re-BUS

You have downloaded a document from
RE-BUŚ
repository of the University of Silesia in Katowice

**Title:** Databases and SQL

**Author:** Pavel Turčínek, Martin Drlík, Ján Skalka, Beata Zielosko, Zenón José Hernández-Figueroa, José Daniel González-Domínguez, Juan Carlos Rodríguez-del-Pino, Jaroslav Reichel

**Citation style:** Turčínek Pavel, Drlík Martin, Skalka Ján, Zielosko Beata, Hernández-Figueroa Zenón José, González-Domínguez José Daniel, Rodríguez-del-Pino Juan Carlos, Reichel Jaroslav. (2021). Databases and SQL. Nitra : Constantine the Philosopher University in Nitra.

Uznanie autorstwa - Użycie niekomercyjne - Bez utworów zależnych Polska - Licencja ta zezwala na rozpowszechnianie, przedstawianie i wykonywanie utworu jedynie w celach niekomercyjnych oraz pod warunkiem zachowania go w oryginalnej postaci (nie tworzenia utworów zależnych).

UNIWERSYTET ŚLĄSKI W KATOWICACH
Biblioteka Uniwersytetu Śląskiego
Ministerstwo Nauki i Szkolnictwa Wyższego

FITPED

# Databases and SQL

Pavel Turčínek (MENDELU)
Martin Drlík (UKF)
Ján Skalka (UKF)
Beata Zielosko (US)
Zenón José Hernández-Figueroa (ULPGC)
José Daniel González-Domínguez (ULPGC)
Juan Carlos Rodríguez-del-Pino (ULPGC)
Jaroslav Reichel (UKF)

www.fitped.eu

2021

Co-funded by the
Erasmus+ Programme
of the European Union

Work-Based Learning in Future
IT Professionals Education
(Grant. no. 2018-1-SK01-KA203-046382)

# Databases and SQL

**Published on**

November 2021

**Authors**

Pavel Turčínek | Mendel University in Brno, Czech Republic

Martin Drlík | Constantine the Philosopher University in Nitra, Slovakia

Ján Skalka | Constantine the Philosopher University in Nitra, Slovakia

Beata Zielosko | University of Silesia in Katowice, Poland

Zenón José Hernández-Figueroa | University of Las Palmas de Gran Canaria, Spain

José Daniel González-Domínguez | University of Las Palmas de Gran Canaria, Spain

Juan Carlos Rodríguez-del-Pino | University of Las Palmas de Gran Canaria, Spain

Jaroslav Reichel | Constantine the Philosopher University in Nitra, Slovakia

**Reviewers**

Jozef Kapusta | Pedagogical University of Cracow, Poland

Piet Kommers | Helix5, Netherland

Eugenia Smyrnova-Trybulska | University of Silesia in Katowice, Poland

Peter Švec | Teacher.sk, Slovakia

**Graphics**

Ľubomír Benko | Constantine the Philosopher University in Nitra, Slovakia

David Sabol | Constantine the Philosopher University in Nitra, Slovakia

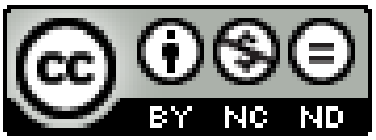
Erasmus+ FITPED

Work-Based Learning in Future IT Professionals Education

Project 2018-1-SK01-KA203-046382

Co-funded by the
Erasmus+ Programme
of the European Union

The European Commission support for the production of this publication does not constitute an endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.




Licence (licence type: Attribution-Non-commercial-No Derivative Works) and may be used by third parties as long as licensing conditions are observed. Any materials published under the terms of a CC Licence are clearly identified as such.

All trademarks and brand names mentioned in this publication and all trademarks and brand names mentioned that may be the intellectual property of third parties are unconditionally subject to the provisions contained within the relevant law governing trademarks and other related signs. The mere mention of a trademark or brand name does not imply that such a trademark or brand name is not protected by the rights of third parties.

© 2021 Constantine the Philosopher University in Nitra

**ISBN 978-80-558-1782-8**

# Table of Contents

# Basic Terminology

Chapter 1

# 1.1 Basic terminology

## 🕮 1.1.1

Data

Data are properties of objects typically obtained by measurement or observation. In order for data to be processed, they must be expressed. Data can be expressed by text, speech, image (graphically), electronically, etc. When we simplify it, we can say that the data are expressed using signs or signals.

## 🕮 1.1.2

Data record / data sentence

In order to work with the data, the data is put together into higher logical units, and these are called records (sentences). The record is marked as a logical data unit. However, a record is not the smallest data element. Record (sentence) can be decomposed. As in everyday life, the sentence consists of words and the issue of data will divide the sentence (a record) into attributes. The attribute is the least addressable part of the sentence. Just as the word is decomposable to individual letters, in some cases it is possible to divide an attribute. However, as with words, they would lose their meaning.

## 🕮 1.1.3

Attributes

Attributes may be atomic or structured. An example of a structured attribute can be an address (street, house number, city, zip code, …). Such attributes are good to avoid and divide into atomic attributes. Attributes have a certain position in the sentence (a record), they have their meaning. In order for an attribute to play its role properly, the values it will acquire must be meaningful. The set of allowable values that an attribute can acquire is called a domain.

## 🕮 1.1.4

Domain

The domain does not just specify that the attribute NAME OF A PERSON is a string of characters. This is the determination of the data type (it will be explained later in

more detail). The domain is more specific. It represents all meaningful values for the given attribute. Into the domain of the attribute NAME OF A PERSON can be included a value "Pavel", but the value of "x7br_15" does not fall into the domain of this attribute, even if it is a text string.

## 📝 1.1.5

Which of the following values are suitable for the domain of the attribute SPORT.

- swimming
- reading
- programming
- football
- curling

## 🕮 1.1.6

Type of record (sentence)

The type of record determines which attributes (including domains) the record has. The order of individual attributes also plays an important role. The type of record (sentence) is specified by its attributes for example NAME OF A PERSON, ACTIVITY, OBJECT. The sentence of this type can then be: "John drives a car."

## 📝 1.1.7

Let us suppose, there is a type of record described by these attributes: ANIMAL, ACTIVITY, PLACE. Chose which of the following sentences can be described by this type.

- A cow eats grass.
- A wolf runs in a forest.
- A car stands in a parking place.
- A ladybird flies over a meadow.

## 🕮 1.1.8

Information

Information is intangible. Information is data that has meaning. Information is, therefore, a subset of data. Information can respond to questions, thus reducing ignorance (uncertainty). The information may be contained in both the signs (or signals) themselves and also in their arrangement. Physical appearance may vary (text, image, signals, etc.). Data provide information only to those who understand them, who are able to recognize their syntax and understand semantics. What is information for someone, can be only data for another.

## 1.1.9

Choose the definition that suits the best to term "data".

- All data can be described as information stored on some media.
- Data are properties of objects which you can get by observation or measurement.
- Data brings you a new view of an already known fact.
- Data can never decrease your uncertainty.

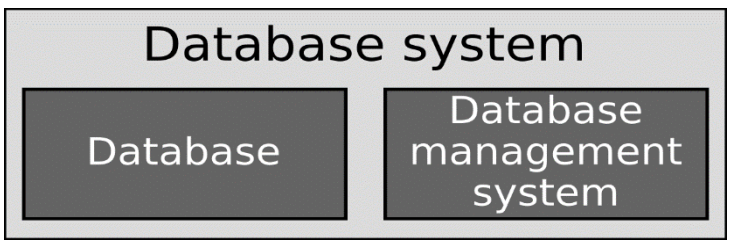
## 1.1.10

Identification of records (sentences)

Records of the same type are grouped into data files. In order to work with them (eg search, delete, edit), each record must be clearly distinguished from others. Therefore, each record must be identified by the file key. A key is a set of attributes that uniquely identify a record. The number of attributes belonging to the key is called k. The total number of attributes of the sentence is denoted as n. Always apply $k \leq n$. The goal is to keep it as small as possible. All minimum keys create space K*. One of the keys is marked as the primary key.

## 1.1.11

The architecture of the Database System



As can be seen from the picture, the database system consists of two basic components:

- database,
- a database management system.

## 🕮 1.1.12

Database

The database is a set of structured homogeneous files. These are data that are intended for further processing. They are meaningfully divided into individual files. All data within a single file has the same structure.

## 🕮 1.1.13

Database management system

In order to access the database, there is a tool called a *database management system*. Often, the DBMS abbreviation is also used. A Database management system is an integrated software tool that allows to define, create, and manage access to the database and work with it. Typically, this is a collection of programs that make up the interface between application programs and stored data.

The database management system has many features. Above all, it allows you to manipulate data. It ensures that only authorized users can access the data. It also allows concurrent access for multiple users. It provides transaction management. Depending on the model used, it creates a database and defines its schema (structure). In case of failure, it allows recovery. It checks data integrity and provides many other tasks.

## 🕮 1.1.14

Benefits of DBMS

The benefits that the DBMS brings to direct access to data are indisputable. The main ones include:

- **Abstraction of data** - the user does not work directly with the source files, but with formalized structures at the higher logical level of abstraction.

- **Independence of data** - if physical data changes, it does not affect the work of application programs. The data interface will still look the same on the outside.
- **Centralized data administration** - all data is in one place. Everything is treated in a similar way. It is possible to display a description of the data structure.
- **The ability to formulate ad-hoc queries outside of application programs** - users can randomly query queries on a database through the DBMS. They do not need other programs to do so.

## 1.1.15

Choose features that are allowed thanks to DBMS:

- access to data only through other applications
- concurrent access
- data manipulation
- recovery
- unauthorized access

## 1.1.16

Match together the following terms and sentences about DBMS.

The user does not work directly with the source files, but with formalized structures. _____

If physical data changes, it does not affect the work of application programs. The data interface will still look the same on the outside._____

All data is in one place. Everything is treated in a similar way. It is possible to display a description of the data structure._____

Users can randomly query queries on a database through the DBMS. They do not need other programs to do so. _____

- Abstraction of data
- Centralized data administration
- Independence of data
- The ability to formulate ad-hoc queries outside of application programs

## 1.1.17

To sentences (characteristics) assign the words (terms) that match them best.

Set of structured homogeneous files. _____

The set of allowable values that an attribute can acquire. _____

Logical units of data. _____

Properties of objects. _____

Data that reduce uncertainty. _____

A software that allows to define, create, and manage access to the database and work with it. _____

- Data record
- Data
- Database management system
- Domain
- Information
- Database

# Data Modelling

## Chapter 2

# 2.1 Entities

## 📖 2.1.1

Data modelling is a process that aims to create a data model. The data model describes the data and its structure. Data can be viewed from three views.

The first is an outside view. Sometimes it is referred to as an application or a user view of the data. It's the view of a regular user who is not interested in deeper connections between the data. It's usually just their consumer.

The second view is called logical or conceptual. In both cases, it is about identifying important objects of interest and relationships between them. It does not solve the way of implementation. It covers current needs with the potential for further development.

The third view is a physical view. This view looks at how data is stored. It is a custom implementation proposal in a particular database (or other) system. Contains tables, object structures, and integrity constraints.

## 📖 2.1.2

Conceptual vs logical model

Different authors have a little bit different explanations. Most of them can be summed up in two opinions:

- The conceptual model contains only entity names and links between them. Logic contains keys and attributes.
- Conceptual contains entity names, attributes, and links between them. In a logical view, it further solves the decay of the M:N relationship and the division of attributes into atomic attributes.

Some of the terms, you don't know will be explained in the next texts. For now, try to remember at least that the logical model is a little more specific than the conceptual one.

## 📖 2.1.3

The goal of conceptual modelling is to accurately describe the data storage needs. Conceptual modelling supports discussion. It allows you to communicate with non-technical language, which prevents mistakes and misunderstandings. It defines the

initial "ideal system" documentation. In fact, the standard for creating a conceptual model is the relational diagram (ERD).

ERD consists of entities and relations between them.

## 📝 2.1.4

Choose the right term for the following descriptions.

_____ identifies important objects of interest and relationships between them.

The view of a regular user who is not interested in deeper connections between the data is called _____.

_____ is a custom implementation proposal in a particular database (or other) system.

- Conceptual view
- Outside view
- Physical view

## 📝 2.1.5

Which of following sentences are true about a conceptual modelling?

- It covers current needs with the potential for further development.
- It identifies important objects of interest and relationships between them.
- It defines the initial "ideal system" documentation.
- It supports discussion.
- It allows you to communicate with the non-technical language.
- It contains tables, object structures, and integrity constraints.
- It is a custom implementation proposal in a particular database (or other) system.

## 📖 2.1.6

Entities are objects, persons, things for which is important to keep data about them. It is a naming of a set of similar objects. An entity can be compared to a type of record you've read about earlier. Examples of entities can be:

CAR, MOVIE, PERSON, PRODUCT, ANIMAL.

One particular occurrence of an entity is called an instance.

## 📖 2.1.7

Entity or instance?

Is DOG an entity or an instance?

The data model is usually created based on a scenario from the client. The first task is to identify entities. They can be found as nouns. However, not all nouns are entities. It always depends on the context of the given assignment, so without this assignment, it is not possible to determine whether a dog is an entity or an instance.

In case you are creating a data model for a dog shelter, it is very likely that the dog will be an entity and will have its instances (specific dogs in the shelter). In the other case, the dog may be an instance of an animal entity in some other model. It is, therefore, necessary to look at the problem in its complexity.

## 📝 2.1.8

Choose the appropriate instance for each entity.

SPORT _____

COUNTRY _____

MOVIE _____

DRINK _____

PERSON _____

ANIMAL _____

- Beer
- Bear
- Czech Republic
- Ice hockey
- Edgar Frank Codd
- Schindler's List

## 📖 2.1.9

Attributes

As with the record type, entities are also characterized by their attributes. These allow you to distinguish individual instances of the entity from each other. Attributes describe entity characteristics. Individual attributes describe, quantify, qualify, classify, or specify the entity.

The attribute acquires just one value (number, character string, date, image, sound, ...) of its domain (a subset of values of a particular data type) - eg age is an integer from 0 to 120. However, the conceptual model does not address the domain too much. Attributes may also include an integrity constraint. These are additional policies that ensure that the model matches the reality. More about integrity constraints will be written later.

## 📖 2.1.10

Characteristics of attributes

Attributes, however, do not only distinguish a domain but also whether each instance of an entity must have a value for that attribute. It is discussed whether the attribute is mandatory or optional (optional). The obligation of the individual attributes depends again on the scenario.

Some attributes (such as age) have values that constantly change. These are called volatile attributes. Other attributes (such as order date) will change rarely, if ever. These are nonvolatile attributes. If there is a choice between attributes, use the nonvolatile one. For example, use birth date instead of age.

When looking for attributes in the problem description, again, you need to choose between nouns. For nouns, the decision will be made whether this is an entity or an attribute. However, some nouns may also represent instances, or they may not play a role in the model at all.

## 📝 2.1.11

Choose suitable attributes for an entity PERSON.

- first name
- date of birth
- price
- country of origin
- order date

- last name
- sex

### 📝 2.1.12

Choose suitable attributes for an entity PRODUCT.

- name
- date of birth
- price
- country of origin
- description
- password
- sex

### 📝 2.1.13

When you will create a model based on a given scenario, entities, instances, and attributes can be found among nouns. True or false?

- True
- False

### 📖 2.1.14

In modelling, it is necessary to agree on the rules of how the model will look, so everyone can understand its meaning. The way of how it is written is called a notation. There are multiple notations for creating data models and one can not say which notation is most used. Here, Baker's notation will be used.

In Baker's notation, entities are marked as rectangles. The first line shows the name of the entity. It is usually written in capital letters and is singular. The other rows describe the attributes, including their optionality. If the attribute is mandatory, it has an asterisk before it. If it is optional, it has a circle. You can see the example in the picture.


## 🕮 2.1.15

An identifier is an attribute or combination of attributes that uniquely distinguish one instance from the other. It is a unique identifier of just one instance. It is often possible to meet the UID abbreviation.

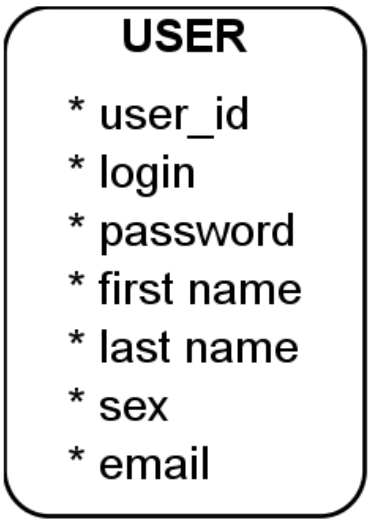
If the identifier is composed, it is a combination of multiple attributes. This is the case when one attribute for identification is not enough. For example, the street name does not uniquely identify a particular street because streets with the same name can be in two different cities. Street + City has definite information.

## 🕮 2.1.16

In some cases, it is not possible to build an identifier even with a combination of all entity attributes. You can see the example in the picture. In such cases, an artificial identifier will come in. As an artificial identifier, a numeric attribute (integer) is usually selected. This attribute is most commonly called id.

## 2.1.17

Choose if an attribute or a combination of attributes could be an identifier for the entity in the picture.




- user_id
- login
- password
- first name
- last name
- sex
- email
- first name, last name
- password, sex

## 2.1.18

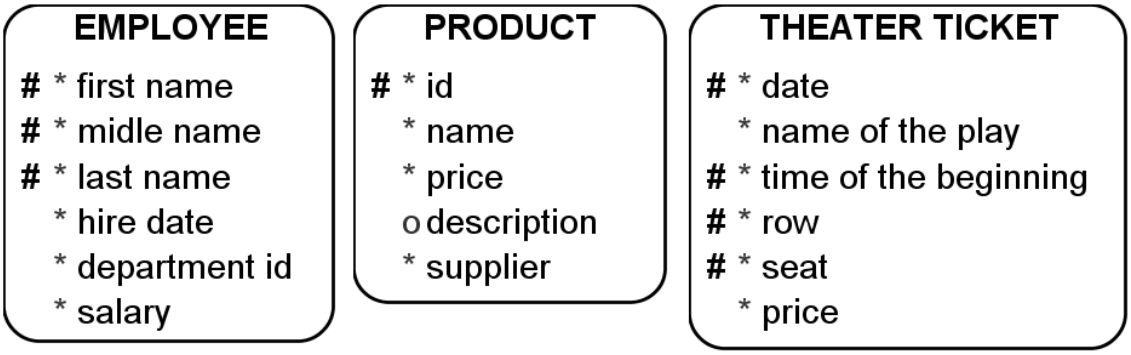
There are also cases where the entity has more identifiers. One is elected as primary. The others are referred to as candidate (secondary) identifiers. An artificial identifier is created even when there is another identifier, but for some reason, it is not suitable as a primary identifier.

When an entity is created, it must always have a specified primary identifier. You must be able to clearly distinguish one instance from another so that you can work with that instance (select, edit, delete). The primary identifier must, therefore, be a mandatory attribute. If it is composed of multiple attributes, all must be mandatory. Secondary attributes can also be marked as optional.

The primary identifier is indicated by the hash sign (#). Secondly, use a large U, as shown in the picture.


### 2.1.19

Out of these three entities chose which of them has not correctly chosen the primary identifier.




- Employee
- Product
- Theater ticket

## 2.2 Relationships

### 2.2.1

Relationships are the connections between entities. They carry important information. Relationships exist always between two entities or entities can have a relationship with themselves. Relationships are always named on both sides. An example of a relationship can be:

STUDENT (entity) STUDIES (relation) COURSE (entity).

COURSE (entity) IS STUDIED BY (relationship) STUDENT (entity).

In that scenario, relationships act as verbs. When identifying them, you need to focus on them.

## 📖 2.2.2

As with attributes, for relationships is important to state their optionality. However, it's a bit different. The optionality applies to instances of entities. The point is whether the instance of an entity must or may be in relation to the instance of another entity. The optionality is determined on the basis of appropriate questions. For example:

- MUST the student study the subject?
- MUST the subject be studied by a student?

Based on the answers to the previous questions, it is possible to modify the previous sentences:

The student MUST study the course.

Course MAY be studied by a student.

When you determine the optionality it always depends on the context of the assignment. In some cases, the relationship is mandatory where else the participation of instances in a relationship is optional.

## 📝 2.2.3

Select the optionality which suites the best the relationship between two persons which represents the relation between the biological mother and her child.

- A person MUST be the biological mother of a child. A person MUST be the child of a biological mother.
- A person MAY be the biological mother of a child. A person MAY be the child of a biological mother.
- A person MAY be the biological mother of a child. A person MUST be the child of a biological mother.
- A person MUST be the biological mother of a child. A person MAY be the child of a biological mother.

## 📝 2.2.4

Select the optionality which suites the best relationship between receipt and product.

- A receipt MUST contain a product. A product MUST be an item of a receipt.
- A receipt MAY contain for a product. A product MAY be an item of a receipt.
- A receipt MAY contain for a product. A product MUST be an item of a receipt.
- A receipt MUST contain for a product. A product MAY be an item of a receipt.

## 📖 2.2.5

Cardinality describes how many times each instance of an entity can participate in a given relationship. To determine cardinality, it is advisable to ask questions such as:

- HOW MANY courses can one student study? One or more?
- HOW MANY students can study one course? One or more?

By answering the previous questions, it is possible to add the cardinality to the sentences in such a way that the optionality and cardinality will be clear:

The student must study ONE OR MORE courses.

The course can be studied by ONE OR MORE students.

Again, it is important to think that it always depends on the particular case. For different assignments, both cardinality and optionality may differ for the same entities.

## 📖 2.2.6

You have certainly noticed that the relationship has two ends. For each end, there is optionality, but also cardinality. There are three types of cardinality:

- 1:1 - the instances of both entities enter the relationship at most one at a time,
- 1:N - an instance of one entity enters into a relationship at most once, whereas an instance of the other entity can enter the relationship multiple times,
- M:N - Instances of both entities can enter the relationship multiple times.

## 2.2.7

Select the cardinality which suites the best relationship between two persons which represents the relation between the biological mother and her child.

- 1:1
- 1:N (one biological mother to more children)
- 1:N (one child to more biological mothers)
- M:N

## 2.2.8

Select the cardinality which suites the best relationship between receipt and product.

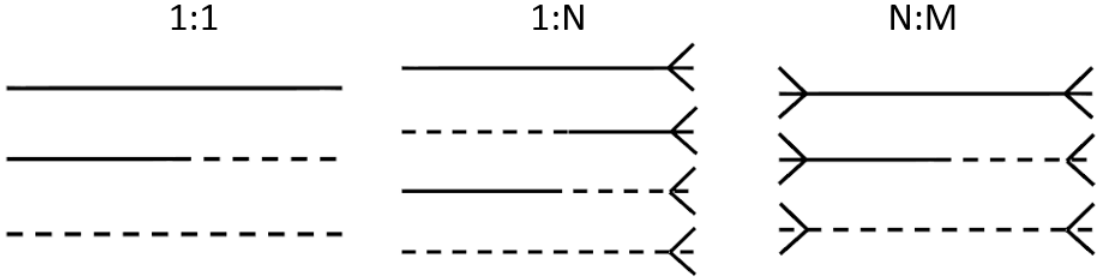
- 1:1
- 1:N (one receipt to more products)
- 1:N (one product to more receipts)
- M:N

## 2.2.9

As with entities and their attributes, there are rules for displaying relationships between them. Baker's notation will still be used.

Relationships appear as lines between entities. The optionality is either a solid line, which means the relationship is mandatory for an instance of the entity, or a dashed line, which determines that the relationship is optional for an instance of the entity.

Cardinality is then determined by a fork at the end of the line. If the line is ended by a "Crow's Foot" (fork), it means that multiple instances of the entity can enter the relationship. If the line terminates in a "single toe" (simple line), then the instance of the relationship can only participate once.



## 📖 2.2.10

ERDish is a language that describes relationships between entities in verbal terms. It helps to understand the modelling problem even for non-IT people. It serves as a prevention to avoid errors. With incomplete ERDish sentences, you have already met in the previous text. So how is an ERDish sentence formed?

1. EACH
2. Entity A
3. OPTIONALITY (must/may)
4. relationship name
5. CARDINALITY (one/one or more)
6. Entity B.

Remember, the relationship has two entities, so it will also need two ERDish sentences. Modifying the previous sentences into the desired form will create the following sentences:

Each student must study one or more courses.

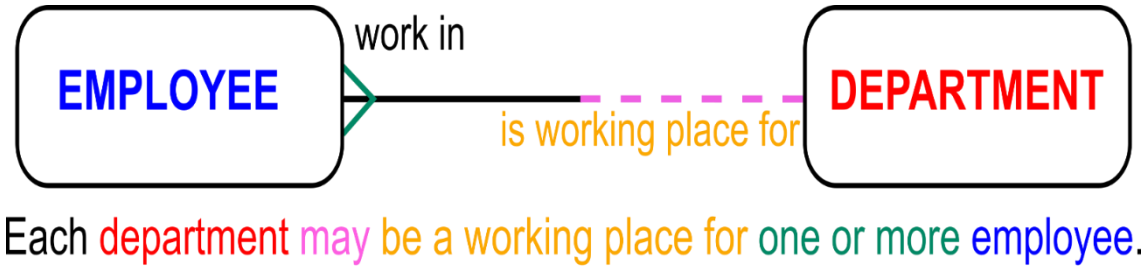
Each course may be studied by one or more students.

These two sentences then perfectly and unequivocally describe the relationship between course and student entities. The sentences thus formulated will be understood by everyone. They are understandable by anybody, no IT education is needed. Using these sentences, it is possible to clarify problems with the non-technical language.

## 📖 2.2.11

Whether an instance of an entity can or must participate in a given relationship is displayed at the end of a line that touches the entity. Cardinality is displayed at the far end of the line.

For a better understanding, the relationship between the employee and the department in which he/she works is analysed. The relationship is described by two ERDish sentences, and you should understand how the notation is properly drawn.

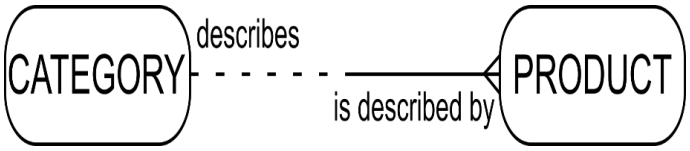



Each department may be a working place for one or more employee.


Each employee must work in one and only one department.

## 2.2.12

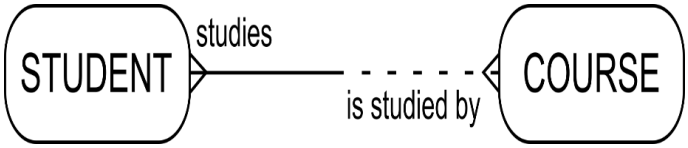
Choose ERDish sentences which suit the best to the relation in the following picture.




- Each category may describe one or more products. Each product must be described by one and only one category.
- Each category must describe one or more products. Each product may be described by one and only one category.
- Each category may describe one or more products. Each product must be described by one or more category.
- Each category may be described by one or more products. Each product must describe one and only one category.

## 2.2.13

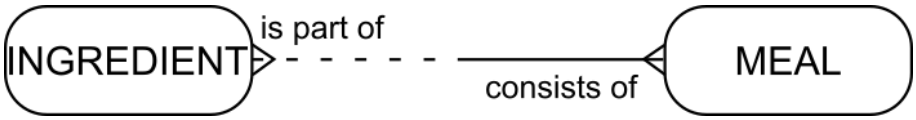
Choose ERDish sentences which suit the best to the relation in the following picture.



- Each student must study one or more courses. Each course may be studied by one or more students.
- Each student must study one or more courses. Each course may be studied by one and only one student.
- Each student may study one and only one course. Each course must be studied by one or more students.
- Each student may study one or more courses. Each course may be studied by one or more students.

## 2.2.14

Choos ERDish sentences which suit the best to the relation in the following picture.



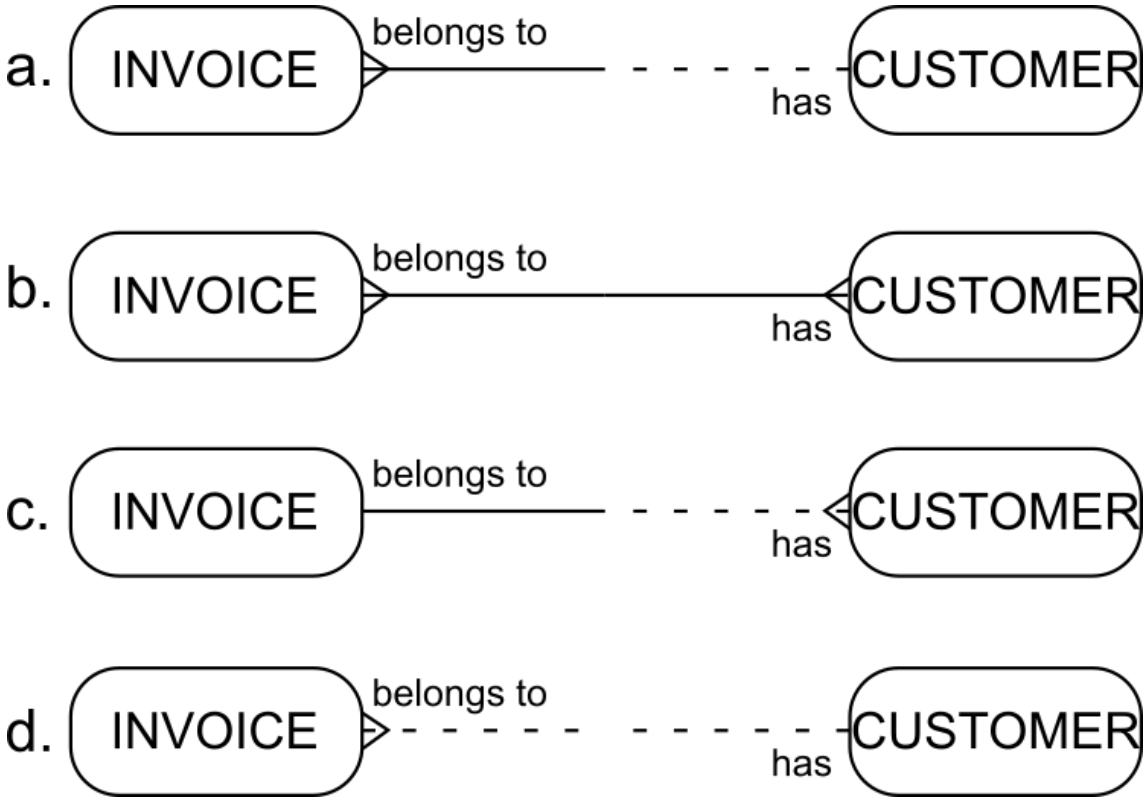

- Each ingredient may be part of one or more meals. Each meal must consist of one or more ingredients.
- Each ingredient must be part of one or more meals. Each meal may consist of one or more ingredients.
- Each ingredient may be part of one and only one meal. Each meal must consist of one and only one ingredient.
- Each ingredient must be part of one and only one meal. Each meal may consist of one and only one ingredient.

## 2.2.15

Choose a picture that suits the best to following ERDish sentences:

Each invoice must belong to one and only one customer.

Each customer may have one or more invoices.




- a.
- b.
- c.
- d.

## 2.2.16

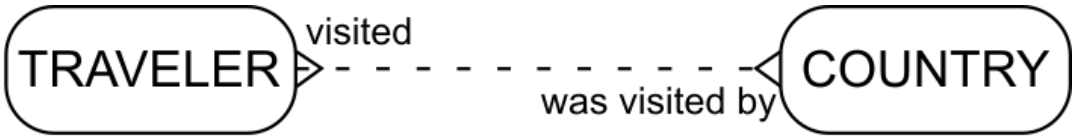
Choose a picture that suits the best to following ERDish sentences:

Each traveller may have visited one or more countries.

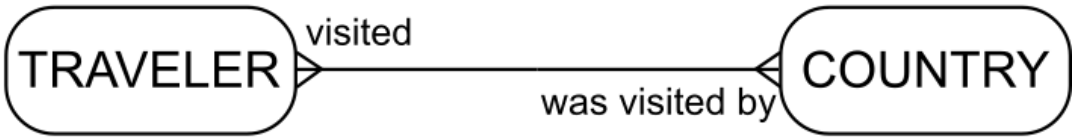
Each country may have been visited by one or more customers.

Select one:

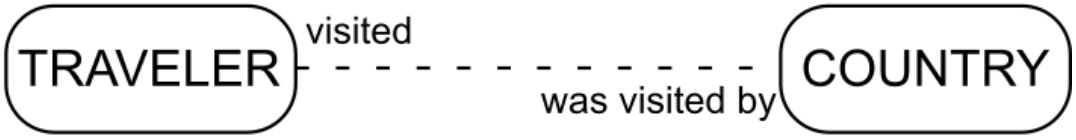
a.



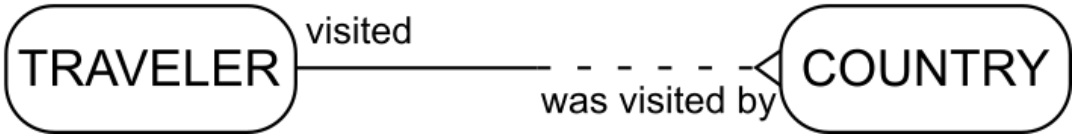

b.




c.




d.




- a.
- b.
- c.
- d.

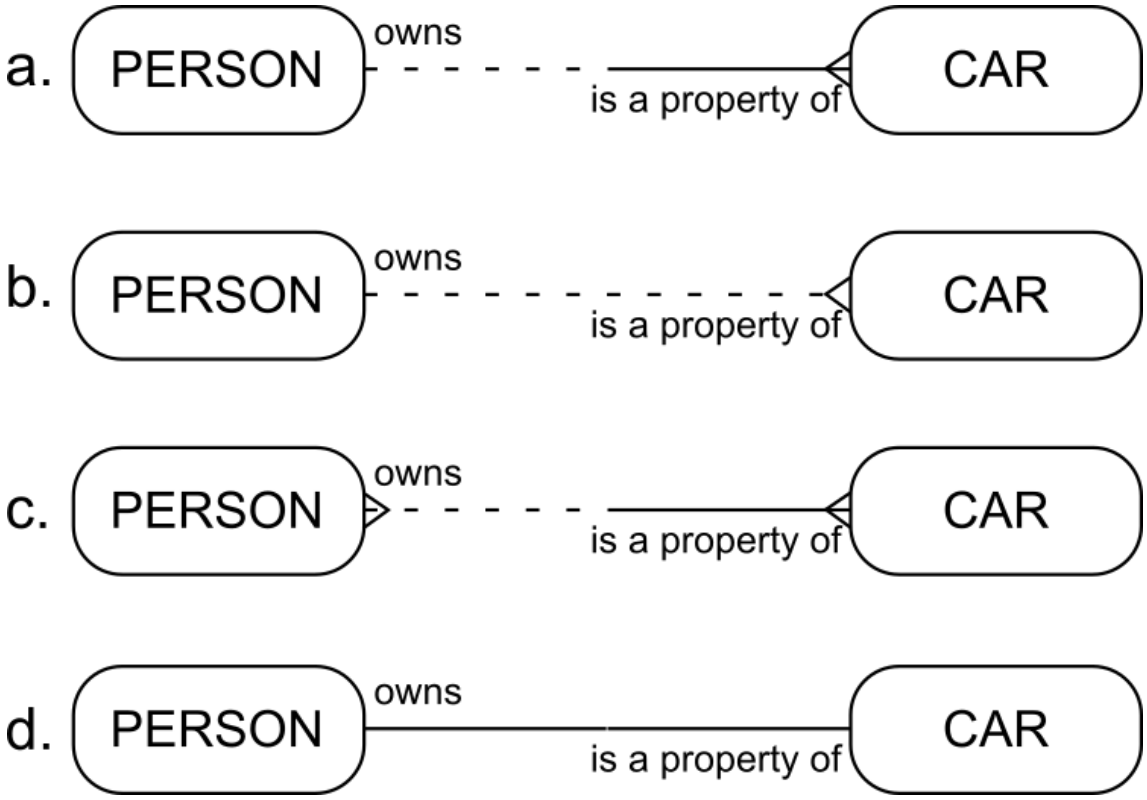
## 2.2.17

Choose a picture that suits the best to following ERDish sentences:

Each person may own one or more cars.

Each car must be the property of one and only one person.

Select one:




- a.
- b.
- c.
- d.

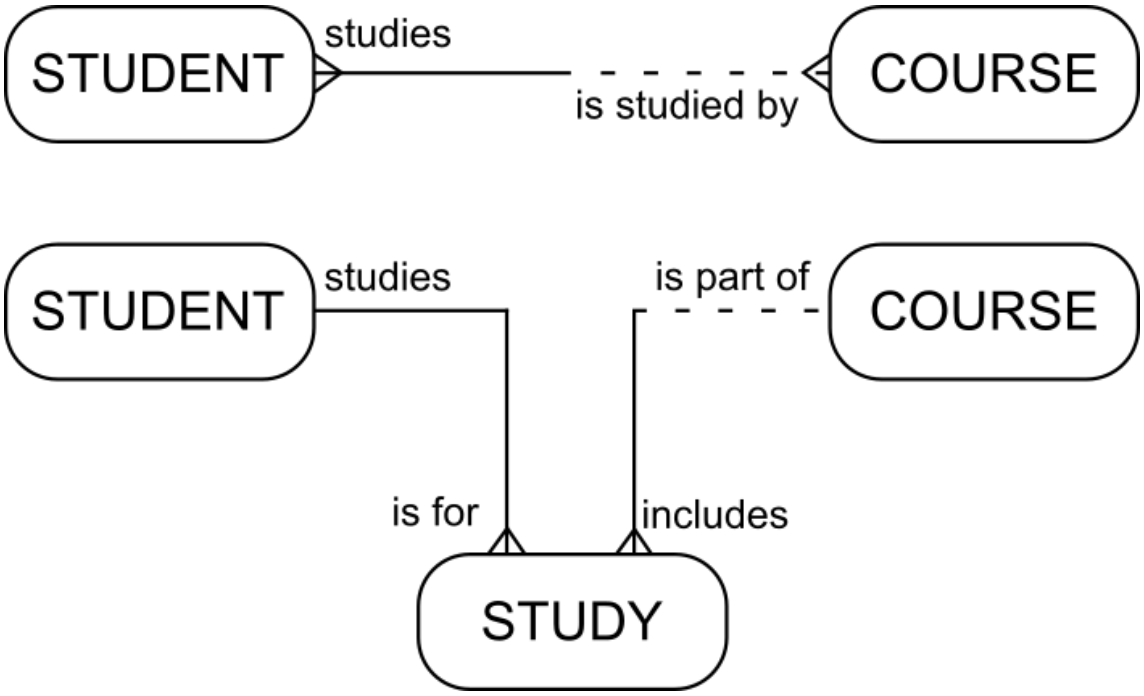
## 🕮 2.2.18

Decomposition of M:N relationship is a decomposition of this relation to two relations 1:N. This brings also a new so-called associative entity. It is not always necessary to perform decomposition within a conceptual model. You may remember that it can be solved by a logical model or even a physical model. When decomposition is good, it will be explained later. How decomposition is processed, will be illustrated in the relationship described by the following ERDish sentences:

Each student must study one or more courses.

Each course may be studied by one or more students.

You have already encountered this relationship in one of the questions. However, to be sure, once again it is illustrated in the following figure.

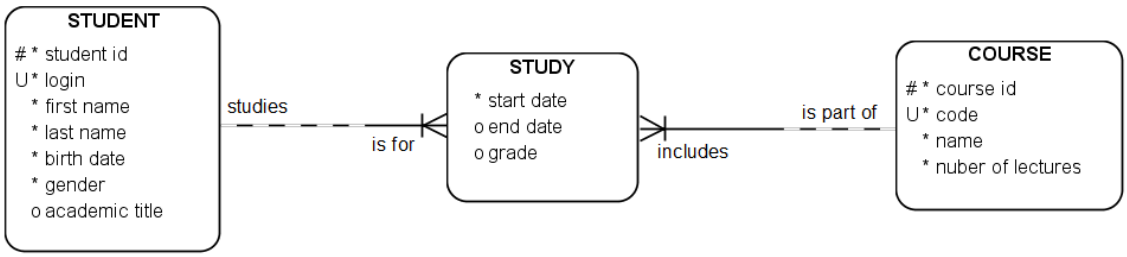



A newly created entity STUDY is referred to as an associated entity. As you can see, two new ones have emerged from one relationship. For an association entity, both relationships are mandatory. Without these relationships, it would lack its meaning. The optionality of the relationship remains the same for the original entities. With regard to cardinality, it is apparent that the Crow's Feet are in the newly formed association entity.

The attributes and primary identifiers are not addressed here. An association entity is also an entity, and so it must have its primary identifier. This usually consists of transmitted (foreign) identifiers.

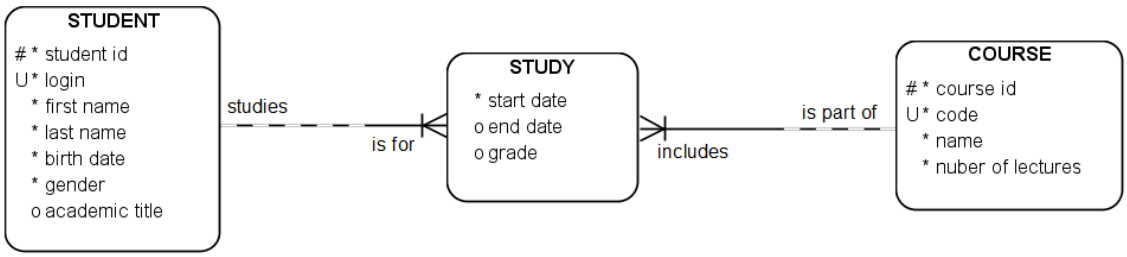
## 🕮 2.2.19

Let's now face the question of when to perform decomposition and when not? If you will do it always, you won't make a mistake. However, this is necessary if the associative entity has more attributes than just identifiers from the source entities. There is no need for decomposition if the purpose of the study is to just cover the fact that a specific course way studied by a specific student. However, if you want to include a grade student received for an exam, decomposition is necessary. An example of the necessary decomposition is shown in the following figure.




## 📖 2.2.20

Let’s have a look at the previous picture one more time.



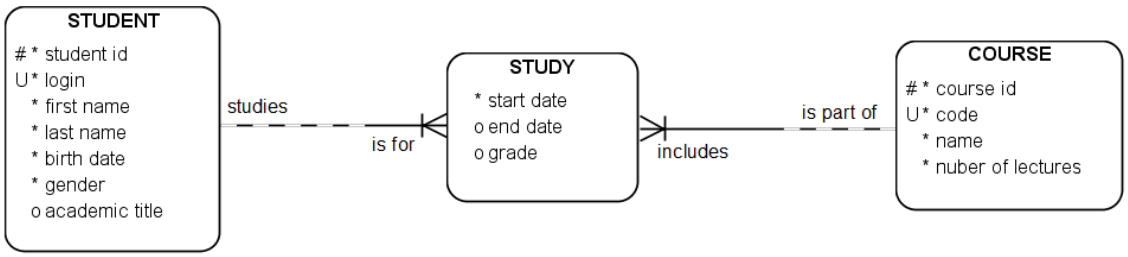

Relationships always have two entities. However, these entities are not entirely equal. One is in the source position and the other in the target position. This means that the so-called foreign identifier is transferred from the source entity into the target entity. In the figure, the source entity is the STUDENT entity and the target the entity STUDY. STUDY is the target entity even for the second relationship.

That means that the primary identifiers of the source entities are transferred to the target entity. However, they will be called foreign identifiers within the STUDY entity because they will identify the instance of another (foreign) entity. These attributes are no longer written into the rectangle because they are represented by a relation.

For 1:N relationships, the source, and target entities can be recognized easily. The source does not carry a fork while the target has it. At 1:1 relationship it is not visible at first glance, but you will soon recognize it by gaining experience. With the relation M:N, it is no longer meaningful to talk about the source and target entities, because this relationship breaks down into two relationships 1:N.

## 📖 2.2.21

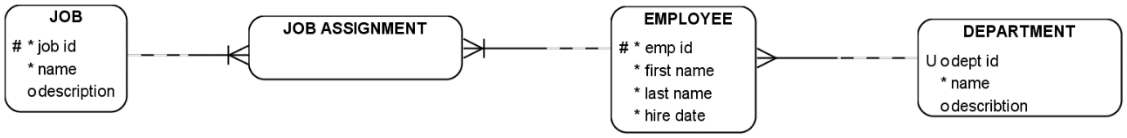
Another problem is that the STUDY entity doesn't seem to have a primary identifier.




But even this is marked in the diagram. The primary identifier will be composed of two attributes. It will be just the two that don't appear (student id and course id). The fact that these foreign identifiers become part of the primary identifier can be identified by the bars in front of the fork.

### 2.2.22

Which entity in the figure doesn't have a correctly marked primary identifier?




- DEPARTMENT
- EMPLOYEE
- JOB
- JOB ASSIGNMENT

## 2.3 Normal forms

### 2.3.1

Normal forms are a set of rules that help ensure the correct structure of a data model. The goal is to reduce redundancy and data dependencies, which makes it easier to modify, for example. The process of modifying the data model according to these rules is called normalization. Usually, normalization takes place up to the third normal form. However, there are more normal forms and the most important ones will be described here.

### 2.3.2

The data model is in the first normal form (1NF) if all attributes are atomic (indivisible). What is indivisible is probably best understood in the following example.

Imagine that you are the owner of the store and want to register the products sold. The easiest way to do this would be to create a sale entity in the form you see in the following figure.


At first glance, there is no problem to see. Violation of normal forms is evident only on the specific values of attributes that instances of this entity acquire. Let the following table be an example.

| id | sale time | total cost | products |
|---|---|---|---|
| 1 | 2. 4. 2019 8:01 | 150 | table, chair |
| 2 | 2. 4. 2019 8:04 | 300 | sofa, chair |
| 3 | 2. 4. 2019 8:06 | 1180 | table, sofa, refrigerator |

From this table, it is quite clear that the attribute products are not atomic and therefore the 1st normal form is violated. This imperfection needs to be removed.

## 🕮 2.3.3

In this figure, you see the modified entity sale so that all attributes remain atomic.

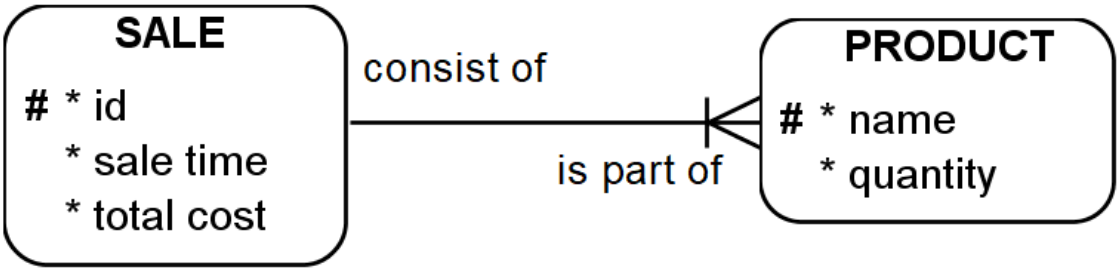

If this model is used, the table of the entity is transformed into the following form.

| id | sale time | total cost | product 1 | product 2 | product 3 |
|---|---|---|---|---|---|
| 1 | 2. 4. 2019 8:01 | 150 | table | chair | |
| 2 | 2. 4. 2019 8:04 | 300 | sofa | chair | |
| 3 | 2. 4. 2019 8:06 | 1180 | table | sofa | refrigerator |

Although all attributes are in this case atomic, it is certainly not the correct solution. How many such attributes will be needed? Will you limit customers by purchasing a maximum of three (ten, a hundred,…) products? A repeatable attribute is a totally unsuitable way to deal with the 1 st normal form. According to some authors, this is also understood as a violation of the 1NF.

## 🕮 2.3.4

The following figure will present the correct solution. Even though it is not how you would model this in reality, it fulfils the first normal form.

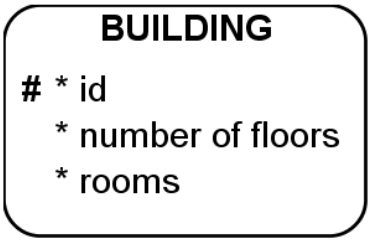



Sale can have as many products as needed. The product has a primary identifier composed of sale id and name. An even better solution could be seen in the next figure.


The total cost can be calculated based on the price of a product and its quantity.

## 2.3.5

Which attribute is the reason that this entity is not in 1st normal form?




- id
- number of floors
- rooms
- There is no attribute that breaks the 1st normal form.

## 2.3.6

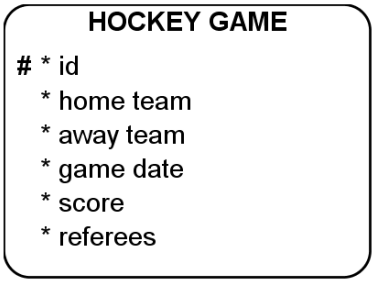
Which attribute is the reason that this entity is not in 1st normal form?


- id
- first name
- last name
- birth date
- address

- job
- children

## 2.3.7

Which attribute is the reason that this entity is not in 1st normal form?

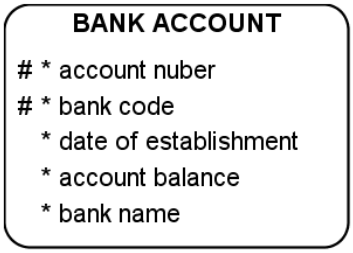



- id
- home team
- away team
- game date
- score
- referees

## 2.3.8

The second normal form

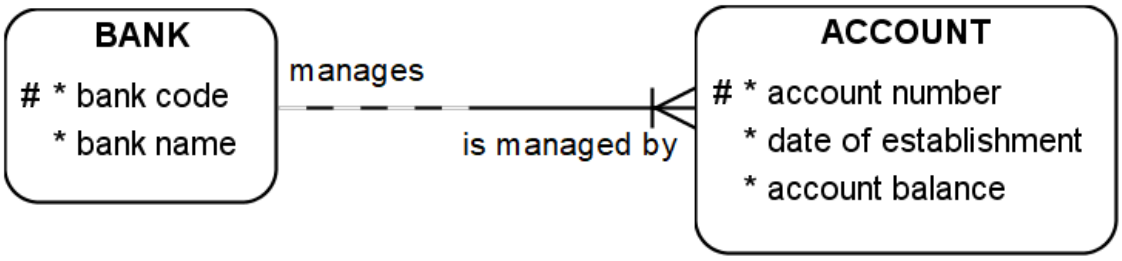
To meet the second normal form (2NF), the first normal form needs to be met. Furthermore, each non-key attribute (not part of the identifier) must be fully dependent on an identifier. The model must contain no partial dependencies of non-key attributes on the key (each non-key value must depend on the entire key). This is best understood in the following example.




The bank name attribute is not dependent on the entire primary identifier, but only on the bank code. This is what violates 2NF.

## 2.3.9

The second normal form can be easily achieved by fulfilling 1NF and introducing simple identifiers. To see how to resolve your bank account problem, see the following figure.



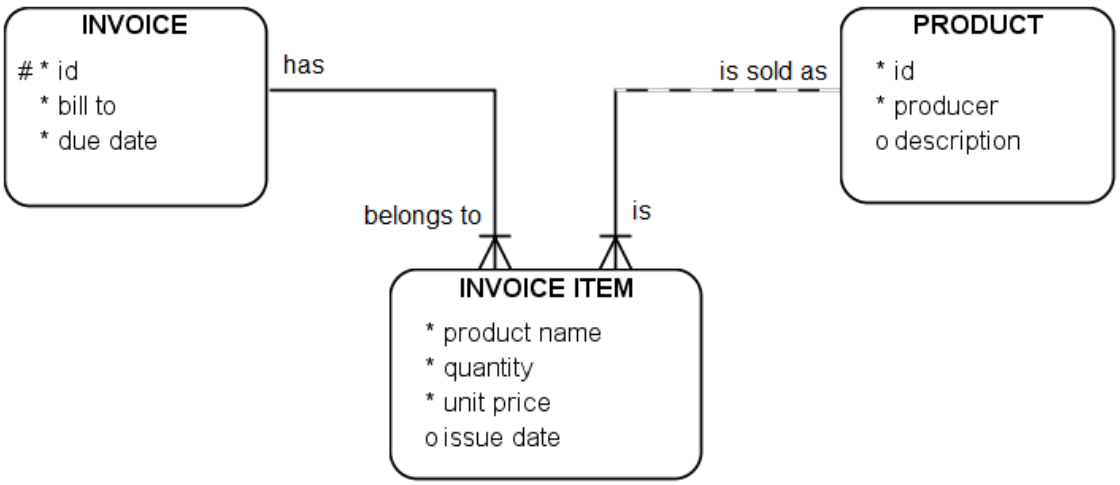

## 2.3.10

Select one or more attributes that violate the second normal form in the following figure.


- name
- description
- duration
- date of event
- title
- artist

## 2.3.11

Select one or more attributes that violate the second normal form in the following figure.




- product name
- description
- bill to
- issue date
- quantity
- unit price

## 🕮 2.3.12

To ensure the third normal form the second normal form must be fulfilled. Third Normal Form prohibits transitive dependencies. A transitive dependency exists when any attribute in an entity is dependent on any other non-UID attribute in that entity. For clarity, everything will be explained in the following example. You have the task of registering orders and at the same time needing information about the warehouse that handles the order, so you create the following entity.

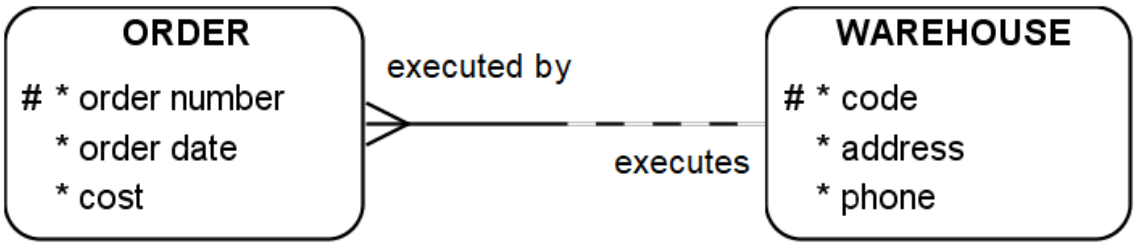

But this proposal is not right. It could be argued that the address is not atomic, so it does not meet even 1NF. But this is not a problem that should bother you now.

Think about what if a phone number changes, for example? Then you have to change this information everywhere. The violation of the third normal form is due to the attribute dependency of the warehouse address and the warehouse telephone

number on the warehouse code, which is dependent on the order number. Dependence between address (phone number) is not directly on the order number (non-sensitive). This is a problem that needs to be eliminated.

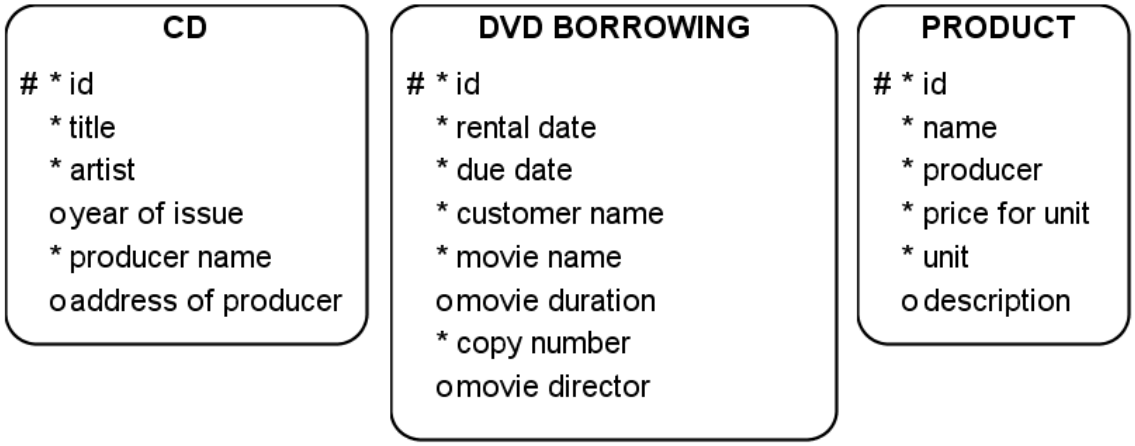
## 2.3.13

The solution will surely come to your mind. By creating a separate entity (warehouse) transitive dependency will disappear, as shown in the following figure.




## 2.3.14

Choose all entities which break the third normal form.

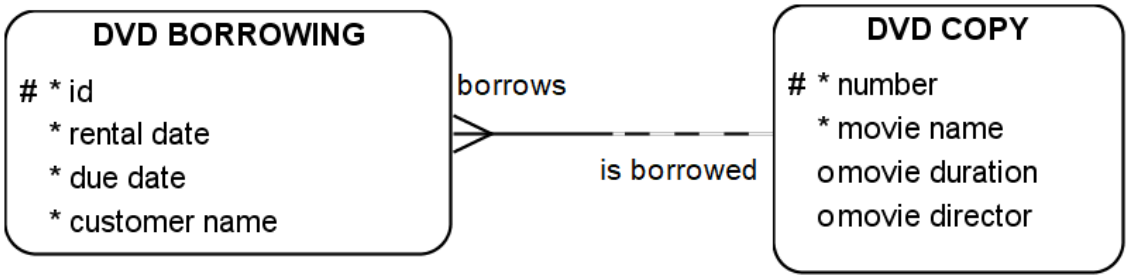



- CD
- DVD BORROWING
- PRODUCT

## 2.3.15

In the previous question, the entity DVD BORROWING did not fulfil the third normal form.


Is this solution correct, so there is no violation of 3NF?




- True
- False

### 🕮 2.3.16

There are other levels of normalization, but this study material will not include them. If you are interested in this topic you must search for it yourself.

## 2.4 Physical model

### 🕮 2.4.1

The physical model determines how data will be stored in a relational database. The conceptual and physical models are very similar. In many cases, the terminology used in the conceptual and physical models is often confused. The following table may be understood as a dictionary between models.

| Conceptual | Physical |
|---|---|
| entity | table |
| instance | row |
| attribute | column |
| primary identifier | primary key |
| secondary identifier | unique key |
| relationship | foreign key |

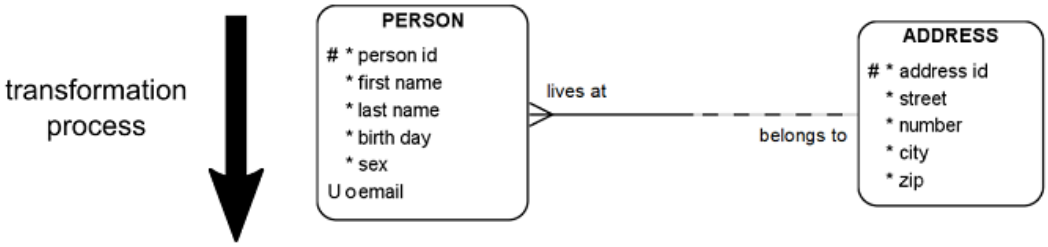
## 📖 2.4.2

Within the relational database, all data will be stored in tables. Each table is horizontally divided into rows and vertically on columns. The columns are named and the column name must be unique within the same table. Certain rules apply to name tables and columns. These may vary across database systems. Usually, you can use letters, numbers, and some special characters such as _ (underscore). The name should begin with a letter. The maximum name length is also limited.

## 📖 2.4.3

The notation of the physical model is logical. The first line is the name of the table. Surely you have noticed that entities are usually used the singular nouns. For table names, it is customary to use plural nouns.

### Conceptual model (ERD)




### Physical model of relational database

| persons | | | |
|---|---|---|---|
| key type | optionality | column name | data type |
| PK | * | persons_id | int |
| | * | first_name | text |
| | * | last_name | text |
| | * | birth_day | text |
| | * | sex | text |
| UK | o | email | text |
| FK | * | adresses_id | int |

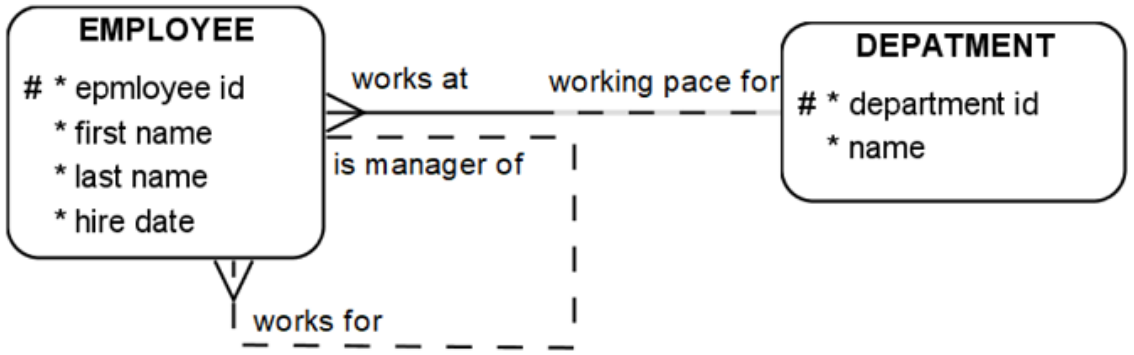
| adresses | | | |
|---|---|---|---|
| key type | optionality | column name | data type |
| PK | * | adresses_id | int |
| | * | street | text |
| | * | number | int |
| | * | city | text |
| | * | zip | text |

The table itself describing the table contains three columns: key type (pk, uk, fk), optionality (*, o), column name. In some cases (one in the picture), a fourth column is added to specify the data type. The data type specifies what values a given column can hold. Data type names vary by database system. Column names remain in most cases. Spaces are traditionally replaced by an underscore.

## 2.4.4

From relationships modelled in ERD, the physical model becomes foreign keys that must be entered as columns (sometimes multiple columns) of the table.

Depending on the optionality of the relationship, the optionality of the relevant column is given.




| employees | | |
|---|---|---|
| key type | optionality | column name |
| PK | * | emploeeys_id |
| | * | first_name |
| | * | last_name |
| | * | hire_date |
| FK | o | manager_id |
| FK | * | departments_id |

| departments | | |
|---|---|---|
| key type | optionality | column name |
| PK | * | departments_id |
| UK | * | name |

The optionality of relationship in a relational database can only be enforced on one side of the constraint (the one that is projected as a new column).

## 2.4.5

Match the terms between conceptual and physical models.

primary identifier _____

instance _____

relationship _____

secondary identifier _____

attribute _____

entity _____

- row
- foreign key
- secondary key
- table
- column
- primary key

## 2.4.6

The name of any column:

- must be unique within the whole database.
- must be unique within the table where it belongs.
- can be used many times even within the same table.

## 2.4.7

Each table is horizontally divided into

## 2.4.8

Each table is vertically divided into

## 2.4.9

The optionality of the relationship in a relational database:

- can only be enforced on one side of the constraint.
- can be enforced on both sides of the constraint.
- cannot be enforced on either side of the constraint.

## 2.4.10

To uniquely identify each row from others the table should have:

## 2.4.11

If a column is marked as a unique key, it must be mandatory.

- True
- False

## 2.4.12

For each table, there can be only one primary key.

- True
- False

# Structured Query Language

Chapter 3

# 3.1 SQL

## 🕮 3.1.1

**Structured Query Language** (SQL) is a declarative language. It is also referred to as a non-procedural language. It is, therefore, a different type of language than Java, C++, Pascal and others, which are referred to as procedural. For declarative languages, the programmer describes what he/she wants to achieve, but he/she does not care how this will be achieved. SQL allows to define data structures, insert, delete, edit, read data, or assign access permissions.

In many database systems, there is also a procedural SQL extension that allows you to use elements such as conditions, cycles, and so on. However, this is not part of SQL.

## 🕮 3.1.2

Individual data processing operations can be divided into related areas. These areas correspond to the SQL language, which is further divided into:

- Data Definition Language (DDL),
- Data Manipulation Language (DML),
- Data Query Language (DQL),
- Data Control Language (DCL).

## 🕮 3.1.3

**Data Definition Language (DDL)** - a language used to work with data structures. Allows you to create, edit, and delete them. This category includes CREATE, ALTER, and DROP.

## 🕮 3.1.4

**Data Manipulation Language (DML)** - This language is used to edit data content. Allows you to insert, edit, and delete data. Includes INSERT, UPDATE, DELETE or MERGE statements.

## 📖 3.1.5

**Data Query Language (DQL)** - many authors do not see this naming convention. Usually, this part is included in the DML because data acquisition is part of the manipulation. However, this section is specific to other DML operations in that it does not affect data content but only allows data to be read. The most important command is SELECT. It also includes SHOW and EXPLAIN commands.

## 📖 3.1.6

**Data Control Language (DCL)** - This part of the SQL is used to control the database. On the contrary, some authors this part of SQL into smaller units. These are transaction management issues and access privileges specifications. Commands included in this category are GRANT, REVOKE, COMMIT, or ROLLBACK.

## 📝 3.1.7

SQL is a typical representative of procedural programming language.

- False
- True

## 📝 3.1.8

DML stands for:

- Data Manipulation Language
- Data Modification Language
- Data Markup Language
- Data Movement Language

## 📝 3.1.9

The MERGE statement belongs to:

- DDL
- DML
- DQL
- DCL

### 📝 3.1.10

Choose statements that belong to DCL.

- REVOKE
- EXPLAIN
- COMMIT
- ALTER
- INSERT

### 📝 3.1.11

Match categories of SQL and their statements.

**DCL** _____

**DDL** _____

**DQL** _____

**DML** _____

- DROP
- GRANT
- PUSH
- TAKE
- SELECT
- DELETE

## 3.2 DDL

### 🕮 3.2.1

Each column within a relational database must be specified by a certain data type. It is then limited by the range of valid values and the set of operations that can be performed. When creating a table, you must always specify a data type for each column.

The names of specific data types may vary by name for each relational database system. In principle, they can be divided into the following groups:

- character,
- numeric,
- time,
- binary,
- others.

In almost all, you can use **integer (int)** for integers, **text** for character strings, and **date** for date. There are many data types, but for now, these are enough.

## 📖 3.2.2

Creating a table is quite easy. The only problem in different database systems may be different names of data types that need to be specified when creating. The table creation command looks like this:

**CREATE TABLE table_name (specification of columns)**

The column specification is as follows:

**column_name data_type [DEFAULT default_value] [constraint]**

The default value is defined by the DEFAULT keyword followed by the value (expression, function) that is inserted when no other specific value is specified when inserting. The default value doesn't have to be specified.

## 📖 3.2.3

Five different types of restrictions are commonly used:

**NOT NULL** indicates that the column must not contain empty values. There must always be value.

**PRIMARY KEY** is used to define the primary table key. The table can have a maximum of one primary key. Although this is not required by the database, each table should have its primary key.

**UNIQUE** is a constraint that requires uniqueness. Can be used on one or more columns. This restriction can be used even if the column is not required. Two empty (undefined) values are not considered the same.

**FOREIGN KEY** is designed to determine referential integrity. When creating a foreign key, you must specify the name of the table and the column on which it is dependent.

*REFERENCES referenced_table (referenced_column)*

**CHECK** this constraint can control specific values. E.g. the age must be between 0 and 130 years. The restriction is defined by the logical expression (condition) that must be met.

Restrictions may apply to individual columns, but some may be applied over multiple columns. If a constraint is defined over multiple columns, it must be specified separately after the column enumeration.

## 🕮 3.2.4

| persons | | | |
|---|---|---|---|
| key type | optionality | column name | data type |
| PK | * | persons_id | int |
| | * | first_name | text |
| | * | last_name | text |
| | * | birth_day | text |
| | * | sex | text |
| UK | o | email | text |
| FK | * | adresses_id | int |

| adresses | | | |
|---|---|---|---|
| key type | optionality | column name | data type |
| PK | * | adresses_id | int |
| | * | street | text |
| | * | number | int |
| | * | city | text |
| | * | zip | text |

```
CREATE TABLE addresses (
 addresses_id int NOT NULL PRIMARY KEY,
 street text NOT NULL,
 number int NOT NULL,
 city text NOT NULL,
 zip text NOT NULL
);
CREATE TABLE persons (
 persons_id int NOT NULL PRIMARY KEY,
 first_name text NOT NULL,
 last_name text NOT NULL,
 birth_day date NOT NULL,
 sex text NOT NULL,
 email text UNIQUE,
 addresses_id int NOT NULL REFERENCES addresses (addresses_id)
);
```

## 3.2.5

If you would copy the previous commands, they would probably end up with an error. In some database systems, the "number" is a name of data type, and so cannot be used for the name of the column. The use of a unique key in combination with a "text" data type could also cause a problem.

## 3.2.6

Based on the model in the picture, you need to ensure that the column that the name of any department will differ from any other departments' name.

| employees | | |
|---|---|---|
| **key type** | **optionality** | **column name** |
| PK | * | emploeeys_id |
| ??? | * | first_name |
| ??? | * | last_name |
| ??? | * | hire_date |
| ??? | * | dept_id |

| departments | | |
|---|---|---|
| **key type** | **optionality** | **column name** |
| PK | * | departments_id |
| ??? | * | name |

Which type of constraint has to be used?

- NOT NULL
- UNIQUE
- CHECK
- FOREIGN KEY

## 3.2.7

Based on the model in the picture, you need to ensure that the column dept_id will contain only values which are stored as primary keys of the table departments.

| employees | | |
|---|---|---|
| **key type** | **optionality** | **column name** |
| PK | * | emploeeys_id |
| ??? | * | first_name |
| ??? | * | last_name |
| ??? | * | hire_date |
| ??? | * | dept_id |

| departments | | |
|---|---|---|
| **key type** | **optionality** | **column name** |
| PK | * | departments_id |
| ??? | * | name |

Which type of constraint has to be used?

- NOT NULL
- UNIQUE
- CHECK
- FOREIGN KEY

## 3.2.8

Based on the model in the picture, you need to ensure that the column hire_date is mandatory.

| employees | | |
|---|---|---|
| **key type** | **optionality** | **column name** |
| PK | * | emploeeys_id |
| ??? | * | first_name |
| ??? | * | last_name |
| ??? | * | hire_date |
| ??? | * | dept_id |

| departments | | |
|---|---|---|
| **key type** | **optionality** | **column name** |
| PK | * | departments_id |
| ??? | * | name |

Which type of constraint has to be used?

- NOT NULL
- UNIQUE
- CHECK
- FOREIGN KEY
- REFERENCES
- PRIMARY KEY

## 3.2.9

The implicit value is defined by *a keyword* followed by the value (expression, function) that is inserted when no other specific value is specified when inserting. Which keyword it is?

- IMPLICIT
- DEFAULT
- REQUIRE
- REFERENCES
- IS

## 🕮 3.2.10

When the change of table structure is needed the ALTER statement has to be used. By using this statement, it is possible to:

- add a new column,
- modify a column,
- define the implicit value to a column,
- drop a column,
- add integrity constraint.

## 🕮 3.2.11

Adding a new column

```
ALTER TABLE table_name
 ADD (column_name data_type
       [default_value][constraint])
```

It is possible to add more than one column by one ALTER statement. Between each column definition, a comma needs to be used.

## 🕮 3.2.12

When modifying a column, it is possible to do the following:

- Extend the length or precision of a numeric type.
- Increase the maximum length for string types.
- The maximum length can be reduced depending on the data stored. If the table already contains some records, it is not possible to modify the data types so that existing records do not meet them.
- Change the data type if there are no values other than NULL in the column.
- Change the default value. However, this only affects newly inserted records.

```
ALTER TABLE table_name
 MODIFY (column_name new_definition)
```

## 🕮 3.2.13

Drop a column:

```
ALTER TABLE table_name DROP COLUMN column_name;
```

It is possible to drop columns with or without values. Only one column can be deleted during one operation. At least one column must always remain in the table. You cannot retrieve dropped data when the column is dropped.

## 🕮 3.2.14

The restrictions that can be added in this way are PRIMARY KEY, UNIQUE, FOREIGN KEY, and CHECK. To add NOT NULL constraints, you need to modify the column as already mentioned.

```
ALTER TABLE table_name
 ADD CONSTRAINT constraint_name
    constraint_type(column);
```

For some constraints, there can be more than one column.

## 🕮 3.2.15

Removing a constraint is similar to removing a column.

```
ALTER TABLE table_name DROP CONSTRAINT constraint_name;
```

## 🕮 3.2.16

If you need to change the name of a table, it is necessary to use the RENAME statement.

```
RENAME old_name TO new_name;
```

## 🕮 3.2.17

To get rid of the whole table there is the statement DROP.

```
DROP TABLE table_name
```

If there are some dependencies (FOREIGN KEY) between tables it might be not possible to drop a particular table. If you would want to drop the table together with all tables which depends on the dropped table you can use the keyword CASCADE.

```
DROP TABLE table_name CASCADE
```

In some database systems, it can be a little bit different. You may need to use CASCADE CONSTRAINTS. If any problem arises, see the documentation.

## 3.2.18

You need to add a FOREIGN KEY between tables employees and departments.

| employees | | |
|---|---|---|
| **key type** | **optionality** | **column name** |
| PK | * | emploeeys_id |
| | * | first_name |
| | * | last_name |
| | * | hire_date |
| | * | dept_id |

| departments | | |
|---|---|---|
| **key type** | **optionality** | **column name** |
| PK | * | departments_id |
| | * | name |

Which of the following possibilities make sense the most?

- ALTER TABLE departments ADD CONSTRAINT dept_FK FOREIGN KEY (departments_id) REFERENCES employees(dept_id)
- ALTER TABLE employees ADD CONSTRAINT dept_FK FOREIGN KEY (dept_id) REFERENCES departments (departments_id)
- ALTER TABLE employees ADD CONSTRAINT dept_FK FOREIGN KEY (employees_id) REFERENCES departments (departments_id)
- ALTER TABLE departments ADD CONSTRAINT dept_FK FOREIGN KEY (departments_id) REFERENCES employees (employees_id)

## 3.2.19

If you need to add a new mandatory column *gender* to the table employees, which of the following possibilities make sense the most?

| employees | | |
|---|---|---|
| **key type** | **optionality** | **column name** |
| PK | * | emploeeys_id |
| | * | first_name |
| | * | last_name |
| | * | hire_date |
| | * | dept_id |

| departments | | |
|---|---|---|
| **key type** | **optionality** | **column name** |
| PK | * | departments_id |
| | * | name |

Suppose that the table employee is empty.

- ALTER TABLE employees ADD (gender text NOT NULL)
- ALTER TABLE departments ADD (gender text NOT NULL)
- ALTER TABLE employees ADD gender text
- ALTER TABLE employees ADD gender date NOT NULL

## 3.2.20

To change a column name of a table a RENAME statement must be used.

- False
- True

## 3.2.21

Statement:

```
DROP TABLE employees
```

will always remove the table employees from the database independently of other circumstances.

- False
- True

# Select Command

Chapter 4

# 4.1 SELECT Statement - The Basics

## 📖 4.1.1

SELECT Statement

The SELECT statement is the most common SQL statement. It searches for records that meet the conditions that we define. The SELECT statement belongs to a DML group. It does not change the data stored in the tables. It only shows the data that met the conditions.

## 📖 4.1.2

SELECT Statement - The Simplest Form

The SQL SELECT statement must always contain the SELECT and FROM clauses. Their order cannot be changed. Other clauses of the SQL command are optional and will be introduced later. The simplest SQL statement that returns all records from a single table is then as follows

```
SELECT * FROM employees;
```

## 📝 4.1.3

Each SQL SELECT statement must contain the following keywords:

- SELECT
- FROM
- ORDER BY
- DISTINCT
- TABLE

## 📖 4.1.4

SELECT Statement - Columns Selection

The list of table columns, which should be displayed in the query result, must be written between the SELECT and FROM statements. The following rules should be followed:

- if there are only some columns in the result, they should be listed sequentially and separated by a comma, without a comma after the last column name,
- if all columns of the table are to be included in the result, simply sign * can be used instead of listing them.

The order in which the columns appear in the query will match the order in which the columns appear. The following query returns the firstname, lastname and salary of the employees.

```
SELECT firstname, lastname, salary FROM employees;
```

Columns can be combined using operators. E.g. add two numeric columns or concatenate text columns. You can use the AS keyword to rename the column. On some database systems, the AS keyword is optional. It is also possible to use parentheses in expressions.

```
SELECT salary + bonus AS together FROM employees;
```

## 4.1.5

Mark the correct answer about the right order of keywords in the SELECT query.

- The keyword SELECT must appear before keyword FROM.
- The keyword FROM must appear before keyword SELECT.
- The order of the keywords is not important.

## 4.1.6

Statement SELECT DISTINCT

The requirements to show only unique values as well as to remove duplicates are very often in practice. It can be easily realized by the SELECT DISTINCT statement. The following example shows how to use this statement.

```
SELECT DISTINCT column FROM table
```

However, it is important to know, that the statement SELECT DISTINCT is not a function. Therefore, it does not require to enclose attributes in brackets.

### 📝 4.1.7

Which keyword can be used to remove duplicate entries from the selected attributes?

- CONSTRAINT
- DISTINCT
- DIFFERENT
- ONLY

## 4.2 SELECT Statement - ORDER BY Clause

### 📖 4.2.1

ORDER BY Clause - Single Column Ordering

The records that return a SQL SELECT query can be ordered by the values of any column using the ORDER BY clause. Therefore, the ORDER BY clause is always at the end of the query.

```
SELECT * FROM employees ORDER BY firstname;
```

Unless otherwise specified, the values in the column in this clause are sorted in ascending order (A-Z), more precisely:

- numeric values are ordered from smallest to largest,
- date values are ordered from oldest to newest,
- text values are ordered in alphabetical order.

### 📝 4.2.2

The result of the query can be ordered only by the columns, which are written in the SELECT clause. True or false?

- FALSE
- TRUE

### 📖 4.2.3

ORDER BY Clause - Ordering Direction

The predefined direction for ordering the column values specified in the ORDER BY clause is the ascending direction. Therefore, there is no need to specify the ASC operator after the column that provides this direction in SQL.

```
SELECT firstname, lastname, salary FROM employees ORDER BY lastname;
```

However, if it is necessary to order the values in descending order (Z-A), it is necessary to enter the DESC operator after the column name.

```
SELECT firstname, lastname, salary FROM employees ORDER BY lastname DESC;
```

### 4.2.4

Create the correct query by reordering the clauses below. The query should return all employees who work in Bratislava ordered by their salaries from the highest to the lowest.

- FROM employees
- WHERE city = 'Bratislava'
- SELECT firstname, lastname, salary
- ORDER BY salary DESC;

### 4.2.5

ORDER BY Clause - Multiple Column Ordering

The SQL query result can be ordered by multiple columns. Each column can be ordered in ascending or descending order using the ASC and DESC operators. The order of the columns in the ORDER BY clause determines the order in which the records will be ordered in the result.

```
SELECT firstname, lastname, salary FROM employees ORDER BY salary DESC, lastname ASC;
```

## 4.3 SELECT Basics (execises)

### 4.3.1 Get all data from countries

Select all fields from the table Countries

Data Set 1

**Countries**

```
Id  Country    Capital
```

Data Set 2

**Countries**

```
Id  Country      Capital
1   Portugal     Lisbon
2   Belgium      Brussels
3   Spain        Madrid
4   Hungary      Budapest
8   Croatia      Zagreb
9   Greece       Athens
10  Malta        Valletta
```

Data Set 3

**Countries**

```
Id  Country      Capital
3   Spain        Madrid
4   Hungary      Budapest
6   Slovakia     Bratislava
7   Norway       Oslo
8   Croatia      Zagreb
9   Greece       Athens
10  Malta        Valletta
11  Italy        Rome
```

Data Set 4

**Countries**

```
Id  Country      Capital
3   Spain        Madrid
5   Poland       Warsaw
8   Croatia      Zagreb
10  Malta        Valletta
11  Italy        Rome
```

Data Set 5

**Countries**

```
Id  Country      Capital
6   Slovakia     Bratislava
7   Norway       Oslo
8   Croatia      Zagreb
10  Malta        Valletta
11  Italy        Rome
```

### 4.3.2 List of countries

Select country and capital from the table Countries

Data Set 1

**Countries**

```
Id  Country    Capital
```

Data Set 2

**Countries**

```
Id  Country      Capital
1   Portugal     Lisbon
2   Belgium      Brussels
3   Spain        Madrid
4   Hungary      Budapest
8   Croatia      Zagreb
9   Greece       Athens
10  Malta        Valletta
```

Data Set 3

**Countries**

```
Id  Country      Capital
3   Spain        Madrid
4   Hungary      Budapest
6   Slovakia     Bratislava
7   Norway       Oslo
8   Croatia      Zagreb
9   Greece       Athens
10  Malta        Valletta
11  Italy        Rome
```

Data Set 4

**Countries**

| Id | Country | Capital |
|---|---|---|
| 3 | Spain | Madrid |
| 5 | Poland | Warsaw |
| 8 | Croatia | Zagreb |
| 10 | Malta | Valletta |
| 11 | Italy | Rome |

Data Set 5

**Countries**

| Id | Country | Capital |
|---|---|---|
| 6 | Slovakia | Bratislava |
| 7 | Norway | Oslo |
| 8 | Croatia | Zagreb |
| 10 | Malta | Valletta |
| 11 | Italy | Rome |

### 4.3.3 Get all data from employees

Select all fields from the table Employees

Data Set 1

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|

Data Set 2

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 2 | Jan | Nowak | RKB003 |
| 3 | Kamil | Wilmowski | RKB011 |
| 4 | Olga | Milenka | RKB003 |
| 5 | Angela | Wilga | RKB002 |

Data Set 3

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|

```
1    Jan      Kowalski     RKB012
3    Kamil    Wilmowski    RKB011
5    Angela   Wilga        RKB002
```

Data Set 4

**Employees**

```
Id   Name      Surname      JobTitleCode
1    Jan       Kowalski     RKB012
3    Kamil     Wilmowski    RKB011
4    Paco      Ramirez      RKB011
5    Angela    Wilga        RKB002
6    Beatriz   Santana      RKB002
```

Data Set 5

**Employees**

```
Id   Name      Surname      JobTitleCode
7    Joahim    Mesina       RKB001
8    Marta     Frantisek    RKB002
9    Jan       Matalau      RKB001
```

### 4.3.4 List of employees

Select Name and Surname from the table Employees

Data Set 1

**Employees**

```
Id   Name      Surname      JobTitleCode
```

Data Set 2

**Employees**

```
Id   Name      Surname      JobTitleCode
1    Jan       Kowalski     RKB012
2    Jan       Nowak        RKB003
3    Kamil     Wilmowski    RKB011
4    Olga      Milenka      RKB003
5    Angela    Wilga        RKB002
```

Data Set 3

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 3 | Kamil | Wilmowski | RKB011 |
| 5 | Angela | Wilga | RKB002 |

Data Set 4

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 3 | Kamil | Wilmowski | RKB011 |
| 4 | Paco | Ramirez | RKB011 |
| 5 | Angela | Wilga | RKB002 |
| 6 | Beatriz | Santana | RKB002 |

Data Set 5

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 7 | Joahim | Mesina | RKB001 |
| 8 | Marta | Frantisek | RKB002 |
| 9 | Jan | Matalau | RKB001 |

## 4.3.5 Countries - ordered

Select all fields from countries ordered by Country

Data Set 1

**Countries**

| Id | Country | Capital |
|---|---|---|

Data Set 2

**Countries**

| Id | Country | Capital |
|---|---|---|
| 1 | Portugal | Lisbon |
| 2 | Belgium | Brussels |
| 3 | Spain | Madrid |
| 4 | Hungary | Budapest |

```
8   Croatia      Zagreb
9   Greece       Athens
10  Malta        Valletta
```

Data Set 3

**Countries**

```
Id  Country      Capital
3   Spain        Madrid
4   Hungary      Budapest
6   Slovakia     Bratislava
7   Norway       Oslo
8   Croatia      Zagreb
9   Greece       Athens
10  Malta        Valletta
11  Italy        Rome
```

Data Set 4

**Countries**

```
Id  Country      Capital
3   Spain        Madrid
5   Poland       Warsaw
8   Croatia      Zagreb
10  Malta        Valletta
11  Italy        Rome
```

Data Set 5

**Countries**

```
Id  Country      Capital
6   Slovakia     Bratislava
7   Norway       Oslo
8   Croatia      Zagreb
10  Malta        Valletta
11  Italy        Rome
```

### ⌨ 4.3.6 Employees - ordered

Select the name and surname of employees and order them by name.

Data Set 1

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|

Data Set 2

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 2 | Jan | Nowak | RKB003 |
| 3 | Kamil | Wilmowski | RKB011 |
| 4 | Olga | Milenka | RKB003 |
| 5 | Angela | Wilga | RKB002 |

Data Set 3

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 3 | Kamil | Wilmowski | RKB011 |
| 5 | Angela | Wilga | RKB002 |

Data Set 4

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 3 | Kamil | Wilmowski | RKB011 |
| 4 | Paco | Ramirez | RKB011 |
| 5 | Angela | Wilga | RKB002 |
| 6 | Beatriz | Santana | RKB002 |

Data Set 5

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 7 | Joahim | Mesina | RKB001 |
| 8 | Marta | Frantisek | RKB002 |
| 9 | Jan | Matalau | RKB001 |

### 4.3.7 Unique countries

Select unique names of countries (every country should be on the list once).

Data Set 1

**Countries**

| Id | Country | City |
|---|---|---|

Data Set 2

**Countries**

| Id | Country | City |
|---|---|---|
| 1 | Portugal | Lisbon |
| 2 | Belgium | Brussels |
| 3 | Spain | Madrid |
| 4 | Croatia | Zadar |
| 8 | Croatia | Zagreb |
| 9 | Greece | Athens |
| 10 | Croatia | Split |

Data Set 3

**Countries**

| Id | Country | City |
|---|---|---|
| 3 | Spain | Madrid |
| 4 | Italy | Milan |
| 6 | Slovakia | Bratislava |
| 7 | Norway | Oslo |
| 8 | Slovakia | Nitra |
| 9 | Greece | Athens |
| 10 | Italy | Turin |
| 11 | Italy | Rome |

Data Set 4

**Countries**

| Id | Country | City |
|---|---|---|
| 3 | Spain | Madrid |
| 5 | Spain | Barcelona |
| 8 | Spain | Sevilla |
| 10 | Malta | Valletta |

```
11   Italy          Rome
```

Data Set 5

**Countries**

```
Id   Country        City
6    Slovakia       Bratislava
7    Slovakia       Nitra
8    Croatia        Zagreb
10   Malta          Valletta
11   Italy          Rome
```

## ⌨ 4.3.8 Unique names ordered by alphabet

Select Names from the table Employees to avoid repeating them. Order them by alphabet.

Data Set 1

**Employees**

```
Id   Name      Surname       JobTitleCode
```

Data Set 2

**Employees**

```
Id   Name      Surname       JobTitleCode
1    Jan       Kowalski      RKB012
2    Jan       Nowak         RKB003
3    Kamil     Wilmowski     RKB011
4    Olga      Milenka       RKB003
5    Angela    Wilga         RKB002
```

Data Set 3

**Employees**

```
Id   Name      Surname       JobTitleCode
1    Jan       Kowalski      RKB012
3    Kamil     Wilmowski     RKB011
5    Jan       Wilgo         RKB002
```

Data Set 4

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Paco | Kowalski | RKB012 |
| 3 | Kamil | Wilmowski | RKB011 |
| 4 | Paco | Ramirez | RKB011 |
| 5 | Angela | Wilga | RKB002 |
| 6 | Paco | Santana | RKB002 |

Data Set 5

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Joahim | Mesina | RKB001 |
| 2 | Marta | Frantisek | RKB002 |
| 3 | Jan | Matalau | RKB001 |
| 7 | Joahim | Mesana | RKB001 |
| 8 | Marta | Frantis | RKB002 |
| 9 | Jan | Matal | RKB001 |

## 4.3.9 Sum of tickets

Calculate and display the number of tickets for each theatre show if data about online and cash sales are stored in the table Shows. Set the name of the new column to "together".

Choose data about the name, count of tickets bought online, cash and the new column.

Data Set 1

**Shows**

| Id | Name | Online | Cash |
|---|---|---|---|

Data Set 2

**Shows**

| Id | Name | Online | Cash |
|---|---|---|---|
| 1 | Mackbeth | 100 | 20 |
| 2 | Hamlet | 53 | 48 |
| 3 | Romeo and Julia | 22 | 158 |
| 4 | War and Piece | 200 | 68 |
| 5 | Evgenij Onegin | 120 | 140 |

Data Set 3

**Shows**

| Id | Name | Online | Cash |
|---|---|---|---|
| 1 | Mackbeth | 1003 | 3510 |
| 2 | Hamlet | 538 | 9871 |
| 3 | Romeo and Julia | 220 | 897 |

Data Set 4

**Shows**

| Id | Name | Online | Cash |
|---|---|---|---|
| 1 | Mackbeth | 333 | 950 |
| 2 | Hamlet | 253 | 418 |
| 6 | Romeo and Julia | 122 | 158 |
| 4 | War and Piece | 222 | 1068 |
| 5 | Evgenij Onegin | 120 | 140 |
| 3 | Janosik | 303 | 698 |

Data Set 5

**Shows**

| Id | Name | Online | Cash |
|---|---|---|---|
| 1 | Mackbeth | 2502 | 103 |
| 2 | Hamlet | 2503 | 4108 |
| 6 | Romeo and Julia | 1202 | 1508 |
| 5 | Evgenij Onegin | 1200 | 1040 |
| 3 | Janosik | 3003 | 1698 |

## ⌨ 4.3.10 Actors ordered by the year of birth

Show all columns of actors ordered by the year of birth from the oldest to the youngest.

**actors**

| aID | firstname | lastname | year_of_birth | income |
|---|---|---|---|---|
| 1 | Woody | Hopkins | 1975 | 1995 |
| 2 | Sandra | Tandy | 1965 | 2001 |
| 3 | John | Torn | 1994 | 2015 |
| 4 | Natalie | Hunt | 1957 | 2002 |
| 5 | Juan | Pitt | 1978 | 1999 |

```
6    Henry        Nolte      1968            2005
7    Woody        Harris     1945            1984
```

### ⌨ 4.3.11 Incomming actors

Show firstname, lastname and income of all actors ordered by the income in descending order.

**actors**

```
aID    firstname    lastname   year_of_birth    income
1    Woody        Hopkins    1975            1995
2    Sandra       Tandy      1965            2001
3    John         Torn       1994            2015
4    Natalie      Hunt       1957            2002
5    Juan         Pitt       1978            1999
6    Henry        Nolte      1968            2005
7    Woody        Harris     1945            1984
```

### ⌨ 4.3.12 Unique countries of actors

Show the list of unique countries from which the actors come.

**actors**

```
aID    firstname    lastname   year_of_birth    income   country
1    Woody        Hopkins    1975            1995     Australia
2    Sandra       Tandy      1965            2001     France
3    John         Torn       1994            2015     Spain
4    Natalie      Hunt       1957            2002     France
5    Juan         Pitt       1978            1999     Australia
6    Henry        Nolte      1968            2005     United
Kingdom
7    Woody        Harris     1945            1984     Australia
```

## 4.4 SELECT Statement - Functions

### 🕮 4.4.1

Calculated columns

In addition to table attributes, a SELECT statement can return other columns that are not included in the original table. These columns are called calculated

columns/fields. Various calculations can be performed directly in the SQL command to get a different view of the data stored in the tables. The calculation does not only allow basic arithmetic operations, but also the use of functions and formulas. Expressions can be constants or values stored in columns. It may happen that an undefined/unknown value (NULL) appears in the expressions. If this unknown value is not the expected input of a specific function that works with an undefined value, then the result of the entire expression is NULL. The same rules as in other applications for writing formulas and functions must be applied in this case.

## 🕮 4.4.2

Alias

All columns used in the SELECT query must have a unique name. The same must be also applied to all kinds of calculated columns. The SQL keyword AS allows giving the unique and eloquent name to the newly created columns.

The following query shows how to rename the columns easily.

```
SELECT firstname AS forename, lastname AS surname FROM
employees;
```

The alias can be also used for renaming the tables. This approach will be shown later in the chapter, which deals with multi-table queries. The keyword AS can be omitted in this case. The alias is written after the name of the table. The example shows how to rename the table employees to our_company.

```
SELECT firstname, lastname FROM employees our_company;
```

## 🕮 4.4.3

Types of SQL Row Functions

The SELECT clause can contain different kinds of calculated columns. As was mentioned earlier, besides these columns can besides the arithmetic operations contain also functions, which can preprocess the raw data stored in the data tables to the expected or more suitable form. These functions are called row-functions because they are applied gradually to all individual rows of the tables included in the query. The functions can be divided into the following groups based on the data type of their argument:

- Numeric Functions
- String Functions
- Temporal Functions

- Flow Control Functions
- NULL-related Functions

The complete list of the functions exceeds the possibilities of this course but can be easily found on the websites of the vendors of the database systems.

The parenthesis follows the name of the function without any space. For example, the query

```
SELECT ROUND(salary) FROM employees;
```

returns the salary of all employees rounded to zero decimal places.

## 4.4.4

If any of the values in the arithmetic expression contains NULL, then

- the result will be NULL.
- SQL will not be able to process the query.
- the system will return error message.
- the result will be equal to zero.

## 4.4.5

Numeric Functions

The SQL provides a lot of standard numeric functions, which works in the same way as in other math applications. The following functions belong to the most frequently used numeric functions:

- ROUND() rounds the argument to the required decimal places or tens
- FLOOR() returns the largest integer value not greater than the argument
- CEIL() returns the smallest integer value not less than the argument
- MOD() returns the remainder
- ABS() returns the absolute value

For example, the following query returns the salary rounded on the thousands

```
SELECT firstname, lastname, ROUND(salary, -3) AS
rounded_salary FROM employees;
```

## 4.4.6

What will be the result of the following query, if the value of the argument will be Null?

```
SELECT ROUND(salary,2) FROM employees;
```

- NULL
- 0
- NaN
- 0.00

## 4.4.7

String Functions

This group of functions perform operations on strings. They allow

- calculating string lengths - LENGTH(),
- finding the occurrence of a string in another string – INSTR(), LOCATE(), POSITION(),
- getting a part of a string – SUBSTR(), LEFT(), RIGHT(),
- combining strings to a new string – CONCAT(),
- changing the letter case of a string – LOWER(), LCASE(), UPPER(), UCASE(),
- trimming or padding strings – TRIM(), LPAD(), RPAD().

For example, the following query returns the full name of the employees with the lastname in uppercase.

```
SELECT CONCAT(firstname, ' ', UCASE(lastname)) AS full_name
FROM employees;
```

## 4.4.8

Mark the function which removes the spaces from the values of the column used as the argument.

- TRIM()
- LTRIM()
- LEFT()
- SUBSTR()
- LOCATE()

## 📖 4.4.9

Temporal Functions

Temporal functions closely relate to the date and time data types. These functions can be used for

- providing information about the current date and time – CURDATE(), NOW(), SYSDATE(),
- extracting part of the date – DAY(), MONTH(), YEAR(), EXTRACT(),
- formating date and time DATE(), DATE_FORMAT(),
- calculating with dates – ADDDATE(), DATEDIFF().

The names of the temporal functions can partially differ in different database systems, but they provide the same functionality.

The following query returns the year and the month of the birthday for all employees

```
SELECT fistname, lastname, YEAR(birthday) AS year_birthday,
MONTH(birthday) AS month_birthday FROM employees;
```

## 📝 4.4.10

Create a query, which returns the year from the current date.

_____(____()) _____;

- FROM
- NOW
- SELECT
- TODAY
- YEAR
- employees
- DATE

## 📖 4.4.11

Control Flow Functions

The following special functions provide an option to make assign a new value to a column based on the evaluation of the logical expression and current value in the column, which is taken as the argument of the function.

- IF() for the decision based on the simple logical expression, which can be fulfiled or not,
- CASE for more complex cases.

The following query returns information if the employee has more than three children and is, therefore, qualified for the bonus

```
SELECT firstname, lastname, IF(children_count > 3, 'yes',
'no') AS bonus FROM employees;
```

The next query divides the employees into three groups based on the number of years spent on the position

```
SELECT firstname, lastname,
CASE
  WHEN years_in_position < 3 THEN 'level 1'
  WHEN years_in_position  < 10 THEN 'level 2'
  ELSE 'level 3'
END AS level FROM employees;
```

## 📖 4.4.12

NULL-related Functions

Sometimes can be useful to replace the null value with a more suitable value, for example with zero in calculations, and vice versa. The following two null-related functions can be used in these cases:

- IFNULL()
- NULLIF()

The first query replaces the null value with zero in the column year_in_position.

```
SELECT firstname, last name, IFNULL(years_in_position, 0) AS
years_on_position FROM employees;
```

The NULLIF() returns NULL if the compared expressions are equal otherwise returns the first expression. The query returns null if the two email contacts are equal. Otherwise, it returns the first one.

```
 SELECT firstname, lastname, NULLIF(email1, email2) AS
contact_email FROM employees;
```

### 4.4.13

Which function will be evaluated first in the following query?

```
SELECT CONCAT(lastname, (SUBSTR(LOWER(firstname),4))) FROM
employees;
```

- LOWER
- CONCAT
- SUBSTR
- Functions can not be combined in such a way.

## 4.5 SELECT Statement - WHERE Clause

### 4.5.1

WHERE Clause - Search Conditions

The SELECT statement is often used to determine which records meet the specified conditions. The WHERE clause is the part of the SQL query that is used to specify these conditions. The condition most often contains the following parts:

- The name of the column whose values should meet the condition,
- comparison operator (<, <=, =,> =,>, =!),
- the value to compare the values in the column with.

The query returns all employees with a salary of less than 1000 Euro.

```
SELECT * FROM employees WHERE salary < 1000;
```

### 4.5.2

WHERE Clause - Multiple Condition

Simple conditions can be combined into multiple conditions using logical operators AND, OR, NOT, XOR. Their use has the same rules as in mathematics. For the sake of clarity, it is recommended to enclose the conditions in parentheses, so that their order of execution is clear. Multiple conditions may refer to the same column or multiple columns.

For example, the following query returns all employees, who work in Prague or Bratislava and work for the company for more than 5 years.

```
SELECT * FROM employees WHERE (city = 'Prague'  OR city =
'Bratislava') AND year_in_company > 5;
```

## 4.5.3

The number of conditions in the WHERE clause is limited to:

- There is no limit except the querytext limit
- 5
- 100
- 1000

## 4.5.4

WHERE Clause - Range Search Condition

If the SQL query should return rows whose value in the selected column is from the searched interval/range, a combination of logical operator and limit values can be used

```
SELECT firstname, lastname FROM employees WHERE salary >= 500
AND salary <= 1000;
```

or BETWEEN ... AND

```
SELECT firstname, lastname FROM employees WHERE salary BETWEEN
500 AND 1000;
```

Note: The BETWEEN .. AND operator always considers the interval/range border values.

## 4.5.5

The following two queries are equivalent, they return the same rows. True or False?

```
SELECT * FROM employees WHERE salary > 800 AND salary < 900;
SELECT * FROM employees WHERE salary BETWEEN 800 AND 900;
```

- FALSE
- TRUE

## 📖 4.5.6

WHERE Clause - Set Membership

It is often necessary to list all the values that an attribute can take in multiple conditions. Reuse of a condition requires repeating logical operators and can become confusing. Therefore, SQL provides an option to replace it with IN operator. A SQL query with an IN operator will not be more efficient in terms of performance but will be clearer for the human.

```
WHERE year_of_birth IN (1986, 1988, 1990, 1996)
```

## 📝 4.5.7

Mark correct queries that return the employees who work in Krakow, Katowice or Warsaw.

- SELECT * FROM employees WHERE city = 'Krakow' OR city = 'Katowice' OR city = 'Warsaw';
- SELECT * FROM employees WHERE city IN ('Krakow', 'Katowice', 'Warsaw');
- SELECT * FROM employees WHERE city = 'Krakow' AND city = 'Katowice' AND city = 'Warsaw';
- SELECT * FROM employees WHERE city = 'Krakow' OR 'Katowice' OR 'Warsaw';

## 📖 4.5.8

WHERE Clause - Pattern Matching

If the column has the data type text, it is necessary to specify exactly what the text value looks like in that column. For example, if the text value differs in only one character, the operator "=" does not find the value. An operator LIKE was added to SQL for these cases. It allows to enter into the searching condition only part of the text and the rest of the text replace with the following wildcard symbols:

- %: sequence of zero or more characters,
- _ (underscore): any single character.

If you are looking for a percent character or an underscore, you must override wildcard characters. It is done by the \ (backslash) character.

Example of use:

```
WHERE name LIKE '_a%'
```

The example given would be the names of what they have as the second letter a, such as Martin, Jana, Carolina etc.

### 📝 4.5.9

Which from the following query returns the employees, whose lastname begins with the letter S?

- SELECT firstname, lastname FROM employees WHERE lastname LIKE 'S%';
- SELECT firstname, lastname FROM employees WHERE firstname LIKE 'S%';
- SELECT firstname, lastname FROM employees WHERE lastname LIKE '%S%';
- SELECT firstname, lastname FROM employees WHERE lastname = 'S%';
- SELECT firstname, lastname FROM employees WHERE lastname LIKE '%S';

### 🕮 4.5.10

WHERE Clause - Searching for Null

Null represents an unknown or nonexistent value. Because its value is not exactly known, common operators of comparison can not be applied. For example, if it is necessary to determine which rows do not contain a value, the operator "=" returns a query with zero rows as a result. Therefore, it is necessary to use IS NULL operator.

Example:

```
WHERE email IS NULL
```

condition selects those records where no email was specified.

However, it is often necessary to find out whether a value in a given column is not empty, so the IS operator is combined with a negation of NOT.

```
WHERE email IS NOT NULL
```

## 4.6 SELECT - WHERE clause (exercises)

### ⌨ 4.6.1 Employees named Jan

Select Name and JobTitleCode of employees with the name Jan

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 2 | Jan | Nowak | RKB003 |
| 3 | Kamil | Wilmowski | RKB011 |
| 4 | Olga | Milenka | RKB003 |
| 5 | Angela | Wilga | RKB002 |

### ⌨ 4.6.2 Movies created at 1990

Find all movies created in 1990.

**Movies**

| mID | title | length | year |
|---|---|---|---|
| 1 | Catch me, please | 86 | 2017 |
| 2 | A Touching Saga of a Hunter | 126 | 2013 |
| 3 | Agent Truman | 143 | 1990 |
| 4 | Tomatoes antitrust | 203 | 2003 |
| 5 | Savannah | 160 | 1990 |
| 6 | Armageddon again | 106 | 2002 |
| 7 | Happy people | 217 | 1987 |

### ⌨ 4.6.3 Movies created at ‘80s

Find all movies created in the ‘80s.

**Movies**

| mID | title | length | year |
|---|---|---|---|
| 1 | Catch me, please | 86 | 2017 |
| 2 | A Touching Saga of a Hunter | 126 | 2013 |
| 3 | Agent Truman | 143 | 1990 |
| 4 | Tomatoes antitrust | 203 | 2003 |
| 5 | Savannah | 160 | 1990 |
| 6 | Armageddon again | 106 | 1986 |
| 7 | Happy people | 217 | 1987 |

### ⌨ 4.6.4 Movies with length greather than 110 minutes

List all movies with a length greater than 110 minutes

**Movies**

| mID | title | length | year |
|---|---|---|---|
| 1 | Catch me, please | 86 | 2017 |
| 2 | A Touching Saga of a Hunter | 126 | 2013 |
| 3 | Agent Truman | 143 | 1990 |
| 4 | Tomatoes antitrust | 203 | 2003 |
| 5 | Savannah | 160 | 1990 |
| 6 | Armageddon again | 106 | 1986 |
| 7 | Happy people | 217 | 1987 |

### ⌨ 4.6.5 Movies with length 100 - 140 minutes

List all movies with a length between 100 and 140 minutes

**Movies**

| mID | title | length | year |
|---|---|---|---|
| 1 | Catch me, please | 86 | 2017 |
| 2 | A Touching Saga of a Hunter | 126 | 2013 |
| 3 | Agent Truman | 143 | 1990 |
| 4 | Tomatoes antitrust | 203 | 2003 |
| 5 | Savannah | 160 | 1990 |
| 6 | Armageddon again | 106 | 1986 |
| 7 | Happy people | 217 | 1987 |

### ⌨ 4.6.6 Movies length in seconds

Show movie titles and the length expressed in seconds named seclength.

**Movies**

| mID | title | length | year |
|---|---|---|---|
| 1 | Catch me, please | 86 | 2017 |
| 2 | A Touching Saga of a Hunter | 126 | 2013 |
| 3 | Agent Truman | 143 | 1990 |
| 4 | Tomatoes antitrust | 203 | 2003 |
| 5 | Savannah | 160 | 1990 |
| 6 | Armageddon again | 106 | 1986 |
| 7 | Happy people | 217 | 1987 |

### ⌨ 4.6.7 Movies begun with A

Show all columns of all movies, which title begins with A.

**Movies**

| mID | title | length | year |
|---|---|---|---|
| 1 | Catch me, please | 86 | 2017 |
| 2 | A Touching Saga of a Hunter | 126 | 2013 |
| 3 | Agent Truman | 143 | 1990 |
| 4 | Tomatoes antitrust | 203 | 2003 |
| 5 | Savannah | 160 | 1990 |
| 6 | Armageddon again | 106 | 1986 |
| 7 | Happy people | 217 | 1987 |

## 4.6.8 T - actors

Show all the data of all actors whose last name begins with T.

actors

| aID | firstname | lastname | year_of_birth | income |
|---|---|---|---|---|
| 1 | Woody | Hopkins | 1975 | 1995 |
| 2 | Sandra | Tandy | 1965 | 2001 |
| 3 | John | Torn | 1994 | 2015 |
| 4 | Natalie | Hunt | 1957 | 2002 |
| 5 | Juan | Pitt | 1978 | 1999 |
| 6 | Henry | Nolte | 1968 | 2005 |
| 7 | Woody | Harris | 1945 | 1984 |

## 4.6.9 S, T - actors

Show all the data of all actors whose last name begins with S or T.

**actors**

| aID | firstname | lastname | year_of_birth | income |
|---|---|---|---|---|
| 1 | Woody | Hopkins | 1975 | 1995 |
| 2 | Sandra | Tandy | 1965 | 2001 |
| 3 | John | Torn | 1994 | 2015 |
| 4 | Natalie | Hunt | 1957 | 2002 |
| 5 | Juan | Stock | 1978 | 1999 |
| 6 | Henry | Nolte | 1968 | 2005 |
| 7 | Woody | Seller | 1945 | 1984 |

### 4.6.10 TT - actors

Show all the data of all actors whose last name and the first name begins with T.

**actors**

| aID | firstname | lastname | year_of_birth | income |
|---|---|---|---|---|
| 1 | Woody | Hopkins | 1975 | 1995 |
| 2 | Tibor | Tandy | 1965 | 2001 |
| 3 | John | Torn | 1994 | 2015 |
| 4 | Natalie | Hunt | 1957 | 2002 |
| 5 | Juan | Stock | 1978 | 1999 |
| 6 | Henry | Nolte | 1968 | 2005 |
| 7 | Woody | Seller | 1945 | 1984 |

# Group by

Chapter 5

# 5.1 SELECT Statement - Aggregates

## 📖 5.1.1

SELECT Statement – Aggregates

We often need to know basic common characteristics of the values in the selected column, like sum, average, minimum or maximum value. SQL provides a set of functions, called aggregate functions, which do not operate on each row separately, like single-row functions, but on a single column of the table. They always return a single value as a result. There are five aggregate functions:

- COUNT returns the number of values in the specified column.
- SUM returns the sum of values in the specified column.
- AVG returns the average of values in the specified column.
- MIN returns the smallest value in the specified column.
- MAX returns the largest value in the specified column.

There are several other aggregate functions, which are used very rarely, like STDEV, VAR.

## 📝 5.1.2

Which functions belong to the most common aggregation functions?

- MIN
- AVG
- SUM
- ROUND
- DISTINCT
- AVERAGE

## 📖 5.1.3

SELECT Statement – Characteristics of Aggregates

The aggregate functions have the following important features, which should be always considered.

- COUNT, MIN, and MAX can be applied to numeric and non-numeric fields.
- On the other hand, SUM and AVG may be used on numeric fields only.

- All functions can use DISTINCT before column name to eliminate duplicates. However, DISTINCT has no effect with MIN/MAX but may have with SUM/AVG.
- Apart from a special case of function COUNT - COUNT(*) - each function eliminates nulls first and operates only on remaining non-null values.
- Aggregate functions can be used only in the SELECT clause and in the HAVING clause which will be introduced later.

## 5.1.4

Which of the following aggregate functions can be applied to numeric and non-numeric columns?

- MAX
- MIN
- COUNT
- SUM
- AVG

## 5.1.5

COUNT

COUNT is a function that allows to count the number of records in a selected column or a number of rows in one table or joined tables. The following query returns the number of employees.

```
SELECT COUNT(ID) AS number_of_employees FROM employees;
```

COUNT expects only one argument, which can be mainly a column name but can also contain constant or any kind of combination of arithmetic operations and functions, which returns one value. The next query counts the number of unique cities, in which the employees work.

```
SELECT COUNT(DISTINCT city) AS number_of_unique_cities FROM
employees;
```

COUNT applied on a primary key column of the table returns the number of rows in the table. The same result can be obtained using COUNT(*). The symbol asterisk (*) represents again all columns of the table. COUNT(*) counts all rows of a table, regardless of whether nulls or duplicate values occur.

```
SELECT COUNT(*) AS number_of_employees FROM employees;
```

## 5.1.6

What is the result of the following query, if the ID is the primary key of the table employees?

```
SELECT COUNT(*) - COUNT(ID) FROM employees;
```

- 0
- NULL
- If the ID contains null, the result will be greater than 0.
- 1

## 5.1.7

SUM and AVG

SUM and AVG functions ignore the null values and can not be applied to string and date data types.

```
SELECT COUNT(ID) AS number_of_employees, AVG(salary) AS
average_salary FROM employees;
```

The query returns the number of employees and the average salary in the company. Both aggregate functions can be used in one query as other calculated columns.

If DISTINCT is applied to the argument of SUM and AVG functions, the result can be different, because the duplicates are eliminated before the calculation occurs.

## 5.1.8

Which of the following queries return the average number of years, which the employees spent in the company?

- SELECT AVG(years_in_company) AS average_years FROM employees;
- SELECT SUM(years_in_company)/COUNT(ID) AS average_years FROM employees;
- SELECT AVERAGE(years_in_company) AS average_years FROM employees;
- SELECT MEAN(years_in_company) AS mean_years FROM employees;

## 📖 5.1.9

MIN and MAX

MIN and MAX can be applied to the columns of the following generic data types

- Integer or Float numbers – return the highest or the lowest value
- Date – return the oldest or the newest date
- String – return the string, which appears as the first in the list of values considering the alphabet order and vice versa.

The following query returns the lowest and the highest salary stored in the table employees.

```
SELECT MIN(salary) AS minimum, MAX(salary) AS maximum FROM employees;
```

## 📝 5.1.10

What is the result of the following query?

```
SELECT MAX(firstname) FROM employees;
```

- The firstname of the employee, which begins with the last letter of the alphabet.
- The firstname of the employee, which begins with the first letter of the alphabet.
- The shortest firstname of the employee.
- The firstname of the employee with the lowest salary.

## 📖 5.1.11

Aggregate Functions in the SELECT statement

Aggregate functions can be used in the SELECT statement in the following situations:

- A SELECT clause can contain several aggregate functions in one query but the SELECT clause can not reference a column without an aggregate function in this case.
- If the SELECT clause contains aggregate function and references columns, on which an aggregate function is not applied, the query must contain GROUP BY clause.

For example, the following is illegal

```
SELECT firstname, lastname, MAX(salary) FROM employees;
```

## 5.2 SELECT Statement - Grouping

### 📖 5.2.1

SELECT Statement – Grouping

Sometimes it is useful to calculate the average or sum from the subset of data stored in the table or find the minimum or maximum values within the values with the same characteristics. The aggregate functions can be used in these cases, together with the clause GROUP BY. Clause GROUP BY allows to define of the groups, which will be created before the given aggregate function is applied.

SELECT and GROUP BY clauses are closely integrated. Each item in the SELECT clause must be single-valued per group, and the SELECT clause may only contain:

- column names,
- aggregate functions,
- constants,
- an expression involving combinations of the above.

The following query counts the number of employees, who work in the different cities:

```
SELECT city, COUNT(ID) AS number_of_employees_in_city
FROM employees
GROUP BY city;
```

The order of the columns in the SELECT clause will be shown also in the result. It is recommended to always also show column, which defines the groups for easier understanding of the result.

### 📝 5.2.2

If the list of columns in the SELECT clause contains column names as well as any of the aggregate functions, which additional clause must still be included in the query?

- GROUP BY
- HAVING
- ORDER BY

- WHERE
- DISTINCT

## 📖 5.2.3

SELECT Statement – Grouping

All column names in the SELECT clause must appear in the GROUP BY clause unless the name is used only in an aggregate function.

If the WHERE clause is used with GROUP BY, WHERE must be applied first, then groups are formed from remaining rows satisfying the condition defined in the WHERE clause. The following query returns the number of employees who earn less than 1000 Euro grouped by the cities,

```
SELECT city, COUNT(ID) AS number_of_employees_in_city
FROM employees
WHERE salary < 1000
GROUP BY city;
```

In contrast to other situations, where the null value appears, two nulls are considered equal for purposes of the GROUP BY clause.

## 📝 5.2.4

Create a query, which returns the average salaries of employees, who work in different cities in the position of Sales Representative. Order the result from the highest to the lowest value of the average salary.

_____

_____

_____

_____

_____

- WHERE position = 'Sales Representative'
- ORDER BY AVG(salary)
- SELECT city, AVG(salary) AS average_salary

- ORDER BY AVG(salary) DESC
- FROM employees
- GROUP BY salary
- GROUP BY city

## 5.3 SELECT Statement - HAVING Clause

### 📖 5.3.1

HAVING Clause - Restricting Groupings

HAVING clause is designed for use with GROUP BY to restrict groups that appear in the result table.

HAVING filters groups whereas WHERE clause filters individual rows. WHERE clause must forego the HAVING clause in the SQL query.

Column names in the HAVING clause must also appear in the GROUP BY clause or must be included within an aggregate function.

### 📝 5.3.2

The HAVING clause performs in an aggregate query a similar function to the clause WHERE.

- WHERE
- ORDER BY
- GROUP BY
- SELECT
- JOIN

### 📝 5.3.3

Write the right keyword to finish the following sentence correctly.

The HAVING clause performs in an aggregate query a similar function to the clause ..... .

# 5.4 Aggregation and grouping (exercises)

## ⌨ 5.4.1 Average length of the movies

What is the average length of the movies stored in the database?

**Movies**

| mID | title | length | year |
|---|---|---|---|
| 1 | Catch me, please | 86 | 2017 |
| 2 | A Touching Saga of a Hunter | 126 | 2013 |
| 3 | Agent Truman | 143 | 1990 |
| 4 | Tomatoes antitrust | 203 | 2003 |
| 5 | Savannah | 160 | 1990 |
| 6 | Armageddon again | 106 | 1986 |
| 7 | Happy people | 217 | 1987 |

## ⌨ 5.4.2 The longest movie

How long does the longest movie take?

**Movies**

| mID | title | length | year |
|---|---|---|---|
| 1 | Catch me, please | 86 | 2017 |
| 2 | A Touching Saga of a Hunter | 126 | 2013 |
| 3 | Agent Truman | 143 | 1990 |
| 4 | Tomatoes antitrust | 203 | 2003 |
| 5 | Savannah | 160 | 1990 |
| 6 | Armageddon again | 106 | 1986 |
| 7 | Happy people | 217 | 1987 |

## ⌨ 5.4.3 Money for actors

How much the actors have been earned together.

**actors**

| aID | firstname | lastname | year_of_birth | income | earned |
|---|---|---|---|---|---|
| 1 | Woody | Hopkins | 1975 | 1995 | 100000 |
| 2 | Tibor | Tandy | 1965 | 2001 | 2458771 |
| 3 | John | Torn | 1994 | 2015 | 7778852 |
| 4 | Natalie | Hunt | 1957 | 2002 | 15000001 |

| | | | | | |
|---|---|---|---|---|---|
| 5 | Juan | Stock | 1978 | 1999 | 154557 |
| 6 | Henry | Nolte | 1968 | 2005 | 1154775 |
| 7 | Woody | Seller | 1945 | 1984 | 8425771 |

## 5.4.4 Average profit of actors

How much the actors have been earned on average?

**actors**

| aID | firstname | lastname | year_of_birth | income | earned |
|---|---|---|---|---|---|
| 1 | Woody | Hopkins | 1975 | 1995 | 100000 |
| 2 | Tibor | Tandy | 1965 | 2001 | 2458771 |
| 3 | John | Torn | 1994 | 2015 | 7778852 |
| 4 | Natalie | Hunt | 1957 | 2002 | 15000001 |
| 5 | Juan | Stock | 1978 | 1999 | 154557 |
| 6 | Henry | Nolte | 1968 | 2005 | 1154775 |
| 7 | Woody | Seller | 1945 | 1984 | 8425771 |

## 5.4.5 The longest and the shortest movie

How long do the longest and the shortest movies stored in the database take? Show the answer in one record.

**Movies**

| mID | title | length | year |
|---|---|---|---|
| 1 | Catch me, please | 86 | 2017 |
| 2 | A Touching Saga of a Hunter | 126 | 2013 |
| 3 | Agent Truman | 143 | 1990 |
| 4 | Tomatoes antitrust | 203 | 2003 |
| 5 | Savannah | 160 | 1990 |
| 6 | Armageddon again | 106 | 1986 |
| 7 | Happy people | 217 | 1987 |

## 5.4.6 The oldest and the youngest movie

How many years have passed between the oldest and newest movie stored in the database? Show the answer in one record.

**Movies**

| mID | title | length | year |
|---|---|---|---|

| | | | |
|---|---|---|---|
| 1 | Catch me, please | 86 | 2017 |
| 2 | A Touching Saga of a Hunter | 126 | 2013 |
| 3 | Agent Truman | 143 | 1990 |
| 4 | Tomatoes antitrust | 203 | 2003 |
| 5 | Savannah | 160 | 1990 |
| 6 | Armageddon again | 106 | 1986 |
| 7 | Happy people | 217 | 1987 |

## 5.4.7 Movies in countries

How many movies were created in each country? Show name of country and count of movies.

**Movies**

| mID | title | length | year | country |
|---|---|---|---|---|
| 1 | Catch me, please | 86 | 2017 | Australia |
| 2 | A Touching Saga of a Hunter | 126 | 2013 | France |
| 3 | Agent Truman | 143 | 1990 | USA |
| 4 | Tomatoes antitrust | 203 | 2003 | USA |
| 5 | Savannah | 160 | 1990 | Australia |
| 6 | Armageddon again | 106 | 1986 | France |
| 7 | Happy people | 217 | 1987 | France |

## 5.4.8 Actors from countries

How many actors come from the countries? Show country and count of actors from the country.

**actors**

| aID | firstname | lastname | year_of_birth | income | earned | country |
|---|---|---|---|---|---|---|
| 1 | Woody | Hopkins | 1975 | 1995 | 100000 | Australia |
| 2 | Tibor | Tandy | 1965 | 2001 | 2458771 | France |
| 3 | John | Torn | 1994 | 2015 | 7778852 | USA |
| 4 | Natalie | Hunt | 1957 | 2002 | 15000001 | USA |
| 5 | Juan | Stock | 1978 | 1999 | 154557 | Australia |
| 6 | Henry | Nolte | 1968 | 2005 | 1154775 | France |
| 7 | Woody | Seller | 1945 | 1984 | 8425771 | Australia |

## 5.4.9 Countries with more than 3 actors

Show the list of countries with more than 3 actors. Show the name of the country and count of actors.

**actors**

| aID | firstname | lastname | year_of_birth | income | earned | country |
|---|---|---|---|---|---|---|
| 1 | Woody | Hopkins | 1975 | 1995 | 100000 | Australia |
| 2 | Tibor | Tandy | 1965 | 2001 | 2458771 | France |
| 3 | John | Torn | 1994 | 2015 | 7778852 | USA |
| 4 | Natalie | Hunt | 1957 | 2002 | 15000001 | USA |
| 5 | Juan | Stock | 1978 | 1999 | 154557 | Australia |
| 6 | Henry | Nolte | 1968 | 2005 | 1154775 | France |
| 7 | Woody | Seller | 1945 | 1984 | 8425771 | Australia |
| 8 | Hypolia | Genders | 1995 | 2002 | 333345 | Australia |

# Join

## Chapter 6

# 6.1 Multi-table Queries

## 🕮 6.1.1

Multi-Table Queries

As a result of the database design and normalisation process, almost all databases contain many tables. This approach minimises data redundancy. Simultaneously, it ensures data consistency and integrity. The relationship between the data stored in different tables is expressed by the primary and foreign keys between the related tables. However, there are plenty of situations, in which the information collected from different tables is required. Therefore, SQL provides several options, how to join tables.

## 🕮 6.1.2

Computing a Join

The general procedure for generating results of a join of Table1 and Table2 is as follows:

1. Cartesian product of the tables named in FROM clause is formed.
2. If there is a WHERE clause, the search condition is applied to each row of the product table, retaining those rows that satisfy the condition.
3. For each remaining row, the value of each item in the SELECT clause is determined to produce a single row in the result table.
4. If DISTINCT has been specified, any duplicate rows are eliminated from the result table.
5. If there is an ORDER BY clause, the rows of the result table are ordered as required.

```
SELECT Table1.*, Table2.*
FROM Table1, Table2
WHERE Table1.Table1ID = Table2.Table1ID
ORDER BY Table1ID;
```

The query shows all columns from joined tables ordered by the primary key of Table1. The FROM clause must list the names of all tables used in the query. The WHERE clause is used for joining the primary key of Table1 with the corresponding foreign key in Table2.

## 6.1.3

The result of omitting the join between the tables in the query is

- Cartesian product
- outer join
- inner join
- NULL

## 6.1.4

Sorting the Result of Joined Tables

No matter how many tables will be joined in the query and which SQL joining clause will be used, the ORDER BY clause can be used only once as the last clause of the SQL query. The rules, which define the final order of the values in the result table, are the same as in the case of one simple table.

```
SELECT a.full_name, b.title, b.pages
FROM authors a
INNER JOIN authors_books ab ON a.authorID = ab.authorID
INNER JOIN books b ON ab.bookID = b.bookID
ORDER BY pages DESC;
```

The query returns the list of authors, their books ordered by the number of pages in descending order. The query simultaneously shows how to effectively use aliases to name tables. The ORDER BY clause at the end of the query is used for ordering the result.

## 6.1.5

Mark all true statements.

- The multi-table query can contain only one WHERE clause.
- The FROM clause must list all tables used in query separated by space.
- The number of tables, which can be joined in multi-table query is limited to 10.
- The alias can be used for renaming the tables in the multi-table query.

### 6.1.6

The number of tables, which can be joined in SQL query, is theoretically unlimited.

- TRUE
- FALSE

### 6.1.7

Aggregate Functions in Multi-Table Queries

All aggregate functions can be also used without any limits in the multi-table queries. The same set of rules for their writing must be followed. The GROUP BY clause is used only once in the query after all tables are joined. The same is true also for the HAVING clause.

Assume, the database contains two tables offices, employees with relation 1:N. The query counts the number of employees at the different offices of the company.

```
SELECT offices.name, COUNT(employeeID) AS number_of_employees
FROM offices
INNER JOIN employees ON offices.officeID = employees.officeID
GROUP BY offices.name;
```

## 6.2 Simple Join Using WHERE Clause

### 6.2.1

Simple Join

If the columns required in the result come from more than one table, a join between the tables must be defined. The simplest way is to use the WHERE clause and write an SQL query considering the following steps:

- The relationship between tables in the form of related primary and foreign keys must exist.
- Tables, which contain the required columns, must be named in the FROM clause, separated by a comma.
- The aliases can be assigned to the tables in the FROM clause to ensure simplicity. Alias is a unique short name of the table, separated from the table name with space.
- If the tables contain columns with the same names, and these columns will be used in the query, the alias must be assigned to each of them to avoid ambiguity.

- WHERE clause must define the condition based on which the data from the tables is joined. Primary and foreign keys are often used in this condition.

As a result, only those rows from both tables that have identical values in the condition defined in the WHERE clause is included in the result.

For example, the following query shows the employees' names, city and the name of the office, where they work.

```
SELECT firstname, lastname, city, office_name
FROM employees, offices
WHERE employees.officeID = offices.officeID
```

While the names of the columns are different, it is not necessary to use aliases to rename them.

## 📖 6.2.2

Joining More Tables

The same approach can be applied to joining more tables. Between all tables must exist a relationship based on the comparison of related primary and foreign keys.

Therefore, for example, in the case of joining three tables, two conditions, which define the relationship, must be defined in minimum, for four tables three conditions, etc. In the opposite case, the result will contain an unexpected number of rows, because the Cartesian product occurred.

The number of tables, which can be joined using the WHERE clause is not limited in general. However, there are some practical and technical limitations, which should be considered to ensure the highest performance of the database system.

## 📝 6.2.3

What is the minimum number of uses of the JOIN clause we need to prevent the Cartesian product from joining four tables?

- 3
- 2
- 0
- 4
- 1

### 📝 6.2.4

Two tables are given:

- countries (countryID, country_name, continentID, population)
- continents (continentID, continent_name)

Create a query, which returns the list of countries and related continents ordered by the population from the largest to the lowest.

- FROM countries, continents
- SELECT country_name, continent_name, population
- ORDER BY population DESC;
- WHERE countries.continentID = continents.continentID

## 6.3 Preferred Approaches to Join Tables based on JOIN Clause

### 🕮 6.3.1

Preferred Approaches to Join Tables based on JOIN Clause

The requirement to join many tables in one query is very often in practice. The WHERE clause becomes disarranged. The SQL provides the following constructs based on the new clause JOIN, which allow solving this situation:

- INNER JOIN - allows joining tables similarly to WHERE clause, only the rows, which fulfil the joining condition will be included in the result,
- OUTER JOIN – allows joining tables based on the defined joining condition, while the result will contain not only data, which fulfilled the defined condition but also data, which does not have a corresponding value in the joined table,
- NATURAL JOIN and JOIN USING allow joining tables based on the same names of the table columns,
- CROSS JOIN allows the creation of a Cartesian product explicitly.

These clauses differ in the way, how they identify the columns, which define the relationship between tables and how they work with the null values.

### 🕮 6.3.2

INNER JOIN Clause

The INNER JOIN clause represents the equivalent to the cases, in which the WHERE clause can be used for joining the tables. It joins two tables based on the quality of the values in the columns, which represent the relationship between tables, mostly between primary and foreign keys. If one row of a joined table is unmatched, a row is also omitted from the result table. The following example shows how to join two tables using the JOIN clause instead of the WHERE clause.

The query, which uses the WHERE clause, seems like this

```
SELECT firstname, lastname, city, office_name
FROM employees, offices
WHERE employees.officeID = offices.officeID
```

It can be rewritten using the INNER JOIN clause.

```
SELECT firstname, lastname, city, office_name
FROM employees
INNER JOIN offices ON employees.officeID = offices.officeID
```

The relation between the joined tables is written in the ON part of the INNER JOIN clause.

The keyword INNER can be omitted. However, it is recommended to use it consistently.

## 6.3.3

Create a query, which shows the genre of all movies from the table movies together with the title of the movie.

_____

_____

_____ _____

- JOIN INNER movies
- FROM genres, movies
- SELECT genre, title
- ON genres.genreID = movies.genreID
- INNER JOIN movies
- FROM genres

## 📖 6.3.4

INNER JOIN Clause in Queries with Many Tables

The INNER JOIN clause can be used for joining many tables. In comparison to other SQL clauses, the INNER JOIN clause must be repeated for each statement, which represents the relation between tables.

```
SELECT a.full_name, b.title, b.pages
FROM authors a
INNER JOIN authors_books ab ON a.authorID = ab.authorID
INNER JOIN books b ON ab.bookID = b.bookID
ORDER BY pages DESC;
```

The query above joins three tables. Therefore, the SQL query must have two INNER JOIN clauses. The order, in which the tables will be joined, does not matter. However, it is important to write all relations between the tables correctly using the right columns.

## 📝 6.3.5

Create a query, which shows the orderdate, the name of the product and the quantity of product items, if the quantity is greater than 10.

- INNER JOIN products ON products.productnumber = orderdetails.productnumber
- INNER JOIN orderdetails ON orders.ordernumber = orderdetails.ordernumber
- FROM orders
- SELECT orderdate, producttitle, quantity
- WHERE quantity > 10;

## 📖 6.3.6

OUTER JOIN

There are some specific situations, which require showing the values from the first table included in the join, which do have not the corresponding values in the second table. The clause OUTER JOIN was added to the SQL for that reason. It retains rows that do not satisfy the join condition in the result. The following three types of outer join can be used based on the requirements, which table's data has to be included in the query result:

LEFT OUTER JOIN – the result will contain all data from the left table (the first table in the query) and corresponding data from the right table (the second one in the query). If there is not a corresponding value in the second table, the null value will be used instead of it.

RIGHT OUTER JOIN - the result will contain all data from the right table (the second table in the query) and corresponding data from the left table (the first one in the query). If there is not a corresponding value in the left table, the null value will be used instead of it.

FULL OUTER JOIN - the result will contain all data from the left and right table, in which the joining condition is true. Moreover, it will add all other rows from both tables, in which the corresponding value in the joined table is missing. The null value will be used in all these cases.

It is evident that each RIGHT OUTER JOIN can be easily rewritten to the LEFT OUTER JOIN. Therefore, several database systems support only LEFT OUTER JOIN.

## 📝 6.3.7

What kind of join is required to show all continents regardless, there is not any country on them?

SELECT continent, country

FROM continents

_____ JOIN countries ON

continents.continentID = countries.continentID

- INNER
- RIGHT
- LEFT

## 🕮 6.3.8

Other types of JOIN Clause

There are several other SQL keywords, which allow joining tables in a special situation.

NATURAL JOIN allows joining tables based on the columns with the same names. For example, if the names of the primary and foreign keys are used consistently

throughout the data model, it is not necessary to write the INNER JOIN ON clause. The NATURAL JOIN clause can be used instead of it. On the other hand, it is necessary to bear in mind that all table columns with the same names will be included in the joining condition, which may not be expected behaviour.

```
SELECT firstname, lastname, office FROM employees NATURAL JOIN offices
```

Therefore, it is recommended to use the JOIN USING clause instead of the NATURAL JOIN. The columns with the same name, which will define the relation, are explicitly defined in this case in the USING clause.

```
SELECT firstname, lastname, office FROM employees JOIN offices USING (officeID)
```

Finally, the CROSS JOIN clause should be mention for completeness sake. It allows for creating the Cartesian product explicitly. The result will have the same characteristics, as in the case of omitting joining conditions between the tables in the query.

For example, if the table employees (20 rows) will be joined with the table offices (5 rows) using CROSS JOIN, the resulting table will contain 100 rows (20x5). The query seems as follows

```
SELECT* FROM employees CROSS JOIN offices
```

## 6.4 JOIN (exercises)

### ⌨ 6.4.1 Movies and genres

List genres for all movies. Use tables genres and movies.

**Movies**

| mID | title | length | year | gID |
|---|---|---|---|---|
| 1 | Catch me, please | 86 | 2017 | 2 |
| 2 | A Touching Saga of a Hunter | 126 | 2013 | 2 |
| 3 | Agent Truman | 143 | 1990 | 1 |
| 4 | Tomatoes antitrust | 203 | 2003 | 3 |
| 5 | Savannah | 160 | 1990 | 4 |
| 6 | Armageddon again | 106 | 1986 | 1 |
| 7 | Happy people | 217 | 1987 | 4 |

**Genres**

| gID | genre |
|---|---|

```
1     comedy
2     drama
3     thriller
4     documentary
```

## 6.4.2 Actors from countries

List the actor and the country, from which she comes. Show the first name, last name and country.

### Actors

```
aID firstname   lastname   year_of_birth   income    earned     cID
1   Woody       Hopkins    1975            1995      100000     1
2   Tibor       Tandy      1965            2001      2458771    1
3   John        Torn       1994            2015      7778852    3
4   Natalie     Hunt       1957            2002      15000001   2
5   Juan        Stock      1978            1999      154557     1
6   Henry       Nolte      1968            2005      1154775    7
7   Woody       Seller     1945            1984      8425771    1
```

### Countries

```
cID    country
1      Australia
2      France
3      Spain
4      Russia
5      USA
6      United Kingdom
7      Italia
```

## 6.4.3 Movies and genres II.

List genres for all movies using the INNER JOIN clause. Use tables genres and movies.

### Movies

```
mID   title                          length     year     gID
1     Catch me, please               86         2017     2
2     A Touching Saga of a Hunter    126        2013     2
3     Agent Truman                   143        1990     1
4     Tomatoes antitrust             203        2003     3
5     Savannah                       160        1990     4
```

```
6    Armageddon again                106        1986    1
7    Happy people                    217        1987    4
```

**Genres**

```
gID  genre
1    comedy
2    drama
3    thriller
4    documentary
```

## 6.4.4 Movies in 1990

List genres for all movies created in 1990. Show the title, genre and year.

**Movies**

```
mID  title                           length     cr_year gID
1    Catch me, please                86         2017    2
2    A Touching Saga of a Hunter     126        2013    2
3    Agent Truman                    143        1990    1
4    Tomatoes antitrust              203        2003    3
5    Savannah                        160        1990    4
6    Armageddon again                106        1986    1
7    Happy people                    217        1987    4
```

**Genres**

```
gID  genre
1    comedy
2    drama
3    thriller
4    documentary
```

## 6.4.5 Movie genres

Count the number of movies in different genres. Show the genre and count of movies.

**Movies**

```
mID  title                           length     cr_year gID
1    Catch me, please                86         2017    2
```

```
2    A Touching Saga of a Hunter   126       2013    2
3    Agent Truman                  143       1990    1
4    Tomatoes antitrust            203       2003    3
5    Savannah                      160       1990    4
6    Armageddon again              106       1986    1
7    Happy people                  217       1987    4
```

**Genres**

```
gID  genre
1    comedy
2    drama
3    thriller
4    documentary
```

## 6.4.6 Actors in movies

List actors' first name and last name, movie title and the name of the role the actor played in the movie.

**Actors**

```
aID firstname   lastname   year_of_birth   income    earned      cID
1   Woody       Hopkins    1975            1995      100000      1
2   Tibor       Tandy      1965            2001      2458771     1
3   John        Torn       1994            2015      7778852     3
4   Natalie     Hunt       1957            2002      15000001    2
5   Juan        Stock      1978            1999      154557      1
6   Henry       Nolte      1968            2005      1154775     7
7   Woody       Seller     1945            1984      8425771     1
```

**Movies**

```
mID  title                         length    cr_year gID
1    Catch me, please              86        2017    2
2    A Touching Saga of a Hunter   126       2013    2
3    Agent Truman                  143       1990    1
4    Tomatoes antitrust            203       2003    3
5    Savannah                      160       1990    4
6    Armageddon again              106       1986    1
7    Happy people                  217       1987    4
```

**Casting**

```
cid  aID  mID  role
1    4    2    mother
```

```
2     7     4     father
3     5     2     Luis
4     5     6     child
5     8     4     child
6     6     2     hunter
7     9     2     wizard
```

## 6.4.7 Actors with more than 2 movies

Show the list of actors, who played in more than 2 movies. Show the first name, last name and count of movies.

### Actors

```
aID firstname   lastname   year_of_birth   income    earned     cID
1   Woody       Hopkins    1975            1995      100000     1
2   Tibor       Tandy      1965            2001      2458771    1
3   John        Torn       1994            2015      7778852    3
4   Natalie     Hunt       1957            2002      15000001   2
5   Juan        Stock      1978            1999      154557     1
6   Henry       Nolte      1968            2005      1154775    7
7   Woody       Seller     1945            1984      8425771    1
```

### Movies

```
mID  title                          length    cr_year gID
1    Catch me, please               86        2017    2
2    A Touching Saga of a Hunter    126       2013    2
3    Agent Truman                   143       1990    1
4    Tomatoes antitrust             203       2003    3
5    Savannah                       160       1990    4
6    Armageddon again               106       1986    1
7    Happy people                   217       1987    4
```

### Casting

```
cid  aID  mID  role
1    4    2    mother
2    7    4    father
3    5    2    Luis
4    5    6    child
5    8    4    child
6    6    2    hunter
7    9    2    wizard
```

## 6.4.8 All genres

Show genres and movie titles. Include a genre even if it does not include any movies. Order the result by genre name. Use LEFT JOIN

**Movies**

| mID | title | length | cr_year | gID |
|---|---|---|---|---|
| 1 | Catch me, please | 86 | 2017 | 2 |
| 2 | A Touching Saga of a Hunter | 126 | 2013 | 2 |
| 3 | Agent Truman | 143 | 1990 | 1 |
| 4 | Tomatoes antitrust | 203 | 2003 | 2 |
| 5 | Savannah | 160 | 1990 | 4 |
| 6 | Armageddon again | 106 | 1986 | 1 |
| 7 | Happy people | 217 | 1987 | 4 |

**Genres**

| gID | genre |
|---|---|
| 1 | comedy |
| 2 | drama |
| 3 | thriller |
| 4 | documentary |
| 5 | romance |

## 6.4.9 All movies

Show movie titles and genres. Include a movie even if it does not have a registered genre. Order the result by movie title. Use LEFT JOIN

**Movies**

| mID | title | length | cr_year | gID |
|---|---|---|---|---|
| 1 | Catch me, please | 86 | 2017 | 2 |
| 2 | A Touching Saga of a Hunter | 126 | 2013 | 7 |
| 3 | Agent Truman | 143 | 1990 | 1 |
| 4 | Tomatoes antitrust | 203 | 2003 | 2 |
| 5 | Savannah | 160 | 1990 | 4 |
| 6 | Armageddon again | 106 | 1986 | 1 |
| 7 | Happy people | 217 | 1987 | 8 |

**Genres**

| gID | genre |
|---|---|
| 1 | comedy |

```
2    drama
3    thriller
4    documentary
5    romance
```

# Insert

Chapter 7

# 7.1 INSERT INTO statement

## 🕮 7.1.1

The INSERT INTO statement is used to insert a new row (tuple) into the table (relation).

In the following statement, you need to specify both the column (attribute) names and the corresponding values.

```
INSERT INTO table_name (column1, column2, ... , columnN)
VALUES (value1, value2, ... , valueN);
```

Using this form of INSERT INTO statement it is possible to insert values into selected columns only. However, it is necessary to include all NOT NULL columns (and corresponding values) if they don't have set up the default value.

## 🕮 7.1.2

It is possible to not specify the column names in the INSERT INTO query if you are adding values for all the columns of the table. In this case, the order of the values has to be the same as the order of the columns in the table. The syntax is as follows:

```
INSERT INTO table_name
 VALUES (value1, value2, ... , valueN);
```

## 🕮 7.1.3

When you create a new table it is possible to define DEFAULT values for a column or specify that column can be empty (a NULL value). Using the INSERT INTO statement it is possible to insert a new row with DEFAULT or NULL values. However, in this case, all the columns should have a default or null value set. If a single column doesn't fulfil this requirement you cannot use this possibility.

The syntax is as follows:

```
INSERT INTO table_name DEFAULT VALUES;
```

Please remember that this statement doesn't work in all database systems.

## 7.1.4

Which of the INSERT INTO statements is incorrect if you want to insert data into the table with the schema defined in SQLite DBMS:

```
Products (ProductID INTEGER PRIMARY KEY AUTOINCREMENT NOT
NULL,
ProductName VARCHAR (100), ProductDescription TEXT,
ProductPrice NUMERIC)
```

SQLite is one of the relational database systems.

- INSERT INTO Products DEFAULT VALUES;
- INSERT INTO Products (ProductID, ProductName, ProductDescription, ProductPrice) VALUES (27,'Onkyo TX','Amplituner', 2000.00);
- INSERT INTO Products (ProductID,ProductName) VALUES (30,'Yamaha RX');
- all of them
- none of them

## 7.1.5

Some database systems allow inserting multiple rows into a table, by one command. In this case, you will use the following form:

```
INSERT INTO table_name
VALUES (valueA1, valueA2, … , valueAN),
        (valueB1, valueB2, … , valueBN),
        (valueC1, valueC2, … , valueCN);
```

## 7.1.6

You want to insert three rows into empty table Customers with the following schema:

```
Customers (CustomerID INTEGER PRIMARY KEY,
CustomerName VARCHAR (50), CustomerSurname VARCHAR (50));
```

Is the INSERT INTO statement correct?

```
INSERT INTO Customers
VALUES (1,'Sofia','Bednar'),
        (2,'Jan','Zachar'),
        (3,'Nela','Walach');
```

- True
- False

## 7.1.7

The INSERT INTO statement can be combined with a SELECT statement to insert rows into a table. The SELECT statement allows selecting data from a different table.

The following syntax copy rows from different tables and inserts them into the table.

```
INSERT INTO table_name SELECT * FROM different_table;
```

The syntax may slightly vary between database systems.

The SELECT statement is a way how to get data from a database table. It is described in detail in the SELECT statement topic.

## 7.1.8

You want to copy capital cities from table Countries to table Cities column CityName. Use the appropriate statement. The schema of the tables are as follows:

```
Countries (CountryID INTEGER PRIMARY KEY, CountryName CHAR
(40), CountryCapital VARCHAR (50))

Cities (CityID INTEGER PRIMARY KEY AUTOINCREMENT, CityName
VARCHAR (50))
```

```
INSERT INTO _____ _____ SELECT _____ FROM _____ ;
```

- Cities
- CountryCapital
- Countries
- Cities
- (CityName)
- CityName
- CountryCapital
- Countries

# 7.2 INSERT exercise I.

## 7.2.1 INSERT 01

Write a query that inserts a new employee Anna Kowalska into the Employees table with the following data: Id equals 10, JobTitleCode is RKB011.

Data Set 1

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|

Data Set 2

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 2 | Jan | Nowak | RKB003 |
| 3 | Kamil | Wilmowski | RKB011 |
| 4 | Olga | Milenka | RKB003 |
| 5 | Angela | Wilga | RKB002 |

Data Set 3

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 3 | Kamil | Wilmowski | RKB011 |
| 5 | Angela | Wilga | RKB002 |

Data Set 4

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 3 | Kamil | Wilmowski | RKB011 |
| 4 | Paco | Ramirez | RKB011 |
| 5 | Angela | Wilga | RKB002 |
| 6 | Beatriz | Santana | RKB002 |

Data Set 5

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 7 | Joahim | Mesina | RKB001 |
| 8 | Marta | Frantisek | RKB002 |
| 9 | Jan | Matalau | RKB001 |

## 7.2.2 INSERT 02

Write a query that inserts into the Employees table a new employee Barbara Nowak, her Id equals 100 and she is working as a Shop Director - JobTitleCode is RKB100.

Data Set 1

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|

Data Set 2

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 2 | Jan | Nowak | RKB003 |
| 3 | Kamil | Wilmowski | RKB011 |
| 4 | Olga | Milenka | RKB003 |
| 5 | Angela | Wilga | RKB002 |

Data Set 3

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 3 | Kamil | Wilmowski | RKB011 |
| 5 | Angela | Wilga | RKB002 |

Data Set 4

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 3 | Kamil | Wilmowski | RKB011 |
| 4 | Paco | Ramirez | RKB011 |

```
5   Angela   Wilga       RKB002
6   Beatriz  Santana     RKB002
```

Data Set 5

**Employees**

```
Id  Name     Surname     JobTitleCode
7   Joahim   Mesina      RKB001
8   Marta    Frantisek   RKB002
9   Jan      Matalau     RKB001
```

### 7.2.3 INSERT 03

Write a query that using insert into statement copies all rows from the Countries table into the CountriesCopy table.

Data Set 1

**Countries**

```
Id  Country    Capital
```

**CountriesCopy**

```
Id  Country    Capital
```

Data Set 2

**Countries**

```
Id  Country      Capital
1   Portugal     Lisbon
2   Belgium      Brussels
3   Spain        Madrit
4   Hungary      Budapest
8   Croatia      Zagreb
9   Greece       Athens
10  Malta        Valletta
```

**CountriesCopy**

```
Id  Country  Capital
```

Data Set 3

**Countries**

```
Id  Country      Capital
3   Spain        Madrit
4   Hungary      Budapest
6   Slovakia     Bratislava
7   Norway       Oslo
8   Croatia      Zagreb
9   Greece       Athens
10  Malta        Valletta
11  Italy        Rome
```

**CountriesCopy**

```
Id  Country  Capital
```

Data Set 4

**Countries**

```
Id  Country      Capital
3   Spain        Madrit
5   Poland       Warsaw
8   Croatia      Zagreb
10  Malta        Valletta
11  Italy        Rome
```

**CountriesCopy**

```
Id  Country   Capital
```

Data Set 5

**Countries**

```
Id  Country      Capital
6   Slovakia     Bratislava
7   Norway       Oslo
8   Croatia      Zagreb
10  Malta        Valletta
11  Italy        Rome
```

**CountriesCopy**

```
Id  Country    Capital
```

## 7.2.4 INSERT 04

Write a query that using insert into statement copies a row from the Cities table into the CitiesCopy table where city Id is equal to 2.

Data Set 1

**Cities**

```
Id  Name
```

**CitiesCopy**

```
Id  Name
```

Data Set 2

**Cities**

```
Id  Name
1   Bratislava
2   Nitra
3   Trnava
4   Katowice
5   Telde
6   Paris
8   Galdar
10  Krakow
```

**CitiesCopy**

```
Id  Name
```

Data Set 3

**Cities**

```
Id  Name
2   Nitra
4   Katowice
6   Paris
8   Galdar
```

**CitiesCopy**

```
Id  Name
```

Data Set 4

**Cities**

```
Id  Name
2   Nitra
3   Vienna
4   Katowice
5   Sevilla
8   Galdar
```

**CitiesCopy**

```
Id  Name
```

Data Set 5

**Cities**

```
Id  Name
2   Nitra
4   Katowice
6   Paris
7   Madrit
9   Brussels
```

**CitiesCopy**

```
Id  Name
```

## 7.2.5 INSERT 05

Write a query that inserts into the Countries table a new row with default values, the column Id has the autoincrement option set.

Data Set 1

**Countries**

```
Id  Country    Capital
```

Data Set 2

**Countries**

```
Id  Country      Capital
```

```
1    Portugal      Lisbon
2    Belgium       Brussels
3    Spain         Madrit
4    Hungary       Budapest
8    Croatia       Zagreb
9    Greece        Athens
10   Malta         Valletta
```

Data Set 3

**Countries**

```
Id   Country       Capital
3    Spain         Madrit
4    Hungary       Budapest
6    Slovakia      Bratislava
7    Norway        Oslo
8    Croatia       Zagreb
9    Greece        Athens
10   Malta         Valletta
11   Italy         Rome
```

Data Set 4

**Countries**

```
Id   Country       Capital
3    Spain         Madrit
5    Poland        Warsaw
8    Croatia       Zagreb
10   Malta         Valletta
11   Italy         Rome
```

Data Set 5

**Countries**

```
Id   Country       Capital
6    Slovakia      Bratislava
7    Norway        Oslo
8    Croatia       Zagreb
10   Malta         Valletta
11   Italy         Rome
```

## 7.2.6 INSERT 06

Write a query that inserts into the Employees table a new employee Martin Socha with employee Id equals 12 and who has no JobTitleCode.

Data Set 1

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|

Data Set 2

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 2 | Jan | Nowak | RKB003 |
| 3 | Kamil | Wilmowski | RKB011 |
| 4 | Olga | Milenka | RKB003 |
| 5 | Angela | Wilga | RKB002 |

Data Set 3

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 3 | Kamil | Wilmowski | RKB011 |
| 5 | Angela | Wilga | RKB002 |

Data Set 4

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 3 | Kamil | Wilmowski | RKB011 |
| 4 | Paco | Ramirez | RKB011 |
| 5 | Angela | Wilga | RKB002 |
| 6 | Beatriz | Santana | RKB002 |

Data Set 5

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|

```
7   Joahim   Mesina      RKB001
8   Marta    Frantisek   RKB002
9   Jan      Matalau     RKB001
```

### 7.2.7 INSERT 07

Write a query that inserts into the Countries table the following countries with capitals:

Ireland, Dublin

Bulgaria, Sofia

The column Id has the autoincrement option set.

Data Set 1

**Countries**

```
Id  Country    Capital
```

Data Set 2

**Countries**

```
Id  Country      Capital
1   Portugal     Lisbon
2   Belgium      Brussels
3   Spain        Madrit
4   Hungary      Budapest
8   Croatia      Zagreb
9   Greece       Athens
10  Malta        Valletta
```

Data Set 3

**Countries**

```
Id  Country      Capital
3   Spain        Madrit
4   Hungary      Budapest
6   Slovakia     Bratislava
7   Norway       Oslo
8   Croatia      Zagreb
9   Greece       Athens
10  Malta        Valletta
```

```
11   Italy          Rome
```

Data Set 4

**Countries**

```
Id   Country        Capital
3    Spain          Madrit
5    Poland         Warsaw
8    Croatia        Zagreb
10   Malta          Valletta
11   Italy          Rome
```

Data Set 5

**Countries**

```
Id   Country        Capital
6    Slovakia       Bratislava
7    Norway         Oslo
8    Croatia        Zagreb
10   Malta          Valletta
11   Italy          Rome
```

### 7.2.8 INSERT 08

Write a query that inserts into the JobTitles table a new job title Seller Assistant with the JobTitleCode RKB115.

Data Set 1

**JobTitles**

```
JobTitleCode   JobTitle
```

Data Set 2

**JobTitles**

```
JobTitleCode   JobTitle
RKB012         Seller
RKB100         Shop Director
RKB001         Worker
RKB011         Manager
RKB002         Cashier
RKB003         Technician
```

Data Set 3

**JobTitles**

```
JobTitleCode  JobTitle
RKB001        Worker
RKB011        Manager
RKB002        Cashier
```

Data Set 4

**JobTitles**

```
JobTitleCode  JobTitle
RKB012        Seller
RKB100        Shop Director
RKB002        Cashier
RKB003        Technician
```

Data Set 5

**JobTitles**

```
JobTitleCode  JobTitle
RKB001        Worker
RKB011        Manager
RKB002        Cashier
RKB003        Technician
```

## 7.2.9 INSERT 09

Write a query that inserts into the EmployeesCopy table all rows from the Employees table without the position (JobTitleCode).

Data Set 1

**Employees**

```
Id  Name    Surname     JobTitleCode
```

**EmployeesCopy**

```
Id  Name    Surname     JobTitleCode
```

Data Set 2

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 2 | Jan | Nowak | RKB003 |
| 3 | Kamil | Wilmowski | RKB011 |
| 4 | Olga | Milenka | RKB003 |
| 5 | Angela | Wilga | RKB002 |

**EmployeesCopy**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|

Data Set 3

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 3 | Kamil | Wilmowski | RKB011 |
| 5 | Angela | Wilga | RKB002 |

**EmployeesCopy**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|

Data Set 4

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 3 | Kamil | Wilmowski | RKB011 |
| 4 | Paco | Ramirez | RKB011 |
| 5 | Angela | Wilga | RKB002 |
| 6 | Beatriz | Santana | RKB002 |

**EmployeesCopy**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|

Data Set 5

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 7 | Joahim | Mesina | RKB001 |

```
8   Marta     Frantisek   RKB002
9   Jan       Matalau     RKB001
```

**EmployeesCopy**

```
Id Name Surname JobTitleCode
```

## 7.2.10 INSERT 10

Write a query that inserts into the Countries table a new country Latvia with the capital Riga. The column Id has the autoincrement option set.

Data Set 1

**Countries**

```
Id  Country    Capital
```

Data Set 2

**Countries**

```
Id  Country      Capital
1   Portugal     Lisbon
2   Belgium      Brussels
3   Spain        Madrit
4   Hungary      Budapest
8   Croatia      Zagreb
9   Greece       Athens
10  Malta        Valletta
```

Data Set 3

**Countries**

```
Id  Country      Capital
3   Spain        Madrit
4   Hungary      Budapest
6   Slovakia     Bratislava
7   Norway       Oslo
8   Croatia      Zagreb
9   Greece       Athens
10  Malta        Valletta
11  Italy        Rome
```

Data Set 4

**Countries**

| Id | Country | Capital |
|---|---|---|
| 3 | Spain | Madrit |
| 5 | Poland | Warsaw |
| 8 | Croatia | Zagreb |
| 10 | Malta | Valletta |
| 11 | Italy | Rome |

Data Set 5

**Countries**

| Id | Country | Capital |
|---|---|---|
| 6 | Slovakia | Bratislava |
| 7 | Norway | Oslo |
| 8 | Croatia | Zagreb |
| 10 | Malta | Valletta |
| 11 | Italy | Rome |

## 7.3 INSERT exercise II.

### 7.3.1 INSERT 11

Write a query that inserts into the Countries table all cities from the Cities table. The column Id in the Countries table has an autoincrement option set.

Data Set 1

**Countries**

| Id | Country | Capital |
|---|---|---|

**Cities**

| Id | Name |
|---|---|

Data Set 2

**Countries**

| Id | Country | Capital |
|---|---|---|
| 1 | Portugal | Lisbon |
| 2 | Belgium | Brussels |

```
3   Spain         Madrit
4   Hungary       Budapest
8   Croatia       Zagreb
9   Greece        Athens
10  Malta         Valletta
```

**Cities**

```
Id  Name
1   Bratislava
2   Nitra
3   Trnava
4   Katowice
5   Telde
6   Paris
8   Galdar
10  Krakow
```

Data Set 3

**Countries**

```
Id  Country       Capital
3   Spain         Madrit
4   Hungary       Budapest
6   Slovakia      Bratislava
7   Norway        Oslo
8   Croatia       Zagreb
9   Greece        Athens
10  Malta         Valletta
11  Italy         Rome
```

**Cities**

```
Id  Name
2   Nitra
4   Katowice
6   Paris
8   Galdar
```

Data Set 4

**Countries**

```
Id  Country       Capital
3   Spain         Madrit
```

```
5   Poland      Warsaw
8   Croatia     Zagreb
10  Malta       Valletta
11  Italy       Rome
```

**Cities**

```
Id  Name
2   Nitra
3   Vienna
4   Katowice
5   Sevilla
8   Galdar
```

Data Set 5

**Countries**

```
Id  Country     Capital
6   Slovakia    Bratislava
7   Norway      Oslo
8   Croatia     Zagreb
10  Malta       Valletta
11  Italy       Rome
```

**Cities**

```
Id  Name
2   Nitra
4   Katowice
6   Paris
7   Madrit
9   Brussels
```

## 7.3.2 INSERT 12

Write a query that inserts into the JobTitles table two rows with JobTitleCodes: AKB500, AKB300.

Data Set 1

**JobTitles**

```
JobTitleCode  JobTitle
```

Data Set 2

**JobTitles**

```
JobTitleCode  JobTitle
RKB012        Seller
RKB100        Shop Director
RKB001        Worker
RKB011        Manager
RKB002        Cashier
RKB003        Technician
```

Data Set 3

**JobTitles**

```
JobTitleCode  JobTitle
RKB001        Worker
RKB011        Manager
RKB002        Cashier
```

Data Set 4

**JobTitles**

```
JobTitleCode  JobTitle
RKB012        Seller
RKB100        Shop Director
RKB002        Cashier
RKB003        Technician
```

Data Set 5

**JobTitles**

```
JobTitleCode  JobTitle
RKB001        Worker
RKB011        Manager
RKB002        Cashier
RKB003        Technician
```

## 7.3.3 INSERT 13

Write a query that inserts into the Customers table two customers Anna Nowak, Jan Nowak with Id equals 10 and 11 respectively and City_Id equals 2.

Data Set 1

**Customers**

```
Id  Name     Surname      City_Id
```

Data Set 2

**Customers**

```
Id  Name       Surname      City_Id
1   Sofia      Bednar       10
2   Jan        Zachar       1
3   Nela       Walach       1
4   Witold     Nowak        5
5   Katarzyna  Kowalska     6
6   Iwona      Bednarz      8
```

Data Set 3

**Customers**

```
Id  Name       Surname      City_Id
2   Jan        Zachar       2
4   Katarzyna  Kowalska     6
5   Witold     Nowak        4
6   Iwona      Bednarz      8
7   Katy       Bernard      8
```

Data Set 4

**Customers**

```
Id  Name       Surname    City_Id
2   Jan        Zachar     2
3   Jose       Sanchez    2
4   Witold     Nowak      3
5   Iwona      Bednarz    8
6   Katarzyna  Kowalska   5
```

Data Set 5

**Customers**

```
Id  Name         Surname    City_Id
8   Magdalena    Meredit    6
9   Mateus       Vettel     7
```

```
10  Fryderyk   Hanke      9
```

### 7.3.4 INSERT 14

Write a query that inserts into the Employees table a new employee Martin Nowak with JobTitleCode RKB012 and Id equals 11.

Data Set 1

**Employees**

```
Id  Name     Surname      JobTitleCode
```

Data Set 2

**Employees**

```
Id  Name     Surname      JobTitleCode
1   Jan      Kowalski     RKB012
2   Jan      Nowak        RKB003
3   Kamil    Wilmowski    RKB011
4   Olga     Milenka      RKB003
5   Angela   Wilga        RKB002
```

Data Set 3

**Employees**

```
Id  Name     Surname      JobTitleCode
1   Jan      Kowalski     RKB012
3   Kamil    Wilmowski    RKB011
5   Angela   Wilga        RKB002
```

Data Set 4

**Employees**

```
Id  Name      Surname      JobTitleCode
1   Jan       Kowalski     RKB012
3   Kamil     Wilmowski    RKB011
4   Paco      Ramirez      RKB011
5   Angela    Wilga        RKB002
6   Beatriz   Santana      RKB002
```

Data Set 5

**Employees**

```
Id  Name      Surname      JobTitleCode
7   Joahim    Mesina       RKB001
8   Marta     Frantisek    RKB002
9   Jan       Matalau      RKB001
```

## 7.3.5 INSERT 15

Write a query that inserts into the Customers table a new customer Ala Nowak from city name Nitra (which corresponds to City_Id value equals 2) and customer Id is equal to 15.

Data Set 1

**Customers**

```
Id  Name     Surname      City_Id
```

**Cities**

```
Id  Name
2 Nitra
```

Data Set 2

**Customers**

```
Id  Name       Surname      City_Id
1   Sofia      Bednar       10
2   Jan        Zachar       1
3   Nela       Walach       1
4   Witold     Nowak        5
5   Katarzyna  Kowalska     6
6   Iwona      Bednarz      8
```

**Cities**

```
Id  Name
1   Bratislava
2   Nitra
3   Trnava
4   Katowice
5   Telde
6   Paris
```

```
8   Galdar
10  Krakow
```

Data Set 3

**Customers**

```
Id  Name       Surname      City_Id
2   Jan        Zachar       2
4   Katarzyna  Kowalska     6
5   Witold     Nowak        4
6   Iwona      Bednarz      8
7   Bernard    Katy         8
```

**Cities**

```
Id  Name
2   Nitra
4   Katowice
6   Paris
8   Galdar
```

Data Set 4

**Customers**

```
Id  Name       Surname    City_Id
2   Jan        Zachar     2
3   Jose       Sanchez    2
4   Witold     Nowak      3
5   Iwona      Bednarz    8
6   Katarzyna  Kowalska   5
```

**Cities**

```
Id  Name
2   Nitra
3   Vienna
4   Katowice
5   Sevilla
8   Galdar
```

Data Set 5

**Customers**

```
Id   Name        Surname    City_Id
8    Magdalena   Meredit    6
9    Mateus      Vettel     7
10   Fryderyk    Hanke      9
```

**Cities**

```
Id   Name
2    Nitra
4    Katowice
6    Paris
8    Galdar
```

## 7.3.6 INSERT 16

Write a query that inserts into the Countries table (column Capital) the following cities: Frankfurt, Porto. The column Id has an autoincrement option set.

Data Set 1

**Countries**

```
Id   Country     Capital
```

Data Set 2

**Countries**

```
Id   Country       Capital
1    Portugal      Lisbon
2    Belgium       Brussels
3    Spain         Madrit
4    Hungary       Budapest
8    Croatia       Zagreb
9    Greece        Athens
10   Malta         Valletta
```

Data Set 3

**Countries**

```
Id   Country       Capital
3    Spain         Madrit
4    Hungary       Budapest
6    Slovakia      Bratislava
```

```
7   Norway      Oslo
8   Croatia     Zagreb
9   Greece      Athens
10  Malta       Valletta
11  Italy       Rome
```

Data Set 4

**Countries**

```
Id  Country     Capital
3   Spain       Madrit
5   Poland      Warsaw
8   Croatia     Zagreb
10  Malta       Valletta
11  Italy       Rome
```

Data Set 5

**Countries**

```
Id  Country     Capital
6   Slovakia    Bratislava
7   Norway      Oslo
8   Croatia     Zagreb
10  Malta       Valletta
11  Italy       Rome
```

## 7.3.7 INSERT 17

Write a query that inserts into the CustomersCopy table all customers from the Customers table which are from city name Galdar (which corresponds to City_Id equals 8).

Data Set 1

**Customers**

```
Id  Name    Surname     City_Id
```

**Cities**

```
Id  Name
```

**CustomersCopy**

```
Id  Name     Surname      City_Id
```

Data Set 2

**Customers**

```
Id  Name       Surname      City_Id
1   Sofia      Bednar       10
2   Jan        Zachar       1
3   Nela       Walach       1
4   Witold     Nowak        5
5   Katarzyna Kowalska     6
6   Iwona      Bednarz      8
```

**Cities**

```
Id  Name
1   Bratislava
2   Nitra
3   Trnava
4   Katowice
5   Telde
6   Paris
8   Galdar
10  Krakow
```

**CustomersCopy**

```
Id  Name     Surname      City_Id
```

Data Set 3

**Customers**

```
Id  Name       Surname      City_Id
2   Jan        Zachar       2
4   Katarzyna Kowalska     6
5   Witold     Nowak        4
6   Iwona      Bednarz      8
7   Bernard    Katy         8
```

**Cities**

```
Id  Name
2   Nitra
4   Katowice
```

```
6   Paris
8   Galdar
```

**CustomersCopy**

```
Id  Name     Surname      City_Id
```

Data Set 4

**Customers**

```
Id  Name       Surname    City_Id
2   Jan        Zachar     2
3   Jose       Sanchez    2
4   Witold     Nowak      3
5   Iwona      Bednarz    8
6   Katarzyna  Kowalska   5
```

**Cities**

```
Id  Name
2   Nitra
3   Vienna
4   Katowice
5   Sevilla
8   Galdar
```

**CustomersCopy**

```
Id  Name     Surname      City_Id
```

Data Set 5

**Customers**

```
Id  Name         Surname    City_Id
8   Magdalena    Meredit    6
9   Mateus       Vettel     7
10  Fryderyk     Hanke      9
```

**Cities**

```
Id  Name
2   Nitra
4   Katowice
6   Paris
7   Madrit
```

```
9   Brussels
```

**CustomersCopy**

```
Id  Name    Surname   City_Id
```

## 7.3.8 INSERT 18

Write a query that inserts into the EmployeesCopy table all employees from the Employees table whose Name begins with the letter J.

Data Set 1

**Employees**

```
Id  Name    Surname     JobTitleCode
```

**EmployeesCopy**

```
Id  Name    Surname     JobTitleCode
```

Data Set 2

**Employees**

```
Id  Name    Surname     JobTitleCode
1   Jan     Kowalski    RKB012
2   Jan     Nowak       RKB003
3   Kamil   Wilmowski   RKB011
4   Olga    Milenka     RKB003
5   Angela  Wilga       RKB002
```

**EmployeesCopy**

```
Id  Name    Surname     JobTitleCode
```

Data Set 3

**Employees**

```
Id  Name    Surname     JobTitleCode
1   Jan     Kowalski    RKB012
3   Kamil   Wilmowski   RKB011
5   Angela  Wilga       RKB002
```

**EmployeesCopy**

```
Id  Name     Surname      JobTitleCode
```

Data Set 4

**Employees**

```
Id  Name      Surname      JobTitleCode
1   Jan       Kowalski     RKB012
3   Kamil     Wilmowski    RKB011
4   Paco      Ramirez      RKB011
5   Angela    Wilga        RKB002
6   Beatriz   Santana      RKB002
```

**EmployeesCopy**

```
Id  Name     Surname      JobTitleCode
```

Data Set 5

**Employees**

```
Id  Name      Surname      JobTitleCode
7   Joahim    Mesina       RKB001
8   Marta     Frantisek    RKB002
9   Jan       Matalau      RKB001
```

**EmployeesCopy**

```
Id Name Surname JobTitleCode
```

## 7.3.9 INSERT 19

Write a query that inserts into the EmployeesCopy table all employees from the Employees table whose Name ends with the letter a.

Data Set 1

**Employees**

```
Id  Name     Surname      JobTitleCode
```

**EmployeesCopy**

```
Id  Name     Surname      JobTitleCode
```

Data Set 2

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 2 | Jan | Nowak | RKB003 |
| 3 | Kamil | Wilmowski | RKB011 |
| 4 | Olga | Milenka | RKB003 |
| 5 | Angela | Wilga | RKB002 |

**EmployeesCopy**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|

Data Set 3

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 3 | Kamil | Wilmowski | RKB011 |
| 5 | Angela | Wilga | RKB002 |

**EmployeesCopy**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|

Data Set 4

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 3 | Kamil | Wilmowski | RKB011 |
| 4 | Paco | Ramirez | RKB011 |
| 5 | Angela | Wilga | RKB002 |
| 6 | Beatriz | Santana | RKB002 |

**EmployeesCopy**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|

Data Set 5

**Employees**

```
Id  Name      Surname      JobTitleCode
7   Joahim    Mesina       RKB001
8   Marta     Frantisek    RKB002
9   Jan       Matalau      RKB001
```

**EmployeesCopy**

```
Id Name Surname JobTitleCode
```

## 7.3.10 INSERT 20

Write a query that using insert into statement copies all rows from the Cities table into the CitiesCopy table where city Name begins with the letter N.

Data Set 1

**Cities**

```
Id  Name
```

**CitiesCopy**

```
Id  Name
```

Data Set 2

**Cities**

```
Id  Name
1   Bratislava
2   Nitra
3   Trnava
4   Katowice
5   Telde
6   Paris
8   Galdar
10  Krakow
```

**CitiesCopy**

```
Id  Name
```

Data Set 3

**Cities**

```
Id  Name
2   Nitra
4   Katowice
6   Paris
8   Galdar
```

**CitiesCopy**

```
Id  Name
```

Data Set 4

**Cities**

```
Id  Name
2   Nitra
3   Vienna
4   Katowice
5   Sevilla
8   Galdar
```

**CitiesCopy**

```
Id  Name
```

Data Set 5

**Cities**

```
Id  Name
2   Nitra
4   Katowice
6   Paris
7   Madrit
9   Brussels
```

**CitiesCopy**

```
Id  Name
```

# Update

Chapter 8

# 8.1 UPDATE statement

## 📖 8.1.1

The UPDATE statement is used to modify (update) the data contained in a table. The basic form of this statement is the following:

```
UPDATE table_name
SET column_name=value;
```

It will set new values to the particular column. It is possible to update values in more than one column. The statement is the following:

```
UPDATE table_name
SET column1_name=value1, column2_name=value2, …,
columnN_name=valueN;
```

## 📖 8.1.2

The UPDATE statement can be combined with the WHERE clause which contains a condition to select rows for which the values of columns needs to be updated. It is possible to combine many conditions using, for example, the AND or the OR operators.

```
UPDATE table_name
SET column1_name=value1, column2_name=value2
WHERE condition;
```

## 📝 8.1.3

Which statement will you use to obtain the result presented in a picture - Countries table:

| CountryID | CountryName | CountryCapital |
|---|---|---|
| 1 | Poland | London |
| 2 | Belgium | London |
| 3 | Spain | London |
| 4 | Slovakia | London |

- UPDATE Countries SET CountryCapital = 'London';
- UPDATE Countries SET CountryCapital = 'London' where CountryName='Spain';
- UPDATE Countries SET CountryName='Poland', CountryCapital = 'London';

## 8.1.4

You want to update data in the Countries table with the following schema:

```
Countries (CountryID INTEGER PRIMARY, CountryName VARCHAR
(50), CountryCapital VARCHAR (50))
```

Set the value Krakow in the CountryCapital column where CountryName is Poland. Create an appropriate statement.

```
_____ Countries SET _____='Krakow' _____ CountryName='Poland';
```

- CountryCapital
- WHERE
- UPDATE

## 8.1.5

You want to update data in the Products table with the following schema:

```
Products (ProductID INTEGER PRIMARY KEY, ProductName VARCHAR
(50), Category VARCHAR (50))
```

Set the value Car in the ProductName column for all rows where the Category is Toy. Which of the following statements is incorrect?

- UPDATE Products SET ProductName='Car' where Category='Toy';
- UPDATE Products SET ProductName=’Car’;
- UPDATE Products SET Category=’Car’;
- Update Products;

## 8.1.6

You want to update data in the Customers table with the following schema:

```
Customers (CustomerID INTEGER PRIMARY KEY, CustomerName
VARCHAR (50), CustomerSurname VARCHAR (50), CustomerSex
BOOLEAN);
```

Is the UPDATE statement correct?

```
UPDATE Customers SET Name='Ala' WHERE CustomerSurname
='Nowak';
```

- False
- True

### 8.1.7

Update data values in the Countries table. Set, for all rows, value to Slovakia in the CountryName column.

```
_____ _____ SET CountryName=' _____ ';
```

- Slovakia
- Countries
- INSERT
- UPDATE

## 8.2 UPDATE exercise I.

### 8.2.1 UPDATE 01

Write a query that modifies a city name to Gdansk, from the Cities table, where city Id is equal to 4.

Data Set 1

**Cities**

```
Id   Name
```

Data Set 2

**Cities**

```
Id  Name
1   Bratislava
2   Nitra
3   Trnava
4   Katowice
5   Telde
6   Paris
8   Galdar
10  Krakow
```

Data Set 3

**Cities**

```
Id  Name
2   Nitra
4   Katowice
6   Paris
8   Galdar
```

Data Set 4

**Cities**

```
Id  Name
2   Nitra
3   Vienna
4   Katowice
5   Sevilla
8   Galdar
```

Data Set 5

**Cities**

```
Id Name
2   Nitra
4   Katowice
6   Paris
7   Madrit
9   Brussels
```

### ⌨ 8.2.2 UPDATE 02

Write a query that modifies a city name Nitra to Porto from the Cities table.

Data Set 1

**Cities**

```
Id   Name
```

Data Set 2

**Cities**

```
Id  Name
1   Bratislava
```

```
2   Nitra
3   Trnava
4   Katowice
5   Telde
6   Paris
8   Galdar
10  Krakow
```

Data Set 3

**Cities**

```
Id  Name
2   Nitra
4   Katowice
6   Paris
8   Galdar
```

Data Set 4

**Cities**

```
Id  Name
2   Nitra
3   Vienna
4   Katowice
5   Sevilla
8   Galdar
```

Data Set 5

**Cities**

```
Id Name
2   Nitra
4   Katowice
6   Paris
7   Madrit
9   Brussels
```

## 8.2.3 UPDATE 03

Write a query that modifies a Country name to Austria and a Capital name to Vienna, from the Countries table, where country Id equals 9.

Data Set 1

**Countries**

```
Id  Country    Capital
```

Data Set 2

**Countries**

```
Id  Country      Capital
1   Portugal     Lisbon
2   Belgium      Brussels
3   Spain        Madrit
4   Hungary      Budapest
8   Croatia      Zagreb
9   Greece       Athens
10  Malta        Valletta
```

Data Set 3

**Countries**

```
Id  Country      Capital
3   Spain        Madrit
4   Hungary      Budapest
6   Slovakia     Bratislava
7   Norway       Oslo
8   Croatia      Zagreb
9   Greece       Athens
10  Malta        Valletta
11  Italy        Rome
```

Data Set 4

**Countries**

```
Id  Country      Capital
3   Spain        Madrit
5   Poland       Warsaw
8   Croatia      Zagreb
10  Malta        Valletta
11  Italy        Rome
```

Data Set 5

**Countries**

```
Id  Country      Capital
6   Slovakia     Bratislava
7   Norway       Oslo
8   Croatia      Zagreb
10  Malta        Valletta
11  Italy        Rome
```

## ⌨ 8.2.4 UPDATE 04

Write a query that modifies a Capital name to Krakow, from the Countries table, where the Country name is Poland.

Data Set 1

Countries

```
Id  Country    Capital
```

Data Set 2

Countries

```
Id  Country      Capital
1   Portugal     Lisbon
2   Belgium      Brussels
3   Spain        Madrit
4   Hungary      Budapest
8   Croatia      Zagreb
9   Greece       Athens
10  Malta        Valletta
```

Data Set 3

Countries

```
Id  Country      Capital
3   Spain        Madrit
4   Hungary      Budapest
6   Slovakia     Bratislava
7   Norway       Oslo
8   Croatia      Zagreb
9   Greece       Athens
10  Malta        Valletta
```

```
11   Italy          Rome
```

Data Set 4

Countries

```
Id   Country        Capital
3    Spain          Madrit
5    Poland         Warsaw
8    Croatia        Zagreb
10   Malta          Valletta
11   Italy          Rome
```

Data Set 5

Countries

```
Id   Country        Capital
6    Slovakia       Bratislava
7    Norway         Oslo
8    Croatia        Zagreb
10   Malta          Valletta
11   Italy          Rome
```

## 8.2.5 UPDATE 05

Write a query that modifies the JobTitleCode to RKB002, from the Employees table, all employees whose Surname ends with the letter a.

Data Set 1

**Employees**

```
Id    Name
```

Data Set 2

**Employees**

```
Id   Name      Surname       JobTitleCode
1    Jan       Kowalski      RKB012
2    Jan       Nowak         RKB003
3    Kamil     Wilmowski     RKB011
4    Olga      Milenka       RKB003
5    Angela    Wilga         RKB002
```

Data Set 3

**Employees**

```
Id  Name    Surname     JobTitleCode
1   Jan     Kowalski    RKB012
3   Kamil   Wilmowski   RKB011
5   Angela  Wilga       RKB002
```

Data Set 4

**Employees**

```
Id  Name      Surname     JobTitleCode
1   Jan       Kowalski    RKB012
3   Kamil     Wilmowski   RKB011
4   Paco      Ramirez     RKB011
5   Angela    Wilga       RKB002
6   Beatriz   Santana     RKB002
```

Data Set 5

**Employees**

```
Id  Name      Surname     JobTitleCode
7   Joahim    Mesina      RKB001
8   Marta     Frantisek   RKB002
9   Jan       Matalu      RKB001
```

## 8.2.6 UPDATE 06

Write a query that modifies a JobTitle to Manager Assistant, from the JobTitles table, where JobTitleCode is RKB001.

Data Set 1

**JobTitles**

```
JobTitleCode  JobTitle
```

Data Set 2

**JobTitles**

```
JobTitleCode  JobTitle
RKB012        Seller
```

```
RKB100          Shop Director
RKB001          Worker
RKB011          Manager
RKB002          Cashier
RKB003          Technician
```

Data Set 3

**JobTitles**

```
JobTitleCode    JobTitle
RKB001          Worker
RKB011          Manager
RKB002          Cashier
```

Data Set 4

**JobTitles**

```
JobTitleCode    JobTitle
RKB012          Seller
RKB100          Shop Director
RKB002          Cashier
RKB003          Technician
```

Data Set 5

**JobTitles**

```
JobTitleCode    JobTitle
RKB001          Worker
RKB011          Manager
RKB002          Cashier
RKB003          Technician
```

## 8.2.7 UPDATE 07

Write a query that modifies JobTitleCode to RKB011 for all Employees who are working as Sellers.

Data Set 1

**Employees**

```
Id   Name      Surname       JobTitleCode
```

**JobTitles**

```
JobTitleCode  JobTitle
```

Data Set 2

**Employees**

```
Id  Name    Surname     JobTitleCode
1   Jan     Kowalski    RKB012
2   Jan     Nowak       RKB003
3   Kamil   Wilmowski   RKB011
4   Olga    Milenka     RKB003
5   Angela  Wilga       RKB002
```

**JobTitles**

```
JobTitleCode  JobTitle
RKB012        Seller
RKB100        Shop Director
RKB001        Worker
RKB011        Manager
RKB002        Cashier
RKB003        Technician
```

Data Set 3

**Employees**

```
Id  Name    Surname     JobTitleCode
1   Jan     Kowalski    RKB012
3   Kamil   Wilmowski   RKB011
5   Angela  Wilga       RKB002
```

**JobTitles**

```
JobTitleCode  JobTitle
RKB001        Worker
RKB012        Seller
RKB011        Manager
RKB002        Cashier
```

Data Set 4

**Employees**

```
Id   Name      Surname      JobTitleCode
1    Jan       Kowalski     RKB012
3    Kamil     Wilmowski    RKB011
4    Paco      Ramirez      RKB011
5    Angela    Wilga        RKB002
6    Beatriz   Santana      RKB002
```

**JobTitles**

```
JobTitleCode  JobTitle
RKB012        Seller
RKB100        Shop Director
RKB002        Cashier
RKB003        Technician
RKB011        Manager
```

Data Set 5

**Employees**

```
Id   Name      Surname      JobTitleCode
7    Joahim    Mesina       RKB001
8    Marta     Frantisek    RKB002
9    Jan       Matalau      RKB001
```

**JobTitles**

```
JobTitleCode  JobTitle
RKB001        Worker
RKB011        Manager
RKB002        Cashier
RKB003        Technician
```

## ⌨ 8.2.8 UPDATE 08

Write a query that modifies data all employees whose Name is Jan to Joahim, from the Employees table.

Data Set 1

**Employees**

```
Id    Name
```

Data Set 2

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 2 | Jan | Nowak | RKB003 |
| 3 | Kamil | Wilmowski | RKB011 |
| 4 | Olga | Milenka | RKB003 |
| 5 | Angela | Wilga | RKB002 |

Data Set 3

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 3 | Kamil | Wilmowski | RKB011 |
| 5 | Angela | Wilga | RKB002 |

Data Set 4

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 3 | Kamil | Wilmowski | RKB011 |
| 4 | Paco | Ramirez | RKB011 |
| 5 | Angela | Wilga | RKB002 |
| 6 | Beatriz | Santana | RKB002 |

Data Set 5

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 7 | Joahim | Mesina | RKB001 |
| 8 | Marta | Frantisek | RKB002 |
| 9 | Jan | Matalu | RKB001 |

## ⌨ 8.2.9 UPDATE 09

Write a query that modifies an employee Name to Ola all employees who are working as a Cashier.

Data Set 1

**Employees**

```
Id  Name    Surname     JobTitleCode
```

**JobTitles**

```
JobTitleCode  JobTitle
```

Data Set 2

**Employees**

```
Id  Name    Surname     JobTitleCode
1   Jan     Kowalski    RKB012
2   Jan     Nowak       RKB003
3   Kamil   Wilmowski   RKB011
4   Olga    Milenka     RKB003
5   Angela  Wilga       RKB002
```

**JobTitles**

```
JobTitleCode  JobTitle
RKB012        Seller
RKB100        Shop Director
RKB001        Worker
RKB011        Manager
RKB002        Cashier
RKB003        Technician
```

Data Set 3

**Employees**

```
Id  Name    Surname     JobTitleCode
1   Jan     Kowalski    RKB012
3   Kamil   Wilmowski   RKB011
5   Angela  Wilga       RKB002
JobTitleCode  JobTitle
RKB001        Worker
RKB012        Seller
RKB011        Manager
RKB002        Cashier
```

Data Set 4

**Employees**

```
Id   Name      Surname      JobTitleCode
1    Jan       Kowalski     RKB012
3    Kamil     Wilmowski    RKB011
4    Paco      Ramirez      RKB011
5    Angela    Wilga        RKB002
6    Beatriz   Santana      RKB002
```

**JobTitles**

```
JobTitleCode  JobTitle
RKB012        Seller
RKB100        Shop Director
RKB002        Cashier
RKB003        Technician
RKB011        Manager
```

Data Set 5

**Employees**

```
Id   Name      Surname      JobTitleCode
7    Joahim    Mesina       RKB001
8    Marta     Frantisek    RKB002
9    Jan       Matalau      RKB001
```

**JobTitles**

```
JobTitleCode  JobTitle
RKB001        Worker
RKB011        Manager
RKB002        Cashier
RKB003        Technician
```

## 8.2.10 UPDATE 10

Write a query that modifies data of an employee Jan Kowalski, from the Employees table to Janina Kowalska and JobTitleCode to RKB011.

Data Set 1

**Employees**

```
Id    Name
```

Data Set 2

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 2 | Jan | Nowak | RKB003 |
| 3 | Kamil | Wilmowski | RKB011 |
| 4 | Olga | Milenka | RKB003 |
| 5 | Angela | Wilga | RKB002 |

Data Set 3

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 3 | Kamil | Wilmowski | RKB011 |
| 5 | Angela | Wilga | RKB002 |

Data Set 4

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 3 | Kamil | Wilmowski | RKB011 |
| 4 | Paco | Ramirez | RKB011 |
| 5 | Angela | Wilga | RKB002 |
| 6 | Beatriz | Santana | RKB002 |

Data Set 5

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 7 | Joahim | Mesina | RKB001 |
| 8 | Marta | Frantisek | RKB002 |
| 9 | Jan | Matalu | RKB001 |

# 8.3 UPDATE exercise II.

## ⌨ 8.3.1 UPDATE 11

Write a query that modifies country name to Estonia and capital to Tallinn from the Countries table.

Data Set 1

**Countries**

```
Id  Country    Capital
```

Data Set 2

**Countries**

```
Id  Country      Capital
1   Portugal     Lisbon
2   Belgium      Brussels
3   Spain        Madrit
4   Hungary      Budapest
8   Croatia      Zagreb
9   Greece       Athens
10  Malta        Valletta
```

Data Set 3

**Countries**

```
Id  Country      Capital
3   Spain        Madrit
4   Hungary      Budapest
6   Slovakia     Bratislava
7   Norway       Oslo
8   Croatia      Zagreb
9   Greece       Athens
10  Malta        Valletta
11  Italy        Rome
```

Data Set 4

**Countries**

```
Id  Country      Capital
3   Spain        Madrit
5   Poland       Warsaw
8   Croatia      Zagreb
10  Malta        Valletta
11  Italy        Rome
```

Data Set 5

**Countries**

```
Id  Country      Capital
6   Slovakia     Bratislava
7   Norway       Oslo
8   Croatia      Zagreb
10  Malta        Valletta
11  Italy        Rome
```

## 8.3.2 UPDATE 12

Write a query that modifies a city Name to London, from the Cities table, where city Names begins with the letter N.

Data Set 1

**Cities**

```
Id   Name
```

Data Set 2

**Cities**

```
Id  Name
1   Bratislava
2   Nitra
3   Trnava
4   Katowice
5   Telde
8   Galdar
```

Data Set 3

**Cities**

```
Id  Name
2   Nitra
4   Katowice
8   Galdar
```

Data Set 4

**Cities**

```
Id  Name
```

```
2   Nitra
3   Vienna
4   Katowice
5   Sevilla
8   Galdar
```

Data Set 5

**Cities**

```
Id  Name
2   Nitra
4   Katowice
6   Paris
7   Madrit
9   Brussels
```

## 8.3.3 UPDATE 13

Write a query that modifies City_Id to 4 from the Customers table, all customers from the Nitra city. Write a query using SQL subquery.

Data Set 1

**Customers**

```
Id  Name    Surname     City_Id
```

**Cities**

```
Id  Name
```

Data Set 2

**Customers**

```
Id  Name       Surname     City_Id
1   Sofia      Bednar      10
2   Jan        Zachar      1
3   Nela       Walach      1
4   Witold     Nowak       5
5   Katarzyna  Kowalska    6
6   Iwona      Bednarz     8
```

**Cities**

```
Id  Name
1   Bratislava
2   Nitra
3   Trnava
4   Katowice
5   Telde
6   Paris
8   Galdar
10  Krakow
```

Data Set 3

**Customers**

```
Id  Name       Surname      City_Id
2   Jan        Zachar       2
4   Katarzyna  Kowalska     6
5   Witold     Nowak        4
6   Iwona      Bednarz      8
7   Katy       Bernard      8
```

**Cities**

```
Id  Name
2   Nitra
4   Katowice
6   Paris
8   Galdar
```

Data Set 4

**Customers**

```
Id  Name       Surname    City_Id
2   Jan        Zachar     2
3   Jose       Sanchez    2
4   Witold     Nowak      3
5   Iwona      Bednarz    8
6   Katarzyna  Kowalska   5
```

**Cities**

```
Id  Name
2   Nitra
3   Vienna
4   Katowice
```

```
5   Sevilla
8   Galdar
```

Data Set 5

**Customers**

```
Id  Name        Surname   City_Id
8   Magdalena   Meredit   6
9   Mateus      Vettel    7
10  Fryderyk    Hanke     9
```

**Cities**

```
Id  Name
2   Nitra
4   Katowice
6   Paris
7   Madrit
9   Brussels
```

### 8.3.4 UPDATE 14

Write a query that modifies the customer Name to Ola, all customers from the city which ends with the letter s. Write a query using SQL subquery.

Data Set 1

**Customers**

```
Id  Name    Surname     City_Id
```

**Cities**

```
Id  Name
```

Data Set 2

**Customers**

```
Id  Name       Surname     City_Id
1   Sofia      Bednar      10
2   Jan        Zachar      1
3   Nela       Walach      1
4   Witold     Nowak       5
5   Katarzyna  Kowalska    6
```

```
6    Iwona      Bednarz      8
```

**Cities**

```
Id   Name
1    Bratislava
2    Nitra
3    Trnava
4    Katowice
5    Telde
6    Paris
8    Galdar
10   Krakow
```

Data Set 3

**Customers**

```
Id   Name       Surname      City_Id
2    Jan        Zachar       2
4    Katarzyna  Kowalska     6
5    Witold     Nowak        4
6    Iwona      Bednarz      8
7    Katy       Bernard      8
```

**Cities**

```
Id   Name
2    Nitra
4    Katowice
6    Paris
8    Galdar
```

Data Set 4

**Customers**

```
Id   Name       Surname    City_Id
2    Jan        Zachar     2
3    Jose       Sanchez    2
4    Witold     Nowak      3
5    Iwona      Bednarz    8
6    Katarzyna  Kowalska   5
```

**Cities**

```
Id  Name
2   Nitra
3   Vienna
4   Katowice
5   Sevilla
8   Galdar
```

Data Set 5

**Customers**

```
Id  Name        Surname   City_Id
8   Magdalena   Meredit   6
9   Mateus      Vettel    7
10  Fryderyk    Hanke     9
```

**Cities**

```
Id  Name
2   Nitra
4   Katowice
6   Paris
7   Madrit
9   Brussels
```

## ⌨ 8.3.5 UPDATE 15

Write a query that modifies a customer City_Id to 2, from the Customers table, where customer Name begins with the letter K.

Data Set 1

**Customers**

```
Id Name        Surname    City_Id
```

Data Set 2

**Customers**

```
Id Name        Surname    City_Id
1  Sofia       Bednar     10
2  Jan         Zachar     1
3  Nela        Walach     1
4  Witold      Nowak      5
```

```
5  Katarzyna  Kowalska   6
6  Iwona      Bednarz    8
```

Data Set 3

**Customers**

```
Id Name        Surname    City_Id
2  Jan         Zachar     2
4  Katarzyna   Kowalska   6
5  Witold      Nowak      4
6  Iwona       Bednarz    8
7  Katy        Bernard    8
```

Data Set 4

**Customers**

```
Id Name       Surname    City_Id
2  Jan        Zachar     2
3  Jose       Sanchez    2
4  Witold     Nowak      3
5  Iwona      Bednarz    8
6  Katarzyna Kowalska   5
```

Data Set 5

**Customers**

```
Id Name        Surname    City_Id
8  Magdalena   Meredit    6
9  Mateus      Vettel     7
10 Fryderyk    Hanke      9
```

## ⌨ 8.3.6 UPDATE 16

Write a query that modifies customers Names to Anna, from the Customers table, where customers are from the city which begins with the letter G. Write a query using SQL subquery.

Data Set 1

**Customers**

```
Id  Name     Surname      City_Id
```

**Cities**

| Id | Name |
|---|---|

Data Set 2

**Customers**

| Id | Name | Surname | City_Id |
|---|---|---|---|
| 1 | Sofia | Bednar | 10 |
| 2 | Jan | Zachar | 1 |
| 3 | Nela | Walach | 1 |
| 4 | Witold | Nowak | 5 |
| 5 | Katarzyna | Kowalska | 6 |
| 6 | Iwona | Bednarz | 8 |

**Cities**

| Id | Name |
|---|---|
| 1 | Bratislava |
| 2 | Nitra |
| 3 | Trnava |
| 4 | Katowice |
| 5 | Telde |
| 6 | Paris |
| 8 | Galdar |
| 10 | Krakow |

Data Set 3

**Customers**

| Id | Name | Surname | City_Id |
|---|---|---|---|
| 2 | Jan | Zachar | 2 |
| 4 | Katarzyna | Kowalska | 6 |
| 5 | Witold | Nowak | 4 |
| 6 | Iwona | Bednarz | 8 |
| 7 | Bernard | Katy | 8 |

**Cities**

| Id | Name |
|---|---|
| 2 | Nitra |
| 4 | Katowice |
| 6 | Paris |
| 8 | Galdar |

Data Set 4

**Customers**

```
Id  Name       Surname    City_Id
2   Jan        Zachar     2
3   Jose       Sanchez    2
4   Witold     Nowak      3
5   Iwona      Bednarz    8
6   Katarzyna  Kowalska   5
```

**Cities**

```
Id  Name
2   Nitra
3   Vienna
4   Katowice
5   Sevilla
8   Galdar
```

Data Set 5

**Customers**

```
Id  Name         Surname    City_Id
8   Magdalena    Meredit    6
9   Mateus       Vettel     7
10  Fryderyk     Hanke      9
```

**Cities**

```
Id  Name
2   Nitra
4   Katowice
6   Paris
7   Madrit
9   Brussels
```

## 8.3.7 UPDATE 17

Write a query that modifies an employee Surname to Wieluch, from the Employees table, where employee JobTitle is Seller. Write a query using SQL subquery.

Data Set 1

**Employees**

```
Id  Name    Surname     JobTitleCode
```

**JobTitles**

```
JobTitleCode  JobTitle
```

Data Set 2

**Employees**

```
Id  Name    Surname     JobTitleCode
1   Jan     Kowalski    RKB012
2   Jan     Nowak       RKB003
3   Kamil   Wilmowski   RKB011
4   Olga    Milenka     RKB003
5   Angela  Wilga       RKB002
```

**JobTitles**

```
JobTitleCode  JobTitle
RKB012        Seller
RKB100        Shop Director
RKB001        Worker
RKB011        Manager
RKB002        Cashier
RKB003        Technician
```

Data Set 3

**Employees**

```
Id  Name    Surname     JobTitleCode
1   Jan     Kowalski    RKB012
3   Kamil   Wilmowski   RKB011
5   Angela  Wilga       RKB002
```

**JobTitles**

```
JobTitleCode  JobTitle
RKB001        Worker
RKB012        Seller
RKB011        Manager
RKB002        Cashier
```

Data Set 4

**Employees**

```
Id   Name      Surname      JobTitleCode
1    Jan       Kowalski     RKB012
3    Kamil     Wilmowski    RKB011
4    Paco      Ramirez      RKB011
5    Angela    Wilga        RKB002
6    Beatriz   Santana      RKB002
```

**JobTitles**

```
JobTitleCode  JobTitle
RKB012        Seller
RKB100        Shop Director
RKB002        Cashier
RKB003        Technician
RKB011        Manager
```

Data Set 5

**Employees**

```
Id   Name      Surname      JobTitleCode
7    Joahim    Mesina       RKB001
8    Marta     Frantisek    RKB002
9    Jan       Matalau      RKB001
```

**JobTitles**

```
JobTitleCode  JobTitle
RKB001        Worker
RKB011        Manager
RKB002        Cashier
RKB003        Technician
```

## ⌨ 8.3.8 UPDATE 18

Write a query that modifies a JobTitleCode to RKB011 for all employees whose Id is less than 2.

**Employees**

```
Id  Name    Surname     JobTitleCode
```

**JobTitles**

```
JobTitleCode  JobTitle
```

Data Set 2

**Employees**

```
Id  Name    Surname     JobTitleCode
1   Jan     Kowalski    RKB012
2   Jan     Nowak       RKB003
3   Kamil   Wilmowski   RKB011
4   Olga    Milenka     RKB003
5   Angela  Wilga       RKB002
```

**JobTitles**

```
JobTitleCode  JobTitle
RKB012        Seller
RKB100        Shop Director
RKB001        Worker
RKB011        Manager
RKB002        Cashier
RKB003        Technician
```

Data Set 3

**Employees**

```
Id  Name    Surname     JobTitleCode
1   Jan     Kowalski    RKB012
3   Kamil   Wilmowski   RKB011
5   Angela  Wilga       RKB002
```

**JobTitles**

```
JobTitleCode  JobTitle
RKB001        Worker
RKB012        Seller
RKB011        Manager
RKB002        Cashier
```

Data Set 4

**Employees**

```
Id   Name      Surname      JobTitleCode
1    Jan       Kowalski     RKB012
3    Kamil     Wilmowski    RKB011
4    Paco      Ramirez      RKB011
5    Angela    Wilga        RKB002
6    Beatriz   Santana      RKB002
```

**JobTitles**

```
JobTitleCode  JobTitle
RKB012        Seller
RKB100        Shop Director
RKB002        Cashier
RKB003        Technician
RKB011        Manager
```

Data Set 5

**Employees**

```
Id   Name      Surname      JobTitleCode
7    Joahim    Mesina       RKB001
8    Marta     Frantisek    RKB002
9    Jan       Matalau      RKB001
```

**JobTitles**

```
JobTitleCode  JobTitle
RKB001        Worker
RKB011        Manager
RKB002        Cashier
RKB003        Technician
```

## ⌨ 8.3.9 UPDATE 19

Write a query that modifies a customer Name to Jose all customers from Nitra. Write a query using SQL subquery.

Data Set 1

**Customers**

```
Id  Name     Surname      City_Id
```

**Cities**

```
Id  Name
```

Data Set 2

**Customers**

```
Id  Name       Surname      City_Id
1   Sofia      Bednar       10
2   Jan        Zachar       1
3   Nela       Walach       1
4   Witold     Nowak        5
5   Katarzyna  Kowalska     6
6   Iwona      Bednarz      8
```

**Cities**

```
Id  Name
1   Bratislava
2   Nitra
3   Trnava
4   Katowice
5   Telde
6   Paris
8   Galdar
10  Krakow
```

Data Set 3

**Customers**

```
Id  Name       Surname      City_Id
2   Jan        Zachar       2
4   Katarzyna  Kowalska     6
5   Witold     Nowak        4
6   Iwona      Bednarz      8
7   Katy       Bernard      8
```

**Cities**

```
Id  Name
2   Nitra
4   Katowice
```

```
6   Paris
8   Galdar
```

Data Set 4

**Customers**

```
Id  Name       Surname    City_Id
2   Jan        Zachar     2
3   Jose       Sanchez    2
4   Witold     Nowak      3
5   Iwona      Bednarz    8
6   Katarzyna  Kowalska   5
```

**Cities**

```
Id  Name
2   Nitra
3   Vienna
4   Katowice
5   Sevilla
8   Galdar
```

Data Set 5

**Customers**

```
Id  Name         Surname    City_Id
8   Magdalena    Meredit    6
9   Mateus       Vettel     7
10  Fryderyk     Hanke      9
```

**Cities**

```
Id  Name
2   Nitra
4   Katowice
6   Paris
7   Madrit
9   Brussels
```

## 8.3.10 UPDATE 20

Write a query that modifies the employee JobTitleCode to RKB011, from the Employees table, all employees where employee Id is between 8 and 10.

Data Set 1

**Employees**

```
Id   Name
```

Data Set 2

**Employees**

```
Id  Name     Surname      JobTitleCode
1   Jan      Kowalski     RKB012
2   Jan      Nowak        RKB003
3   Kamil    Wilmowski    RKB011
4   Olga     Milenka      RKB003
5   Angela   Wilga        RKB002
```

Data Set 3

**Employees**

```
Id  Name    Surname     JobTitleCode
1   Jan     Kowalski    RKB012
3   Kamil   Wilmowski   RKB011
5   Angela  Wilga       RKB002
```

Data Set 4

**Employees**

```
Id  Name      Surname     JobTitleCode
1   Jan       Kowalski    RKB012
3   Kamil     Wilmowski   RKB011
4   Paco      Ramirez     RKB011
5   Angela    Wilga       RKB002
6   Beatriz   Santana     RKB002
```

Data Set 5

**Employees**

```
Id  Name      Surname     JobTitleCode
```

```
7   Joahim   Mesina     RKB001
8   Marta    Frantisek  RKB002
9   Jan      Matalu     RKB001
```

# Delete

Chapter 9

# 9.1 DELETE statement

## 📖 9.1.1

The DELETE statement is used to delete rows from a table.

It allows deleting single as well as multiple rows depending on the condition provided in the WHERE clause.

```
DELETE FROM table_name
WHERE condition;
```

## 📖 9.1.2

It is possible to omit the WHERE clause. In this case, the DELETE query deletes all rows from a table. The syntax is as follows:

```
 DELETE FROM table_name;
```

## 📝 9.1.3

You want to delete all rows from table Products with the following schema:

```
Products (ProductID INTEGER PRIMARY KEY, ProductName VARCHAR
(100), ProductDescription TEXT, ProductPrice NUMERIC).
```

Which of the DELETE statements will you use?

- DELETE FROM Products ALL VALUES;
- DELETE FROM Products (ProductID, ProductName, ProductDescription, ProductPrice) VALUES =ALL;
- DELETE FROM Products;
- DELETE FROM Products WHERE ProductID=1 OR ProductID=2;

## 📝 9.1.4

Table Employees contains data presented in a picture.

| EmployeeID | EmployeeName | EmployeeSurname | JobTitleCod |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 2 | Jan | Nowak | RKB003 |
| 3 | Kamil | Wilmowski | RKB011 |
| 4 | Olga | Milenka | RKB012 |
| 5 | Angela | Wilga | RKB002 |
| 6 | Joahim | Mesina | RKB100 |
| 7 | Marta | Frantisek | RKB002 |
| 8 | Armin | Matalu | RKB001 |

Which of the following statements is correct if you want to delete all employees whose name is Jan?

- DELETE FROM Employees;
- DELETE FROM Employees WHERE EmployeeName = 'Jan';
- DELETE FROM Employees WHERE EmployeeName = 'Jan' AND EmployeeSurname='Kowalski';
- DELETE FROM Employees WHERE EmployeeSurname = 'Kowalski' OR EmployeeSurname='Nowak';
- DELETE FROM Employees WHERE EmployeeSurname = 'Kowalski' AND EmployeeSurname='Nowak';

## 9.1.5

Which of the following statements delete all rows from the table Countries presented in a picture?

| CountryID | CountryName | CountryCapital |
|---|---|---|
| 1 | Poland | London |
| 2 | Belgium | London |
| 3 | Spain | London |
| 4 | Slovakia | London |

- DELETE FROM Countries;
- DELETE from Countries WHERE CountryCapital='London';
- DELETE FROM Countries WHERE CountryName='Poland' OR CountryName='Slovakia' OR CountryName='Belgium' OR CountryName='Spain';
- DELETE FROM Countries WHERE CountryCapital='London' OR CountryName='Slovakia' OR CountryName='Spain' OR CountryName='Belgium';

# 9.2 DELETE exercises

## 9.2.1 DELETE 01

Write a query that deletes city from the Cities table where city Id is equal to 2.

Data Set 1

**Cities**

```
Id    Name
```

Data Set 2

**Cities**

```
Id  Name
1   Bratislava
2   Nitra
3   Trnava
4   Katowice
5   Telde
6   Paris
8   Galdar
10  Krakow
```

Data Set 3

**Cities**

```
Id  Name
2   Nitra
4   Katowice
6   Paris
8   Galdar
```

Data Set 4

**Cities**

```
Id  Name
2   Nitra
3   Vienna
4   Katowice
5   Sevilla
8   Galdar
```

Data Set 5

**Cities**

```
Id Name
2   Nitra
4   Katowice
6   Paris
7   Madrit
9   Brussels
```

## 9.2.2 DELETE 02

Write a query that deletes all rows from the Countries table.

Data Set 1

**Countries**

```
Id  Country    Capital
```

Data Set 2

**Countries**

```
Id  Country      Capital
1   Portugal     Lisbon
2   Belgium      Brussels
3   Spain        Madrit
4   Hungary      Budapest
8   Croatia      Zagreb
9   Greece       Athens
10  Malta        Valletta
```

Data Set 3

**Countries**

```
Id  Country      Capital
3   Spain        Madrit
4   Hungary      Budapest
6   Slovakia     Bratislava
7   Norway       Oslo
8   Croatia      Zagreb
9   Greece       Athens
```

```
10  Malta         Valletta
11  Italy         Rome
```

Data Set 4

**Countries**

```
Id  Country       Capital
3   Spain         Madrit
5   Poland        Warsaw
8   Croatia       Zagreb
10  Malta         Valletta
11  Italy         Rome
```

Data Set 5

**Countries**

```
Id  Country       Capital
6   Slovakia      Bratislava
7   Norway        Oslo
8   Croatia       Zagreb
10  Malta         Valletta
11  Italy         Rome
```

### 9.2.3 DELETE 03

Write a query that deletes from the Employees table an employee Jan Kowalski.

Data Set 1

**Employees**

```
Id  Name    Surname     JobTitleCode
```

Data Set 2

**Employees**

```
Id  Name    Surname     JobTitleCode
1   Jan     Kowalski    RKB012
2   Jan     Nowak       RKB003
3   Kamil   Wilmowski   RKB011
4   Olga    Milenka     RKB003
5   Angela  Wilga       RKB002
```

Data Set 3

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 3 | Kamil | Wilmowski | RKB011 |
| 5 | Angela | Wilga | RKB002 |

Data Set 4

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 3 | Kamil | Wilmowski | RKB011 |
| 4 | Paco | Ramirez | RKB011 |
| 5 | Angela | Wilga | RKB002 |
| 6 | Beatriz | Santana | RKB002 |

Data Set 5

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 7 | Joahim | Mesina | RKB001 |
| 8 | Marta | Frantisek | RKB002 |
| 9 | Jan | Matalau | RKB001 |

## 9.2.4 DELETE 04

Write a query that deletes from the Customers table all customers whose Id is greater than 2.

Data Set 1

**Customers**

| Id | Name | Surname | City_Id |
|---|---|---|---|

Data Set 2

**Customers**

| Id | Name | Surname | City_Id |
|---|---|---|---|
| 1 | Sofia | Bednar | 10 |

```
2  Jan         Zachar     1
3  Nela        Walach     1
4  Witold      Nowak      5
5  Katarzyna   Kowalska   6
6  Iwona       Bednarz    8
```

Data Set 3

**Customers**

```
Id Name        Surname    City_Id
2  Jan         Zachar     2
4  Katarzyna   Kowalska   6
5  Witold      Nowak      4
6  Iwona       Bednarz    8
7  Katy        Bernard    8
```

Data Set 4

**Customers**

```
Id Name        Surname    City_Id
2  Jan         Zachar     2
3  Jose        Sanchez    2
4  Witold      Nowak      3
5  Iwona       Bednarz    8
6  Katarzyna Kowalska   5
```

Data Set 5

**Customers**

```
Id Name        Surname    City_Id
8  Magdalena   Meredit    6
9  Mateus      Vettel     7
10 Fryderyk    Hanke      9
```

## ⌨ 9.2.5 DELETE 05

Write a query that deletes from the Employees table all employees whose Name is Jan or Joahim.

Data Set 1

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|

Data Set 2

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 2 | Jan | Nowak | RKB003 |
| 3 | Kamil | Wilmowski | RKB011 |
| 4 | Olga | Milenka | RKB003 |
| 5 | Angela | Wilga | RKB002 |

Data Set 3

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 3 | Kamil | Wilmowski | RKB011 |
| 5 | Angela | Wilga | RKB002 |

Data Set 4

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 1 | Jan | Kowalski | RKB012 |
| 3 | Kamil | Wilmowski | RKB011 |
| 4 | Paco | Ramirez | RKB011 |
| 5 | Angela | Wilga | RKB002 |
| 6 | Beatriz | Santana | RKB002 |

Data Set 5

**Employees**

| Id | Name | Surname | JobTitleCode |
|---|---|---|---|
| 7 | Joahim | Mesina | RKB001 |
| 8 | Marta | Frantisek | RKB002 |
| 9 | Jan | Matalau | RKB001 |

## 9.2.6 DELETE 06

Write a query that deletes from the Employees table all employees who are working as Cashier.

Data Set 1

**Employees**

```
Id  Name    Surname     JobTitleCode
```

**JobTitles**

```
JobTitleCode  JobTitle
```

Data Set 2

**Employees**

```
Id  Name    Surname     JobTitleCode
1   Jan     Kowalski    RKB012
2   Jan     Nowak       RKB003
3   Kamil   Wilmowski   RKB011
4   Olga    Milenka     RKB003
5   Angela  Wilga       RKB002
```

**JobTitles**

```
JobTitleCode  JobTitle
RKB012        Seller
RKB100        Shop Director
RKB001        Worker
RKB011        Manager
RKB002        Cashier
RKB003        Technician
```

Data Set 3

**Employees**

```
Id  Name    Surname     JobTitleCode
1   Jan     Kowalski    RKB012
3   Kamil   Wilmowski   RKB011
5   Angela  Wilga       RKB002
```

**JobTitles**

```
JobTitleCode  JobTitle
RKB001        Worker
RKB012        Seller
RKB011        Manager
RKB002        Cashier
```

Data Set 4

**Employees**

```
Id  Name      Surname     JobTitleCode
1   Jan       Kowalski    RKB012
3   Kamil     Wilmowski   RKB011
4   Paco      Ramirez     RKB011
5   Angela    Wilga       RKB002
6   Beatriz   Santana     RKB002
```

**JobTitles**

```
JobTitleCode  JobTitle
RKB012        Seller
RKB100        Shop Director
RKB002        Cashier
RKB003        Technician
RKB011        Manager
```

Data Set 5

**Employees**

```
Id  Name      Surname     JobTitleCode
7   Joahim    Mesina      RKB001
8   Marta     Frantisek   RKB002
9   Jan       Matalau     RKB001
```

**JobTitles**

```
JobTitleCode  JobTitle
RKB001        Worker
RKB011        Manager
RKB002        Cashier
RKB003        Technician
```

## 9.2.7 DELETE 07

Write a query that deletes from the Customers table all customers from the Galdar city.

Data Set 1

**Customers**

```
Id  Name     Surname      City_Id
```

**Cities**

```
Id  Name
```

Data Set 2

**Customers**

```
Id  Name       Surname      City_Id
1   Sofia      Bednar       10
2   Jan        Zachar       1
3   Nela       Walach       1
4   Witold     Nowak        5
5   Katarzyna  Kowalska     6
6   Iwona      Bednarz      8
```

**Cities**

```
Id  Name
1   Bratislava
2   Nitra
3   Trnava
4   Katowice
5   Telde
6   Paris
8   Galdar
10  Krakow
```

Data Set 3

**Customers**

```
Id  Name       Surname      City_Id
2   Jan        Zachar       2
4   Katarzyna  Kowalska     6
5   Witold     Nowak        4
```

```
6   Iwona     Bednarz     8
7   Katy      Bernard     8
```

**Cities**

```
Id  Name
2   Nitra
4   Katowice
6   Paris
8   Galdar
```

Data Set 4

**Customers**

```
Id  Name       Surname    City_Id
2   Jan        Zachar     2
3   Jose       Sanchez    2
4   Witold     Nowak      3
5   Iwona      Bednarz    8
6   Katarzyna  Kowalska   5
```

**Cities**

```
Id  Name
2   Nitra
3   Vienna
4   Katowice
5   Sevilla
8   Galdar
```

Data Set 5

**Customers**

```
Id  Name        Surname    City_Id
8   Magdalena   Meredit    6
9   Mateus      Vettel     7
10  Fryderyk    Hanke      9
```

**Cities**

```
Id  Name
2   Nitra
4   Katowice
6   Paris
```

```
7   Madrit
9   Brussels
```

### 9.2.8 DELETE 08

Write a query that deletes from the Customers table all customers where City_Id is equal to 8.

Data Set 1

**Customers**

```
Id Name       Surname     City_Id
```

Data Set 2

**Customers**

```
Id Name        Surname     City_Id
1  Sofia       Bednar      10
2  Jan         Zachar      1
3  Nela        Walach      1
4  Witold      Nowak       5
5  Katarzyna   Kowalska    6
6  Iwona       Bednarz     8
```

Data Set 3

**Customers**

```
Id Name        Surname     City_Id
2  Jan         Zachar      2
4  Katarzyna   Kowalska    6
5  Witold      Nowak       4
6  Iwona       Bednarz     8
7  Katy        Bernard     8
```

Data Set 4

**Customers**

```
Id Name       Surname    City_Id
2  Jan        Zachar     2
3  Jose       Sanchez    2
4  Witold     Nowak      3
5  Iwona      Bednarz    8
```

```
6  Katarzyna Kowalska  5
```

Data Set 5

**Customers**

```
Id Name        Surname   City_Id
8  Magdalena   Meredit   6
9  Mateus      Vettel    7
10 Fryderyk    Hanke     9
```

## 9.2.9 DELETE 09

Write a query that deletes from the Countries table all countries which names begin with the letter S.

Data Set 1

**Countries**

```
Id  Country    Capital
```

Data Set 2

**Countries**

```
Id  Country      Capital
1   Portugal     Lisbon
2   Belgium      Brussels
3   Spain        Madrit
4   Hungary      Budapest
8   Croatia      Zagreb
9   Greece       Athens
10  Malta        Valletta
```

Data Set 3

**Countries**

```
Id  Country      Capital
3   Spain        Madrit
4   Hungary      Budapest
6   Slovakia     Bratislava
7   Norway       Oslo
8   Croatia      Zagreb
9   Greece       Athens
```

```
10  Malta         Valletta
11  Italy         Rome
```

Data Set 4

**Countries**

```
Id  Country       Capital
3   Spain         Madrit
5   Poland        Warsaw
8   Croatia       Zagreb
10  Malta         Valletta
11  Italy         Rome
```

Data Set 5

**Countries**

```
Id  Country       Capital
6   Slovakia      Bratislava
7   Norway        Oslo
8   Croatia       Zagreb
10  Malta         Valletta
11  Italy         Rome
```

## 9.2.10 DELETE 10

Write a query that deletes all rows from the Countries table where Country has value NULL.

Data Set 1

**Countries**

```
Id  Country     Capital
```

Data Set 2

**Countries**

```
Id  Country       Capital
1   Portugal      Lisbon
2   Belgium       Brussels
3   Spain         Madrit
4   Hungary       Budapest
8   Croatia       Zagreb
```

```
9   Greece      Athens
10  Malta       Valletta
```

Data Set 3

**Countries**

```
Id  Country     Capital
3   Spain       Madrit
4   Hungary     Budapest
6   Slovakia    Bratislava
7   Norway      Oslo
8   Croatia     Zagreb
9   Greece      Athens
10  Malta       Valletta
11  Italy       Rome
```

Data Set 4

**Countries**

```
Id  Country     Capital
3   Spain       Madrit
5   Poland      Warsaw
8   Croatia     Zagreb
10  Malta       Valletta
11  Italy       Rome
```

Data Set 5

**Countries**

```
Id  Country     Capital
6   Slovakia    Bratislava
7   Norway      Oslo
8   Croatia     Zagreb
10  Malta       Valletta
11  Italy       Rome
```

# PRISCILLA

priscilla.fitped.eu