***# Secondary school of electrotechnical engineering Jecna 30***

***# 2022-2023***

***# Matěj Šturma***

***# C4c***

## **Introduction**

This database stores information about customers, products, and orders for a furniture store. The database includes tables for storing information about countries, cities, categories, subcategories, colors, stores, and product availability.

I decided to use **MySQL** for this database.

### **Usage**

The database can be maintained in MySQL Workbench or any other IDE that supports either MySQL or MariaDB.

### **Conceptual Model**

Conceptual model was created with the reverse engineer function in MySQL Workbench.

It is stored in **/img/** folder in the **diagram.png** file.

## **Tables**

### **customer**

The customer table stores information about customers of the furniture store.

Columns:

* id: a unique identifier for each customer.
* id\_city: a foreign key that references the id column in the cities table.
* surname: the surname of the customer.
* first\_name: the first name of the customer.
* address: the address of the customer.
* ICO: the identification number of the customer.
* IC\_DPH: the identification number for value-added tax of the customer.
* total\_spending: the total amount spent by the customer.

### **countries**

The countries table stores information about countries.

Columns:

* id: a unique identifier for each country.
* name: the name of the country.

### **cities**

The cities table stores information about cities.

Columns:

* id: a unique identifier for each city.
* id\_country: a foreign key that references the id column in the countries table.
* name: the name of the city.
* zipcode: the zip code of the city.

### **products**

The products table stores information about products.

Columns:

* id: a unique identifier for each product.
* id\_subcategory: a foreign key that references the id column in the subcategories table.
* id\_color: a foreign key that references the id column in the colors table.
* name: the name of the product.
* price: the price of the product.
* price\_exc\_dph: the price of the product excluding value-added tax.
* dph: the value-added tax applied to the product.

### **categories**

The categories table stores information about categories of products.

Columns:

* id: a unique identifier for each category.
* name: the name of the category.

### **subcategories**

The subcategories table stores information about subcategories of products.

Columns:

* id: a unique identifier for each subcategory.
* id\_category: a foreign key that references the id column in the categories table.
* name: the name of the subcategory.

### **colors**

The colors table stores information about colors.

Columns:

* id: a unique identifier for each color.
* name: the name of the color.
* hex: the hexadecimal code for the color.

### **stores**

The stores table stores information about stores.

Columns:

* id: a unique identifier

### **product\_availability**

The product\_availability table stores information about which products are available in which stores.

Columns:

* id: a unique identifier for each product availability record.
* id\_store: a foreign key that references the id column in the stores table.
* id\_product: a foreign key that references the id column in the products table.

### **orders**

The orders table stores information about orders placed by customers.

Columns:

* id: a unique identifier for each order.
* id\_customer: a foreign key that references the id column in the customer table.
* id\_product\_availability: a foreign key that references the id column in the product\_availability table.
* date: the date and time that the order was placed.
* price\_sum: the total price of the order, including value-added tax.
* price\_sum\_exc\_dph: the total price of the order, excluding value-added tax.
* dph\_sum: the total value-added tax applied to the order.

## **Views**

A view is a virtual table that is created based on a SELECT statement.

### **list\_of\_countries**

The list\_of\_countries view displays all countries in the table countries.

Columns:

* id: a unique identifier for each country in the view.
* name: a name of the country.

### **product\_sales**

The product\_sales view displays the number of sales and the total revenue of each product.

Columns:

* id: a unique identifier for each order in the view.
* num\_sales: the total number of orders placed for each product.
* total\_revenue: the sum of the price of all products in the view.

### **top\_customers**

The top\_customers view displays all customers and their total spending.

Columns:

* surname: the surname of each customer.
* first\_name: the first name of each customer.
* address: the address of each customer.
* total\_spending: the sum of all prices of the orders, which the customer placed.

## **Stored procedures**

A stored procedure is a program that can be called to perform a specific task.

### **add\_customer**

The add\_customer procedure adds a new customer to the customer table.

Input parameters:

* surname: the surname of the customer.
* first\_name: the first name of the customer.
* address: the address of the customer.
* ico: the identification number of the customer.
* ic\_dph: the identification number for value-added tax of the customer.
* country: the country where the customer is located.
* city: the city where the customer is located.
* zip\_code: zip code of the city where the customer is located.
* address: the address of the customer.

### **delete\_customer**

The delete\_customer procedure deletes a customer from the customer table.

Input parameters:

* first\_name: the first name of the customer.
* last\_name: the surname of the customer.
* ico: the identification number of the customer.
* ic\_dph: the identification number for value-added tax of the customer.

### **update\_product\_price**

The update\_product\_price procedure updates the price of the specified product.

Input parameters:

* id: id of the product that is being updated.
* new\_price: the new price you want to set.

## **Triggers**

A trigger is a stored program that is automatically executed in response to a specific event, such as the insertion of a row into a table.

### **update\_customer\_spending**

The update\_customer\_spending trigger updates the customer’s total spending value after a new order is inserted.

Action:

* When a new row is inserted into the orders table, the trigger retrieves the id\_customer and price\_sum values from the new row, and adds the price\_sum value to the total\_spending value of the corresponding customer in the customer table.

## **Integrity**

Integrity refers to the accuracy and consistency of the data stored in the database.

### **Entity integrity**

Entity integrity is a type of integrity that is enforced at the level of individual rows in a table. It ensures that each row in a table is uniquely identified by a primary key, and that the primary key value is not NULL.

In this database, entity integrity is enforced through the use of primary keys and the NOT NULL constraint. For example, the customer table has a primary key column called id, which is defined as an integer data type and is marked with the NOT NULL constraint. This ensures that each row in the customer table has a unique id value and that the id value is not NULL.

### 

### 

### **Referential integrity**

Referential integrity is a type of integrity that is enforced at the level of relationships between tables. It ensures that foreign key values in one table match valid primary key values in another table, and that the foreign key values are not NULL.

In this database, referential integrity is enforced through the use of foreign keys and the FOREIGN KEY constraint. For example, the customer table has a foreign key column called id\_city, which is defined as an integer data type and is marked with the FOREIGN KEY constraint. This ensures that the id\_city values in the customer table match valid id values in the cities table, and that the id\_city values are not NULL.

### **Domain integrity**

Domain integrity is a type of integrity that is enforced at the level of individual columns in a table. It ensures that the values stored in a column are of the correct data type, and that they meet any other specified constraints such as minimum or maximum values, allowed values, or patterns.

In this database, domain integrity is enforced through the use of data types and constraints. For example, the price column in the products table is defined as a FLOAT data type, which ensures that only numeric values with decimal places can be stored in that column. The surname column in the customer table is defined as a VARCHAR(50) data type, which ensures that only character strings of up to 50 characters in length can be stored in that column.

## **Additional information**

### **Insertion from a CSV file**

The data in tables countries and colors are inserted from an external CSV file. The SQL statement I use to do that is “LOAD DATA INFILE”

The LOAD DATA INFILE statement reads rows from a text file and inserts them into a table in a database.

Parameters:

* file\_name: the name of the file to be imported.
* table\_name: the name of the table into which the data will be imported.
* FIELDS TERMINATED BY: the string that separates fields in the file. [`,` for CSV files]
* LINES TERMINATED BY: the string that terminates lines in the file. [usually ‘\n’]
* (column\_names): names of the columns to be inserted.

Example:



### **Transactions**

Transactions are temporary units of work that can be rolled back until a commit operation is performed.

Syntax:

* START\_TRANSACTION: starts a transaction.
* COMMIT: commits a transaction.
* ROLLBACK: rollbacks a transaction.

### **Users and roles**

Database contains 3 users that can perform various actions

Users:

* app\_user: is able to insert, select and update data in the customer table.
* readonly\_user: is able to select data in all tables.
* dataentry\_user: is able insert and update data to the customer and orders table.

### **Indexes**

An index is a data structure that can be created on a table to improve the performance of certain types of queries. Indexes allow the database management system to quickly locate and retrieve data from a table, rather than having to search through the entire table.

Used indexes:

* **idx\_customer\_id\_city:** non-unique index on the customer table using the id\_city column.
* **idx\_customer\_surname:** non-unique index on the customer table using the surname column.
* **idx\_countries\_name:** non-unique index on the countries table using the name column.
* **idx\_cities\_id\_country:** non-unique index on the cities table using the id\_country column.
* **idx\_cities\_name:** non-unique index on the cities table using the name column.
* **idx\_products\_id\_subcategory:** non-unique index on the products table using the id\_subcategory column.
* **idx\_products\_id\_color:** non-unique index on the products table using the id\_color column.
* **idx\_categories\_name:** non-unique index on the categories table using the name column.
* **idx\_subcategories\_id\_category:** non-unique index on the subcategories table using the id\_category column.
* **idx\_colors\_name:** non-unique index on the colors table using the name column.
* **idx\_stores\_id\_city:** non-unique index on the stores table using the id\_city column.

### **Backup and restore**

MySQL does not provide backup and restore functions. The database can be backuped in phpMyAdmin using the **mysqldump** command.