Table Relations

Database Design and Rules

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

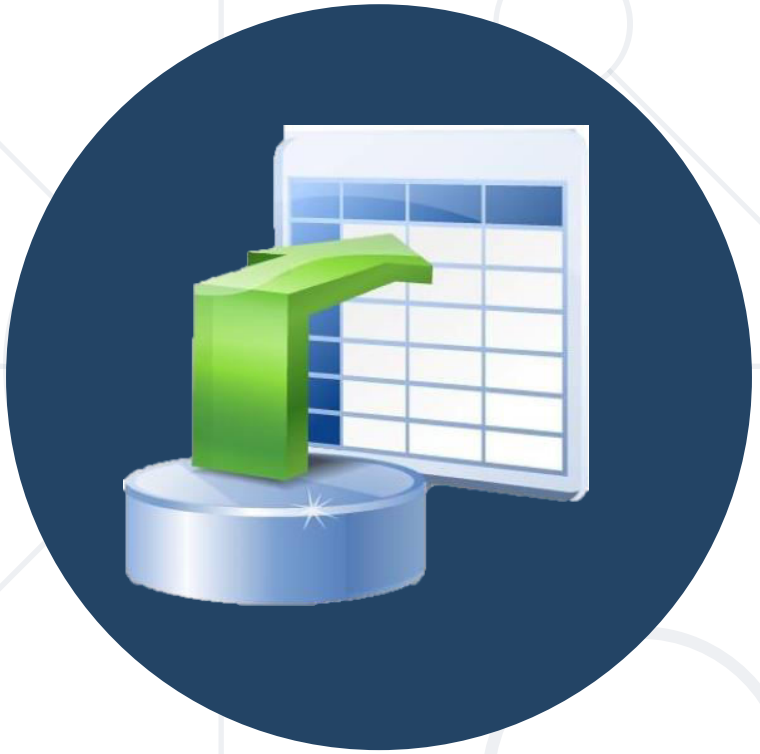
Table of Contents

Questions

Software University

sli.do

#Java-DB



Database Design

Fundamental Concepts

Steps in Database Design

Software University

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 clarifications for the entities in the text of 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.
```

- Students have the following characteristics:
  - Name, faculty number, photo, date of enlistment 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 SSN)
  - Can be an integer number
  - Must be declared as a `PRIMARY KEY`
  - Use `AUTO_INCREMENT` to implement auto-increment
  - Put the primary key as 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 the 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

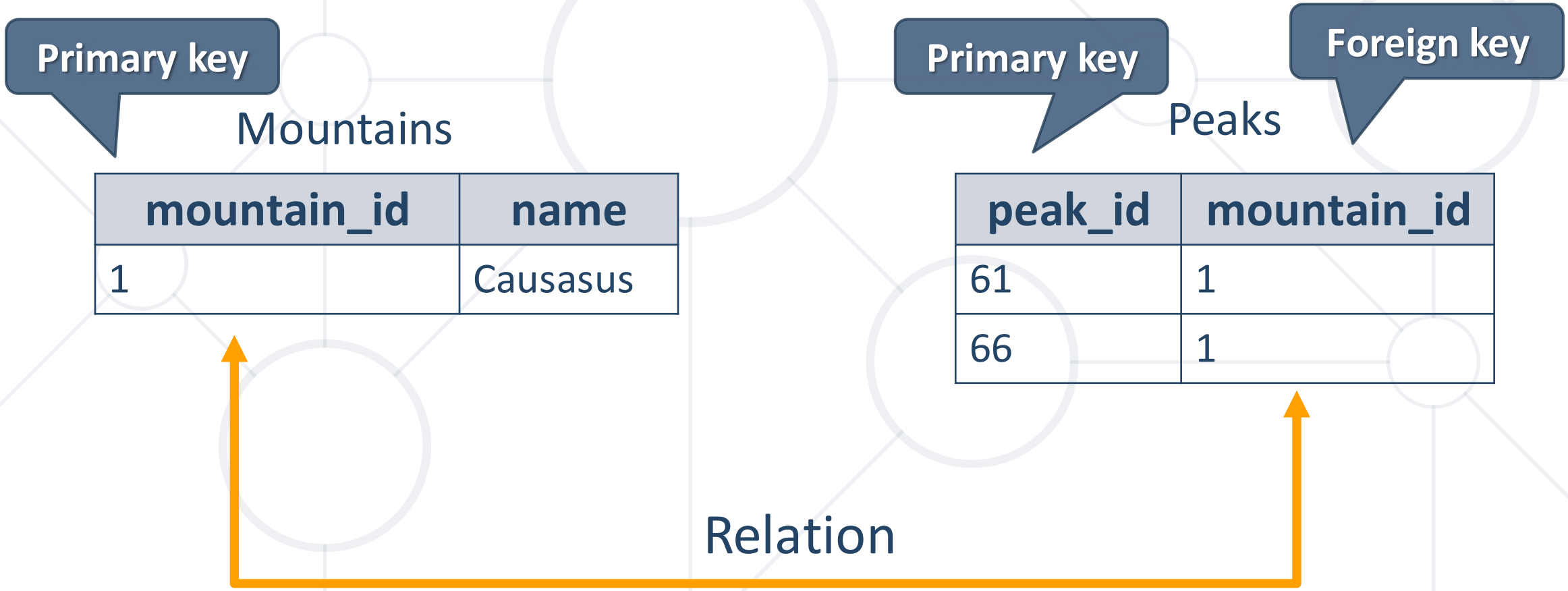
id	name
1	Bulgaria
2	Germany
3	Russia

Relationships

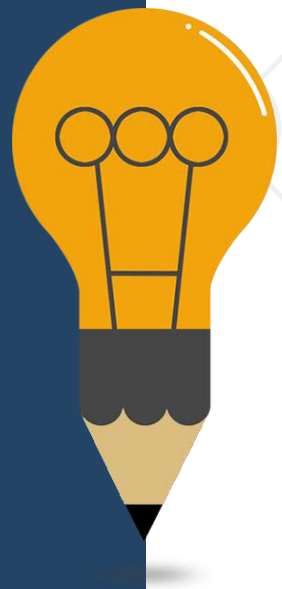
Relationships (2)

- 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. mountains / peaks
  - **Many-to-many** – e.g. student / course
  - **One-to-one** – e.g. example driver / car

One-to-Many/Many-to-One



Foreign Key



```
CONSTRAINT fk_peaks_mountains
FOREIGN KEY (mountain_id)
REFERENCES mountains(mountain_id);
```

Constraint Name

Foreign Key

Referent Table

Primary Key

Problem: Mountains and Peaks

- Create two tables – **mountains** and **peaks**
- **Link their fields** properly
  - Mountains:
    - **Id**
    - **name**
  - Peaks:
    - **id**
    - **name**
    - **mountain_id**

Solution: Mountains and Peaks

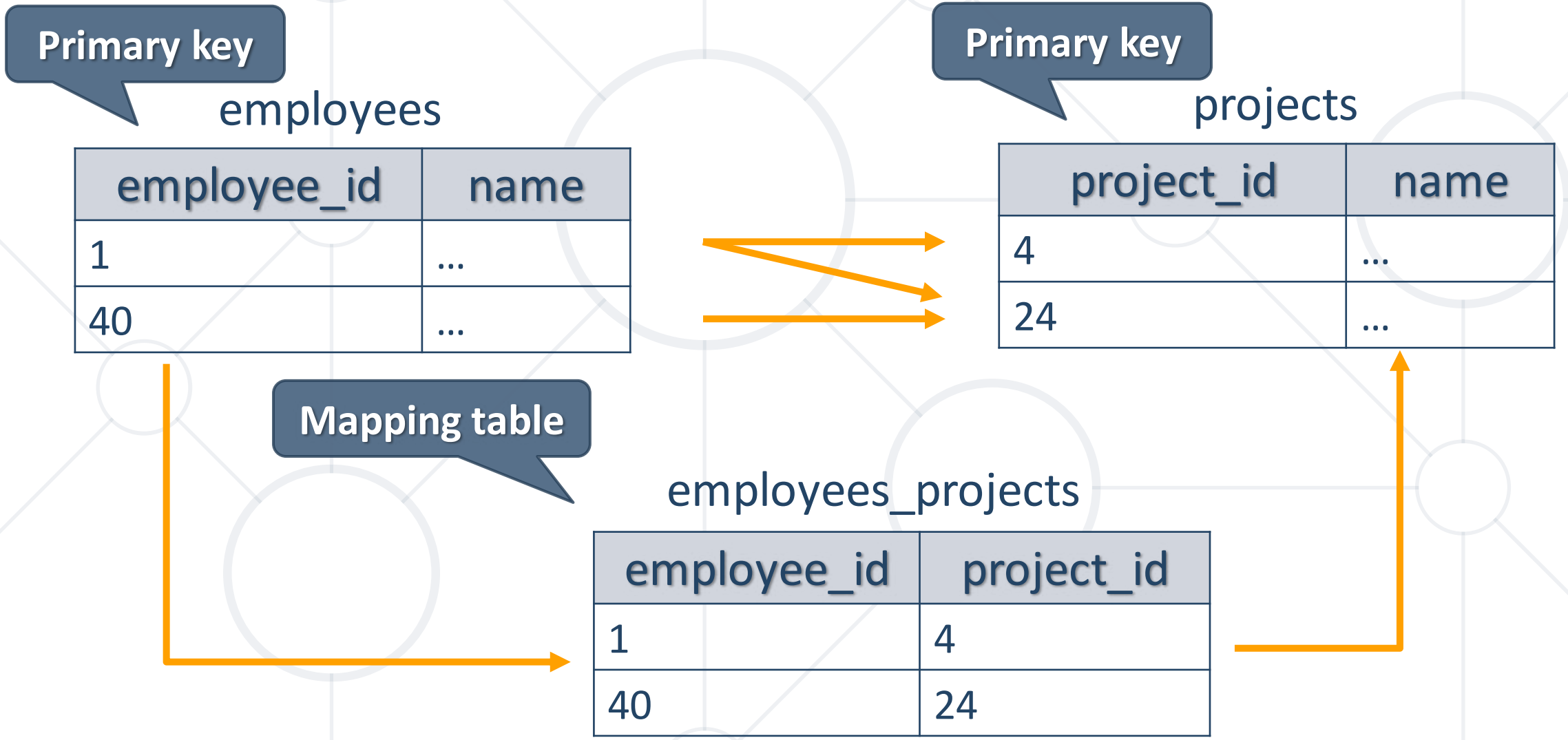
```
CREATE TABLE mountains(
    id INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(50) NOT NULL
);
CREATE TABLE peaks(
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(50) NOT NULL,
  mountain_id INT,
  CONSTRAINT fk_peaks_mountains
  FOREIGN KEY (mountain_id)
  REFERENCES mountains(id)
);
```

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
40
24

Setup(1)

```
CREATE TABLE employees(
  employee_id INT PRIMARY KEY,
  employee_name VARCHAR(50)
);
```

Table Employees

```
CREATE TABLE projects(
  project_id INT PRIMARY KEY,
  project_name VARCHAR(50)
);
```

Table Projects

Setup(2)

```
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(employee_id),
  CONSTRAINT fk_employees_projects_projects
  FOREIGN KEY(project_id)
  REFERENCES projects(project_id)
);
```

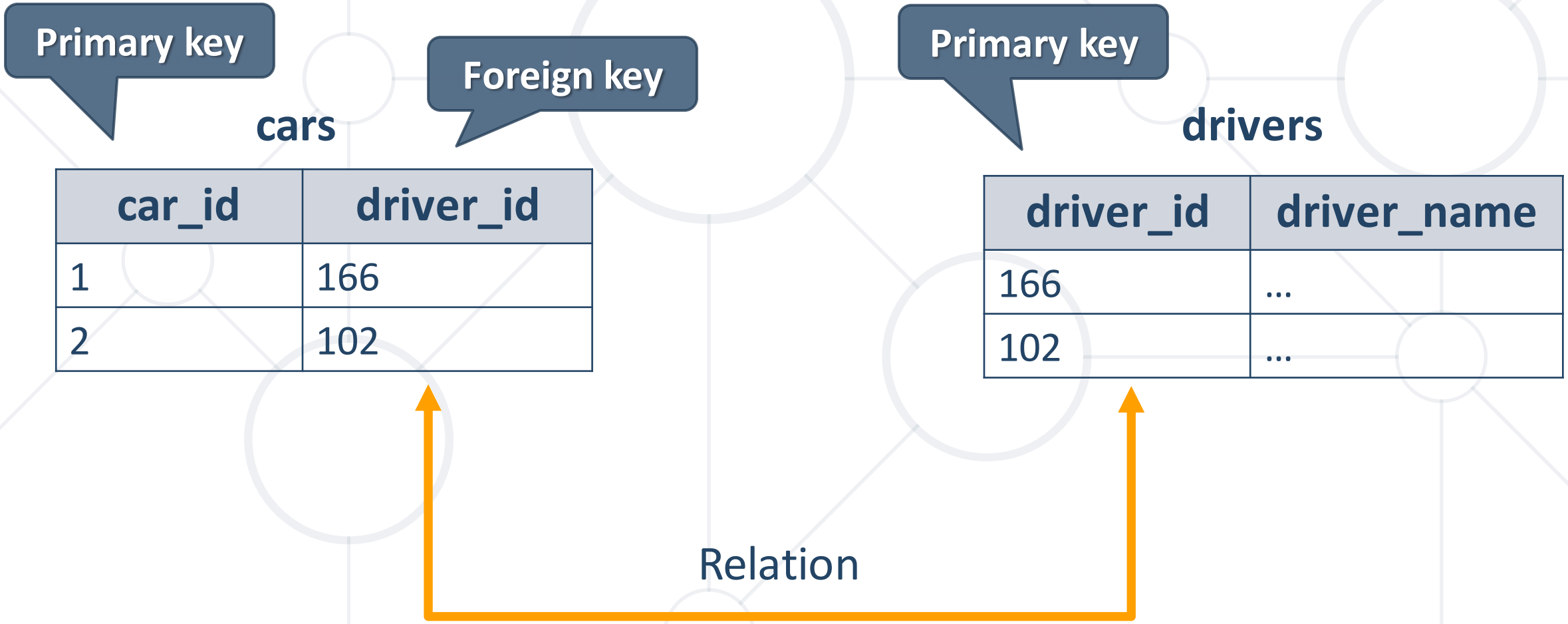
Mapping Table

Primary Key

Foreign Key

Foreign Key

One-to-One


Primary key
Foreign key
cars
car_id
driver_id
1
166
2
102
Primary key
drivers
driver_id
driver_name
166
...
102
...
Relation

Setup

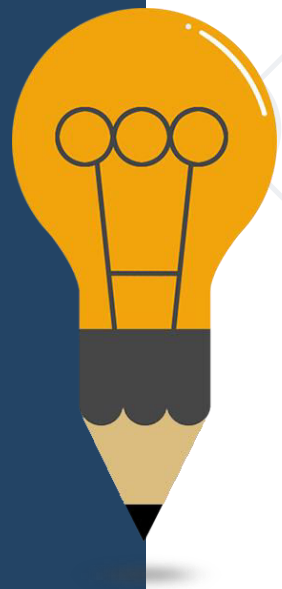
```
CREATE TABLE drivers(
  driver_id INT PRIMARY KEY,
  driver_name VARCHAR(50)
);

CREATE TABLE cars(
  car_id INT PRIMARY KEY,
  driver_id INT UNIQUE,
  CONSTRAINT fk_cars_drivers FOREIGN KEY
  (driver_id) REFERENCES drivers(driver_id)
);
```

Primary key

One driver per car

Foreign Key

Foreign Key



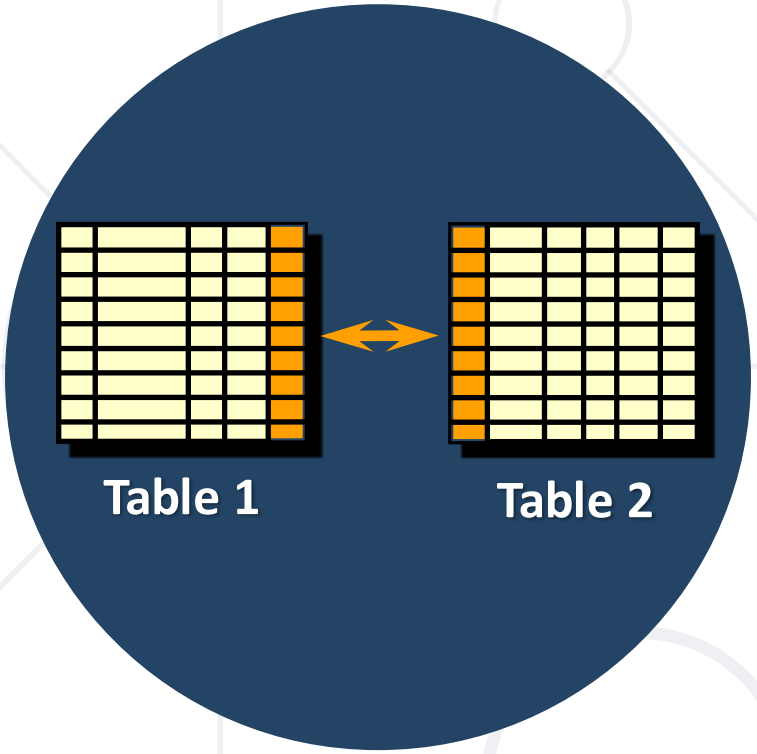
```
CONSTRAINT fk_cars_drivers
FOREIGN KEY (driver_id)
REFERENCES drivers(driver_id)
```

Constraint Name

Foreign Key

Referent Table

Primary Key



Retrieving Related Data

Using Simple JOIN Statements

Joins

- Table relations are useful when combined with JOINS
- With JOINS we can get data from two tables **simultaneously**
  - JOINS require at least two tables and 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

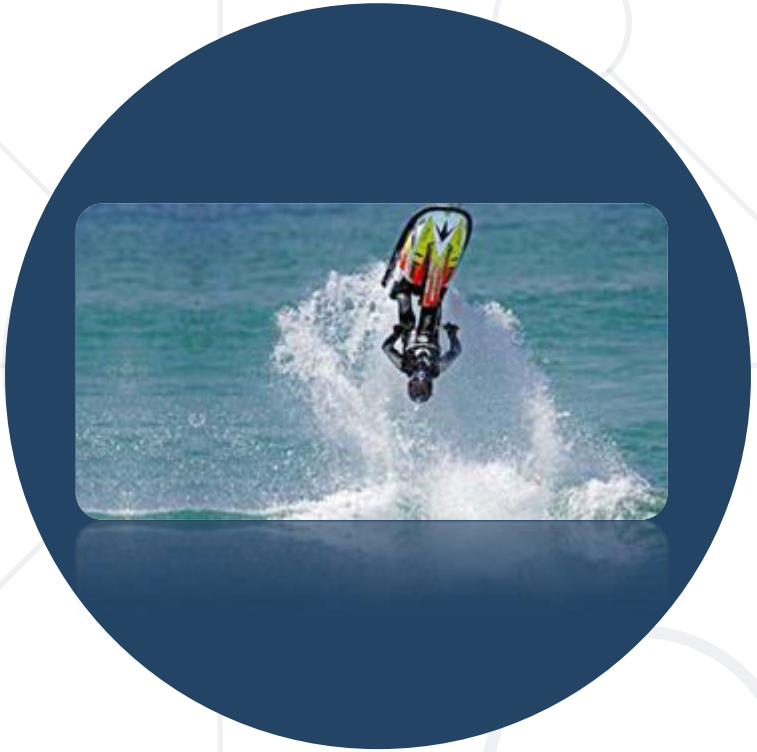
- Write a query to retrieve information about the SoftUni camp's transportation organization.
- Get information about the people who drive(**name** and **age**) and their vehicle **type**
  - Use database "camp".

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;
```

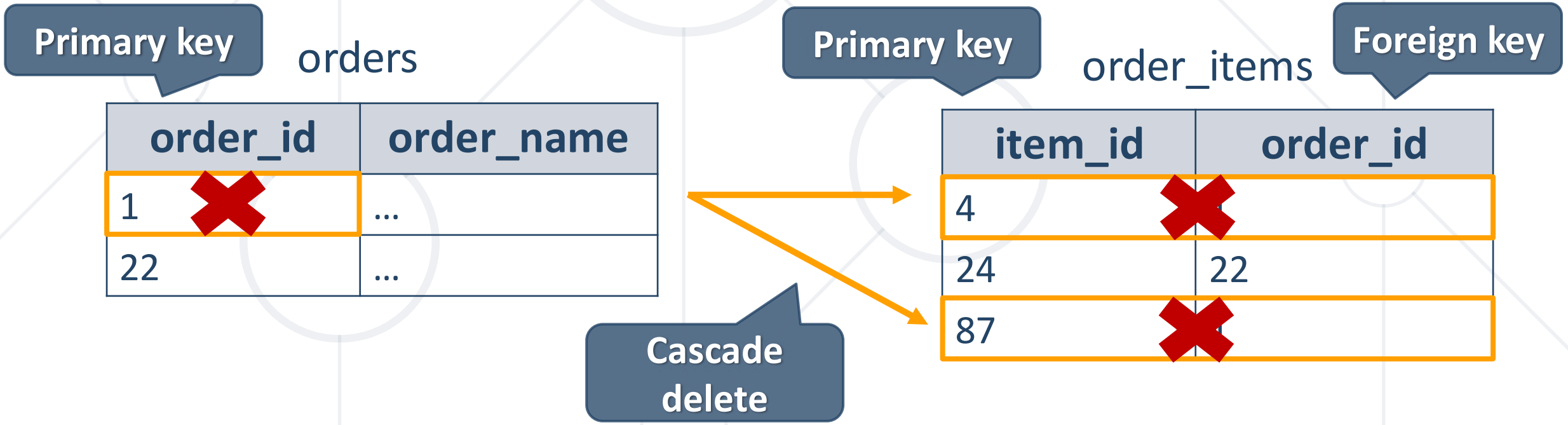
Join Condition



Cascade Operations

Definition

- Cascading allows when a change is made to certain entity, this change to apply to all related entities



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

Problem: Delete Mountains

- Write a query to create a one-to-many relationship
- When an mountains gets removed from the database, all of his peaks are deleted too

```
CREATE TABLE `mountains`(
`id` INT PRIMARY KEY AUTO_INCREMENT,
`name` VARCHAR(20) NOT NULL
);
```

Solution: Delete Mountains (2)

```
CREATE TABLE `peaks`(
`id` INT PRIMARY KEY AUTO_INCREMENT,
`name` VARCHAR(20) NOT NULL,
`mountain_id` INT,
CONSTRAINT `fk_mountain_id`
FOREIGN KEY(`mountain_id`)
REFERENCES `mountains`(`id`)
ON DELETE CASCADE
);
```

CASCADE UPDATE

- Use **CASCADE UPDATE** when:
  - The primary key is **NOT** identity (not **`auto-increment`**) and therefore it **can** be changed
  - Best used with **UNIQUE** constraint
- Do **not** use **CASCADE UPDATE** when:
  - The primary is identity (**`auto-increment`**)
- Cascading can be avoided using triggers or procedures

Foreign Key Delete 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 DELETE CASCADE
);
```

Table Drivers

Table Cars

Foreign Key

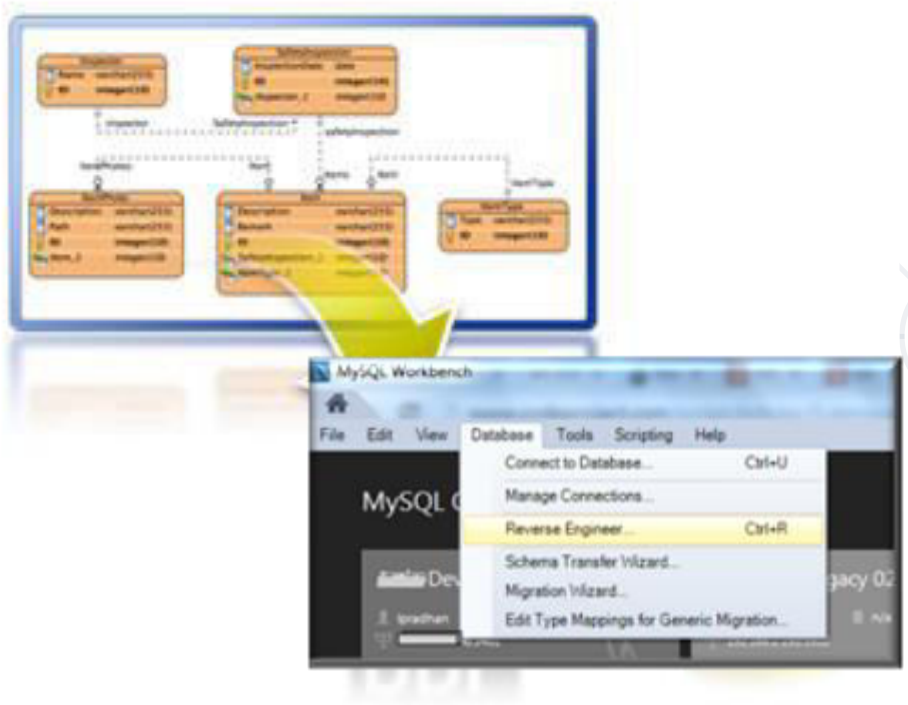
Foreign Key Update Cascade

```
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


MySQL Workbench
File
Edit
View
Database
Tools
Scripting
Help
Connect to Database... Ctrl+U
Manage Connections...
Reverse Engineer... Ctrl+R
Schema Transfer Wizard...
Migration Wizard...
Edit Type Mappings for Generic Migration...

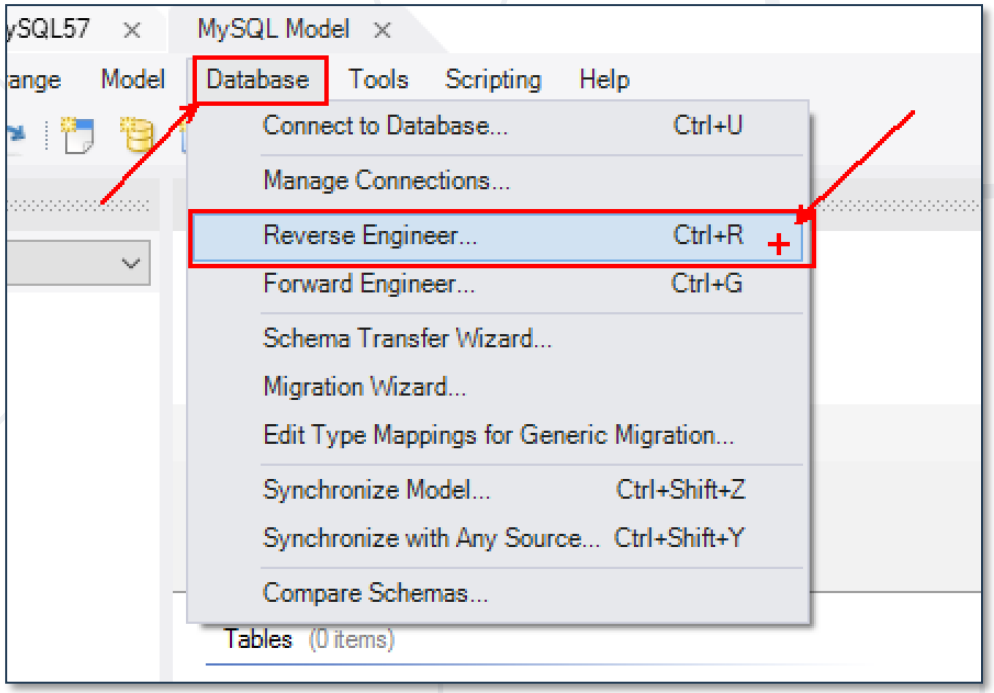
E/R Diagrams

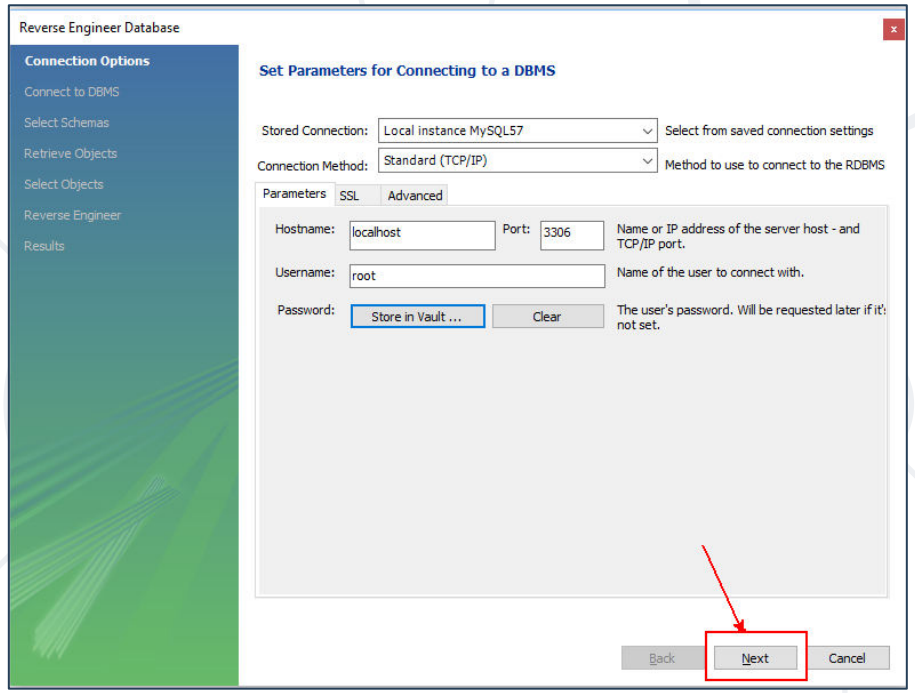
Relational Schema

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

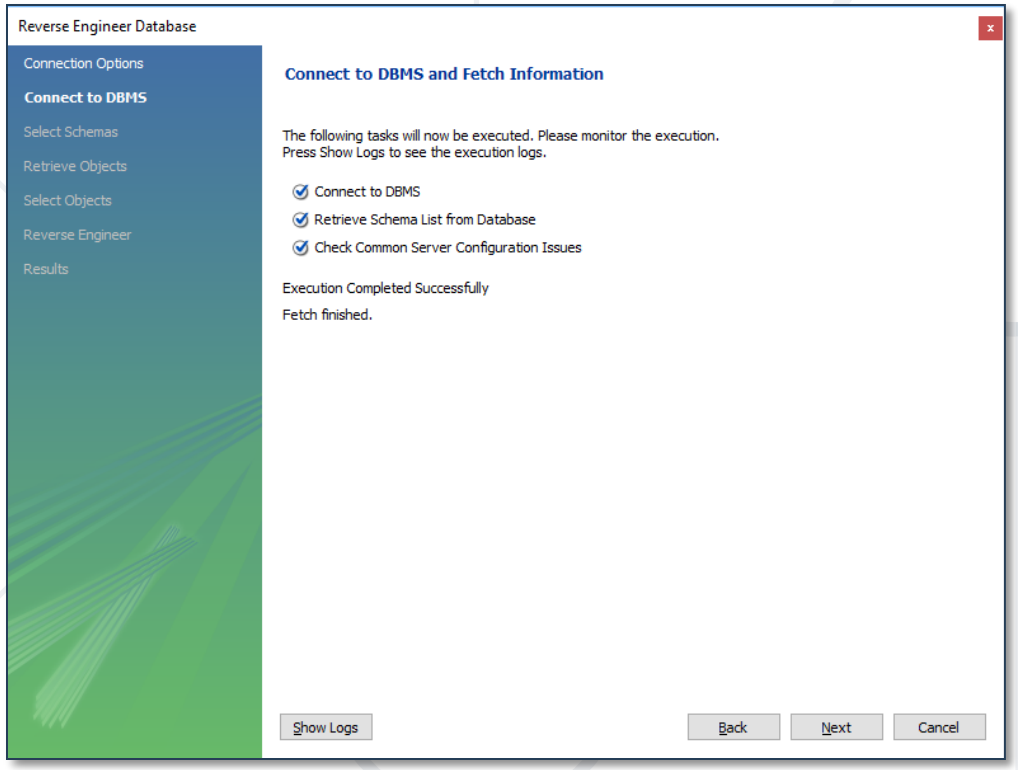
E/R Diagram

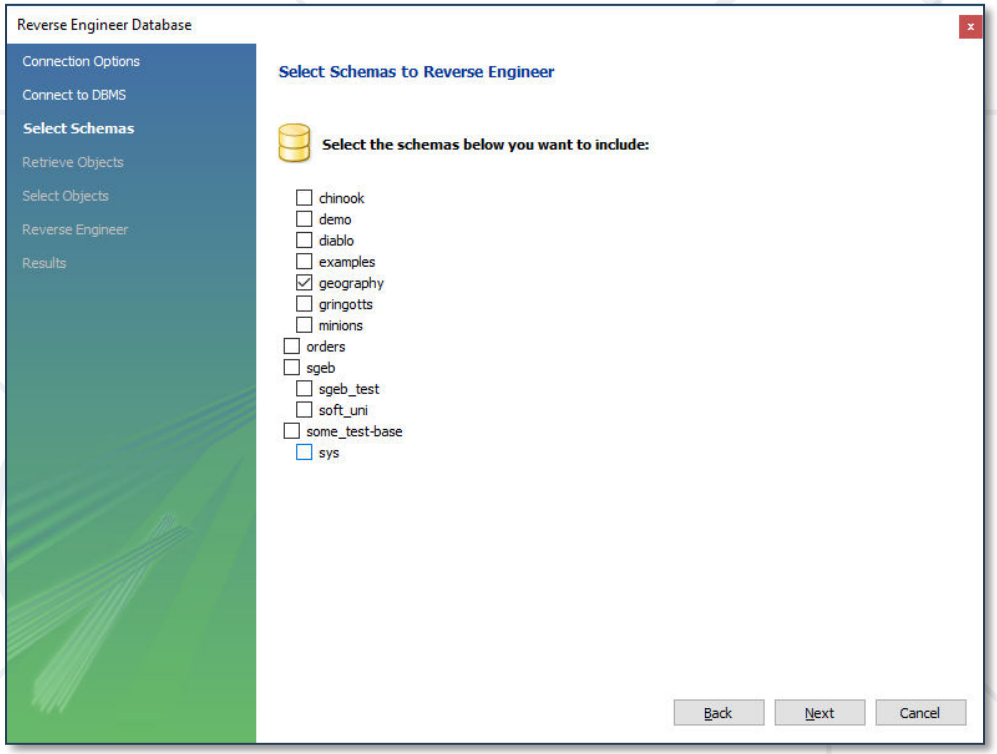
- Click on "Database" then select "Reverse Engineer"



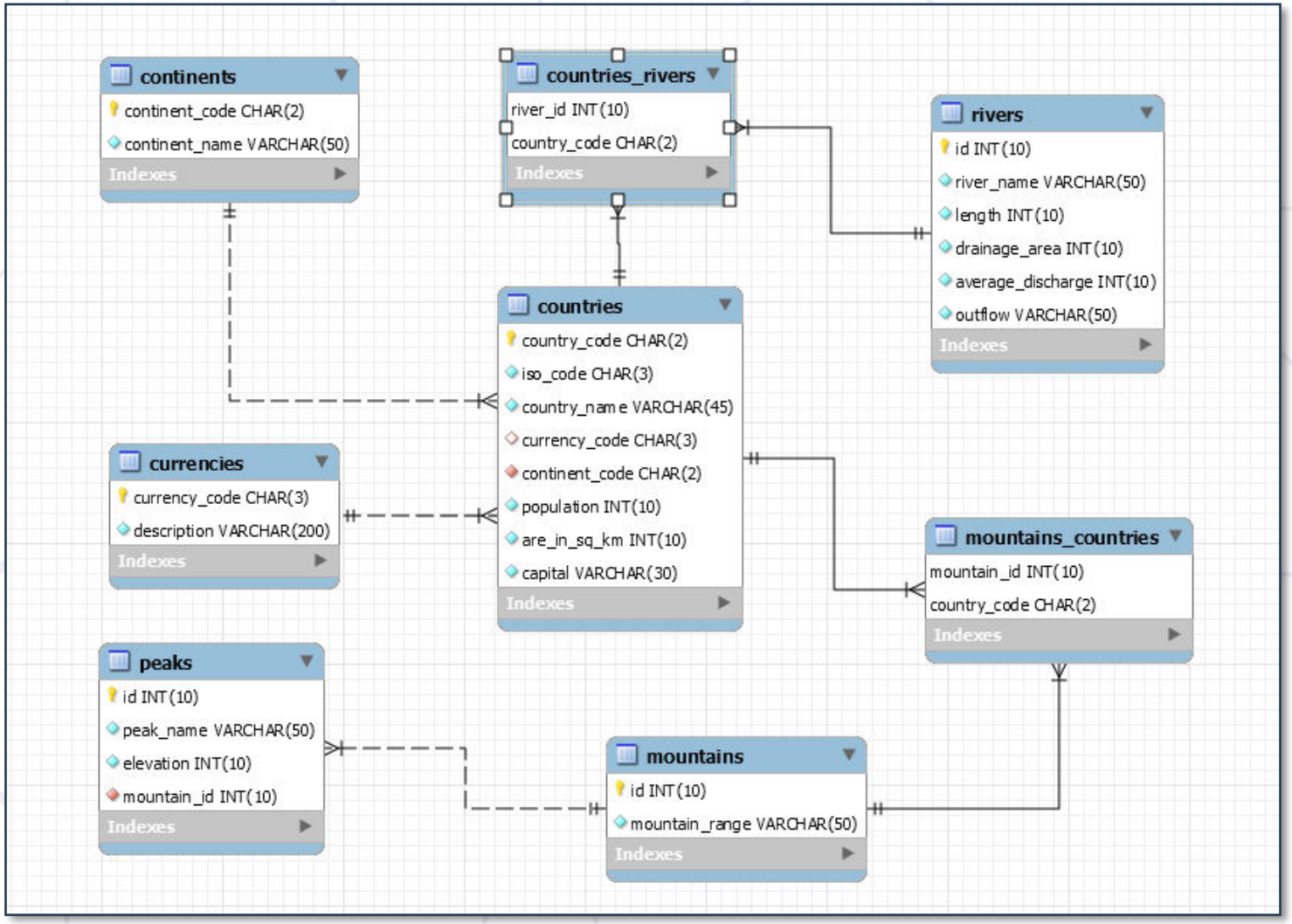


E/R Diagram


Reverse Engineer Database
Connection Options
Connect to DBMS
Select Schemas
Retrieve Objects
Select Objects
Reverse Engineer
Results
Connect to DBMS and Fetch Information
The following tasks will now be executed. Please monitor the execution.
Press Show Logs to see the execution logs.
Connect to DBMS
Retrieve Schema List from Database
Check Common Server Configuration Issues
Execution Completed Successfully
Fetch finished.
Show Logs
Back
Next
Cancel


Reverse Engineer Database
Connection Options
Connect to DBMS
Select Schemas
Retrieve Objects
Select Objects
Reverse Engineer
Results
Select Schemas to Reverse Engineer
Select the schemas below you want to include:
chinook
demo
diablo
examples
geography
gringotts
minions
orders
sgeb
sgeb_test
soft_uni
some_test-base
sys
Back
Next
Cancel

E/R Diagram


continents
continent_code CHAR(2)
continent_name VARCHAR(50)
Indexes
countries_rivers
river_id INT(10)
country_code CHAR(2)
Indexes
rivers
id INT(10)
river_name VARCHAR(50)
length INT(10)
drainage_area INT(10)
average_discharge INT(10)
outflow VARCHAR(50)
Indexes
countries
country_code CHAR(2)
iso_code CHAR(3)
country_name VARCHAR(45)
currency_code CHAR(3)
continent_code CHAR(2)
population INT(10)
are_in_sq_km INT(10)
capital VARCHAR(30)
Indexes
currencies
currency_code CHAR(3)
description VARCHAR(200)
Indexes
mountains_countries
mountain_id INT(10)
country_code CHAR(2)
Indexes
peaks
id INT(10)
peak_name VARCHAR(50)
elevation INT(10)
mountain_id INT(10)
Indexes
mountains
id INT(10)
mountain_range VARCHAR(50)
Indexes

Summary

- We design databases by specification **entities** and their **characteristics**
- Types of relations:
  - **One-to-one**
  - **One-to-many**
  - **Many-to-many**
- We visualize relations via E/R diagrams

Questions?

SoftUni

Software University

SoftUni Svetlina

SoftUni Creative

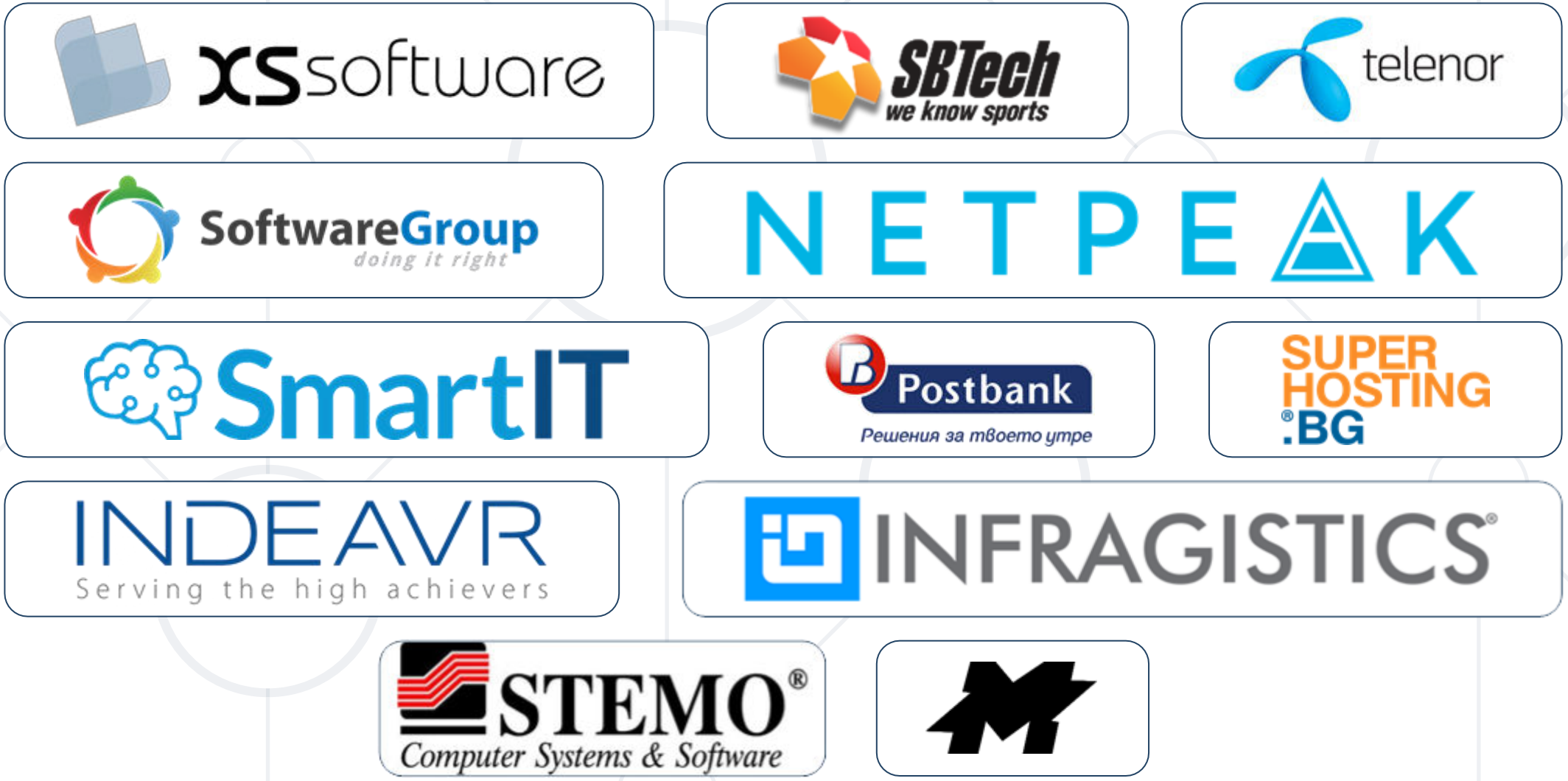
SoftUni Digital

SoftUni Foundation

SoftUni Kids

SoftUni Diamond Partners

Software University



SoftUni Organizational Partners

Software University

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
- Software University Forums
  - forum.softuni.bg

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://softuni.org
- © Software University – https://softuni.bg