Table Relations

Database Design and Rules
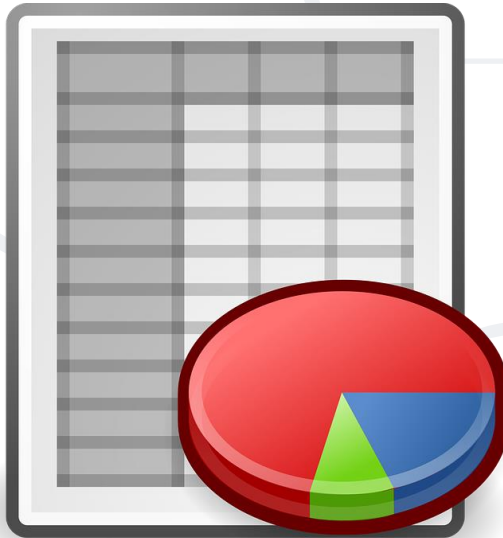
SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

Table of Contents

sli.do

#csharp-db

Database Design

Fundamental Concepts

Steps in Database Design

- Steps in the database design process:
  - Identify entities
  - Identify table columns
  - Define a primary key for each table
  - Identify and model relationships
  - Define other constraints
  - Fill tables with test data

DB Design: Identify 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**

- 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)

  - Must be an **integer** number

  - Must be **declared** as a primary key

  - Use **IDENTITY** 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)

DB Design: Identify Entity 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

Database Normalization

Database Normalization

- It is a technique of organizing the data in the database

- **Normalization** is a systematic approach of **decomposing** tables to eliminate data redundancy (repetition) and undesirable characteristics like insertion, update and deletion **anomalies**

- It is a multi-step process that puts data into **tabular form** removing duplicated data from the relation tables

Normal Forms

- **First Normal Form (1NF)**

  - Table should only have single (atomic) valued attributes/columns

  - Values stored in a column should be of the same domain (same type)

  - All the columns in a table should have unique names

  - The order in which data is stored should not matter

- **Second Normal Form (2NF)**

  - The table should be in the **First Normal form**

  - It shouldn't have **Partial Dependency** (dependency on part of the primary key)

- **Third Normal Form (3NF)**

  - The table is in the **Second Normal form**

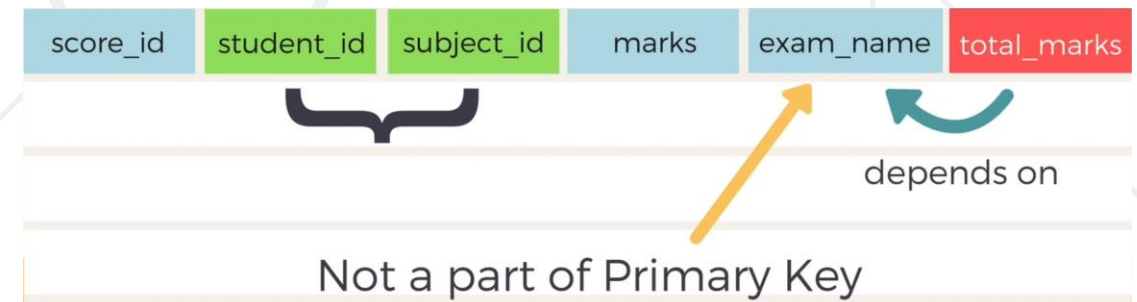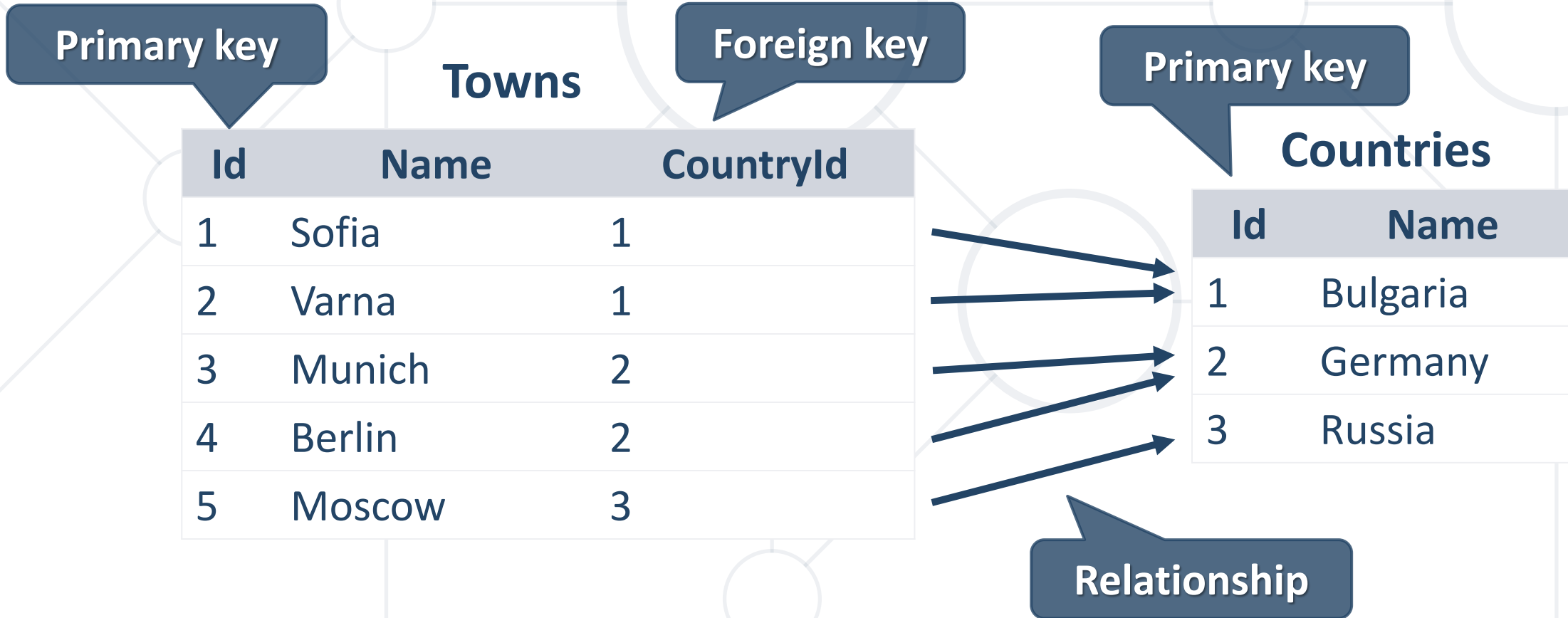 - It doesn't have **Transitive Dependency**

score_id	student_id	subject_id	marks	exam_name	total_marks

depends on

Not a part of Primary Key

Table Relations
Relational Database Model in Action

Table Relations

- **Relationships** between tables are based on interconnections: **primary key ➔ foreign key**

Primary key

Towns

Foreign key

Primary key

Countries

Id	Name	CountryId
1	Sofia	1
2	Varna	1
3	Munich	2
4	Berlin	2
5	Moscow	3

Id	Name
1	Bulgaria
2	Germany
3	Russia

Relationship

Custom Column Properties

- Primary Key

```
Id INT NOT NULL PRIMARY KEY
```

- Identity (auto-increment)

```
Id INT PRIMARY KEY IDENTITY
```

- Unique constraint – no repeating values in entire table

```
Email VARCHAR(50) UNIQUE
```

Table Relations: Foreign Key

- The **foreign key** is an **identifier** of a record located in **another table** (usually a primary key)

- Using relationships, we **refer** to data instead of **repeating** data

  - Country name is **not repeated**, it is **referred** to by its **primary key**

Towns

Id	Name	CountryId
1	Sofia	1
2	Varna	1
3	Munich	2
4	Berlin	2

Countries

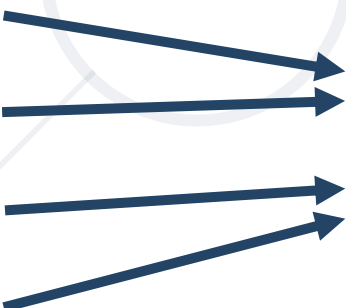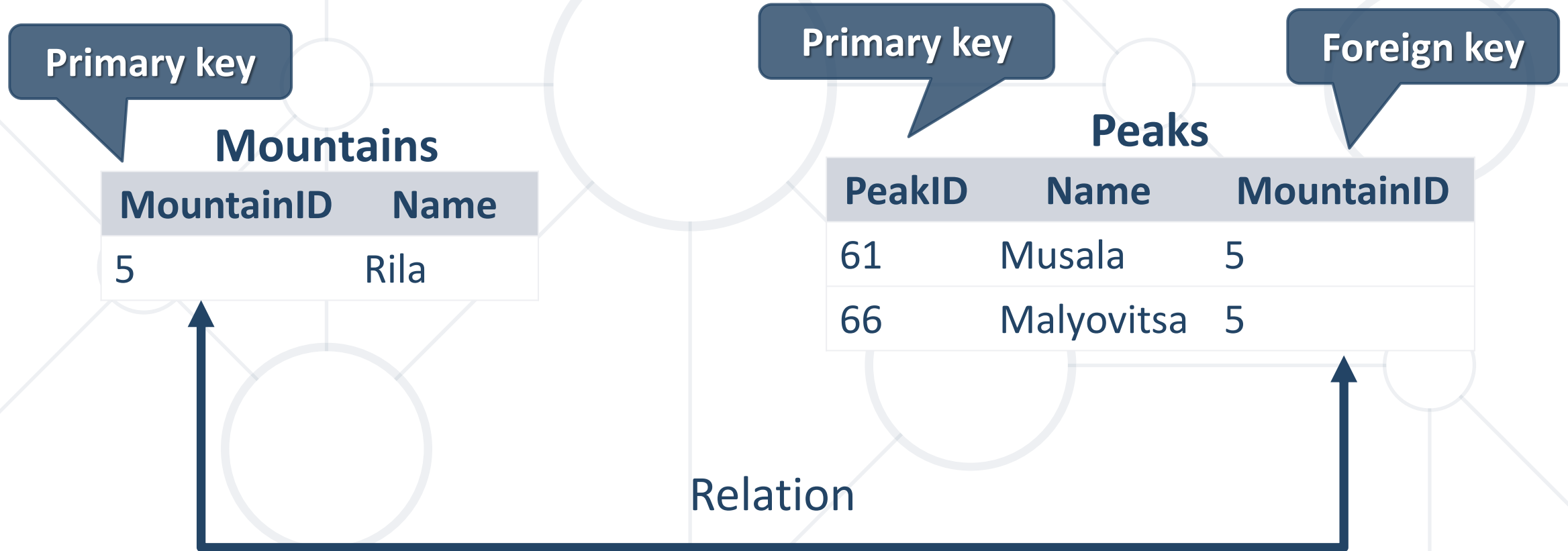| Id	Name
1	Bulgaria
2	Germany

Table Relations: Multiplicity

- **One-to-many** – e.g. country / towns
  - One country has many towns
- **Many-to-many** – e.g. student / course
  - One student has many courses
  - One course has many students
- **One-to-one** – e.g. example driver / car
  - One driver has only one car
  - Rarely used

One-to-Many/Many-to-One

Primary key

Mountains

MountainID	Name
5	Rila

Primary key

Peaks

PeakID	Name	MountainID
61	Musala	5
66	Malyovitsa	5

Foreign key

Relation

```sql
CREATE TABLE Mountains(
    MountainID INT PRIMARY KEY,
    MountainName VARCHAR(50)
)

CREATE TABLE Peaks(
    PeakId INT PRIMARY KEY,
    MountainID INT,
    CONSTRAINT FK_Peaks_Mountains
    FOREIGN KEY (MountainID)
    REFERENCES Mountains(MountainID)
)
```

Primary key

Foreign Key

One-to-Many: Foreign Key

- The table holding the **foreign key** is the **child table**

- The table holding the **referenced primary key** is the **parent/referenced table**

> Constraint Name

```
CONSTRAINT FK_Peaks_Mountains
FOREIGN KEY (MountainID)
REFERENCES Mountains(MountainID)
```

> Foreign Key

> Parent Table

> Primary Key

Many-to-Many

- **Many-to-many relations** use a **mapping/join table**

Primary key

Primary key

Employees

EmployeeID	EmployeeName
1	…
40	…

Projects

ProjectID	ProjectName
4	..
24	…

EmployeesProjects

Mapping table

EmployeeID	ProjectID
1	4
1	24
40	24

```
CREATE TABLE Employees(
    EmployeeID INT PRIMARY KEY,
    EmployeeName VARCHAR(50)
)

CREATE TABLE Projects(
    ProjectID INT PRIMARY KEY,
    ProjectName VARCHAR(50)
)
```

```
CREATE TABLE EmployeesProjects(
    EmployeeID INT,
    ProjectID INT,
    CONSTRAINT PK_EmployeesProjects
    PRIMARY KEY(EmployeeID, ProjectID),
    CONSTRAINT FK_EmployeesProjects_Employees
    FOREIGN KEY(EmployeeID)
    REFERENCES Employees(EmployeeID),
    CONSTRAINT FK_EmployeesProjects_Projects
    FOREIGN KEY(ProjectID)
    REFERENCES Projects(ProjectID)
)
```
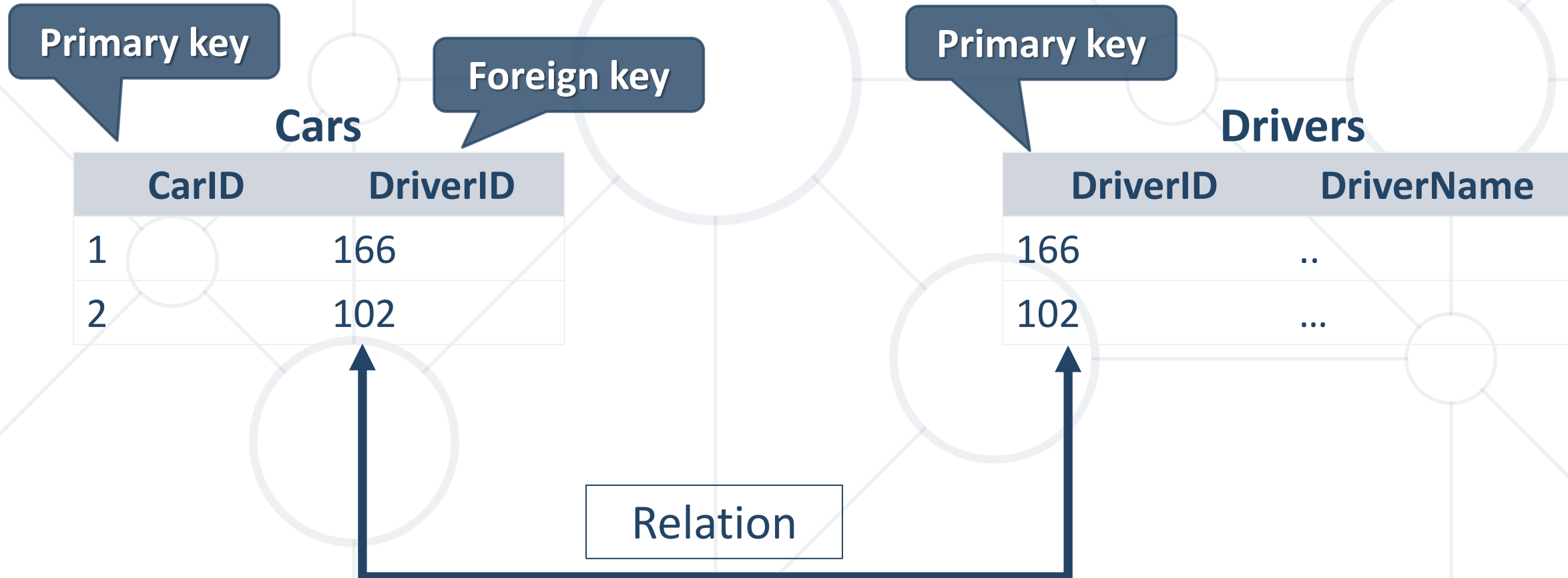
Composite Primary Key

Foreign Key to Employees

Foreign Key to Projects

23

One-to-One

Primary key

Foreign key

Primary key

Cars

Drivers

CarID	DriverID
1	166
2	102

DriverID	DriverName
166	..
102	...

Relation

One-to-One

```
CREATE TABLE Drivers(
    DriverID INT PRIMARY KEY,
    DriverName VARCHAR(50)
)

CREATE TABLE Cars(
    CarID INT PRIMARY KEY,
    DriverID INT UNIQUE,
    CONSTRAINT FK_Cars_Drivers FOREIGN KEY
    (DriverID) REFERENCES Drivers(DriverID)
)
```

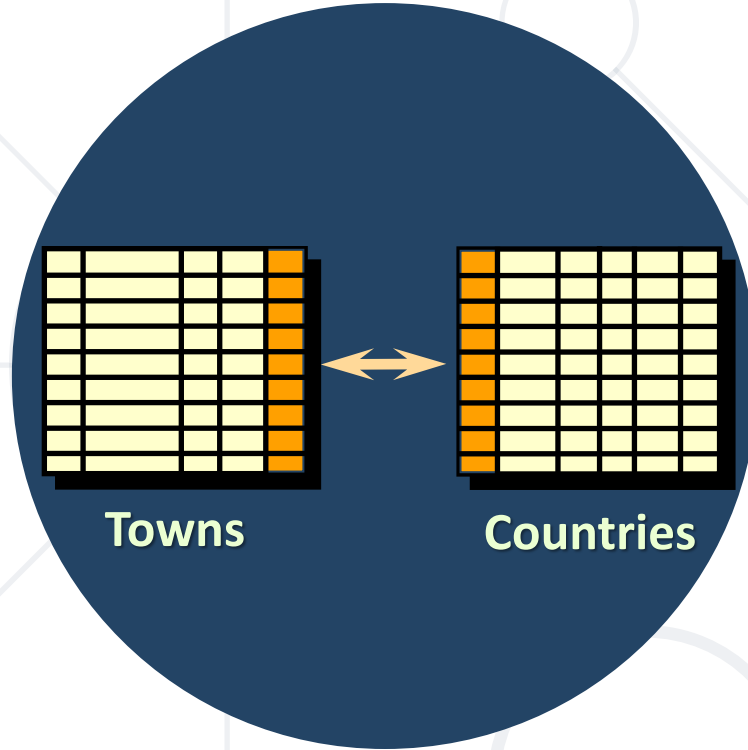Primary key

One driver per car

Foreign Key

One-to-One: Foreign Key

Constraint Name

Foreign Key

```
CONSTRAINT FK_Cars_Drivers
FOREIGN KEY (DriverID)
REFERENCES Drivers(DriverID)
```

Referenced Table

Primary Key

Retrieving Related Data
Using Simple JOIN Statements

JOIN Statements

- With a **JOIN** statement, we can get data from two tables **simultaneously**

  - **JOINs** require at least two tables and a "**join condition**"

```
SELECT * FROM Towns
    JOIN Countries ON
        Countries.Id = Towns.CountryId
```

Join Condition

28

- Use database "**Geography**". Report all peaks for "**Rila**" mountain.

  - Report includes mountain's name, peak's name and also peak's elevation

  - Peaks should be **sorted** by elevation descending

	MountainRange	PeakName	Elevation
1	Rila	Musala	2925
2	Rila	Malka Musala	2902
3	Rila	Malyovitsa	2729
4	Rila	Orlovets	2685

Check your solution here: https://judge.softuni.org/Contests/Compete/Index/292#6

Solution: Peaks in Rila

Cross Table Selection

```
SELECT m.MountainRange, p.PeakName, p.Elevation
  FROM Mountains AS m
  JOIN Peaks As p ON p.MountainId = m.Id
 WHERE m.MountainRange = 'Rila'
ORDER BY p.Elevation DESC
```

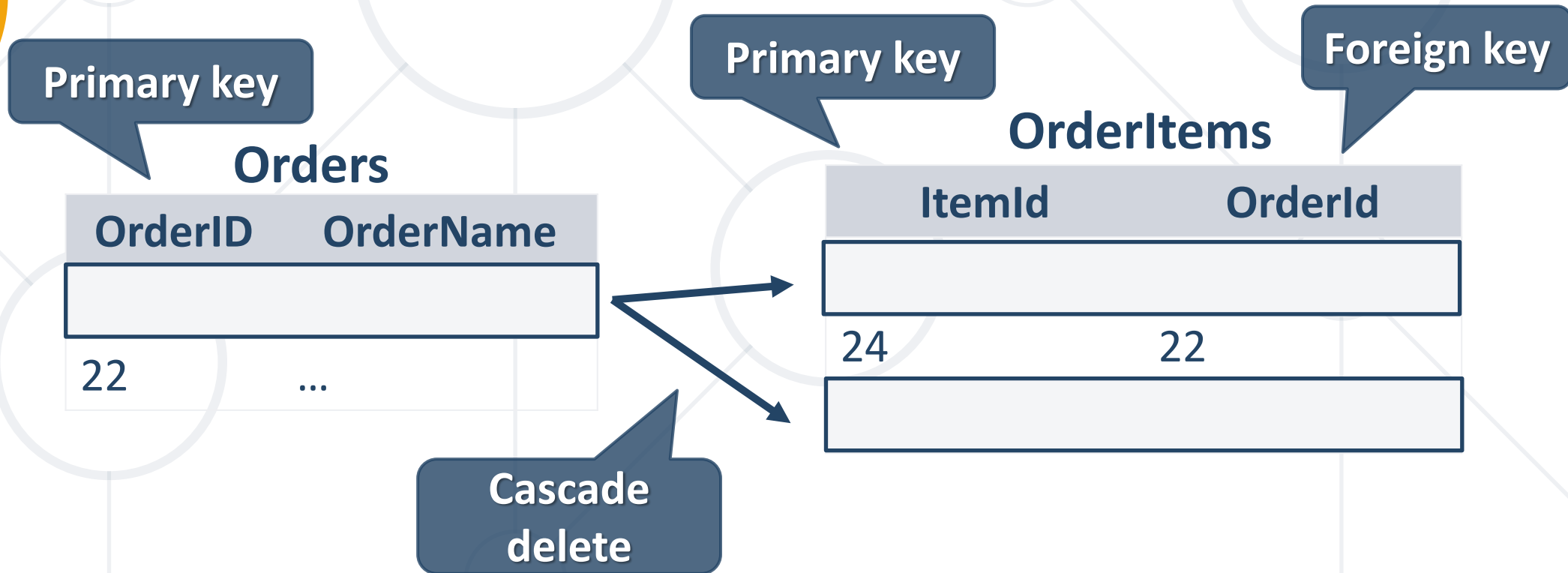Join Condition

Check your solution here: https://judge.softuni.org/Contests/Compete/Index/292#6
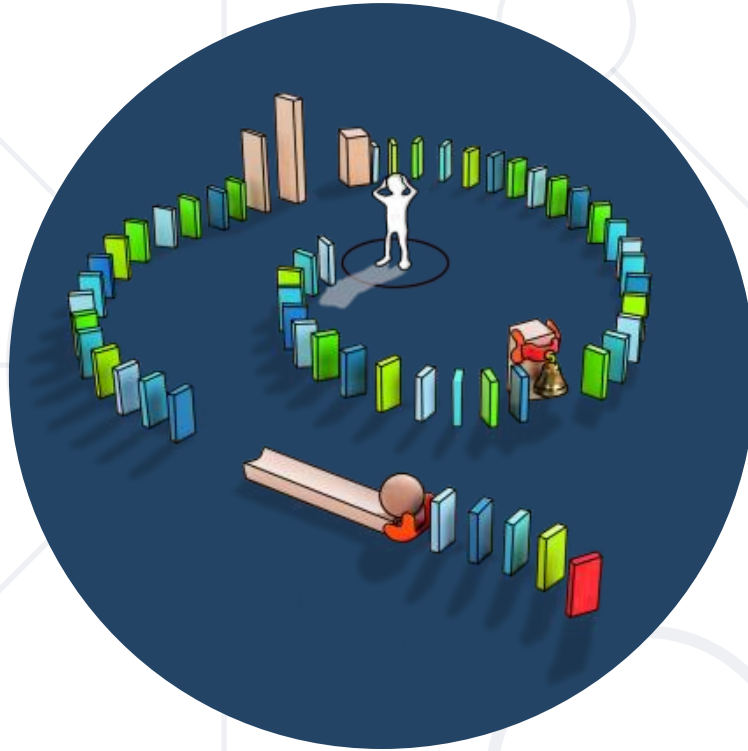
Cascade Operations
Cascade Delete/Update

Definition

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

Primary key

Primary key

Foreign key

Orders

OrderItems

OrderID	OrderName
22	...

ItemId	OrderId
24	22

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 perform a "**logical delete**"

    - Entities are **marked** as deleted (but not actually deleted)

  - In more complicated relations, cascade delete won't work with **circular** references

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**

```
CREATE TABLE Drivers(
    DriverID INT PRIMARY KEY,
    DriverName VARCHAR(50)
)

CREATE TABLE Cars(
    CarID INT PRIMARY KEY,
    DriverID INT,
    CONSTRAINT FK_Car_Driver FOREIGN KEY(DriverID)
    REFERENCES Drivers(DriverID) ON DELETE CASCADE
)
```

Foreign Key

Cascade

```
CREATE TABLE Products(
  BarcodeId INT PRIMARY KEY,
  Name VARCHAR(50)
)

CREATE TABLE Stock(
  Id INT PRIMARY KEY,
  Barcode INT,
  CONSTRAINT FK_Stock_Products FOREIGN KEY(BarcodeId)
  REFERENCES Products(BarcodeId) ON UPDATE CASCADE
)
```

Foreign Key

Cascade

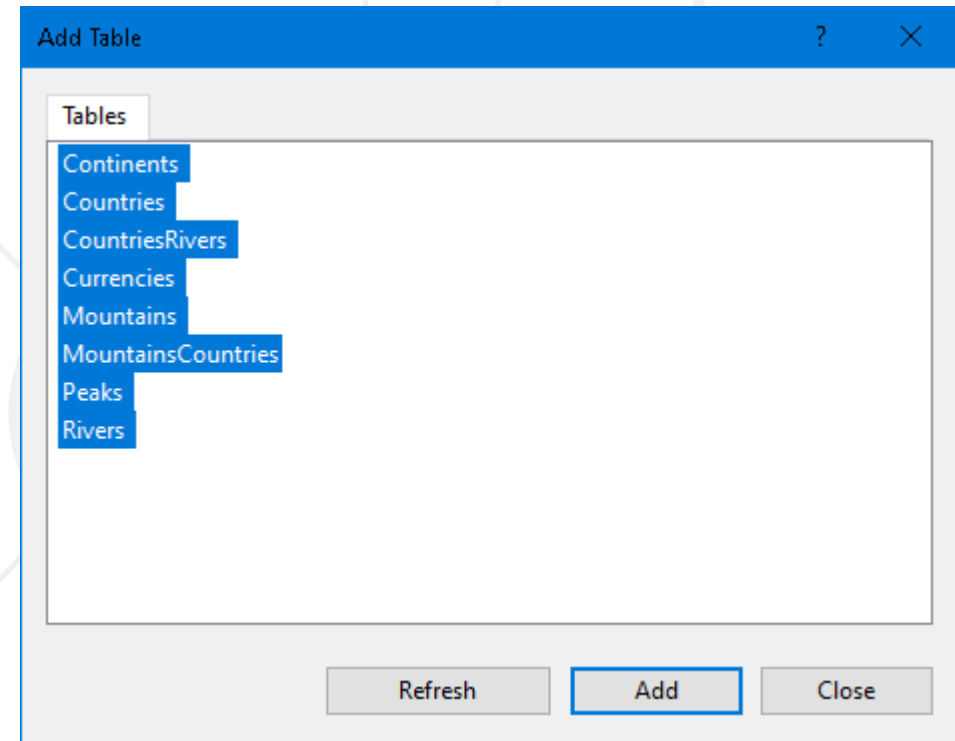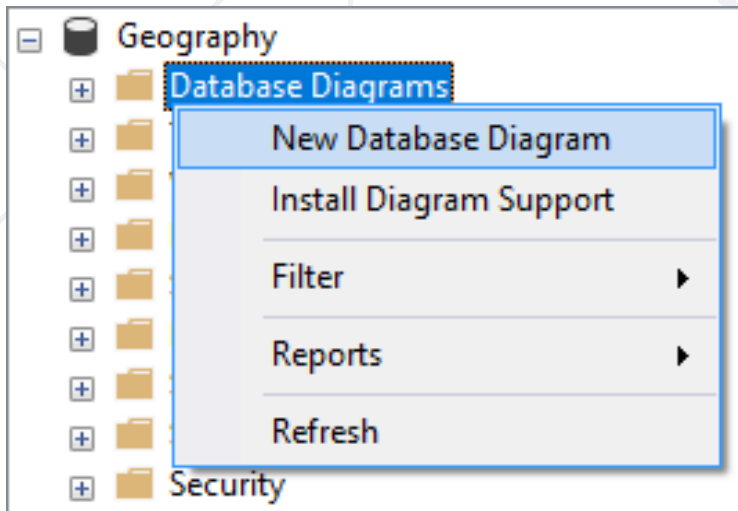# E/R Diagrams
Entity / Relationship Diagrams

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
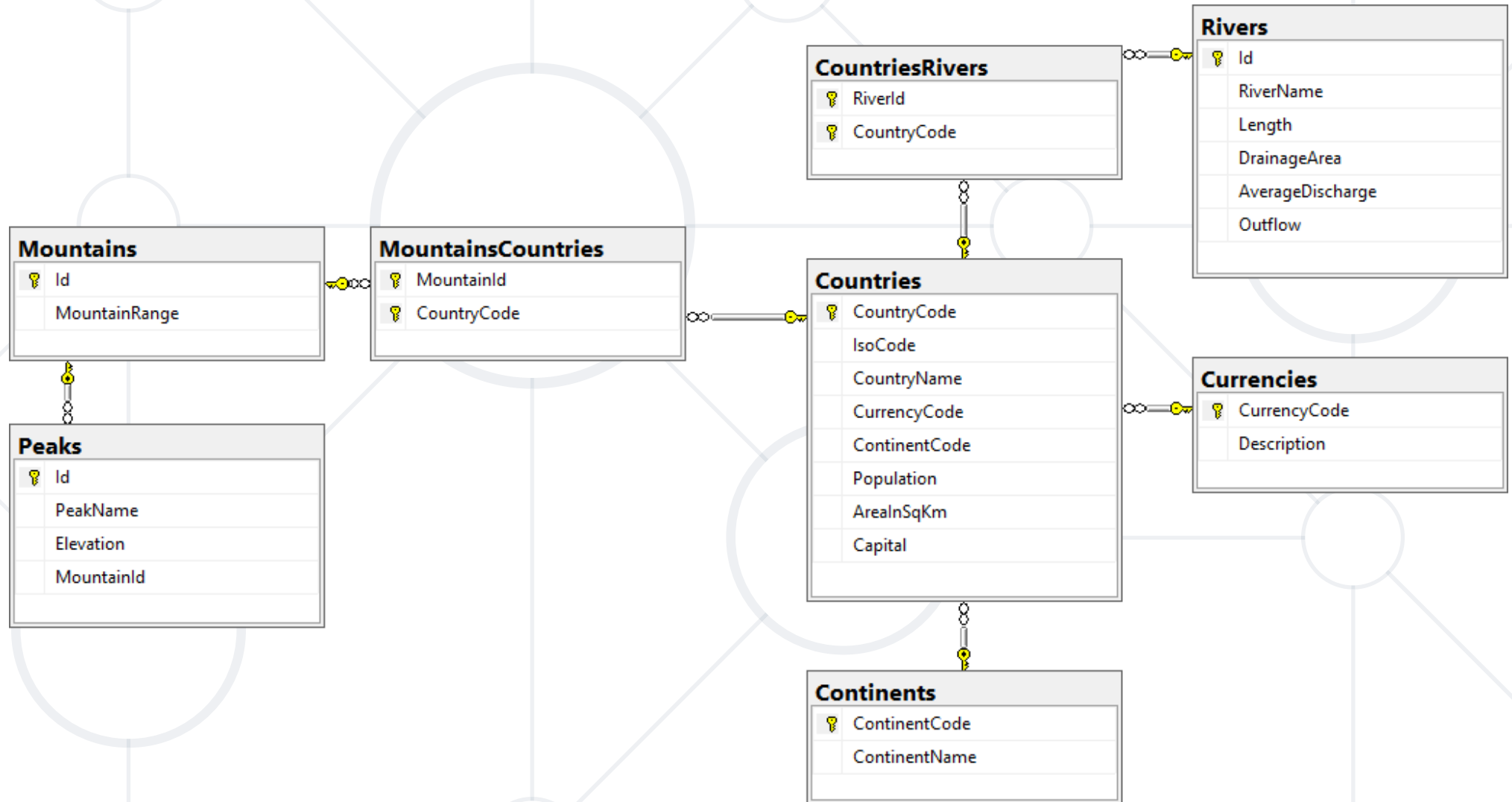- Relational schemas are graphically displayed in Entity / Relationship diagrams (**E/R Diagrams**)

SSMS E/R Diagram: Usage

- Expand a database in **Object Explorer**

  - Right click "**Database Diagrams**" then select "**New Database Diagram**"
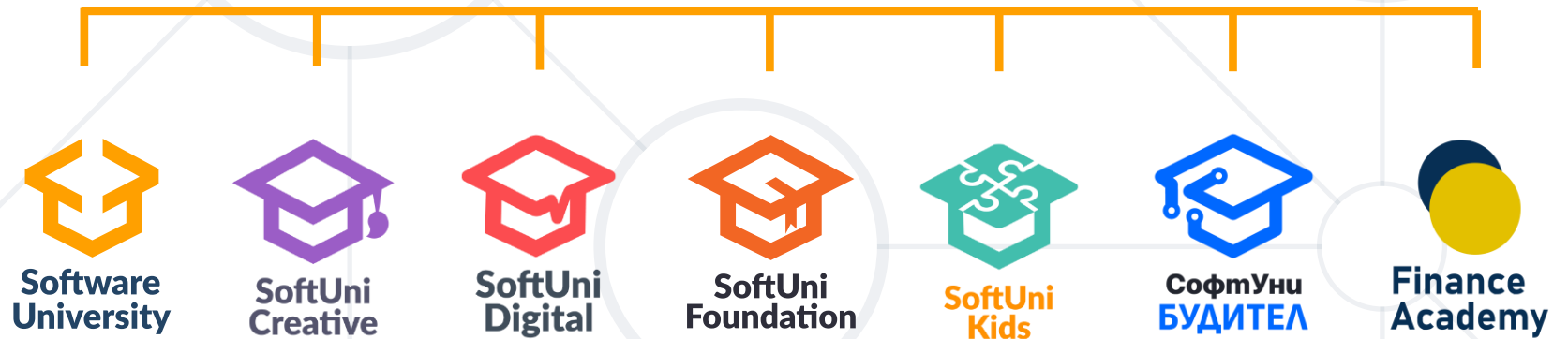
SSMS E/R Diagram

Summary

- Design **multiple** tables with related data
- Types of table relations
- **Cascading** – Pros and Cons
- Entity / Relationship Diagrams

Questions?

SoftUni Diamond Partners

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- 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