Title: Creation of database applications

Code:

Author: Ing. Romana Garzinová, Ph.D.

Edition: first, 2015

Number of pages: 96

Academic materials for the Automation and Computing in Industrial technologies study programme at the Faculty of Metallurgy and Materials Engineering.

Proofreading has not been performed.

Execution: VŠB - Technical University of Ostrava
STUDY INSTRUCTIONS

Creation of Database Applications

You have received a study package containing the integrated university mimeographed for combined study containing the study instructions for the course of the Creation of Database Applications of the 2nd semester of the follow-up master's degree of the study programme of Automation and Computer Technology in Metallurgy.

Course objective and learning outputs

The course objective is to introduce the basic architectures of information systems, the standard of T-SQL programming language, the basics of object programming and, last but not least, the C# programming language enabling the creation of modern database applications in MS.NET Framework environment.

After studying the course, students should be able to:

knowledge outputs:

• students will be able to create basic database inquires using T-SQL
• students will be able to present data within the scope of WPF application
• students will be able to present data within the scope of ASP.NET web application
• students will be able to process data within the scope of distributed applications using WCF

skills outputs:

• students will be able to effectively design and use database models in MS SQL Server
• students will be able to apply their theoretical knowledge in technological environments that produce a number of disparate data to be presented or to predict or optimize further control processes of technological processes on the basis of the acquired data

Who is the course intended for

The course is included in the follow-up master's study of the field of study of Automation and Computer Technology in Metallurgy, but it can be studied by an applicant from any other field of study, provided that he/she meets the required prerequisites.
This study support is divided into parts, chapters, which logically divide the studied matter, but are not equally comprehensive. The estimated study time of the chapters may vary considerably, which is why large chapters are further divided into numbered sub-chapters and they correspond to the structure described below.

**Method of communication with the educator**

This matter is presented to students within the frame of their lectures and practical exercises, where they practically learn the topic discussed during the theoretical lectures.
STRUCTURE OF THE CHAPTERS

Study time

The time necessary to study the subject matter is given at the beginning of the chapter. The time is approximate and can serve as a rough guide for the study layout of the entire subject. The time may seem too long to some people or, on the contrary, too short to other ones. There are students who have never encountered this issue and, on the other hand, those who already have extensive experience in this field.

Objective

Immediately after, there are objectives given for you to achieve after reading this chapter - concrete skills, knowledge.

Explication

The following part is the actual presentation of the studied subject matter, the introduction of new terms, their explanations, and all is accompanied by pictures, tables, solved examples, and links to animations.

Time to think

You will encounter questions and problems within the study of the chapters that should be thought over for the sake of mastering the subject matter. Correct solutions follow the questions, so do not continue reading until you have thought everything over well.

Example from practice, Example to solve

Use practical examples of the application of the presented theoretical knowledge to better understand the discussed subject matter.
Animations, Video

The presentation is accompanied by animations to increase the plasticity of the studied subject matter and videos to illustrate and extend the knowledge.

Summary of terms

The main terms you should learn are repeated at the end of the chapter. If you still do not understand any of the terms, go back to them again.

Questions

There are several theoretical questions to verify that you have fully and well mastered the subject matter of the chapter.

There is a list of the used reference sources, from which you can draw additional information on the issue in question, at the very end of the chapter.

Solution key

You will find the solution key to the tasks and answers to the questions at the end of the study supports.

The author of this educational material wishes you a successful and pleasant study using this textbook.

Ing. Romana Garzinová, Ph.D.
CONTENTS

STUDY INSTRUCTIONS ........................................................................................................... 1

STRUCTURE OF THE CHAPTERS .......................................................................................... 5

CONTENTS .............................................................................................................................. 7

1 INTRODUCTION TO THE COURSE – DATA COLLECTION METHODS,
DEFINATION OF THE SOLUTION ARCHITECTURE AND MODERN
TECHNOLOGICAL TRENDS OF SOFTWARE DEVELOPMENT .......... 10
   1.1 History of T-SQL standard .......................................................................................... 10
   1.2 Solution architecture................................................................................................. 11
       1.2.1 Single-layer (central) systems ............................................................................. 12
       1.2.2 Double-layer architecture (client/server) .......................................................... 12
       1.2.3 Triple-layer architecture ................................................................................... 13
       1.2.4 N-layer models .................................................................................................. 13

2 BASIC MS SQL SERVER OBJECTS ................................................................................. 15
   2.1 Basic objects ............................................................................................................. 15
   2.2 Data types of MS SQL server ................................................................................... 16

3 TABLE – DEFINITION OF DATABASE TABLES AND DATA INTEGRITY 21
   3.1 Create table command.............................................................................................. 21
   3.2 Definition of primary keys ....................................................................................... 22
   3.3 Definition of foreign keys ........................................................................................ 23
   3.4 Definition of default constraints ............................................................................. 24

4 DATA COLLECTION FROM SINGLE AND MULTIPLE DATA SOURCES
   (SELECT, JOIN) ................................................................................................................... 26
   4.1 Basic selection of values (select command) ............................................................... 26
   4.2 Natural join ................................................................................................................ 28
   4.3 Modifiers .................................................................................................................... 30
       4.3.1 DISTINCT ........................................................................................................... 30
       4.3.2 TOP ................................................................................................................... 31
   4.4 Modification of column headers ............................................................................... 31

5 DATA SORTING AND FILTERING .................................................................................... 33
5.1 EXISTS operator .................................................................33
5.2 SQL nested queries ...........................................................34
5.3 Operators ........................................................................35
5.4 SQL queries with aggregation ..............................................36
  5.4.1 COUNT(*) .....................................................................36
  5.4.2 SUM(column) ...............................................................37
  5.4.3 AVG(column) ...............................................................37
  5.4.4 MIN (column) .............................................................38
  5.4.5 MAX (column) .............................................................38
  5.4.6 Further use of aggregate functions .................................39

6 THE USE AND WORK WITH STORED PROCEDURES IN MS SQL SERVER 41
6.1 Conditional processing ..........................................................41
  6.1.1 IF, ELSE ......................................................................42
  6.1.2 CASE ........................................................................42
6.2 Flow control ..........................................................................43
  6.2.1 RETURN ......................................................................43
  6.2.2 WHILE .......................................................................44
  6.2.3 GOTO ..........................................................................45
6.3 Cursors ................................................................................45
  6.3.1 Cursor life cycle .............................................................46
  6.3.2 Cursor declaration .........................................................46
  6.3.3 Cursor opening ..............................................................47
  6.3.4 Work with rows ............................................................47
  6.3.5 Cursor closing and deallocation .....................................48
  6.3.6 Complex example: ........................................................49
6.4 Stored procedures ....................................................................50
6.5 User-defined functions ............................................................52
  6.5.1 Scalar user-defined functions ........................................52
  6.5.2 Direct tabular user-defined functions ...............................54
  6.5.3 Multicommand user-defined functions ...............................55

7 CREATION OF DATABASE APPLICATIONS .........................57
7.1 What is a database good for? ................................................57
7.2 Approaches to work with relational databases in .NET .............57
  7.2.1 Non-object programming ..............................................58
  7.2.2 Database Wrapper .......................................................58
  7.2.3 Object relational mapping .............................................58
7.2.4 Object databases

7.3 Possible connections to database

7.3.1 Connected application

7.3.2 Disconnected application - DataSet

7.3.3 LINQ To SQL

7.3.4 Entity Framework

7.4 Communication with database by means of ADO.NET

7.4.1 ADO.NET architecture

7.4.2 Data provider ADO.NET

7.4.3 Basic ADO.NET classes

7.5 ADO.NET Entity Framework

7.5.1 Data queries by means of Entity Framework

7.5.2 Creation of new records

7.5.3 Data change

7.5.4 Records deletion

7.5.5 Calling stored procedures in Entity Framework

8 ASP.NET - WEB TECHNOLOGIES FOR THE PRESENTATION OF DATA

8.1 The basic principles of ASP.NET

8.1.1 Static web (without ASP.NET)

8.1.2 Dynamic web (with ASP.NET)

8.2 WebForms vs. MVC

8.2.1 WebForms

8.2.2 MVC

8.3 How to program in ASP.NET?

8.3.1 Project initialisation

8.3.2 Adding Web Form into the web project

8.3.3 Definition of client Web Form features

8.3.4 Definition of application logic
1 INTRODUCTION TO THE COURSE – DATA COLLECTION METHODS, DEFINITION OF THE SOLUTION ARCHITECTURE AND MODERN TECHNOLOGICAL TRENDS OF SOFTWARE DEVELOPMENT

Study time

1 hour

Objective

The aim of this chapter is to transfer the basic information about the history of the development of database systems and the basic solution architectures.

Explication

1.1 HISTORY OF T-SQL STANDARD

In engineering practice, SQL is an abbreviation of Structured Query Language. SQL is a standard language that was designed for querying and administration of data in relational database management systems (RDMS - relational database management systems). RDMS is based on the relational model, which is based on two mathematical disciplines: the set theory and predicate logic.

This chapter presents the theoretical foundation of SQL, the relational model and the data lifecycle. The goal is not an interpretation of the mathematical theorems, but a presentation of the basic information about the potential of T-SQL language allowing the basic operations with data.

Relational database systems are currently at their prime. Storing data by means of relational database system is very widespread nowadays. However, it is not a new feature. The basic principles of relational database system structure were already defined in the late 1960s by E.F. Codd of IBM, who also created a special language called Structured English QUEry Language or SEQUEL: This name was later
shortened to today's Structured Query Language, or SQL. Relational databases were based on a simple idea - to reduce repetitive data recording and to solve some other database problems, which were common at that time, in order to increase the degree of data integrity and to reduce the cost of databases.

In the 1970s, relational database systems were first offered by Oracle and Sybase companies. The American standardization organization ANSI (American National Standards Institute), which had created the specification of SQL language called ANSI-SQL, joined these two companies in the 1980s. It was a crucial moment for relational database systems, because the standard brought improved compatibility between the different database manufacturers. Furthermore, the outcomes of the development in one relational database system could mostly be used in another system as well. Thanks to this fact, the number of developers in the SQL community could increase at the same time.

Microsoft SQL Server was originally born from Sybase SQL Server. Microsoft Company started collaboration with Sybase in 1989, and it was aimed at the development of a SQL Server for OS/2 systems. The cooperative relationship between the two companies fell apart with the release of version 6.0. Starting with version 6.5, the SQL Server is an exclusive product of Microsoft Company.

At present, the installation of MS SQL Server gives us much more than just a usual relational database system – the main relational database system, full text, analytical tools, replication, data transformation services, cloud solution support, optimization tools, cluster support, etc.

1.2 SOLUTION ARCHITECTURE

Previous text clearly shows that we have a powerful database tool that we want to use for its purposes. Now, however, we come to the fundamental problem – what architecture to integrate the database server into. There are many technical articles and researches focused on suitable architectures of application solutions. The development of mobile applications and the Internet in general means that today and every day we can hear about new principles, new standards, and every report claims that this or that technology or architecture is the best and all other are basically out of the question. However, there is a simple rule, which is often overlooked: Perfect architecture that we can use to solve the problem is such an architecture that is appropriate just for the particular solution we are working on. In everyday life, there are very few questions that have an easy answer, and the question of software architecture for a particular project will hardly be among them. Do not let anybody persuade you and do not fall for any clear-cut opinion that this or that architecture is always the best. Before you determine the final architecture, always carefully consider all the requirements for functionality, the anticipated volume of data, the user control requirements, the mutual cooperation with other systems, etc. Based on this input information, select the most suitable architecture that fits the given requirements.
Today, the application activity is generally divided according to how it deals with three categories of services:

- **Customer services** - this category includes aspects such as user interface, basic formatting, or rules for validation of input data. The entire area of user services takes care of the presentation of data and they ensure the correct data field content.

- **Application services** - this part of application processes various application rules. They can include communication rules between systems, calculation logic, decision logic, etc.

- **Data services** - take care of the processes associated with data loading and storing. Usually, they know the rules of data integrity, but they do not care how the data are collected and transformed. The aforementioned database systems find themselves in this position.

Now, let's take a look at some of the classic architectures and let us see how the above presented services fit into them.

### 1.2.1 Single-layer (central) systems

The old model featuring a central computer belongs to the world of mainframe computers (mainframes) and minicomputers. Client workstations do not contain any separate logic, they operate in terminal mode. The terminal will only receive information about the screen layout and the displayed data.

### 1.2.2 Double-layer architecture (client/server)

This architecture became popular in the early 1990s. This architecture actually had two subtypes: concentrated on the client or on the server.

**Architecture concentrated at client`s place**

It was based on the idea of inexpensive personal computers and the opinion that the highest application performance is achieved when the highest amount of computational demands is cast aside, in other words, distributed. That is why data services were performed on the server in this architecture. The application logic and the user services were processed at the client´s place. The disadvantage of this solution is the width of the communication range - a large volume of transmitted data.

**Architecture concentrated on server**

This architecture is trying to exploit certain advantages of the centralized systems. Only the user services are transferred on the client. Only the necessary and already limited data, which are to be presented to the system user on the screen, are sent to the client. The application and data services are provided by the server.
1.2.3 Triple-layer architecture

Currently, this is the most commonly used model. In this model, all three levels are divided into completely separate logic models. Each client is responsible only for the work of the user interface, similar to the client/server architecture concentrated on the server. The difference between the two models lies in the fact that the application and data services are separated here. This means that the application and data services may or may not run on a single server. Thanks to this fact, we gain higher stability and scalability of the solution, because we can both divide the operational load on multiple servers, and the individual layers are easier to differentiate, because all changes and supplements interfere with the smaller sections of the program code. Another advantage is the independence on the user interface. Nowadays, it is common to have architectures, where the data and application services are shared by multiple user interfaces, e.g. Windows applications, Internet applications, mobile applications on various platforms - iOS, Android, Windows Mobile, etc.

1.2.4 N-layer models

This type is essentially the same as the previous one; the difference is that the components are divided into the smallest logical units. If this system has correctly defined layer of data services, the database can also be located on multiple servers. The condition is the availability of the data services in the application layer by an unambiguous definition of the connection providing the required data.

<table>
<thead>
<tr>
<th>Summary of terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL, relation, database systems, solution architecture</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Questions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. What does the abbreviation SQL mean?</td>
</tr>
<tr>
<td>2. What is the basic division of the activities of current applications?</td>
</tr>
<tr>
<td>3. What are the basic types of architectures and briefly define the differences among them?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reference sources</th>
</tr>
</thead>
</table>
2 BASIC MS SQL SERVER OBJECTS

Study time

1 hour

Objective

The aim of this chapter is to transfer the knowledge about the basic objects and data types of MS SQL database server.

Explication

2.1 BASIC OBJECTS

In the previous chapter, we were dealing with the possible technical solution architectures, questions, where the centralized application logic and data should be, and how to ensure a complex distribution of the performance of the entire solution. In this chapter, we are going to focus our attention only on the application level ensuring an effective storage and distribution of data - databases.

What does a database consist of? Of course, first and foremost there are the data. A relational database system, in fact, does not consist only of data. If you just want to store data somewhere, a plain text file will be sufficient for that. A relational database system allows much more than that – we can define what the data should look like or describe the application rules.

A database server includes the following basic objects:

- **The actual database** - a database consists of a group of certain set of table objects, stored procedures, views that belong to a particular data aggregate stored in the tables of the database. One database server can contain a wide variety of user databases.
- **Transactional protocol** - the database file itself is not the place where most of the things happen. The data are uploaded from a database, but all the performed changes do not go into the database immediately, instead of that they are written into the transaction log. A checkpoint is created in the database at certain moment, and all the changes recorded in the protocol are also transferred into the actual database file at this very moment.
• **Table** - the most basic database object. The tables store the actual data of the given database. Each table definition contains the so-called metadata, i.e. information describing the data themselves, the basic data types of database table columns, the range of their values, and the definition of a set of rules specifying the data that can be saved in the given column. If the rules of the columns are broken, the system can reject not only the newly inserted row, but also the update of the existing row or the deletion of a selected row.

• **Charts** – graphic illustration of the database tables, including the definitions of their columns and the mutual relations between the individual tables.

![Database diagram](image)

**Figure 1 Database diagram**

• **Stored procedures** - are the foundation of programming in SQL Server. They consist of commands of Transact-SQL language that are packaged into a coherent logical unit. Variables and parameters, as well as the selection structures and cycles, are allowed in the stored procedure body.

• **User-defined functions** - are in many ways similar to stored procedures. Unlike them, however, they work with a smaller set of data types of SQL Server and cannot perform all the operations (changes in tables, sending e-mails, etc.).

2.2 **DATA TYPES OF MS SQL SERVER**
<table>
<thead>
<tr>
<th>Name</th>
<th>Class</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td>Integer</td>
<td>1</td>
<td>Can have the values of 0, 1 or NULL. Used for the purpose of logic operations</td>
</tr>
<tr>
<td>Bigint</td>
<td>Integer</td>
<td>8</td>
<td>Integer values range – 9 223 372 036 854 775 808 up to 9 223 372 036 854 775 807</td>
</tr>
<tr>
<td>Int</td>
<td>Integer</td>
<td>4</td>
<td>Integral numbers from the interval of – 2 147 483 648 up to 2 147 483 647</td>
</tr>
<tr>
<td>Smallint</td>
<td>Integer</td>
<td>2</td>
<td>Integral numbers from the interval of –32 768 up to 32 767</td>
</tr>
<tr>
<td>Tinyint</td>
<td>Integer</td>
<td>1</td>
<td>Integral numbers from 0 up to 255</td>
</tr>
<tr>
<td>Decimal or numeric</td>
<td>Decimal / numeric</td>
<td>Variable</td>
<td>Numbers with fixed accuracy and range from -10^{38}-1 to 10^{38}-1</td>
</tr>
<tr>
<td>Money</td>
<td>Monetary</td>
<td>8</td>
<td>Monetary units from 2^{63} to 2^{63} plus accuracy to four decimal places</td>
</tr>
<tr>
<td>SmallMoney</td>
<td>Monetary</td>
<td>4</td>
<td>Monetary units from -214 748,3648 to 214 748,3647</td>
</tr>
<tr>
<td>Float</td>
<td>Approximate</td>
<td>Variable</td>
<td>Takes over the</td>
</tr>
<tr>
<td>Data Type</td>
<td>Description</td>
<td>Size</td>
<td>Details</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------------</td>
<td>------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DateTime</td>
<td>Date/time</td>
<td>8</td>
<td>Date and time information starting with 1.1.1753 and ending with 31.12 9999 with accuracy of one thirtieth of a second</td>
</tr>
<tr>
<td>SmallDateTime</td>
<td>Date/time</td>
<td>4</td>
<td>Date and time information starting with 1.1.1900 and ending with 6.6. 2079 with accuracy of one minute</td>
</tr>
<tr>
<td>Cursor</td>
<td>Special numerical</td>
<td>1</td>
<td>Cursor indicator</td>
</tr>
<tr>
<td>Timestamp/rowversion</td>
<td>Special numerical</td>
<td>8</td>
<td>Special value unique in the given database.</td>
</tr>
<tr>
<td>Uniqueidentifier</td>
<td>Special numerical</td>
<td>16</td>
<td>Special value of a global unique identifier. This identifier is definitely unique in space and time</td>
</tr>
<tr>
<td>Char</td>
<td>Character</td>
<td>Variable</td>
<td>Character data of fixed length. Text values shorter than the fixed defined size will be</td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Length</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>--------------</td>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Varchar</td>
<td>Character</td>
<td>Variable</td>
<td>Character data of variable length. The values are not supplemented by blank spaces.</td>
</tr>
<tr>
<td>Text</td>
<td>Character</td>
<td>Variable</td>
<td>Maximum length of this data type is 2,147,483,647 characters.</td>
</tr>
<tr>
<td>Nchar</td>
<td>Unicode</td>
<td>Variable</td>
<td>Character data of Unicode format with fixed length.</td>
</tr>
<tr>
<td>NVarchar</td>
<td>Unicode</td>
<td>Variable</td>
<td>Character data of Unicode format with variable length.</td>
</tr>
<tr>
<td>NText</td>
<td>Unicode</td>
<td>Variable</td>
<td>Character data of Unicode format with variable length.</td>
</tr>
<tr>
<td>Binary</td>
<td>Binary</td>
<td>Variable</td>
<td>Binary data of fixed length.</td>
</tr>
<tr>
<td>VarBinary</td>
<td>Binary</td>
<td>Variable</td>
<td>Binary data of variable length.</td>
</tr>
<tr>
<td>Image</td>
<td>Binary</td>
<td>Variable</td>
<td>Binary data of variable length, max. $2^{31} - 1$.</td>
</tr>
<tr>
<td>Table</td>
<td>Other</td>
<td>Special</td>
<td>Primarily intended for work with final tables</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>---------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>XML</td>
<td>Other</td>
<td>Special</td>
<td>Serves for data storage in XML format</td>
</tr>
</tbody>
</table>

### Summary of terms

Database, transaction log, table, diagram, stored procedure, user-defined functions, data types

### Questions

1. What are the basic objects of a database server?
2. Which of the database objects is the most basic?

### Reference sources

3  TABLE – DEFINITION OF DATABASE TABLES AND DATA INTEGRITY

Study time

1 hour

Objective

You will become familiar with the creation of a database system basic object - a database table, including the definitions of the primary and foreign keys.

Explication

In the previous chapter, we have learned about the basic objects of a database server. A database table is the elementary unit of each database. Tables store the actual data of the given database. Each table definition contains the so-called metadata, which is information describing the data themselves, the basic data types of database table columns, the range of their values, and the definition of a set of rules specifying the data that can be stored in the given column. In this chapter, we will learn how to create a database table, and how to set up its basic attributes.

3.1  CREATE TABLE COMMAND

The following code will create a database table in TSQL2012 database.

Example from practice, Example to solve

USE TSQL2012

IF OBJECT_ID('dbo.Employees', 'U') IS NOT NULL
    DROP TABLE dbo.Employees;
CREATE TABLE dbo.Employees
(
    empid INT NOT NULL,
    firstname VARCHAR(30) NOT NULL,
    lastname VARCHAR(30) NOT NULL,
    hiredate DATE NOT NULL,
    mgrid INT NULL,
    ssn VARCHAR(20) NOT NULL,
    salary MONEY NOT NULL
);
Primary key is a field or a combination of database table fields that unambiguously identify each record in the database table. No field that is included in the primary key can contain the NULL value. Each table should have just one defined primary key.

In most cases, the primary key is an identification number of a list, in our case, it is the `empid` item.

Definition of primary key:

```
Example from practice, Example to solve

ALTER TABLE dbo.Employees
    ADD CONSTRAINT PK_Employees
    PRIMARY KEY(empid);
```

### 3.3 DEFINITION OF FOREIGN KEYS

A foreign key defines the relationship between two tables. The definition of a foreign key declares the existence of a specific record from one database table in another defined table. We create a connection of one or more columns with a column or more columns of another ("foreign") table.

The foreign key allows you to define actions to take place when you try to change or delete records in a foreign table. For example, after deleting a record from a foreign table, records with corresponding foreign key value in the source table will be deleted as well, or their values will be set to neutral ones, or the deletion of records in the foreign table will be prevented.

The following code creates another database table of `Orders` with the definition of a primary key of `orderid`:

```
Example from practice, Example to solve

IF OBJECT_ID('dbo.Orders', 'U') IS NOT NULL
    DROP TABLE dbo.Orders;
CREATE TABLE dbo.Orders
    ( orderid
    );
```
The requirement is to ensure data integrity. The table of orders must not contain a record with no link to the submitter’s ID, whose data are stored in the *Employees* database table. This will be ensured through the definition of the *empid* foreign key:

Example from practice, Example to solve

```
ALTER TABLE dbo.Orders
ADD CONSTRAINT FK_Orders_Employees
FOREIGN KEY (empid)
REFERENCES dbo.Employees (empid);
```

### 3.4 **Definition of Default Constraints**

The definition of default constraints sets the default values of a defined column of a database table in case its value is not explicitly defined when new records are entered into this table.

The following example defines the default constraints of date and time when a record of new order is entered:

Example from practice, Example to solve

```
ALTER TABLE dbo.Orders
ADD CONSTRAINT DFT_Orders_orderts
```
The current date and time represent the return value of the used SYSDATETIME function.

**Summary of terms**

CREATE TABLE order, primary key, foreign key, default constrains

**Questions**

1. What is the basic syntax of T-SQL for the created database tables?
2. What is the primary key and what is the syntax of its creation in T-SQL?
3. What is the foreign key and what is the syntax of its creation in T-SQL?
4. What is the T-SQL syntax for the creation of default constraints?

**Reference sources**


4 DATA COLLECTION FROM SINGLE AND MULTIPLE DATA SOURCES (SELECT, JOIN)

Study time

1 hour

Objective

Definition of simple T-SQL queries to select records from one or more database tables.

Explication

In the previous chapter, we have learned about the T-SQL commands used to create database tables. We have the elementary unit, and its structure keeps the necessary information we want to use at some stage of the application functionality. How data are "pulled" out of the structures of database tables is described in this chapter.

4.1 BASIC SELECTION OF VALUES (SELECT COMMAND)

The easiest way to collect data from one database table is the T-SQL SELECT command. Its definition contains information on both the data source and the overall description of the data it wants to obtain from the source, including the final formatting.

The syntax of a simple T-SQL SELECT command is as follows:

```
FROM tabulka [alias] [,...]
[WHERE podminka]
```

Example from practice, Example to solve
Let's explain the meaning of all the basic syntax variants step by step:

**Example from practice, Example to solve**

**SELECT * FROM Orders**

This example presents the basic form of SELECT that returns all the records from the database table. Notice the * character that represents a placeholder of all database table columns in T-SQL language. Personally, I do not recommend this form of entry. Albeit it correctly returns the desired result, however, due to the optimization of queries and requests for the quickest possible response of the database server, it is always recommended to define all the columns in SELECT command we want to acquire using the SELECT command. The optimal version of the SELECT command would therefore be:

**Example from practice, Example to solve**

**SELECT orderid, empid, orderts FROM Orders**

The above mentioned command returns only three required items from the entire set of defined columns of the database table - order id, employee id and the date and time of the order.

If you require information about a specific order, we have to specify the condition restricting the returned set of records:

**Example from practice, Example to solve**

**SELECT orderid, empid, orderts FROM Orders WHERE custid=71**
This example returns a list of orders of a single customer whose ID is defined by the value of 71 in the database.

If we wanted to gather a group of orders according to an employee who had created the order and according to the year of creation, we use the following T-SQL command:

```
SELECT orderid, empid, orderts FROM Orders
WHERE custid=71
GROUP BY empid, YEAR(orderdate)
```

If we wanted to arrange the set of returned records according to an employee ID, as well as according to the year of order, it is possible to extend the SELECT command by including the ORDER BY syntax:

```
SELECT orderid, empid, orderts , YEAR(orderdate) AS orederyear FROM Orders
WHERE custid=71
GROUP BY empid, YEAR(orderdate)
ORDER BY empid, orederyear
```

Note the used association of AS (YEAR (orderdate) AS orederyear) syntax. YEAR function returns the value corresponding only to the value determining the year of the DATETIME data type. AS syntax declares that the returned values of this item will be named orederyear.

It is not possible to interchange the correct logical order in SELECT commands.

4.2 NATURAL JOIN
In the previous examples, we have always been limited to a single table. That is why we could excerpt only the data that have been available within the scope of a single table. In most cases, however, we want to excerpt data we will need to read from several tables at the same time. To be able to do this, we need to know one more term, and it is the **natural join**. Natural join is a merger of two or more tables "together". Each row in one table is "extended" by columns from other tables. The system will obtain the corresponding row through the primary key - foreign key link. Natural join is best shown by the following example. We have the tables of BOOK and AUTHORS and WRITTEN_BOOKS table that mediates the link of M: N type between the entities of BOOK and AUTHOR.

### Example from practice, Example to solve

```sql
SELECT prijmeni, jmeno
FROM AUTORI a
INNER JOIN napsane_knihy nk ON nk.autor_id = a.id
INNER JOIN kniha k ON k.id = nk.kniha_id
WHERE k.nazev LIKE 'Jak publikovat články'
```

We extend the last example by including two more database tables of PUBLISHER and PRINTED COPY. Let us request a search of all publishers where the works of Robin Cook were published in 1996. The answer is given by the following example:

### Example from practice, Example to solve

```sql
SELECT nakladatelství.název
FROM nakladatelství, výtisk, kniha, napsané_knihy, autoři
WHERE nakladatelství.id = výtisk.nakladatel_id
AND výtisk.kniha_id = kniha.id
AND kniha.id = napsané_knihy.kniha_id
AND napsané_knihy.autor_id = autoři.id
AND nakladatelství.rok_vydání = 1996
AND autoři.jméno = 'Robin'
AND autoři.příjmení = 'Cook'
```
There can be duplications in the above presented examples. It means that some rows can be repeated in the output. One of the possible tools to eliminate these unnecessary duplications is to use a DISTINCT modifier.

4.3 MODIFIERS

Modifiers are entered immediately after the keyword SELECT, before the names of the projected columns:

Example from practice, Example to solve

SELECT [MODIFIKÁTOR] jména projektovaných sloupců

4.3.1 DISTINCT

Let us extend the last example by including two more database tables of PUBLISHER and PRINTED COPY. Let us request a search of all publishers where the works of Robin Cook were published in 1996.

However, if a publisher published, e.g., 3 different works by Robin Cook in 1996, we would get the name of such a publisher three times in the final printout. DISTINCT modifier serves to eliminate duplications in the replies to SELECT command. It means that the anticipated reply to the query in the last example would be received using the following command:

Example from practice, Example to solve

SELECT DISTINCT nd.nazev
FROM AUTORI a
INNER JOIN napsane_knihy nk ON
    nk.autor_id = a.id
INNER JOIN kniha k ON
    k.id = nk.kniha_id
INNER JOIN VYTISK v ON
    v.kniha_id = k.id
INNER JOIN nakladatelstvi nd ON
4.3.2 TOP

TOP keyword is another frequently used modifier. It can sometimes happen that there are many returned rows matching the query. The TOP modifier is used to ensure the return of the first N rows only. Number N is entered directly in the query behind the TOP keyword. If we wanted to receive the first five authors of the publication of "How to publish articles" in alphabetical order, we would write the following query:

Example from practice, Example to solve

```
SELECT TOP 5 prijmeni, jmeno
FROM AUTORI a
    INNER JOIN napsane_knihy nk ON
        nk.autor_id = a.id
    INNER JOIN kniha k ON
        k.id = nk.kniha_id
WHERE
    k.nazev LIKE 'Jak publikovat články'
```

4.4 MODIFICATION OF COLUMN HEADERS

Whenever we receive a list of rows following a query, the columns in the list are named exactly as written by us during the creation of the table. SQL language allows us to change the column names according to our requirements. AS keyword, which is entered behind the column name, and is followed by a new name, serves for this purpose:

Example from practice, Example to solve
ELECT TOP 10 kniha_id AS cislo, nazev AS dilo FROM kniha

Summary of terms

Commands SELECT, JOIN, DISTINCT, TOP

Questions

1. What is T-SQL syntax of SELECT command?
2. What is the natural join of two and more database tables?
3. What is the function of DISTINCT modifier?
4. What is the function of TOP modifier?

Reference sources

5 DATA SORTING AND FILTERING

Study time

1.5 hours

Objective

To become familiar with the basic T-SQL operators and their meanings.

Explication

5.1 EXISTS OPERATOR

This operator returns us the truth value of TRUE or FALSE based on whether the nested SQL query returns a value.

This means that it is not applied to any column or value. If the nested SQL query returns at least one row, then EXISTS operator returns TRUE. It has another peculiarity. The nested SQL query in the argument of the EXISTS operator is evaluated separately for each row. That is why it is necessary to consider using EXISTS operator, since poorly designed queries could lead to a relatively large load of the SQL server.

While other operators of nested SQL queries forced us to use aliases for the names of the tables, in case of EXISTS operator, it is almost a necessity. This is the result of the fact that a nested query is evaluated separately for each row and it must avoid a collision of two tables with the same names.

Example from practice, Example to solve

```
SELECT název
FROM kniha
WHERE NOT EXISTS (
    SELECT autor_id
)
FROM napsané_knihy
WHERE kniha.id = napsané_knihy.publikace_id

Note that I have not used table aliases in the example – they are useless, of course, since there is no collision of the table names here.

5.2 SQL NESTED QUERIES

We use nested queries where we need to get some information based on other data stored in a database. A nested query is nothing more than a SELECT command nested into another SELECT command. We use nested queries where we first need to find out some information and, depending on this information, we need to find out more information. The following query can serve as a typical example. Write a list of all employees who have above-average salary. Using conventional SQL queries that are currently known, we will receive this result in two steps. First, we find out what the average salary in the company is (use the aggregate function AVG):

Example from practice, Example to solve

```
SELECT AVG(plat)
FROM platy
```

This query will return e.g. the number of 15600. We will remember this number and use it in the construction of the following query:

Example from practice, Example to solve

```
SELECT jméno, příjmení
FROM platy
WHERE plat > 15600
```

You will certainly agree that, although this example is quite simple, it would sometimes be more comfortable if we could put the two steps into one, i.e. we would write a single command to receive the necessary list of employees.
We can find out the above-mentioned query by means of a nested query:

```
Example from practice, Example to solve
```

```
SELECT jméno, příjmení  
FROM platy  
WHERE plat > (SELECT AVG(plat) FROM platy)
```

The restriction of the above-mentioned command includes a condition of salary > calculated value. The calculated value is evaluated by a nested SELECT command to a single value, the condition is entered in the right way and the query is correct. For the sake of completeness, it has to be mentioned that the nested SELECT is evaluated only once, the SQL server "remembers" the value as it compares all the other values in the table with that one.

### 5.3 OPERATORS

Nesting of inquiries extends the conventional set of relational operators by other operators, which are applied from the left side to the column and from the right side to the nested query returning multiple values. They include IN, ANY (SOME), ALL operators.

IN operator is used for simple comparison whether the value of the column on the left appears among the values returned by a nested query. If we, for example, requested the names of books published between 1997 and 2000, such a query using the IN operator could be written as:

```
Example from practice, Example to solve
```

```
SELECT název  
FROM kniha, výtisk  
WHERE kniha.id = výtisk.publikace_id  
```

The expression in brackets is a set of all values in which the given value is to be searched. The listing is usually not presented explicitly, but it is returned by a nested query.
Example from practice, Example to solve

```sql
SELECT název
FROM kniha, autoři, napsané_knihy, výtisk
WHERE kniha.id = napsané_knihy.publikace_id
    AND napsané_knihy.autor_id = autoři.id
    AND kniha.id = výtisk.publikace_id
    AND rok IN ( SELECT rok
                 FROM autoři a, výtisk v, napsané_knihy n
                 WHERE a.id = n.autor_id
                    AND n.publikace_id = v.publikace_id
                    AND a.jméno LIKE 'Alois'
                    AND a.příjmení LIKE 'Jirásek')
```

SOME or ANY operator determine that the relations refer to at least one of the values returned by a nested query, while ALL operator applies the relation to all the values returned by the sub-query.

### 5.4 SQL QUERIES WITH AGGREGATION

Aggregate function is used to obtain total aggregated data, such as arithmetic averages, totals, minima or maxima.

Example from practice, Example to solve

```sql
SELECT seznam sloupců
FROM seznam tabulek
    [WHERE podminky]
    [GROUP BY výrazy pro seskupení]
    [ORDER BY dle čeho třidit]
```

#### 5.4.1 COUNT(*)

It is the most basic aggregate function, which is used only to obtain the number of records within a single group of aggregated rows. The simplest use of aggregate function COUNT is as follows:
Example from practice, Example to solve

```sql
SELECT COUNT(*)
FROM KNIHA
```

This command returns the number of records in the BOOK table. COUNT function can be applied to any table and the reply will always be one number indicating the number of records in the table. COUNT can also be used for counting the number of records within the individual groups.

5.4.2 SUM(column)

Aggregate function SUM (column) returns the sum of values in the given column within the frame of the aggregated group of records. If we requested a list of all titles of books and publisher, and the sum of the prices of copies of the work in the publisher for each book – publisher pair, we would enter the following command:

```sql
SELECT kniha.název, nakladatelství.název, SUM(cena)
FROM kniha, výtisk, nakladatelství
WHERE kniha.id = výtisk.kniha_id
    AND výtisk.nakladatel_id = nakladatelství.id
GROUP BY kniha.název, nakladatelství.název
```

5.4.3 AVG(column)

AVG function calculates the arithmetic average of selected values of the specified column. Let us use the following example. Let us take only publishers from Brno and determine the average weight of all the books published by these publishers:

```sql
SELECT kniha.název, nakladatelství.název, AVG(cena)
FROM kniha, výtisk, nakladatelství
WHERE kniha.id = výtisk.kniha_id
    AND výtisk.nakladatel_id = nakladatelství.id
GROUP BY kniha.název, nakladatelství.název
```
5.4.4 MIN (column)

MIN aggregate function is used to return the minima of the values in the column of aggregate records. The following command lists all publishers, and it writes the year in which the oldest book was published for each of them.

```
SELECT název, MIN(rok)
FROM výtisk, nakladatelství
WHERE nakladatelství.id = výtisk.nakladatel_id
GROUP BY název, kniha_id
```

5.4.5 MAX (column)

MAX function returns the maxima of the values of the specified column. This example will list the publishers, and the most expensive copy of each of them will be indicated.

```
SELECT název, MAX(cena)
FROM nakladatelství, výtisk
WHERE nakladatelství.id = výtisk.nakladatel_id
GROUP BY název
```
5.4.6 Further use of aggregate functions

While it is nice to get all the groups of aggregate records in the output, sometimes we need to select only those records within the scope of such records that will also meet other conditions. In other words, we have the opportunity to perform additional restrictions after the aggregation. Additional restriction is entered in the SELECT command in HAVING section:

Example from practice, Example to solve

```
SELECT kniha.název, nakladatelství.název, SUM(cena)
FROM kniha, výtisk, nakladatelství
WHERE kniha.id = výtisk.kniha_id
    AND výtisk.nakladatel_id = nakladatelství.id
GROUP BY kniha.název, nakladatelství.název
HAVING count(*) >=3
```

The above-presented example generates a list of all the books and a sum of the prices of copies of the work in the given publishing house for each books – publisher pair. However, the list includes only the publishers that issued at least three different editions of the book.

Summary of terms

EXISTS, nested query, operator, aggregation

Questions

1. What purpose is the EXISTS operator used for?
2. Where is it suitable to use nested queries?
3. What operators do you know?
4. What aggregate functions do you know?
6 THE USE AND WORK WITH STORED PROCEDURES IN MS SQL SERVER

Study time

3 hours

Objective

Learning the basic means of conditional processing, flow control, cursors and the subsequent implementation within the scope of stored procedures or user functions, while explaining the difference between stored procedures with user function.

Explication

T-SQL provides a wide range of functionality for programmers or database administrators. Transact SQL facilitates the management of SQL queries, conditional processing using IF, ELSE, CASE, and WHILE structures, the use of built-in functions of MSSQL server, or even sequential processing using cursors. The great strength of T-SQL can be found in stored procedures through which programmers can define SQL batches and reuse code, define triggers, or, e.g., solve part of the application logic on the database server. Administrators can use the system stored procedures to administer the server, since the system procedures allow you to use T-SQL to set a wide range of SQL Server parameters, to create databases and make backups, to set permissions, etc ...

6.1 CONDITIONAL PROCESSING

T-SQL makes it possible to return query values on the basis of further processing and logical evaluation of expressions. We can execute a block of commands, provided that the condition has been met, or execute a completely different code if the condition has not been met. Based on the obtained values, these values can be processed further or completely changed or transformed.
6.1.1 IF, ELSE

IF, ELSE structure makes it possible to evaluate a logical expression and perform a query according to its outcome.

**Example from practice, Example to solve**

```
IF Boolean_expression { sql_statement | statement_block }
    [ ELSE { sql_statement | statement_block } ]
```

**Example:**

```sql
DECLARE @Condition int
SET @Condition = 250
IF @Condition > 100
    BEGIN
        SELECT TOP 100 * FROM TABLE
    END
ELSE
    BEGIN
        SELECT TOP @Condition * FROM TABLE
    END
```

6.1.2 CASE

CASE function allows you to return a value based on the evaluation of one or more expressions.

**Example from practice, Example to solve**

**Syntax:**

```
CASE input_expression
```

```sql
```
WHEN when_expression THEN result_expression [ ...n ]
[ ELSE else_result_expression ]
END

**Example:**

```sql
SELECT firstname, surname,
CASE gender
    WHEN 'male' THEN 'M'
    WHEN 'female' THEN 'F'
    ELSE 'UNKNOWN'
END,
Age, birthdate FROM peoples
```

### 6.2 FLOW CONTROL

T-SQL allows you to process blocks of commands repeatedly, to move within them, or to leave them immediately with a condition. Using WHILE cycle, you can execute a block of commands until a condition defined by us is met, using GOTO command, we can move within the block of commands at will or leave the blocks using RETURN command.

#### 6.2.1 RETURN

RETURN allows you to immediately leave the current query, batch, or procedure and to return to the parent block with a return value.

**Example from practice, Example to solve**

**Syntax:**

```
RETURN [ integer_expression ]
```

**Example:**
CREATE PROCEDURE findjobs @nm sysname = NULL

AS

IF @nm IS NULL
    BEGIN
        PRINT 'You must give a user name'
        RETURN
    END
ELSE
    BEGIN
        SELECT o.name, o.id, o.uid
        FROM sysobjects o INNER JOIN master..syslogins l
            ON o.uid = l.sid
        WHERE l.name = @nm
    END;

6.2.2 WHILE

It allows you to perform a command or a batch of commands in a cycle.

Example from practice, Example to solve

Syntax:

    WHILE Boolean_expression
        { sql_statement | statement_block | BREAK | CONTINUE }

Example:

DECLARE @nomrows int
WHILE(SELECT @numrows=count(*) FROM anytable WHERE columnA>10)
BEGIN
IF @numrows=1 BREAK
UPDATE anytable SET columnA=(columnA+1)
END

6.2.3 GO TO

Allows you to perform leaps in SQL batches.

Example from practice, Example to solve

Syntax:

GOTO label

Example:

... UPDATE anytable SET columnA=1
IF @@ROWCOUNT = 0 GOTO endblock
UPDATE anytable SET columnB=2
...
...
endblock:
RETURN

6.3 CURSORS

Cursors allow sequential processing of rows of queries. Sometimes, it is necessary to process the rows one by one, instead of the whole batch, and this is exactly what cursors are used for. However, unless
it is absolutely necessary, it is better to use different structures. Cursors can in fact cause some problems while they are running, one of which is a large memory requirement, and cursors can even cause the exhaustion of all available memory, and they can also cause locking of the means, but their speed is probably their largest problem. MSSQL server is based on work with sets of rows and not sequential processing, which is why work with sets is always faster. Sometimes, however, your application requires the use of cursors and you will appreciate their functionality a lot. // Cursor is defined by a SQL query that returns a set of rows. The rows can be taken from it one at a time, or in blocks, in both directions, according to the cursor definition. When you're finished with the cursor, it must be explicitly closed and deallocated in order to release all the means the cursor have allocated. Otherwise, the cursor remains in the memory and it will block it. There can be a problem as well if you have allocated two identically named cursors (although not in the same block!), and it is better to prevent it by giving the cursor a different name according to its function in each procedure.

6.3.1 Cursor life cycle

1. cursor declaration
2. cursor opening
3. work with rows
4. cursor closing
5. cursor deallocation

6.3.2 Cursor declaration

As early as during the cursor declaration, we define how to work with it, as well as what data we will work with. We use SQL query to define what data we will work with and additional declarations can determine whether we will open the cursor only for reading, whether we will move only forward or back as well, or, e.g., the cursor visibility.

Example from practice, Example to solve

Syntax:

```
DECLARE cursor_name [ INSENSITIVE ] [ SCROLL ] CURSOR

FOR select_statement

[ FOR { READ ONLY | UPDATE [ OF column_name [ ,...n ] ] } ]
```
Example:

DECLARE cursorABC CURSOR
FOR SELECT columnA, columnB, columnC FROM anytab

6.3.3 Cursor opening

OPEN command will fill the cursor.

Example from practice, Example to solve

Syntax:

OPEN { { [ GLOBAL ] cursor_name } | cursor_variable_name }

Example:

OPEN cursorABC

6.3.4 Work with rows

We can work with the rows using FETCH command, which loads values of the individual rows into the variables. The outcome of fetch call is stored in the variable of @@FETCH_STATUS (0 success, -1 failure, -2 missing row).

Example from practice, Example to solve

Syntax:

FETCH
Example:

FETCH NEXT FROM cursorABC INTO @A, @B, @C

### 6.3.5 Cursor closing and deallocation

- CLOSE command will release the memory the cursor has allocated to load the rows
- DEALLOCATE command will release all the means the cursor has allocated

#### Example from practice, Example to solve

Syntax:

```
CLOSE { { [ GLOBAL ] cursor_name } | cursor_variable_name } 
DEALLOCATE { { [ GLOBAL ] cursor_name } | @cursor_variable_name } 
```

Example:

```
CLOSE cursorABC 
DEALLOCATE cursorABC 
```
-- deklarace proměnných
DECLARE @A int
DECLARE @B varchar(10)
DECLARE @C decimal(10,2)
-- deklarace kurzoru
DECLARE cursorABC CURSOR
FORWRD_ONLY
FOR SELECT columnA,columnB,columnC FROM anytable
-- otevření kurzoru
OPEN cursorABC
-- načte řádek do kurzoru
FETCH NEXT FROM cursorABC INTO @A,@B,@C
-- cyklus pro zpracování všech řádek
WHILE @@FETCH_STATUS = 0
BEGIN
-- zde můžeme zpracovávat řádky
    print @A
    print @B
    print @C
-- načte další řádek
FETCH NEXT
END
-- uzavření kurzoru
CLOSE cursorABC
-- dealokace kurzoru
6.4 STORED PROCEDURES

A stored procedure is a gathering of T-SQL commands into a single object stored on the SQL server. When the procedure is executed for the first time after starting the instance of the SQL Server, a plan for optimal data access is stored in the cache; the server then uses this plan during each following start-up, which can speed up the execution of a query. This is one and not the only one of the advantages of stored procedures. Another advantage is the reusability of the code. Instead of defining a dedicated block of commands wherever it is necessary, it is better to use a stored procedure. If this code is to be executed in some of your applications, the benefits will be multiplied. If, for example, the storage of data in tables changed, you would have to rewrite your application or multiple applications at all locations, in a worst case scenario even recompile them, but if you use a stored procedure, you can just change it and the applications will remain virtually unaffected. Using the procedures is also safer than a direct access to tables from the point of view of the db administrator.

Example from practice, Example to solve

Syntax:

CREATE { PROC | PROCEDURE } [schema_name.] procedure_name [ ; number ]

[ { @parameter [ type_schema_name. ] data_type } ]

[ VARYING ] [ = DEFAULT ] [ OUT | OUTPUT ] [READONLY]

] [ ,,...n ]

[ WITH <procedure_option> [ ,,...n ] ]

[ FOR REPLICAUTION ]
AS { <sql_statement> [ ; ] [ ...n ] | <method_specifier> } 

[ ]

<procedure_option> ::= 

[ ENCRYPTION ]
[ RECOMPILE ]
[ EXECUTE AS Clause ]

<sql_statement> ::= 

{ [ BEGIN ] statements [ END ] }

<method_specifier> ::= 

EXTERNAL NAME assembly_name.class_name.method_name

**Example:**

CREATE procedure [dbo].[MyProcedure]
@pid int,@other int,@sometext varchar(100)
AS

IF ISNULL(@pid,0)=0
  RETURN

EXEC ExecuteAnotherProcedure @pid

IF ISNULL(@other ,0)<>0
BEGIN
  DECLARE @a int
  SELECT @a=a FROM anytable WHERE a=@pid AND b=@other
6.5 USER-DEFINED FUNCTIONS

User-defined functions, like stored procedures, have many advantages – storing into cache, code reuse, etc., unlike procedures, however, we can also use functions where procedures cannot be used (e.g. in the SELECT clause).

Types of user-defined functions:
- Scalar user-defined functions
- Direct tabular user-defined functions
- Multicommand user-defined functions

6.5.1 Scalar user-defined functions

- can take zero or more arguments
- return one value only

Example from practice, Example to solve

Syntax:

```sql
CREATE FUNCTION [ schema_name. ] function_name
( [ [ @parameter_name [ AS ] [ type_schema_name. ] parameter_data_type
    [ = DEFAULT ] [ readonly ] ]
    [ ,...,n ]
] )
RETURNS return_data_type
    [ WITH <function_option> [ ,...,n ] ]
```
Example:

CREATE FUNCTION dbo.udf_DayOfWeek(@dtDate DATETIME)
RETURNS VARCHAR(10)
AS
BEGIN
    DECLARE @rtDayofWeek VARCHAR(10)
    SELECT @rtDayofWeek = CASE DATEPART(weekday,@dtDate)
        WHEN 1 THEN 'Monday'
        WHEN 2 THEN 'Tuesday'
        WHEN 3 THEN 'Wednesday'
        WHEN 4 THEN 'Thursday'
        WHEN 5 THEN 'Friday'
        WHEN 6 THEN 'Saturday'
        WHEN 7 THEN 'Sunday'
    END
    RETURN (@rtDayofWeek)
END
GO

-- použití funkce
SELECT udf_DayOfWeek(getdate())
6.5.2 Direct tabular user-defined functions

- can take zero or more arguments
- return data on the basis of a single SELECT command
- FROM clause refers to them

Example from practice, Example to solve

Syntax:

```sql
CREATE FUNCTION [ schema_name. ] function_name
( [ [ @parameter_name [ AS ] [ type_schema_name. ] parameter_data_type
    [ = DEFAULT ] [ READONLY ] ]
    [ ,...n ]
] )
)

RETURNS TABLE
[ WITH <function_option> [ ,...n ] ]
[ AS ]
RETURN [ ( ] select_stmt [ ) ]
[ ; ]
```

Example:

```sql
CREATE FUNCTION dbo.GetByCity(@city varchar(100))
RETURNS TABLE
AS
RETURN
(
SELECT * FROM peoples WHERE city=@city
```
Example of function use:

```sql
SELECT * FROM GetByCity('Prague')
```

### 6.5.3 Multicommand user-defined functions

Unlike direct table functions, which are limited to only one select command, multicommand functions can return a result defined by means of multiple T-SQL commands.

**Example from practice, Example to solve**

**Syntax:**

```sql
CREATE FUNCTION [ schema_name. ] function_name
( [ [ { @parameter_name [ AS ] [ type_schema_name. ] parameter_data_type
   [ = DEFAULT ] [READONLY] } ]
   [ ,...n ]
 )
RETURNS @return_variable TABLE <table_type_definition>
[ WITH <function_option> [ ,...n ] ]
[ AS ]
BEGIN
    function_body
    RETURN
END
[ ; ]
```
Summary of terms

IF, ELSE, CASE, RETURN, WHILE, GOTO, cursor, stored procedure, user-defined function

Questions

1. What purposes are CASE and IF - ELSE commands used for and what is the difference between them?
2. What commands to control the flow do you know and briefly explain the differences between the individual versions?
3. What is a cursor and what is the syntax for its creation?
4. What is a stored procedure, what is its use and the syntax to create it?
5. What is the difference between the stored procedure and user-defined function?

Reference sources


7 CREATION OF DATABASE APPLICATIONS

Study time
5 hours

Objective
An introduction of the basic options and technologies for the development of database applications in MS .NET Framework environment.

Explication

7.1 WHAT IS A DATABASE GOOD FOR?

You may wonder what we need a database for. We might as well store data in text files, XML, or other data files.

In scholar literature, we can come across the term RDBMS (Relation DataBase Management System) instead of the term database. This term could be loosely translated as "data base management system" or "database machine". It is not just a storage location of data. It is a very sophisticated and finely tuned tool capable of solving a lot of problems instead of us and it is extremely easy to use. We already know that SQL language is a mediator for the communication with databases. In the following chapters, we will learn how to deal with relational databases using the .NET Framework.

7.2 APPROACHES TO WORK WITH RELATIONAL DATABASES IN .NET

The object world and the world of relational databases (relational world) are very different. These are two different philosophies that, I dare say, are incompatible.

Relational databases represent a proven way of how to work with data. Although there are fully object-oriented databases, it is not beneficial for companies to invest money into them, which is the reason why they haven´t established themselves yet. Of course, after the revolution in programming and the
arrival of objects, there was a problem with storing data, since relational databases do not work with objects and cannot store objects. There are several ways to deal with this issue.

7.2.1 Non-object programming

The first possibility is, of course, programming without any objects at all. However, this would mean going against the tide, as we would not be able to use any third-party components and our code would be very poor. Since C# is an object language, it would not even be possible in this type of programming.

7.2.2 Database Wrapper

The so-called wrapper allows us to work with a database as if it was an object, however we still communicate with it in SQL language. We therefore mix the object and relational code. The approach is a compromise and requires a little twist of the OOP philosophy. The advantage is maintaining the performance and capabilities of the database for the price of a slight degradation of OOP ideas.

We can usually see data from a database as values in the table (which is the object), and we lose the opportunity to allocate some functionality to the entities. Instead, we gather it into the so-called managers. It is also possible to partially map data into existing classes; however, we will not achieve a full-fledged concept of object model.

7.2.3 Object relational mapping

Object-relational mapping (ORM) strictly follows the idea of OOP. Instead of field of values, we immediately receive objects from the database and they have methods attached to them. We do not communicate in SQL language at all, we see tables in the database as a collection of objects we can work with using conventional means of the language. We are actually completely shielded from the fact that we are working with a relational database.

The disadvantage of this system is the degradation of the database performance; SQL queries are automatically generated and are often ineffective. Another problem of ORM is that that it is very complicated. Fortunately, C# has perfectly tuned ORM directly in itself, which means we do not have to worry about anything. On the contrary, for example to get ORM running in PHP is not easy, and that is where the wrapper is preferred.

7.2.4 Object databases

In addition to relational databases, there are also the aforementioned object databases. They deal with the problem of incompatibility between the object and relational approach. They provide the same comfort as ORM, but internally there is no need to convert data into tables, they are stored directly as objects. Theoretically, there is no performance or any other reason why they should not replace relational
databases. In practice, unfortunately, they are hardly ever used, and we can only hope that this will change over time. Those who are interested can take a look, e.g., at MongoDB project.

7.3 POSSIBLE CONNECTIONS TO DATABASE

In .NETu, there are several possibilities of the use of a database in our application.

7.3.1 Connected application

We use connected application when we often need to read or change data in real time. Using DataReader, Command and Connection classes, we send commands directly to the database in SQL language and receive the results.

![Figure 2 Example of connected application](image)

7.3.2 Disconnected application - DataSet

Disconnected application works on the basis of the so-called DataSet that we have in short-term memory, which contains data from the database. The application works with the DataSet and sometimes DataSet is synchronized with a live database on the server (disk). For the price of less up-to-date data we gain increased speed and more comfortable work.

DataSet contains tables, a table contains rows and a row contains columns. A table is an object, where we can add rows and modify them without writing the SQL code. When we want to run a command on the database, we use the DataAdapter, which is used to fill the DataSet with data. We no longer have to enter the commands in the SQL language of the database in question.
So we have a certain object abstraction, we work with tables using object approach, but the data are still only columns in the table, not instances of some objects. We also still use SQL language. From the point of view of the above presented approaches, it is a wrapper.

### 7.3.3 LINQ To SQL

LINQ to SQL provides a complete object abstraction over the database (object relational mapping). We work with the database as if it was, e.g., a list of objects and we do not deal with SQL queries, we do not know about tables or columns, all happens automatically in the background, and the query returns us full-fledged objects straight away. The price for such luxury is worse optimization of queries, but that usually does not matter.

There is the so-called DataClasses in our application, which is an object database structure. It includes classes for the individual tables, while the table columns are the characteristics of the classes. We communicate only with DataClasses in the application, and it subsequently communicates with the database and performs SQL queries instead of us in the background using LINQ to SQL. We will not be in touch with SQL language at all, and we work with the database as if it was an object structure in short-term memory.
7.3.4 Entity Framework

Entity Framework goes even further than LINQ to SQL. It is a more complex technology, which is more capable (e.g. M:N link), but is more cumbersome to use. We will learn more about this technology in chapter 7.5.

7.4 COMMUNICATION WITH DATABASE BY MEANS OF ADO.NET

.NET Framework contains its own technology for data access - ADO.NET, which consists of y classes that allow .NET applications to connect to data sources, execute commands, and manage disconnected data.

7.4.1 ADO.NET architecture

ADO.NET uses multi-layer architecture that is concentrated around several key terms, such as Connection (connection object), Command (command object) and a DataSet (data set object). Unlike database technologies in the past, where a general range of objects was used regardless of what the data source was (for example, if you want to get some records from an Oracle database, you use the same Connection class as in the case of access to MS SQL Server), ADO.NET uses data providers to access databases (data provider models), which are available for each database source separately. This means you have data provider objects available for Oracle database and others for MS SQL Server.

7.4.2 Data provider ADO.NET

Data provider is a set of ADO.NET classes which allows access to specific database, efficiently performs SQL commands and retrieves or modifies data.

The classes, which make up each data provider, include:

- **Connection** – allows connection to a data source
- **Command** – using it, you can perform SQL commands or activate processing of stored procedures
- **DataReader** – provides quick access to data obtained from the query. Data are used for reading only and only in the way "forward"
- **DataAdapter** – allows filling of a DataSet object that represents a data set object (disconnected collection of tables and relations), which we obtained from the data source. Another feature is the ability to project data changes into the data source

One of the key features of ADO.NET model is its expansibility, i.e. it allows the proprietary data sources to create their own providers.

.NET Framework is equipped with a set of 4 providers:
7.4.3 Basic ADO.NET classes

ADO.NET contains two types of objects:

- **Objects based on connection** - are objects of Data Providers, including Connection, Command, DataAdapter, and DataReader objects. They allow you to connect to a database, to perform SQL commands, to go through a set of final data and to fill a data set object.
- **Objects based on content** - are only data "boxes". They include data set objects – DataSet, data columns – DataColumn, data rows - DataRow, data relations - DataRelation and several others. They are completely independent on the used data source and are located in the System.Data namespace.

7.4.3.1 **Connection class**

Connection class allows you to connect to the data source which you want to communicate with. The definition of connection is the basis of any data communication.

When you create a Connection object, it is always necessary to define the connection string, which is a series of settings of name/value separated by semicolons. The sequence of settings or the size of the letters is irrelevant. All these settings represent the basic information required to create the actual connection.

A connection string must always contain at least the following information:

- Server, where the database which you want to connect to is located
- Database which we want to work with
- The way the database verifies your identity
A definition of a connection string that you would use to connect to the local Northwind database through an integrated security (for login, you use the user who is currently logged on to Windows system):

```
String connectionString = "Data Source=localhost; Initial Catalog=Northwind; Integrated Security=SSPI";
```

If you do not want to use Windows integrated security system, you must connect to a database using a valid combination of username and password.

```
String connectionString = "Data Source=localhost; Initial Catalog=Northwind; user id=sa; password=opensame";
```

For an effective application structure, it is not suitable to insert the connection string directly into the code. A suitable place to store the connection strings is the <connectionStrings> section of the application configuration file.

Once you have defined the connection string, the connection management is easy - simply open the connection using Open() method and terminate the connection after the end of the data communication using Close().

```
Example from practice, Example to solve
```

```
String connectionString = ConfigurationManager.ConnectionStrings["Northwind"];

SqlConnection con = new SqlConnection(connectionString);
try{
    con.Open();
    string version = con.ServerVersion;
    string connectionState = con.State.ToString();
```
7.4.3.2 **SqlCommand class**

This class is used for the actual execution of an SQL query over the database specified by the Connection class. Using this class, you can execute not only INSERT, UPDATE, DELETE, SELECT operations, but you can also activate the processing of a stored procedure. System.Data.SqlClient class is an implementation of DBCommand class specific to MS SQL Server.

7.4.3.2.1 **ExecuteNonQuery method**

ExecuteNonQuery method is used to execute SQL commands, whose output is zero data set. It is used, e.g. to execute INSERT, UPDATE, and DELETE commands, where we do not expect any return value.

The following example demonstrates the use of INSERT data into the database:

```csharp
SqlConnection cn = new SqlConnection();
cn.ConnectionString = "Server=myServerAddress;Database=myDataBase;User Id=myUsername;Password=myPassword;";
cn.Open();
SqlCommand cmd = new SqlCommand();
cmd.Connection = cn;
cmd.CommandType = CommandType.Text;
cmd.CommandText = "INSERT INTO Person (FirstName, LastName) " + "VALUES ('Joe', 'Smith')";
cmd.ExecuteNonQuery();
```
Note the 3 basic properties of the SqlCommand class which have to be initialized prior to calling the ExecuteNonQuery method. The first of them is the Connection property, which must be set up for proper connection to the database server. Another important property is the specification of the shape of the actual SQL command – CommandType property. If you define the command directly by means of an SQL command within the source code, you will use CommandType.Text setting as in the previous example. If you want to activate the processing of a stored procedure, use the CommandType.StoredProcedure setting, and if the definition of the stored procedure contains a definition of the input parameters, the SqlCommand must contain their definition. The following example shows you how to proceed in such a case.

**Example from practice, Example to solve**

Let us have a defined stored procedure within the scope of MS SQL Server database which will be used to enter data on the name and surname of the person into the PERSON database table. The stored procedure will be named PersonInsert and it has been defined.

```sql
CREATE PROCEDURE PersonInsert
    @FirstName varchar(50),
    @LastName varchar(50)
AS
BEGIN
    INSERT INTO PERSON (FirstName, LastName) VALUES (@FirstName, @LastName)
END
```

The following code then ensures the call for the stored procedure with the input parameters of @FirstName and @LastName.

```csharp
SqlConnection cn = new SqlConnection();
cn.ConnectionString = "Server=myServerAddress;Database=myDataBase;User Id=myUsername;Password=myPassword;";
```
SqlCommand cmd = new SqlCommand();
cmd.Connection = cn;
cmd.CommandType = CommandType.StoredProcedure;
cmd.CommandText = "PersonInsert";
cmd.Parameters.Add(new SqlParameter("@FirstName", "Joe"));
cmd.Parameters.Add(new SqlParameter("@LastName", "Smith"));
cmd.ExecuteNonQuery();

SqlCommand class has Parameters feature, which can be used to define the input parameters of stored procedures.

7.4.3.2.2 ExecuteReader method

ExecuteReader method is used to obtain the output set of query data by means of SqlCommand class. The output of this method is a DBDataReader object, when MS SQL Server is used, it is SqlDataReader. DBDataReader object represents forward-indexed output data set requiring permanently open database connection during the reading of the output data set. Forward indexing means that such behaviour when reading data is possible only in successive steps. It is not possible to change the cursor over the current data by a few steps forward or in reverse direction; it is always possible only by the next step.

Example from practice, Example to solve

The following example demonstrates how to open data about persons and the listing of their names and surnames on the screen.

SqlConnection cn = new SqlConnection();
cn.ConnectionString = "Server=myServerAddress;Database=myDataBase;User Id=myUsername;Password=myPassword;";
cn.Open();
SqlCommand cmd = new SqlCommand();
cmd.Connection = cn;
cmd.CommandType = CommandType.Text;
cmd.CommandText = "SELECT * FROM Person";
SqlDataReader dr = cmd.ExecuteReader();
if (dr.HasRows)
{
    while (dr.Read())
    {
        Debug.WriteLine(string.Format("First Name: {0} , Last Name: {1}",
                        dr["FirstName"], dr["LastName"]));
    }
}
dr.Close();
cn.Close();

Note that after data processing, it is necessary to close the work with DataReader object, the actual object used to work with the database server.

7.4.3.2.3 ExecuteScalar method

ExecuteScalar method is used in cases where we know that the result of an SQL query will include a result set with one column and row.

Example from practice, Example to solve

The following example returns the total number of records of Person database table:

SqlConnection cn = new SqlConnection();
cn.ConnectionString = "Server=myServerAddress;Database=myDataBase;User Id=myUsername;Password=myPassword;";
 cn.Open();
SqlCommand cmd = new SqlCommand();
cmd.Connection = cn;
cmd.CommandType = CommandType.Text;
cmd.CommandText = "SELECT COUNT(*) FROM Person";
ExecuteScalar method always returns an object that must be converted to the correct data type required for further processing in the application logic of the application.

### 7.4.3.2.4 DataSet, DataTable and DataAdapter

The use of DataSet and DataTable objects is another way how to obtain the resulting data set. DataTable object is similar to DBDataReader object, except for the fact that it allows working with data in the mode of closed connection with the database. Another important difference is that it is possible to move in both directions within the data set cursor, both forward and in the direction of records with an index that is lower than the current one. A significant difference compared to DBDataReader is the possibility to change data, restore the database connection and transfer the changes back to the database. DataSet is a container for one or more DataTable objects. Using DataSet object makes it possible to execute multiple SQL commands and store their results in the independent DataTable objects. Subsequently, the data in the DataSet can be filtered, sorted, and edited. All these operations are performed in the memory, without any link to the database server. DataAdapter is used to fill the DataSet or DataTable objects and for a subsequent restoration of the connection to the database allowing you to INSERT, UPDATE, or DELETE the modified data.

**Example from practice, Example to solve**

The following example uses a DataSet object with data loaded into it from the Person table that are subsequently listed on the screen by the users:

```csharp
SqlConnection cn = new SqlConnection();
 cn.ConnectionString = "Server=myServerAddress;Database=myDataBase;
                  User Id=myUsername;Password=myPassword;";
 cn.Open();
```
SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM Person", cn);
DataSet ds = new DataSet();
da.Fill(ds, "Person");
cn.Close();
foreach (DataRow row in ds.Tables[0].Rows)
{
    Debug.WriteLine(string.Format("First Name: {0} , Last Name: {1}",
                              row["FirstName"], row["LastName"]));
}

SqlDataAdapter is used to execute the acquisition of data from the database and to fill DataSet object with data by calling the Fill method. Note that, after you call the Fill method, the connection with the database is terminated, however, the DataSet object is still useable for work with the downloaded data, which is not possible in the case of DBDataReader.

DataSet object contains Tables property representing a reference to the DataTable objects containing the result of an SQL query. Each DataTable object contains Rows property containing a collection of DataRow objects representing the specific data for one requested record. The total number of found records is specified by the Count property.

As already mentioned before, DataAdapter allows you to execute INSERT, UPDATE, DELETE operations within the DataTable object. The following example demonstrates these properties:

SqlConnection cn = new SqlConnection();
cn.ConnectionString = "Server=myServerAddress;Database=myDataBase;
                      User Id=myUsername;Password=myPassword;";
cn.Open();
SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM Person", cn);
//Create the insert command
SqlCommand insert = new SqlCommand();
insert.Connection = cn;
insert.CommandType = CommandType.Text;
insert.CommandText = "INSERT INTO Person (FirstName, LastName) VALUES
(@FirstName,
@LastName)";
//Create the parameters
insert.Parameters.Add(new SqlParameter("@FirstName", SqlDbType.VarChar, 50,
            "FirstName")
insert.Parameters.Add(new SqlParameter("@LastName", SqlDbType.VarChar, 50,
the insert command with the DataAdapter.
    da.InsertCommand = insert;

    //Get the data.
    DataSet ds = new DataSet();
    da.Fill(ds, "Person");

    //Add a new row.
    DataRow newRow = ds.Tables[0].NewRow();
    newRow["FirstName"] = "Jane";
    newRow["LastName"] = "Doe";
    ds.Tables[0].Rows.Add(newRow);

    //Update the database.
    da.Update(ds.Tables[0]);
    cn.Close();

7.5 ADO.NET ENTITY FRAMEWORK

The use of ADO.NET Entity Framework classes is another method of .NET Framework used to perform the basic data operations INSERT, UPDATE, and DELETE. Entity Framework is a friendlier graphic interface that enables using "drag and drop" methodology of placing database objects within the frame of the design layer. Generally, this procedure is called Object-Relational Mapping tool or ORM. There are several ORM tools for application development on the market. Entity Framework together with LINQ to SQL is integrated directly into the development environment of MS Visual Studio.

In essence, the core of the Entity Framework is a model. ADO.NET Entity Framework performs a conversion of a relational database scheme into objects.

Compared with the previous ADO.NET access, ADO.NET Entity Framework allows direct definition of the classes that are independent on the database structure, while it defines the mapping of these classes into database tables, thus associating them with the database. The use of such objects within the application leads to the resistance of the application against changes in the database.

ADO.NET Entity Framework uses several levels to define the database tables mapping into objects. The first level is to create the actual database model, followed by the definition of the entity classes, and the final level is the entity layer mapping with the database model.

Generally, these layers are defined as:

- Logical model - defines the relational database model (SSDL)
- Conceptual model - definition of .NET classes of entities (CSDL)
- Mapping model - provides mapping of .NET classes into relational database tables, including the associations (MSL)

**Example from practice, Example to solve**

The core of the work with ADO.NET Framework is a model. The following procedure is used to create such a model of Northwinds sample database:

1. We start Visual Studio 2013
2. Select New Project item from the start menu
3. Select a template to create a Console Application project
4. We name the project, e.g. NorthwindsConsole
5. It is necessary to add references to the desired Entity Framework libraries into the created project
   a. Open Package Manager Console window in Visual Studio (Tools -> NuGet Package Manager -> Package Manager Console)
   b. Make sure that the correct project (NorthwindsConsole) is selected in its top section in the drop-down list of the Default project
   c. Open Package Manager Console window in Visual Studio (Tools -> NuGet Package Manager -> Package Manager Console)
   d. Enter the following command of install package EntityFramework, which will install the package with Entity Framework into your project

**Figure 5 NuGet Package console interface**

**Figure 6 Adding Entity Framework to the project**
6. Use right click to call up a menu in Solution Explorer, where you select Add item and then select New Item
7. From the installed templates, select that you want to create an ADO.NET Entity Data Model
8. We will change the name, e.g. to NorthwindsModel and click on the Add button. It will show the Entity Data Model Wizard

![Image of Solution Explorer with Entity Data ModelWizard](image)

**Figure 7 Creating Entity Framework data model**

9. Because we will create a model over the existing database, select the option of Generate from Database. This procedure is also called Database First access in Entity Framework terminology. Model First is an alternative, which proceeds in the opposite direction, where all the classes are generated first, and the database model is then generated from these classes.
10. Click on Next button. On the next page, it is necessary to define the database connection.
11. Click on the New Connection button and it will show a dialog for the definition of the connection.
12. Select the name of the database server on which the Northwind database is located.
13. Select the type of authentication to the database server. You can choose from two options - Windows Authentication or SQL Server Authentication, where you must enter a valid SQL login name and a password.

14. Enter or select the name of a database from the list - in our case Nothwind

Figure 8 Definition of database connection within Entity Data Model Wizard in MS Visual Studio environment
15. Press the OK button that will take you back to the wizard to create an Entity Data Model. The currently set database connection is stored in app.config file of your application as the default option.

16. In the next step, you have a choice of database objects that you want to map using the model. Select the Tables, Views and Stored Procedures items. Leave the other settings unchanged. Click on the Finish button.

Entity Data Model Wizard automatically generates a model:

![Generated Entity Data Model](image_url)
Notice that NorthwindsModel.edmx file has been added to your project. This file is a graphic representation of all classes that have been modelled and represent all objects in the given database. In its essence, it is an XML definition including all the layers - SSDL, CSDL and MSL.

Click on the items that are gathered within the scope of NorthwindsModel.edmx file. Let's focus on NorthwindsModel.tt file. It represents the so-called Text Transformation Template Toolkit file, also called T4 template. T4 template is used for the automatic generation of the code within the scope of Visual Studio. T4 template is a mix of text blocks and control commands, which are used to generate the source codes. NorthwindsModel.tt file combines several source files generated by this template. A separate file is created for each table, view, or stored procedure.

Open, for example, Category.cs file that contains a class mapping Category database table in Northwinds database. Each feature of this class corresponds to a column of the database table. Among other things, note the feature named Products, which is defined by the data type of ICollection<Product>. The reasons for this definition are the database links of Products table with Category table using foreign key definitions. Thanks to this link, the category can have links to several products. Entity Framework Wizard evaluates all these links for you in the background, and puts emphasis on the actual database model design with the definition of all the necessary links.

7.5.1 Data queries by means of Entity Framework

If we have a model prepared in this way, we can load data from the database without any knowledge of SQL language. The following example loads records from Category database table and displays them on the screen.

```csharp
using (NorthwindsEntities db = new NorthwindsEntities()) {
    var categories = from c in db.Categories
                     select c;
    foreach (Category category in categories) {
        Debug.WriteLine(string.Format("CategoryId: {0},
                                       CategoryName: {1}",
                                       category.CategoryID,
                                       category.CategoryName); 
```
The result is a listing:

- CategoryId: 1, CategoryName: Beverages
- CategoryId: 2, CategoryName: Condiments
- CategoryId: 3, CategoryName: Confections
- CategoryId: 4, CategoryName: Dairy Products
- CategoryId: 5, CategoryName: Grains/Cereals
- CategoryId: 6, CategoryName: Meat/Poultry
- CategoryId: 7, CategoryName: Produce
- CategoryId: 8, CategoryName: Seafood

As you can see, we have obtained data from the database without a definition of the SQL query, only by means of Entity Framework and LINQ query composition. All commands have been assembled and compiled using C# programming language syntax.

The following example demonstrates how to create a LINQ query to load data that contains a link of two tables.

```csharp
using (NorthwindsEntities db = new NorthwindsEntities())
{
    var products = from c in db.Categories
                    join p in db.Products on c.CategoryID equals p.CategoryID
                    select p;
    foreach (Product product in products)
    {
        Debug.WriteLine(string.Format("ProductName: {0}, CategoryName: {1}",
                                        product.ProductName,
                                        category.CategoryName));
    }
}
```

**Example from practice, Example to solve**
Apart from the information about the categories, this case also lists the information about products in the given category. Based on the definition of the foreign key, Entity Framework correctly determines a link between the Product and Category classes.

### 7.5.2 Creation of new records

Adding data using Entity Framework is very easy. The principle is based on creating a new class instance and its properties setting. This instance is added to the database context, and SaveChanges() method helps us to add the new record to the database. As you can see, this process did not require a re-definition of any SQL script again.

The whole procedure is illustrated in the following example, where we want to insert a record about a new category into the database.

```csharp
using (NorthwindsEntities db = new NorthwindsEntities())
{
    Category category = new Category()
    {
        CategoryName = "Alcohol",
        Description = "Happy Beverages"
    };
    db.Categories.Add(category);
    db.SaveChanges();
}
```
7.5.3 Data change

Changing records by means of Entity Framework is a trivial matter, as in the previous case. In comparison with the data entry, it is not necessary to create new class instances, because the instance whose properties we want to change is loaded. Use command SaveChanges to save the record again.

The procedure changing the description of an existing category is demonstrated by the following example.

```
using (NorthwindsEntities db = new NorthwindsEntities())
{
    Category category = db.Categories.First(c => c.CategoryName == "Alcohol");
    category.Description = "Happy People";
    db.SaveChanges();
}
```

7.5.4 Records deletion

Deleting data is similar to the previous operation focused on changing data. The only difference is that the record to be deleted from the database is removed from the context. This operation is demonstrated by the following example.

```
using (NorthwindsEntities db = new NorthwindsEntities())
{
```

---

78
7.5.5 Calling stored procedures in Entity Framework

As shown in the introduction to Chapter 9.5, it is possible to include stored procedures into the newly created model when creating a model using Entity Data Model Wizard. The stored procedure is thus activated only by calling the appropriate model method.

The following example calls CustOrderHist stored procedure, whose input parameter is a customer ID and the call result is a list of all orders of this customer.

```csharp
using (NorthwindsEntities db = new NorthwindsEntities())
{
    var custOrderHist = db.CustOrderHist("ALFKI");
}
```
foreach (CustOrderHist_Result result in custOrderHist)
{
    Debug.WriteLine(string.Format("ProductName: {0}, Total: {1}", result.ProductName, result.Total));
}

Summary of terms

Wrapper, object relational mapping, DataReader, Command, Connection, DataAdapter, DataSet, DataColumn, DataRow, DataTable, ExecuteNonQuery, ExecuteReader, ExecuteScalar, LINQ, ADO.NET, Entity Framework, logical model, conceptual model, mapping model

Questions

1. What are the basic access options to work with relational databases in MS .NET Framework environment?
2. What are the database connection options in .NET?
3. What ADO.NET object is used to connect to the database?
4. What Command object method does not return any records after its execution?
5. What is the difference between DataAdapter and DataSet objects?
6. In cases where you use ADO.NET Entity Framework to create a model representing objects in the database. What class represents the created model?
7. How is stored procedure represented within the scope of ADO.NET Entity Framework?

Reference sources

8 ASP.NET - WEB TECHNOLOGIES FOR THE PRESENTATION OF DATA

Study time

2 hours

Objective

The introduction of ASP.NET technology, the comparison of the technology properties with other used to create web applications.

Explication

ASP.NET is a web framework; in short, it is a set of libraries that enable the creation of web applications in C# language. The libraries contain ready-made solutions to many of the fundamental problems arising in web technologies. I mean, for example, security, user authentication, work with databases, administration of forms etc. ASP is not a programming language, the programming will be performed in C# and you will therefore need the basic knowledge of this language.

8.1 THE BASIC PRINCIPLES OF ASP.NET

The technology is based on client-server architecture. The application in ASP.NET is therefore a program whose output is an HTML page. ASP runs on the server side. To better understand how ASP works, we will take a look at an ordinary static web first.

8.1.1 Static web (without ASP.NET)

In the case of static web, HTML pages are simply stored on the server. As soon as the client (user with a browser) sends a request to the server, the server will simply return exactly the pages it has saved.

In graphic form, the communication looks like this:
8.1.2 Dynamic web (with ASP.NET)

Static website is difficult to administer and has significant limitations. This led to the creation of server languages that allow you to change HTML pages before they are sent to the client. It allows you to produce active elements, such as discussion forums, user login, upload content using editor and so on.

ASP.NET is running on the server, based on the requests from the client, it generates the website and sends it to the client. The client can see only the final HTML, in which there is no sign of ASP.NET (well, except for the fact that the site has an .aspx ending instead of .htm one).

Graphically, the situation would look like this:

![Figure 11 The principle of dynamic web applications](image)

There are huge differences between C# and the competing PHP. While PHP is a dynamic language, C# is strongly typed. It has turned out that such a language may be suitable for websites as well. The big advantage is the possibility to use the .NET framework, which is suitable for the server, because it contains many useful libraries and functions. ASP is one part of the .NET framework.

ASP.NET runs on IIS server (Internet Information Services) and, in theory, it can also run on Apache web server as a module (like PHP).

8.2 WEBFORMS VS. MVC

Microsoft offers us two main ways of creating a website. They are WebForms and MVC. Both are interesting and both have a little bit different approach to web application.

8.2.1 WebForms

WebForms is an attempt to transfer WinForms (i.e. standard form/window applications as we know them from desktop) to the web. The idea is simply based on an attempt to put together a form from toolbox controls in designer (buttons, labels, text fields, etc.), and to allocate events to these controls. From the outside, the application behaves as a desktop one, but there is a more
complicated logic in the background. The task to simulate a desktop application on the web is somewhat difficult, because the HTTP protocol is **stateless**.

The state means, for example, values filled in the fields, ticked checkboxes, selected date in the calendar, and so on. The server has no idea about the state of the application, it only receives a request from a user, and on the basis of the request it generates HTML pages. After sending the page to the client, most of the information on the server ceases to exist. Imagine that you fill something in the form, tick something and send the page. On the new page, we usually want these values to remain filled in, particularly when it is the same form. The form status must therefore be kept somewhere. In ASP.NET, this is done using the so-called **ViewState**. ViewState is nothing more than a hidden form field that keeps the application state in the final HTML page and sends it to the server with another query. The server restores the same application state it had before, executes an action and generates the ViewState state again and forgets the state after that. This is repeated in the background and the client (and usually the programmer as well) feel like they are working with a desktop application. The advantage is a huge speed of the formation of applications, especially in situations where there are a lot of forms. The downside is, of course, increased size of the HTML page, caused by sending ViewState and somewhat more complicated architecture.

### 8.2.2 MVC

MVC is a newer concept and it has a different approach. It divides the web application into components of 3 types: controllers, models, and views (Model View Controller). A **controller** is a controlling component that receives data from the user and communicates with the model. The **model includes logic**; it works with a database, etc. Data are transmitted from the model by a view controller. The **view is actually a template** into which data are inserted to form the final HTML page. Simply put, the goal of the concept is to streamline the web architecture and to divide it into two parts - Logic and Output. The controller then connects these two parts. **MVC is not a substitute of WebForms**, although it may look like that. It is only another alternative to web creation, and there are situations where WebForms option is more beneficial. We will also deal with MVC here.

### 8.3 HOW TO PROGRAM IN ASP.NET?

We already know that WebForms try to get as close to desktop WinForms application technology as possible. Each dynamic side of web applications defines a Web Form. This Web Form contains HTML, respectively XHTML structure defining the basic look of the client interface. XHTML is not very different from HTML, except for their necessity to make pair tags as in case of XML format.

The main element of the site is a Form. The entire Web Forms is built on these forms.

The following example shows how to create an empty Web Form in MS Visual Studio environment. We create a web application that will contain a simple search form to find products according to defined categories and product names. We will use the Entity Framework technology for this purpose, which is described in Chapter 9.5.
8.3.1 Project initialisation

We'll start by creating a new project in Visual Studio. We select a template for ASP.NET applications from the list of installed project templates.

![Figure 12 Creating a project of web application in MS Visual Studio environment](image)

Once we have confirmed the dialogue, another one with the selection of the template will appear. The template is a pre-generated project structure. For our purposes, we'll start with a blank project, which is why we choose an Empty template.
The web application project structure was generated in this way, and we can now add the definitions of the individual Web forms into it.

### 8.3.2 Adding Web Form into the web project

We add page by selecting Add New Item, which will open a dialogue with the defined templates of items that we can add to the project. We select the Web Form option, where we name the newly added page Default.aspx.
The Default.aspx name is the standard for the definition of a page that will open when we enter the main web page. Other technologies give similar name to the default index page.

Visual Studio will generate a new website Default.aspx with the following content:

```xml

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
        </div>
    </form>
</body>
</html>
```
The page shows a simple HTML structure with a single element, which is a form representing an essential feature of ASP.NET web applications. The runat = "server" attribute indicates that the HTML element is processed on the server; otherwise ASP.NET would ignore it and see it as a simple HTML form.

There is <%= directive at the beginning of the document, which is a kind of header for the web server part. It determines how to handle the ASP.NET document. The C# programming language and the so-called CodeBehind are set on this page, which is similar to WinForms C# code, which serves the form.

8.3.3 Definition of client Web Form features

Now, it is necessary to define the client interface of the WebForm. The control elements must be inserted into the generated form, and they will allow the web application user to define the input parameters, which are transmitted to the application logic and, finally, the web application should present the output data that the user requires on this page.

For these purposes, Visual Studio includes a Design mode, which is a graphical editor which can be easily used to change the page content and to add new control elements.

You will find a button used to switch to design mode at the bottom of the page.

After switching to design mode, a Toolbox with the menu of the basic control elements that can be entered into the form is displayed.
We request the creation of a web form to search assortment according to two possible input values - a category of assortment name. That is why we drag Label, TextBox, DropDownList and Button control elements from the Toolbox into the form. We will create a search form using these control elements.

You already know from your work with .NET Framework that all its characteristics are based on object programming, and it is not different with the control elements. Each control element is
represented by the corresponding class with the defined attributes and methods. In design mode, you can easily set these attributes in the Properties window.

![Properties window image]

Figure 18 Setting the control elements attributes of a web form

The final web form definition can, for example, have the following form:

```csharp

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Northwind Search Web Application</title>
    <style>
        body {
            font-family: Verdana;
            font-size: 12px;
        }
    </style>
</head>
<body>
    <!-- Web form content -->
</body>
</html>
```
8.3.4 Definition of application logic

Like in desktop applications, we will separate the presentation layer from the logic one, i.e. calculations, work with databases and similar tasks will be clearly organized in separate classes.

First, it is necessary to create a class that would be able to load information about the existing categories from the database, and we would be able to use this list to fill the sCategories control element.

Let's create a separate class of CategoriesService to work with the categories. This class will include a public LoadCategories method loading a list of categories from the database. This class can be defined, e.g. as:

```csharp
using System.Collections.Generic;
using System.Linq;
using NorthwindsWeb.Models;

namespace NorthwindsWeb.Services
{
    public class CategoriesService
    {
        public List<Categories> LoadCategories()
```
Within this class definition, we used Entity Framework to load the data.

Now, let us create `initializeControls` private method in Default.aspx application logic page, which will fill the DropDownList control element items, and they will present the individual categories stored in the database. This method can have the following form:

```csharp
private void initializeControls()
{
    CategoriesService categoriesService = new CategoriesService();
    sCategories.DataSource = categoriesService.LoadCategories();
    sCategories.DataTextField = "CategoryName";
    sCategories.DataValueField = "CategoryID";
    sCategories.DataBind();
}
```

Take notice of the object access in the setting of the DropDownList `sCategories` control element. This object contains the following attributes:

- **DataSource** - specifies the collection of data to be displayed
- **DataTextField** - attribute specifies which input data attribute will be presented to the user
- **DataValueField** - attribute specifies the data value of an element on the list. Its application will be introduced later

Now, we have to call on the `initializeControls` method in a suitable place of the website lifecycle. This place is represented by a method representing an event control of page loading - `Page_Load`, which is called during every page calling, including PostBack operation.

```csharp
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        initializeControls();
    }
}
```
Take notice of the fact that calling the `initializeControls` method was conditioned by `Page.IsPostBack` condition, which is the main event of the dynamic page, when a request to reload the page is sent to the server, after changing the state of some of the control elements, during which the ViewState is changed.

This way, we have achieved the filling of `sCategories` control element.

Figure 19 Filling the control element with data

All the control elements to enter the input data are ready now. The next step is the execution of the actual assortment filtering according to the input parameters and displaying the result.

Let's create the `ProductsService` class, which will include the application logic for the assortment search. This can be defined, e.g., as:

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using NorthwindsWeb.Models;
namespace NorthwindsWeb.Services
{
    public class ProductsService
    {
        public List<Products> Search(int CategoryID, string ProductNameCondition)
        {
            using (Models.NORTHWNDEntities db = new NORTHWNDEntities())
            {
                var products = (String.IsNullOrEmpty(ProductNameCondition))?
                db.Products.Where(p => p.CategoryID == CategoryID)
                p.ProductName.Contains(ProductNameCondition));
                return products.ToList();
            }
        }
    }
}
The search method of ProductsService class is always called each time you press the Search button. It is therefore necessary to define the event service by pressing the button on the web form. The definition of this service is called out by double-clicking on the bSearch control element in the design mode of the web form. It can have the following form:

```csharp
protected void bSearch_Click(object sender, EventArgs e)
{
    ProductsService productsService = new ProductsService();
    List<Products> products =
    productsService.Search(Convert.ToInt32(sCategories.SelectedValue), tName.Text);
}
```

Take notice of the fact that we have transferred the values of SelectedValue attributes of DropDownList sCategories control element and Text of TextBox tName control element to the Search method as its input parameters. The Search method result is a list of products that must be displayed in a suitable way. There are several display options. We choose the simplest one, namely listing into the GridView control element used for data presentation purposes.

We add the following control element to the WebForm:

```xml
<tr>
    <td colspan="3">
        <asp:GridView ID="gResult" runat="server"></asp:GridView>
    </td>
</tr>
```

We fill a new control element using the assortment search result based on the input parameters and add it to the event service of bSearch button click.

```csharp
protected void bSearch_Click(object sender, EventArgs e)
{
    ProductsService productsService = new ProductsService();
    List<Products> products =
    productsService.Search(Convert.ToInt32(sCategories.SelectedValue), tName.Text);
    gResult.DataSource = products;
    gResult.DataBind();
}
```

Now, after entering the input parameters and clicking on the Search button, the list of the found assortments is displayed.
We usually require a definition of what data will be displayed to the client. This is achieved by broadening the definition of the columns of GridView control element:

```xml
<asp:GridView ID="gResult" runat="server" AutoGenerateColumns="False">
    <Columns>
        <asp:BoundField HeaderText="Název sortimentu" DataField="ProductName" />
        <asp:BoundField HeaderText="Množství v balení" DataField="QuantityPerUnit" />
        <asp:BoundField HeaderText="Cena za kus" DataField="UnitPrice" />
        <asp:BoundField HeaderText="Počet ks v balení" DataField="UnitsInStock" />
    </Columns>
</asp:GridView>
```

We've added a definition of the columns (Columns attribute), where we had defined the header text using `HeaderText` attribute and the input data item which will be displayed within the scope of this column - `DataField` attribute.

The final form of the user interface is shown in Fig. 29 and 30.
Figure 21 The final client display of assortment filtered according to category

<table>
<thead>
<tr>
<th>Název sortimentu</th>
<th>Množství v balení</th>
<th>Cena za kus</th>
<th>Počet ks v balení</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chai</td>
<td>10 boxes x 20 bags</td>
<td>16,0000</td>
<td>39</td>
</tr>
<tr>
<td>Chang</td>
<td>24 - 12 oz bottles</td>
<td>10,0000</td>
<td>17</td>
</tr>
<tr>
<td>Guarani Fantastica</td>
<td>12 - 335 ml cans</td>
<td>4,5000</td>
<td>20</td>
</tr>
<tr>
<td>Seafroth 4%</td>
<td>24 - 12 oz bottles</td>
<td>14,0000</td>
<td>111</td>
</tr>
<tr>
<td>Stevelye Stout</td>
<td>24 - 12 oz bottles</td>
<td>18,0000</td>
<td>20</td>
</tr>
<tr>
<td>Côte de Blaye</td>
<td>12 - 75 cl bottles</td>
<td>263,0000</td>
<td>17</td>
</tr>
<tr>
<td>Chartreuse verte</td>
<td>750 cc per bottle</td>
<td>18,0000</td>
<td>69</td>
</tr>
<tr>
<td>Iqosh Coffee</td>
<td>16 - 500 g tins</td>
<td>46,0000</td>
<td>17</td>
</tr>
<tr>
<td>Laughing Lumberjack Lager</td>
<td>24 - 12 oz bottles</td>
<td>14,0000</td>
<td>52</td>
</tr>
<tr>
<td>Outback Lager</td>
<td>24 - 335 ml bottles</td>
<td>15,0000</td>
<td>15</td>
</tr>
<tr>
<td>Rhônebra Klosterbier</td>
<td>24 - 0.5 l bottles</td>
<td>7,7500</td>
<td>125</td>
</tr>
<tr>
<td>Láskalkeř</td>
<td>500 ml</td>
<td>18,0000</td>
<td>57</td>
</tr>
</tbody>
</table>

Figure 22 The final client display of assortment filtered according to category and assortment name

We have achieved functional web applications meeting our requirements. It was a very simple web application. The field of ASP.NET technology is much wider, but its demandingness exceeds the objectives of this study area.

Summary of terms

ASP.NET, static web, dynamic web, WebForms, MVC, ViewState, controller, model, view, Form, HTML, CodeBehind, controlling element, PostBack

Questions

1. What is ASP.NET used for?
2. What are the basic principles of ASP.NET?
3. What are the differences between WebForms and MVC?
Reference sources