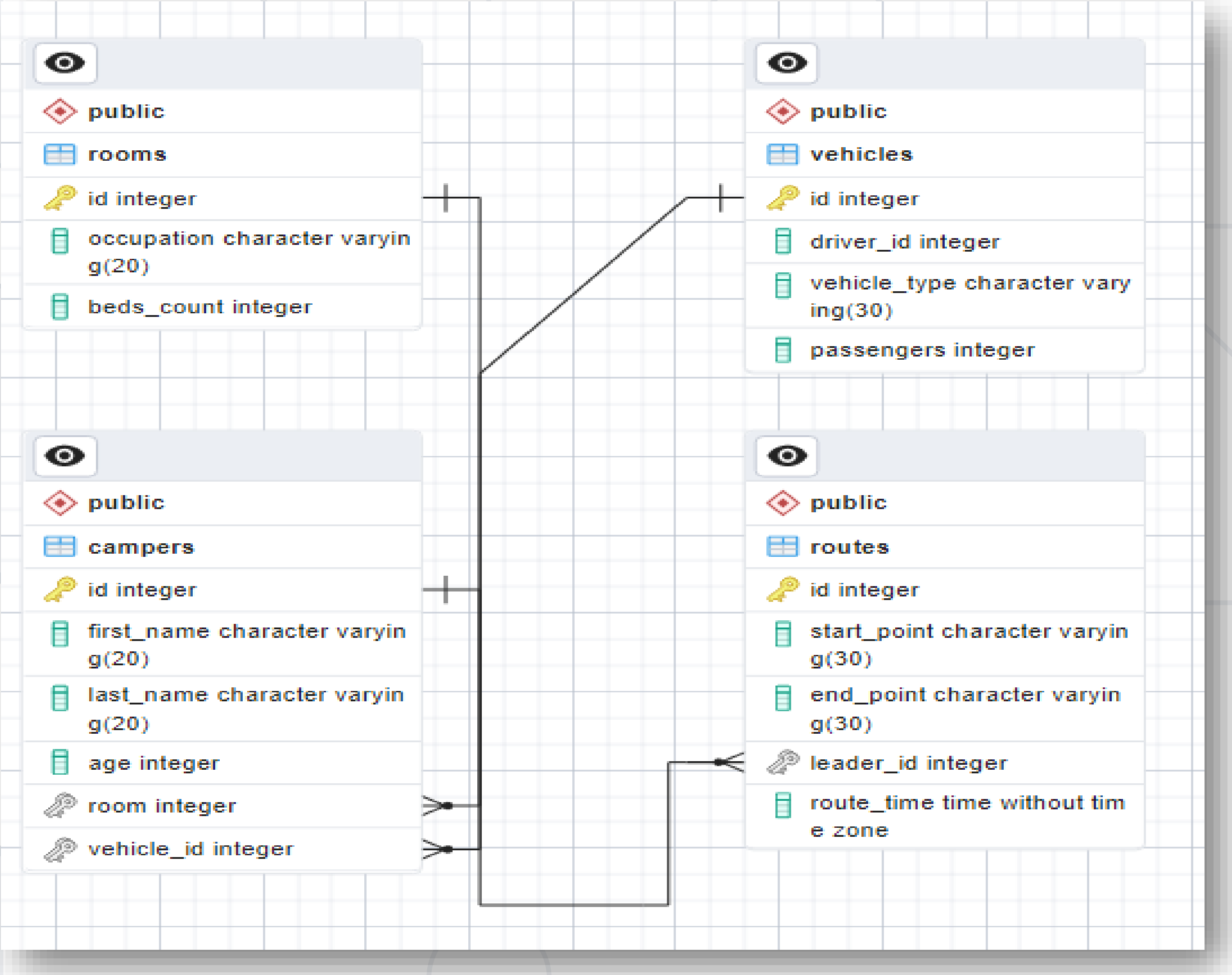
- **Relational schema** of a DB is a collection of:
  - The schemas of all tables
  - Relationships between tables
  - Any other database objects (e.g., constraints)
- The relational schema describes the **structure** of the database
  - Does not contain data, but **metadata**
- Relational schemas are **graphically** displayed in Entity / Relationship diagrams (**E/R Diagrams**)

E/R Diagram

- Right-click on the database name
- Select **`ERD For Database`**



E/R Diagram



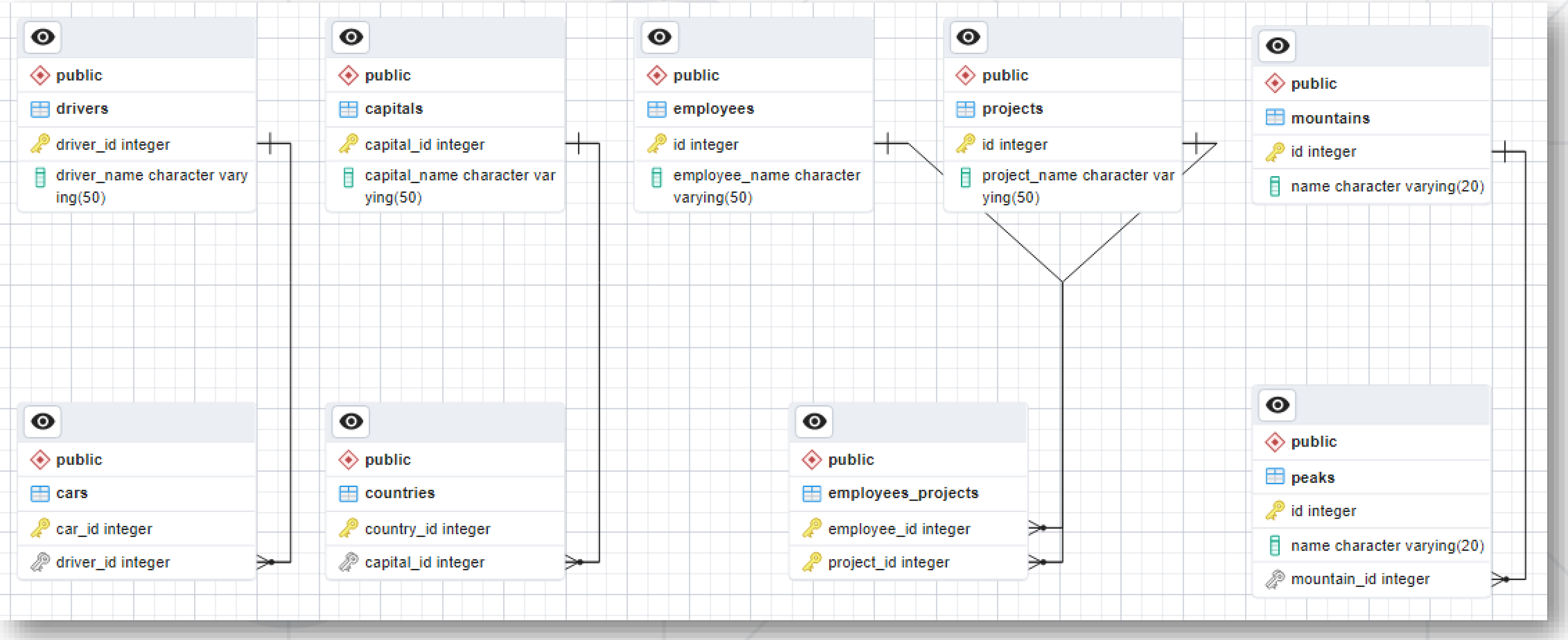
public
rooms
id integer
occupation character varying(20)
beds_count integer

public
vehicles
id integer
driver_id integer
vehicle_type character varying(30)
passengers integer

public
campers
id integer
first_name character varying(20)
last_name character varying(20)
age integer
room integer
vehicle_id integer

public
routes
id integer
start_point character varying(30)
end_point character varying(30)
leader_id integer
route_time time without time zone

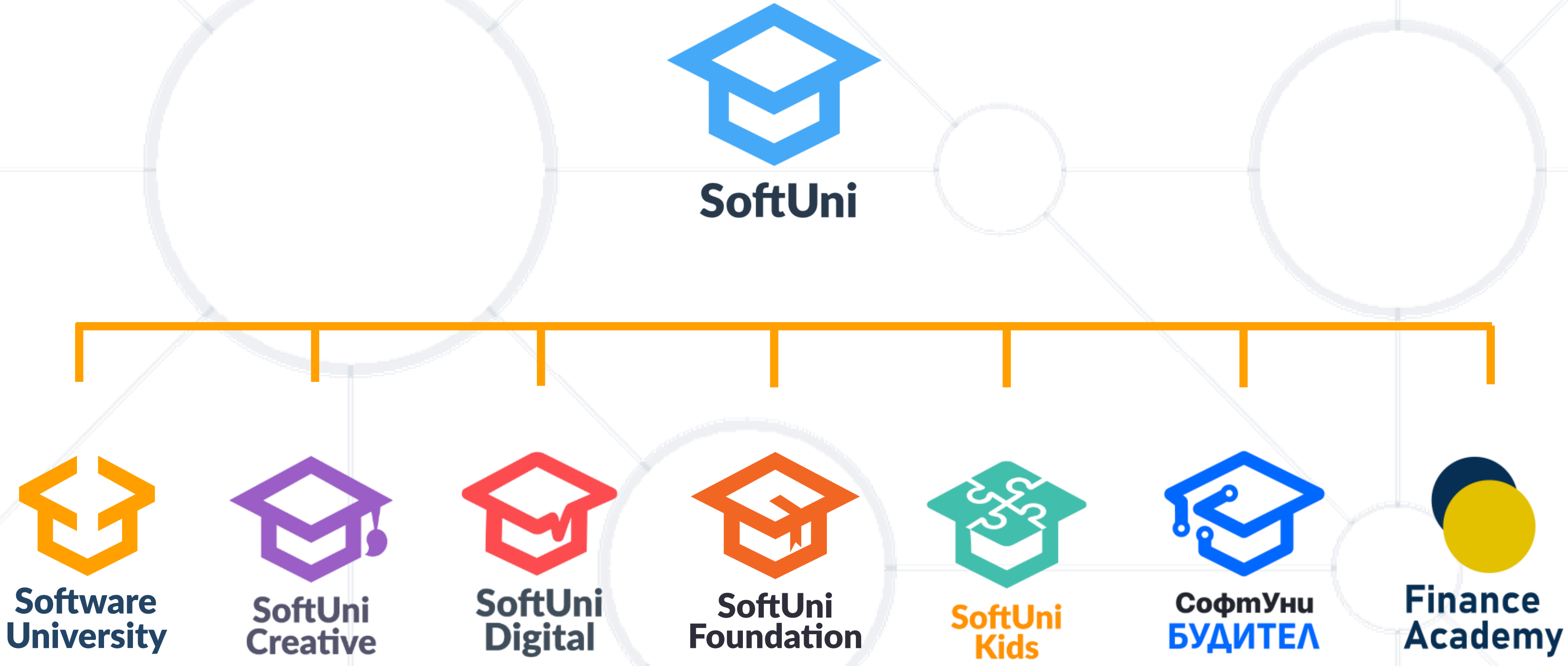
E/R Diagram


public
drivers
driver_id integer
driver_name character varying(50)
public
capitals
capital_id integer
capital_name character varying(50)
public
employees
id integer
employee_name character varying(50)
public
projects
id integer
project_name character varying(50)
public
mountains
id integer
name character varying(20)
public
cars
car_id integer
driver_id integer
public
countries
country_id integer
capital_id integer
public
employees_projects
employee_id integer
project_id integer
public
peaks
id integer
name character varying(20)
mountain_id integer

Summary

- Database design
  - Entities and their characteristics
- Types of relations
  - One-to-one
  - One-to-many
  - Many-to-many
- Relations visualization via E/R diagrams

Questions?



SoftUni

Software University

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

СофтУни БУДИТЕЛ

Finance Academy

SoftUni Diamond Partners

SmartIT

Coca-Cola HBC Bulgaria

SUPER HOSTING .BG

createX

Postbank

Решения за твоето утре

DRAFT KINGS

VIVACOM

AMBITIONED

INDEAVR

Serving the high achievers

SOFTWARE GROUP

PHAR VISION

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, softuni.org
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – https://about.softuni.bg/
- © Software University – https://softuni.bg