Noncommercial Joint-Stock Company

ALMATY UNIVERSITY OF POWER ENGENEERING AND TELECOMMUNICATIONS

Automation and Control Department

DATABASE DESIGN

Methodical guidelines to laboratory works for students of specialty 05070200 - Automation and Control

Almaty 2018

COMPILED BY: Ibrayeva L.K. Database Design. Methodical guidelines to laboratory works for students of specialty 05070200 - Automation and Control - Almaty: AUPET, 2018 - 65 p.

The guidelines describe the problems of database development: creation of DB conceptual model; creation of database objects and execution of complex retrieval requests to information searching in MS SQL Server; development of the user interface.

Ill - 6, bibliography – 6.

Reviewer: Dr.Tech.Sc, prof. Kaim T.T.

Printed according to the plan for publications of NC JSC "Almaty University of Power Engineering and Telecommunications" for 2018 y.

Introduction

One of the most difficult and acute problems associated with creation of an information system is a problem of database design. That is why it is very important to determine this concept, examine the database contents, as well as consider the way of data organizing which will be effective for all its future users and data management tools. The process of database design is a sequence of transitions from informal verbal descriptions of subject domain information structure to formalized description of domain objects in terms of some model. A data structure modeling, based on the meaning of these data that is the *method of semantic modeling* is used in the real design of the database structure. As a tool of semantic modeling, different "Entity-Relationship" diagrams are used. All versions of "Entity-Relationship" diagrams use graphic presentation of subject domain entities, their properties and relationships between entities. The result of database design is a *conceptual* schema (ER-diagram) of the modeled subject domain.

After the development of the database project, you can start implementing it in a specific *database management system* (DBMS). The modern data management technology is based on the use of *relational* databases management systems. All data manipulated languages created before the advent of relational databases have been focused on operations with the data presented as logical file records. It is required a detailed knowledge of the data storage organization and sufficient efforts to specify not only what data is needed, but also where they are placed, and how step by step to get them. A non-procedural SQL (*SQL - Structured Query Language*) language which is used in the guidelines operates with the data presented in the form of logically interrelated sets of tables. This language is focused more on the final result of the data processing than on the procedure of this processing. SQL itself determines where the data resides, define the most effective sequence of operations to be used to obtain them: it is not necessary to specify these details in the query to the database.

Design and creation of a database is not sufficient for successful work with the data: we should be able to access that data, to supplement and modify them. This is the aim of *a client application* which is a special program that allows you to work with data in a user-friendly form. With a single database many different applications can work. When considering applications working with one database, it is assumed that they can work in parallel and independently from each other, and the DBMS is designed to provide of multiple applications work with a single database so that each of them were performed correctly, taking into account all changes in the database made by other applications.

In methodical guidelines to laboratory works the question of database design for the selected subject domain and its implementation in the specific database management system is considered: creating database objects, data manipulation, constructing complex queries for information searching, and tools of professional users: procedures, triggers, development of user applications.

1 Laboratory work №1. Development of a database project. Creation of a database

The purpose of the work: to learn the principles of creating a database project; to create a database in MS SQL Server.

1.1 The task for the laboratory work

In the process of performing the laboratory work the student should:
- introduce the projected subject domain;
- determine the purpose of creating the designed database;
- develop a conceptual model of the database;
- study the logical components of the database;
- create a database in MS SQL Server; execute the commands of saving, deleting, backup and restore the database.

1.2 Basic concepts of conceptual modeling

The methods of data access have been developing over the past few decades from the huge physically oriented methods for the initial period of files processing to various kinds of databases processing. One of the important aspects of the "relational revolution" became the idea of separating the logical structure and data manipulation as they are understood by the final user, from the physical representation required by the computer equipment. This idea was widely accepted and was embodied by the proposal of a generalized database structure called *three-level architecture* of the database: three levels of abstraction, on which a database can be considered. Three-level architecture is a standard database structure consisting of conceptual, external and internal levels.

At the *conceptual* level the conceptual database design is fulfilled. The conceptual database design includes the analysis of information requirements of users and defining the required data elements. The result of the conceptual design is a conceptual schema; it is a single logical description of all data elements and relationships between them.

The *external level* consists of custom views on database data: how different groups of users understand the device of the database, working with it and updating the data.

The *internal level* defines the physical view of the database, by which the DBMS controls and updates the database (physical addresses, indexes, pointers, etc.). Unlike ordinary users, the designers of the physical database meet this level.

The phase of conceptual design involves creation of the conceptual database schema. The specifications are developed to the extent that is required for the transition to implementation. This step produces the detailed models of user views of the data; then they are integrated into a conceptual model fixing all the data elements that the database will contain.

The main elements of the conceptual data model are *entities* and *relationships*.

Entities are real-world objects about which we collect data. *Entity set* is a set of entities of the same type.

Relationship is the connection between the elements of two entity sets.

Composite entity set is the relationship considered as an entity set.

The *cardinality of relationship* refers to the maximum number of elements of one entity associated with one element of another entity.

In the laboratory work a *relational* DBMS is used in which entities and relationships are represented as *tables*.

Conceptual data models are much easier to understand than relational models because they better correspond to the natural view of entities.

1.3 Description of the subject domain

Development of database design starts with a system analysis of the *subject domain*. Subject domain is defined as the topics of interest to the users. It is necessary to carry out a detailed verbal description of the domain objects and the real relations that exist between the described objects.

The process of the subject domain description includes several steps:

- determining the purpose of your database;
- finding and organizing the required information;
- considering the types of reports which must be produced from the database;
- formulating the questions which the database should give answers to.

Suppose we want to create a database that contains information about the teaching/learning process of the current semester: the lists of students; the list of disciplines; the teaching staff of the departments providing training; information about lectures, practical classes and other activities in each group; the results of the exams for each of the classes. Let’s call the database "Education".

As a result of the subject domain analysis the following documents (these are sources of data to create the database) are identified:

- the form of the document with the list of students:

The list of students of group #___ Course # ___

Student ID	Name of student	Birthday	Address

Number of students____

- the form of the document with the list of teachers:

The list of teachers of the Department

Name of the Department_____ Head_____ Phone_____

Department ID	Name of teacher	Degree	Position

- the form of the document with the list of disciplines and types of classes in groups:

The plan of classes in the group #____

Name of discipline	Name of teacher	Type of classes	Hours

- the form of the examination sheet:

The examination sheet

The name of discipline_____Group_____ Teacher_____Date____

#	Name of student	Mark	Signature of teacher

If all this information is stored in one entity (in a single table of a relational database), there will be a lot of duplicate information, which increases the volume of the database and leads to many errors. Therefore we determine different entities.

1.3 Creating a conceptualmodel

The conceptual design is primarily concerned with the attempt to represent the semantics (meaning) of the subject domain in the database model. At the moment the standard in databases conceptual modeling is the Chen model "Entity-Relationship" (ER-model). Using the received information, we will develop a conceptual data model of the educational process.

Note, that in accordance with the previously accepted rules the name of the entity is a noun [1]. If the entity name includes several words, the first letter of each word in the entity name is upper case. The attribute names are in lower case, and if the attribute name includes several words, the first letter of first word is lower case; the first letters of others are upper case. Entity attributes are presented in ovals on the schema.

After analyzing the first document we can create the entity "Students" with attributes: studentID, studentName, studentsBirthday, studentAddress.This entity contains information about the students.

Since the groups have their own attributes, which, in general, is not dependent on the attributes of students (but associated with them!), we extract another entity "Groups" from this form with attributes: groupName, course, studentsNumber; it includes information about training groups.

Similarly, from the second form, we can get the entities:

- "Departments" (information about the departments) with attributes: departmentID, departmentName, departmentHeadName, departmentPhone;

- "Teachers" (information about teachers) with attributes: teacherName, teacherDegree, teacherPosition.

From the document "The plan of classes in the group" we can select the entities:

- "Disciplines" (information about disciplines) with attributes: disciplineName, totalHours, lecturesHours, practiceHours, laboratoryHours;
- "Type of classes" (information about types of classes) with single attribute typeOfClasses.

A lot of questions to the database can be formulated, for example: which group does a student belong to, what departments do the teachers work at, what disciplines, in what groups do they teach, what marks have the students got at certain types of classes in a particular discipline, etc.

To answer a wide range of possible questions to the database, you should consider the relationships between entities.

Between the "Group" and the "Students" there is a "one-to-many" relationship, as one group includes many students, and one student is a member of only one group (see Figure 1.1a). In the schemas below we will display the entities without attributes.

Similarly, the connection between the entities of the "Department" and "Teachers" are also "one-to-many" relationship: one department has many teachers, but each teacher works at a particular department (Figure 1.2b).

In different groups for each discipline many types of classes are provided by different teachers. It defines a "many-to-many" relationship between entities "Groups" and "Subjects", "Groups" and "Teachers", "Disciplines" and "Teachers" (Figures 1.1c,1.1 d, 1.1e) :

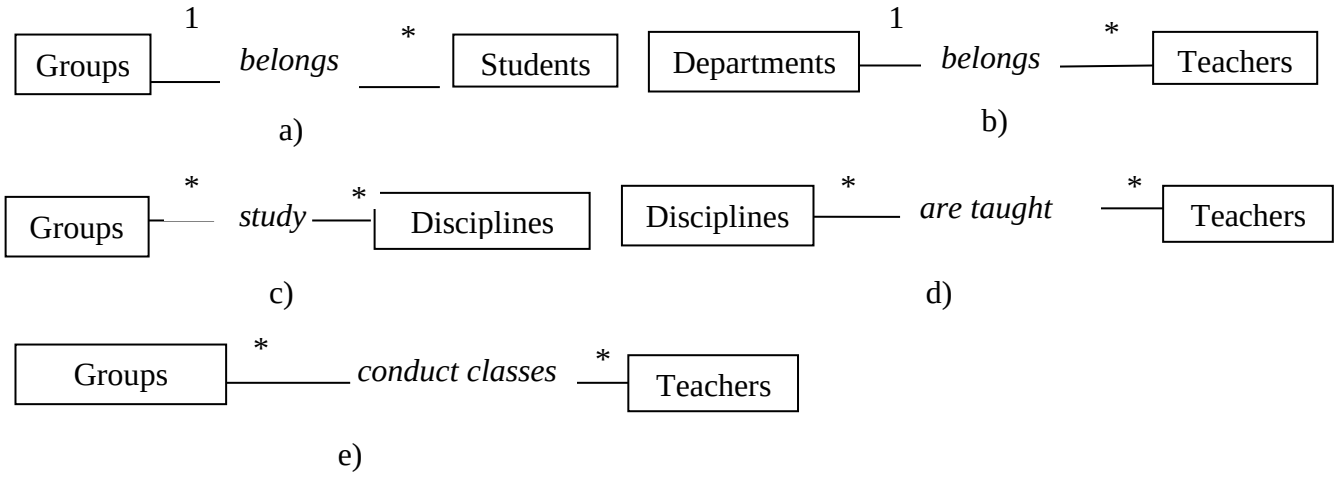

Figure 1.1 – The relationships between the entities

We refine the relationship in figure 1.1d (Figure 1.2).

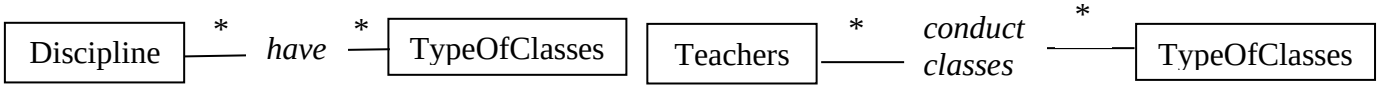

Figure 1.2 – The updated relationships

These relationships between groups, disciplines and teachers with their cardinality are represented in the following diagram (Figure 1.3). The attributes of the

entities on the diagram are not represented. These relationships represent a *composite entity*, call it "Learning". This entity has its own attribute - numberOfClasses (the value of this attribute equals to one of the following attribute values: lecturesHour, practicHours, laboratoryHours of the entity "Disciplines".

The database should provide storage of information about the final assessment of a student per semester for each of the classes displayed in the entity "Learning". The relationship between the "Students" and "Learning" is "many-to-many", as one student attends many types of classes displayed in the entity "Learning", and one lesson is purposed with many students.

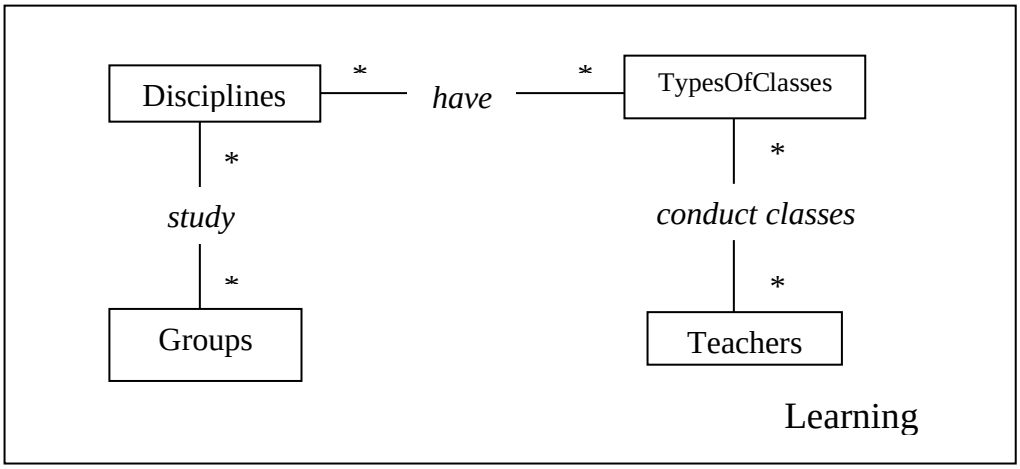

Figure 1.3 - The composite entity "Learning"

The students study the disciplines in the group, but grades are specific to each student. The relationship between students and studied disciplines is reflected in the "Progress" entity. This entity will contain data about the progress of a particular student in a specified class. Therefore, it is associated with the entity "Students" and with the entity "Learning". One student has data on several disciplines but this data always refers to one particular student. This means that the entity "Progress" has "one-to-many" relationship with the entity of "Students".

The entity "Progress" has also "one-to-many "relationship with the entity of "Learning", as there is much data about the progress of different students on one type of classes, but this data always refers to a particular student. This composite entity has its own attributes: dateOfExam, typeOfMonitoring and markOfDiscipline (mark).

The final logical database schema is presented in Figure1.4. The relationships "one-to-many" between entities "Group" and "Students", between "Departments" and "Teachers" are added in the schema. Attributes of the entities are not presented.

It should be noted that the proposed model is *not unique*. Other schemas can be proposed.

1.5 Creating the database in MS SQL Server

1.5.1 Logical components of the database.

MS SQL Server is the relational Database Management System (DBMS). A *relational database* is a database divided into logically complete segments called

tables and within the database these tables are connected through *key* fields. The tables are the main form of data storage in the database.

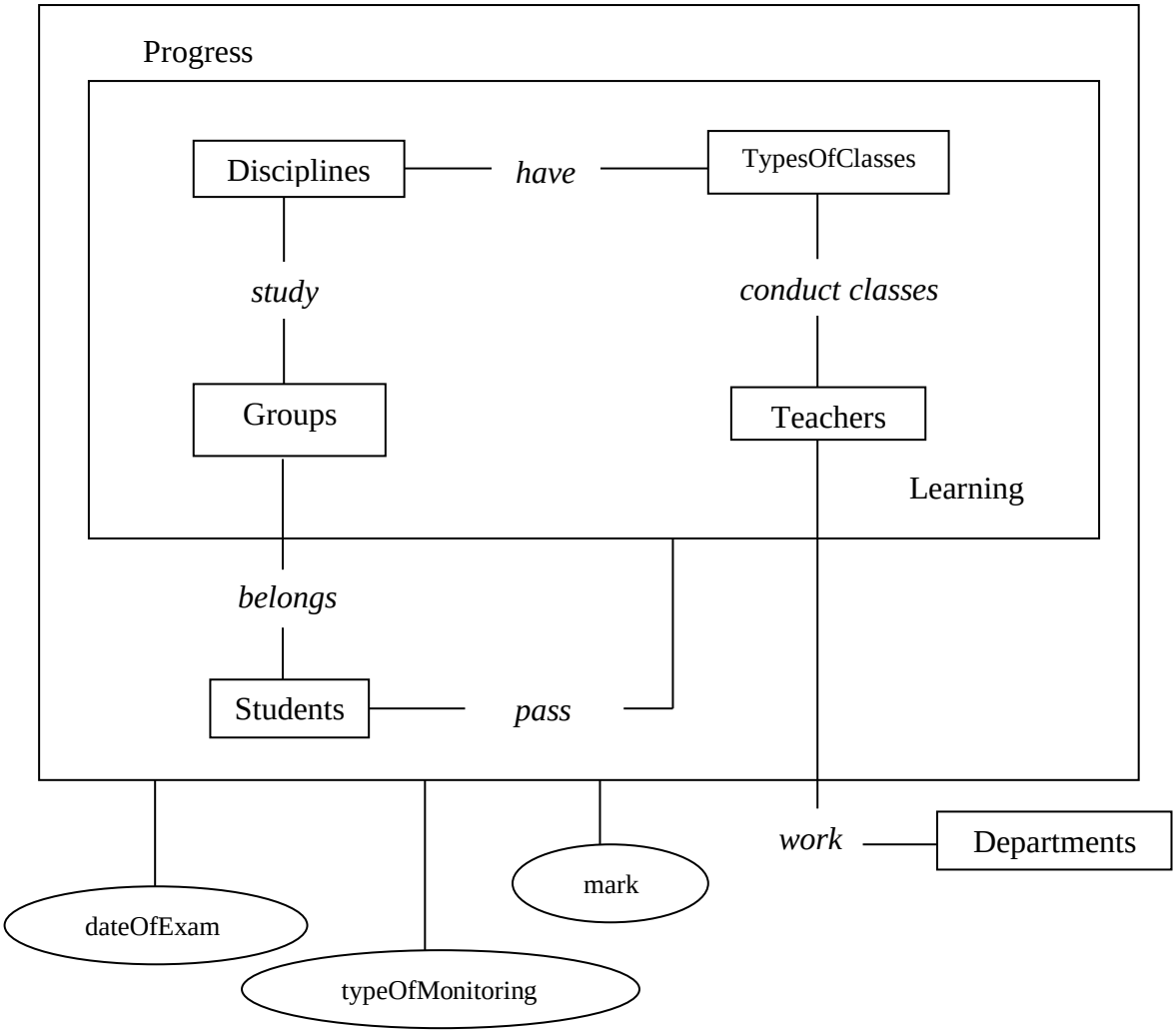

Figure 1.4 - The conceptual data model of the educational process

A relational database allows you to divide the data into logically smaller, more manageable segments that create the optimal representation of the data and the possibility of organizing multiple levels of data access. Due to the presence of the shared *keys* it is possible to combine data from multiple tables into one result set. This is one of the main advantages of the relational databases.

The set of actions executed in the database and considered as a whole, is called a *transaction*. A transaction is an entry of some changes into the database. Each database has a *transaction log*; it is a place where SQL Server writes all transactions before writing them to the database.

The data in SQL Server are organized into several different objects that the user sees when the database is created. These includes: Database Users, Database Roles, Tables, SQL Server Views, Stored Procedures, Rules, Defaults, User Defined Data types, Database Diagrams.

In addition to these visible objects in each database there are still some more: Constraints, Indexes, Keys, and Triggers.

1.5.2 Types of SQL commands.

The basic categories of commands in SQL which implement various functions: DDL - Data Definition Language; DML - Data Manipulation Language; DQL - Data Query Language; DCL - Data Control Language; the commands of data administration; the commands of transaction management.

Among such functions there are: creation of database objects, managing objects, updating database tables with new data, updating the data already existing in tables, executing queries, managing user access to the database and performing general database administration.

1.5.3 Create a database.

Physically the database is located in one or several operating system files.This file contains objects such as *tables* and *indexes*.

One file of operating system cannot contain multiple databases.

Let us name the designed database for the teaching/learning process "Education"

The transaction log is a working area that SQL Server uses to record information before and after the transaction. This information can be used to undo the transaction or to restore the database if the need arises.

The command of database creation *Create Database* has the following syntax:

CREATE DATABASE *name_of the_database*
ON [PRIMARY]
(NAME = *name_of the_database_data*, FILENAME='...\
name_of the_database _data.mdf ', size =*size*, maxsize = *maximum size*,
filegrowth = *increment*)
LOG ON
(NAME = *name_of the_database _log*, FILENAME=*'... \name_of*
the_database _log.ldf ', size = *size*, maxsize = *maximum size*,
filegrowth =*increment*)

Notes:

a) the location of the database and transaction log - 'C:\...' - can vary depending on the version of SQL and the location of *Program Files*;

b) hereinafter, when describing the general form of the command, accommodation of the options in brackets means that this parameter is not always mandatory. For example, in this case the PRIMARY parameter identifies the file containing the logical start of the database and system tables. In the database there can be only one initial (PRIMARY) file. If this parameter is omitted, the first file in the list shall be the primary file. By default the files with type *primary* are assigned by extension *.mdf*;

c) the option separated by a vertical bar means a choice of two alternatives;

d) dots mean the path.

1.5.4 Delete a database.

Deletion of a database leads to the cleaning of all occupied spaces in all files where the database was, and also deletes all contained within it objects:

a) deletion of a database in GUI (Graphical User Interface) mode involves the following steps:

- click the name of the database you wish to delete;
- select in the context menu *Delete* command; in the appearing message box, confirm the deletion of the database;.

b) to delete a database using Transact-SQL, simply run the command:

DROP DATABASE *database_name*

1.5.5 Create a database backup.

To save a copy of the database, transfer it to removable media, you must first create a backup of the database through a special functionality of MS SQL Server:

- select your database from the list of *Databases* and in the context menu *Tasks-Back Up*;
- in the window that appears, create a backup first, clean the list of databases, which are ready to be copied;
- the "Browse" button opens the following window to choose the disk space, type the name of the database backup, confirm the copy operation.

The system should give a message about the successful copying.

It should be noted that the backup is storing the *data* in the database. If there is a need to move the *entire* database completely, then you can do this by copying *.mdf* and *.ldf* files are located at *C:\Program Files\...\Ms SQL\Data\filename*.

1.5.6 Restore the database.

If there is need to transfer the backup file to a computer, it is also performed by a special functionality of MS SQL Server:

- select the database for the restore operation;
- in context menu select *Tasks-Restore*;
- specify the source and location of the backup in the appeared page (here are the general recovery options);
- select the "Options" page, tick the box "Overwrite the existing database" and switch tick in "Restoring state" section;
- confirm the selected operations.

The system will report about a successful recovery. If you receive a message stating that the database restore is impossible and the system will offer to change the script, run it (in a separate window with a script).

1.6 The order of execution of the laboratory work

1.6.1 Select the modeled subject domain (by option) .Fulfill system analysis of this domain: verbal description, the purpose of the designed DB, a possible list of questions to DB.

1.6.2 Define the entities and their attributes.

1.6.3 Define the relationships between entities; define the cardinalities of the relationships.

1.6.4 Develop an ER-diagram for this domain.

1.6.5 Run *MS SQL Server/SQL Management Studio/Server Name* (from the list) – *Browse* (choose the server name) - *Connect* (in accordance with the configurations in the computer class); in the left side of the screen a window of "Objects viewer" appears;

1.6.6 Firstly you will realize the DB "Education".

To create any SQL Server object, there are several ways, based on the fulfilling of a particular command:

a) using interactive development environment (IDE):

- in the list "Objects viewer", using the context menu select *Databases-Create database*;
- you must enter the name of the DB in the appeared window;
- remember the *path* where the DB is saved;
- thenconfirm saving the database;
- nowdelete the created DB using the context menu;

b) create a database “Education” by using Transact-SQL:

- click *Create query*, in the appeared command editor window input the command of DB creation (*Appendix A*);
- click *Run* button;
- if the system gives the errors (messages below), correct them.
- perform the operations of backup and restore of the database;
- copy entire database completely (*.mdf* and *.ldf* files) and locate (paste) in another computer.

1.7 The report on the laboratory work

The report contains:

- a designed ER-diagram (for your subject domain) with explanations;
- commands of creating the database "Education" with explanations and, if necessary, with the relevant screenshots.

1.8 Control questions

1.8.1 Definition of the database.

1.8.2 What is a conceptual data model?

1.8.3 Give definition of entity; what is an attribute of entity?

1.8.4 Give definition of relationship; the types of relationships?

1.8.5 What is a cardinality of relationship?

1.8.7 What is a composite key?

1.8.8 What is a composite entity?

1.8.9 Give the definition of "ER- diagram".

1.8.10 What does the acronym SQL mean?

1.8.11 Explain the file extensions *.mdf* and *.ldf*.
1.8.12 Explain the option *filegrowth* in the command of database file creation.
1.8.12 Where is the DB saved?
1.8.13 What is ON PRIMARY option when you create a DB?
1.8.14 What is LOG ON PRIMAR Yoption when you create a DB?
1.8.15 How can you give a name to DB?

2 Laboratory work №2. Converting of a conceptual model into a relational one. Creation of database tables. Manipulation of data

The purpose of the work: to learn the methods of converting of conceptual models into relational; creation of tables, filling in data.

2.1 The task for the laboratory work

In the process of executing the laboratory work the student should:
- convert the conceptual database model into the relational model;
- determine the structure of database tables "Education";
- create tables in MS SQL Server;
- fill in the database tables with data;
- execute various commands of the data manipulation.

2.2 The attributes of the database "Education" entities

We will consider the conceptual model of the database "Education". In the laboratory work #1 we determined the entities and attributes of this model:

Groups (groupName, course, studentsNumber);

Students (studentID, studentName, studentsBirthday, studentAddress);

Departments (departmentID, departmentName, departmentHeadName, departmentPhone);

Teachers (teacherName, teacherDegree, teacherPosition);

Disciplines (disciplineName, totalHours, lecturesHours, practiceHours, laboratoryHours);

TypeOfClasses (typeOfClasses).

Learning (numberOfClasses);

Progress (dateOfExam, typeOfMonitoring, mark).

2.3 The converting of the conceptual model into the relational one

The conceptual data model consists of entities, attributes, relationships, composite entities. Each of these constructions must be converted into the relational tables. Upon conversion the received tables will have the *fourth normal form*. That is the further normalization of the model will not be required.

2.3.1 Conversion of the entities and attributes.

The entity with the attributes can be converted into the relational table with the name of the entity as the table name and attributes of the entity as the attributes of the table. If some set of these attributes may be used as the key of the table, it is selected as a key of the table. Otherwise, the attribute whose values will uniquely determine the elements of the initial entity, and which thus can serve as a key of the table are added to the table.

The conversion of the entities starts from the entities which are in the side "one" of the relationships. There are entities "Groups", "Disciplines" and "Teachers" entities. Let's convert them into relational tables with the appropriate names. The attributes "name" (of group, of discipline, of teacher) can be chosen as the *potential keys* of these tables, provided that the names will not be repeated. But we have to uniquely identify each element of the table; therefore we enter the "ID" attribute in these tables and will use it as the *primary key*. This key will be easier to use in daily work in data entering.

The entities "Students" and "Departments" already have the attributes "ID" which can be chosen as the *primarykeys*.

Thus we have the relational tables (key fields are in italics):

Groups (*groupID*, groupName, course, studentsNumber;

Students (*studentID*, studentName, studentsBirthday, studentAddress);

Departments (*departmentID*, departmentName, departmentHeadName, departmentPhone);

Teachers (*teacherID*, teacherName, teacherDegree, teacherPosition;

Disciplines (*disciplineID*, disciplineName, totalHours, lecturesHours, practiceHours, laboratoryHours).

Entity TypeOfClassesis transformed into the table with a single attribute: TypeOfClasses (typeOfClasses).

2.3.2 Conversion of the relationships.

The relationship "one-to-many" is converted by the following way: the primary key of the entity from the side of "one" cardinality is added as foreign key to the entity from the side of "many" cardinality.

The relationship between the entities "Groups" and "Students" is characterized by "one-to-many" relationship, as one group includes many students, and one student is the member of only one group. Then according to the rules the relationship between them is carried out by group number. This attribute will be the *foreign key* in the table "Students".

In addition, to the table "Students", we add the attribute "studentLeader"; this field will indicate the ID of the student who is the head of the group, that is, the values of key of the same table. This attribute connects the entity "Students" itself; it is a *recursive* relationship. The attribute "studentLeader" is a *recursive foreign key*.

Thus the structure of "Students" table will look as follows:

Students (*studentID*, studentName, studentsBirthday, studentAddress,groupID, studentLeader).

Note that in this table you could choose a *composite key* of *studentID* + *groupID*. Such an identifier will vividly allow to identity of the student as being part of the group. But in this case to determine the leaders of the groups would require a separate table. We decided to use a *simple* key.

Entities of "Departments" and "Teachers" are also in "one-to-many" relationship. The connection between them is carried out by a unique key "departmentID" of the main entity "Departments" that is included in the subordinate entity:

Teachers (*teacherID*, teacherName, teacherDegree, teacherPosition, departmentID).

Transformation of the relationships "many-to-many": a separate table is created and primary keys of all other entities are included as a foreign key; they form *the composite primary key*. This table can have additional non-key attributes inherent only to it.

The entity of "Learning" determines the "many-to-many" relationship. Let's determine the attributes of table "Learning":

Learning (g*roupID, disciplineID, teacherID, typeOfClasses*, totalHours).

Note. In the future when realizing the project we will not create a table "TypesOf Classes" as the only attribute of this table is part of a key attribute of table "Learning".

The entity "Learning" is an entity-binder in the considered "many-to-many" relationship of the entities.

Series of classes is carried out in each group in accordance with the studied disciplines. On the other hand each type of the class is defined for a particular group. That is why there is the relationship of the "one-to-many" type between entities "Groups" and "Learning".

For each discipline, there is large number of occupations in different groups with different teachers. However, each session is held on a specific discipline that determines the relationship of "one-to-many" type between entities "Disciplines" and "Learning". Similarly the relationship of "one-to-many" between entities "Teachers" and "Learning" is determined.

The "Progress" entity provides keeping of information about the final grades of the student in the semester for each type of classes displayed in the entity "Learning". Such assessment is defined, on the one hand, by the "studentID", on the other hand, by the identifier of the class (groupID + teacherID + disciplineID + typeOfClasses). Their union forms the unique identifier of the entity "Progress". This entity contains information about the progress of the particular student on the specific type of class. By different types of classes at different times different types of control (examination, test, etc.) can be conducted. The attributes "typeOfMonitoring","dateOfExam" and "mark" are the non-key attributes of this table:

Progress (*studentID, groupID, disciplineID, teacherID, typeOfClasses*, typeOfMonitoring, dateOfExam, mark).

So, finally the structure of the relational tables of the designed database has the following form (the primary keys are in italics, specified foreign key definitions):

Groups (*groupID*, groupName, course, studentsNumber);

Departments (*departmentID*, departmentName, departmentHeadName, departmentPhone);

Disciplines (*disciplineID*, disciplineName, totalHours, lecturesHours, practiceHours, laboratoryHours);

Students (*studentID*, studentName, studentsBirthday, studentAddress, groupID, groupLeader);

"groupID" is a foreign key which is the primary key of "Group" table; "groupLeader"is a foreign key which is the primary key of "Student" table;

Teachers (*teacherID*, teacherName, teacherDegree, teacherPosition, departmentID);

"departmentID" is a foreign key which is a primary key of "Department" table;

Learning (g*roupID, disciplineID, teacherID, typeOfClasses*, totalHours);

"groupID" is a foreign key which is a primary key of "Group" table; "disciplineID" is a foreign key which is a primary key of "Disciplines" table, "teacherID" is a foreign key which is a primary key of "Teacher" table.

Progress (*studentID, groupID, disciplineID, teacherID, typeOfClasses*, typeOfMonitoring, dateOfExam, mark);

"studentID" is a foreign key which is a primary key of "Student" table; "groupID, disciplineID, teacherID, typeOfClasses" is a foreign key which is a primary key of "Learning" table.

2.4 Main types of data

Types of data allow you to keep different data in the database. The most common types in SQL, as in most other languages, are: character string, number string, the values of date and time.

Character data consists of a sequence of characters: alphabetic, numeric and specific (for example, ? or >) symbols. When you load in a storage area (such as a column of a table) character data are entered in single or double quotes.

Char(n). When storing data of this type one byte is used for each character. The number *n* determines the maximum number of characters of the column. If you enter a number of characters less than *n*, SQL Server will add spaces after the last character so that the total length will be equal to n.

When values which are stored in the column vary in length, you can use *varchar(n)*. Unlike the previous data type, the size of the storage area for data of this type changes according to the actual number of characters stored in each column of the table, spaces to the end of the entered values are not added.

Type *text*. This type is used for storing large amounts of textual information. To insert data into a column of this data type, they must be enclosed in single quotation marks.

Numeric data types. Standard SQL types are:

- *integer, smallint* to store integers;

- *real* for storing positive or negative fractions with precision up to seven digits;

- *float(n)* stores positive or negative fractions with an accuracy of fifteen digits.

The *datetime* and *smalldatetime* data types. They are used to store date and time. Much easier to store the date and time in the format of one of the intended types of data, not a string of characters. In this case, the date and time displayed in the usual format.

These data types allow you to store up to 90% of the information. In addition to these types, Transact SQL provides a set of specific data types. You can define your own data type - the *user* one, which then will be used for saving structure.

When you assign the data type column of the table it is possible to use constraints *Null/Not Null*, which allow you to specify which of the columns should have values in all rows of the table.

2.5 Determination of the database tables' structure

The data definition language (DDL) is part of SQL that allows a user to create various database objects and redefine their structure, for example, create or delete tables.

Consider the following DDL commands: CREATE TABLE, ALTER TABLE, and DROP TABLE.

To create the table CREATE TABLE operator is used.

The syntax for creating tables:

```
CREATE TABLE name of the table
(Field1 Type Of Data     [Not Null],
 Field2 Type Of Data     [Not Null],
 …)
```

When a table is created the key to it is assigned using the PRIMARY KEY option to one or more fields and the key is constraint:

```
CREATE TABLE name of the table PRIMARY KEY
(Field1     Type Of Data     [Not Null], …)
```

You can create a key directly as a constraint by a comma after the determination of all the columns in the table, and if the key is a composite, all of its components are listed:

```
CREATE TABLE name of the table
(Field1     Type Of Data     [Not Null],
Filed2     Type Of Data     [Not Null], …
PRIMARY KEY (Field1, Field2)  )
```

A foreign key is created using the FOREIGN KEY option. A foreign key is created after defining all the table columns as follows:

```
CREATE TABLE name of the table_1
```

(Field1_1 Type Of Data [Not Null],
Field1_2 Type Of Data [Not Null],
…,
Field2_1 Type Of Data [Not Null],
CONSTRAINT Field2_1_FK FOREIGN KEY (Field2_1) REFERENCES
Name of the table_2 (Field2_1))

Here Column *Field2_1* is assigned the foreign key of the table *name of the table 1*. This foreign key refers to the column *Field2_1* of the table *name of the table 2*.

2.6 The modification of the table

The table can be modified using the ALTER TABLE command. Using this command you can add and remove columns, change column definitions, add and delete constraints.

The standard syntax of ALTER TABLE command is as follows:

ALTER TABLE *name of the table* [Modify] [Column *name of the column*]
[Type Of Data Null|Not Null] [Restrict|Cascade]
DROP] [CONSTRAINT] *name of the constraint*]
[ADD] [COLUMN] *determination of the column*

1) *Modification of table elements*. Column attribute specifies the rules for the submission of data in the column. Using the ALTER TABLE command, you can change column attributes. The attributes here referto the following: the data type of the column; length, precision, or scale of data in the column; the enable or disable to have the NULL value in the column.

When you add a column to an existing table with existing data to the new column the NOT NULL attribute cannot be assigned. NOT NULL means that the column must contain values for each row in the table, so if the added column will have the NOT NULL attribute, you will immediately get a contradiction with this constraint, as available in the table columns do not have values for the new column. And still have the option to add a column that requires data entry as follows:

- add a column by giving it the attribute is NULL (this means that the column does not have to be data);
- enter the data in each row of the new table column;
- making sure that the column contains a value in each of the rows of a table, you can modify the column attribute to NOT NULL.

2) *Changing the columns.* When you change the table columns you need to consider a number of points. The general rules are as follows:

- the column width may be increased to the maximum length allowed for the corresponding data type;
- the column width can be reduced only to the longest available in the column values;
- for columns with numeric data width can always be increased;

- for columns with numeric data width can be reduced only when the number of new characters will be sufficient to accommodate any of the available values in the column;

- for numeric data, you can increase or decrease the number of decimal places;

- the data type of the column can usually be modified.

In some implementations the use of certain options of the ALTER TABLE statement can be denied. For example, you can be not allowed to remove the columns from the tables. Instead, you will want to delete the table and create a new one with the correct number of columns. Problems may arise with the removal of the columns in the table, depending on the column of another table, or to remove a column referenced by other table. On this occasion, carefully review the documentation of the proposed implementation of the SQL, you are working with.

3) *Add the constraints*. This situation may arise, for example, in the case when key fields were not specified in the process of table creation:

ALTER TABLE *name of the table*
ADD CONSTRAINT PK_*name of the table* PRIMARY KEY (*name of the field1, name of the field2*)

Foreign keys you can assign a table are as follows:

ALTER TABLE *name of the table*
ADD CONSTRAINT FK_*name of the table* FOREIGN REY (*name of the field*)
REFERENCES *name of the table (name of the field)*

4) *Drop tables* are probably the most straightforward. The syntax used to delete the tables, following:

DROP TABLE *name of the table* [RESTRICT|CASCADE]

If the option RESTRICT is used either the table is referenced in a view or constraint, DROP operator will return an error. When you use the CASCADE option not only the table itself will delete, but all referencing tables of representations and constraints as well.

2.7 Data Manipulation commands

The data manipulation language (DML) is part of SQL, providing database user with the opportunity to make real changes in the data of a relational database. With DML the user can fill up the tables with new data, update existing data and delete them from the tables.

In SQL there are three basic DML commands: INSERT, UPDATE, DELETE.

2.6.1 Filling the tables with data.

Filling the table with the data is the process of entering the new data into the table manually with the help of separate commands, or automatically by software, or by any other means. What data and in what quantity it will be possible to enter in the

table depends on many factors, the main ones are the constraints that were specified when the table definition, the physical size of tables, data types of the columns, column width, demands integrity in the form of keys and foreign keys.

The following situations are possible:

a) enter new data into the table.

To enter new data into a table, use the INSERT statement. It has the form:

INSERT INTO *name of the table*
VALUES ('value1', 'value2', [NULL])

According to the presented here syntax of the INSERT statement in the VALUES list you must put values for all columns of the corresponding table. The values in the list are separated by commas. Character values and date values must be enclosed in quotation marks. For numeric values and empty values specified by the keyword NULL, the quotation marks are not needed. You must specify values for all columns in the table;

b) enter data to certain columns of a table.

You can enter data at not all, but only in specific columns. In this case with the INSERT statement and a list of values together you need to specify the list of their corresponding columns:

INSERT INTO *name of the table*
('Column1', 'Column2') VALUES ('value1', 'value2');

The order in the list of values must match the order of entering values into the table specified in the column list. The column list in the INSERT statement does not have to match the column list in the definition of the corresponding table, but the inputs must satisfy the list of selected columns;

c) enter NULL values.

To enter a NULL value into the table is simple. This is necessary, in particular, when the value of the corresponding column is not known. The syntax to enter NULL values is the following:

INSERT INTO name of the scheme.name of the table
VALUES ('value1', NULL, 'value3')

Modification commands do not produce output. However, *Query Analyzer* will report that one entry was added. The table must already exist at the time of execution of the command, and type of each value in parentheses after *VALUES* must match the data type of the column into which it is inserted. The first value falls in column 1, the second - in column 2, etc.

2.6.2 Update the existing data.

Existing data in the table can be changed using the UPDATE command. The UPDATE command does not add new records to the table and does not remove them, but allows you to change the data. With the help of one of such commands, you can change the data only of one table, but at the same time, you can change data in several columns. One such operator can change one row of data and a set of rows:

a) updating values in one column. In the simplest form the UPDATE statement modifies one table column. When you change one column, you can edit only one record or several records.

The syntax to change the data in one column is the following:

UPDATE *name of the table*
SET NAME OF THE COLUMN = 'value'
[WHERE CONDITION]

b) update multiple columns at once:

UPDATE *table_name*
SET COLUMN1 = 'value'
[, COLUMN2 = 'value']
[, COLUMN3 = 'value']
[WHERE *condition*]

Please note to use the SET keyword: it is one, but there are a fewdescriptions of columns. Descriptions of the columns are separated by commas.

2.6.3 Deleting data from tables.

To delete data from tables DELETE command is used. The DELETE command is not designed to remove values of individual columns, and to delete entire records. To delete one or more records from a table, use the following syntax of the DELETE operator.

DELETE [FROM] *name of the table*
[WHERE *condition*];

The FROM keyword is optional in the instructions. Before performing the operation, make sure of the correct spelling of the user, because you can inadvertently delete all rows in the table. When you delete rows from a table WHERE keyword is a very important part of the DELETE operator. If the WHERE keyword in DELETE statement is omitted, the command will delete all table rows.

So always use a WHERE keyword in DELETE statement.

To remove the content of a table you can enter the command:

DELETE FROM name table

It is not recommended to run this command frequently! Typically, you will need to delete some specific rows in the table. To determine which rows will be deleted, use the select condition. Unlike the file-based DBMS, SQL Server does not mark the records as deleted and removes them physically, that is, they are not subject to restoration. Be careful with the DELETE command!

2.8 The order of execution of the laboratory work

2.8.1 Run MS SQL Server. Determine the current database with the help of the following command: USE Education or choose the DB in the list.

Now all subsequent commands will be performed in this database.

2.8.2 To create a table, there are two ways:

a) using IDE:

- from the list "Databases" select your DB and in context menu *Create table* (for example, table "Groups");

- in the appeared window you must enter by turns: the columns names (of the table "Groups"); choose the corresponding data type; select to allow or not the NULL values;

- assign the *key* field (use the proper button);

- save the table, assign the name "Groups" to the table;

- in the list of DB objects, in the section "Tables" the name of this table appears; the specification *dbo.groups* (in the left side) shows that it is a DB object;

- open the table; enter several records (according to the chosen data types);

- delete the created table;

b) create the tables "Groups" of the database "Education" by using Transact-SQL (*Appendix B.1*).

2.8.3 Remove the table "Groups"; create this table without determining the primary key.

2.8.4 Make the necessary changes into the structure of the table "Groups" by ALTER TABLE command: set the primary key.

2.8.5 Create all tables of the database "Education" by using Transact-SQL (*Appendix B.1*).

2.8.6 Add into "Students" table the field that will keep the information about students' scholarships. Remove the entered field from the table.

2.8.7 Fill in all the tables of your database by the data (*Appendix B.2* as help). Stick to the following rules:

a) first fill in the tables with the smallest number of connections;

b) *rule of* entity *integrity*: none key attribute of the row can be empty;

c) *rule of reference integrity*: the value of each foreign key must be either empty or equal to one of the current values of the key of another table.

2.8.9 Add the field where GPA will be calculated into the table with information about students' progress. Look through the means of this field.

2.8.10 Update some records in tables.

2.9 The report on the laboratory work

The report on the work must contain:

- the tables structure;
- the examples of changes in the structure of the tables;
- the samples of filling in the tables with data;
- the examples of changes of data in the tables.

2.10 Control questions

2.10.1 Explain the meaning of the terms: primary key; external key; composite key; relational table.

2.10.2 What data manipulation commands do you know?

2.10.3 What is the definition of the field NOT NULL or NULL?

2.10.4 What is specified in the VALUES list of an INSERT command?

2.10.5 Is it really possible to enter data only in certain columns of the table?

2.10.6 Can UPDATE command change the data in multiple tables?

2.10.7 Can UPDATE command change the data in multiple columns in one table?

2.10.8 Is it always necessary to use the condition in the operator of deleting the table records?

2.10.9 What does the definition of the "identity" field mean?

2.10.10What is the difference between the structure of the table and the data of the table?

3 Laboratory work №3. Data retrieval (Data Query Language)

The purpose of the work: development of the query language to the database and ways of forming simple and complex queries.

3.1 The task for the laboratory work

In the process of executing the laboratory work the student must:

- learn the purpose and syntax of the SELECT statement;
- study data selection criteria and operations in the conditions of the selection;
- perform operations for retrieving data from a database: selecting data from one table; selecting data from multiple tables.

3.2 The SELECT statement

This section of the language is represented by only one statement, but for users of a relational database Data Query Language (DQL) is the most important part of SQL. This command is the SELECT command. The command has many options and optional parameters; itis used to build queries to relational databases. With its help, you can build queries of any complexity: from very general to specific and from simple to incredibly complex.

A query is a request to retrieve information from the database. Queries are used to extract data in a form that is convenient to the user.

The SELECT statement is used to create queries to the database. The SELECT statement is not used by itself, and requires that you specify some parameters with the help of key words. In addition to the mandatory, this statement has several optional keywords, expanding its capabilities.

In the SELECT statement the keyword SELECT is used in conjunction with the FROM keyword in order to organize the database to retrieve the data in a readable format. Part of the query specified by the SELECT keyword, determines the source of data selection. The syntax of a simple SELECT statement is as follows:

SELECT * | ALL | DISTINCT [*column1*, *column2*]
FROM *table1* [, *table2*]

The column names that follow the select keyword determine which columns will be returned in the results.

The table's names which follow the keyword FROM specify the tables that will be queried to retrieve the desired results.

You can select as many column names that you'd like, or you can use "*" to select all columns. The ALL option is used when you need to show all column values including duplicates. The option DISTINCT is used to eliminate repetition. The default option is ALL, which is therefore omitted.

Please note, that the column names in the list following the SELECT keyword, are separated by commas, in the same way as the tables names following the FROM keyword.

3.3 Operations in queries

We can retrieve selected data that match the criteria that you specify. Here is the format of such select statement:

SELECT [ALL | * | DISTINCT*column1*, *column2*]
FROM *table1*[,*table2*]
WHERE [*condition1*|*expression1*]
[AND *condition2* | *expression2*]

Conditional selections are used in the WHERE clause:= (Equal), <>(Not equal to), <(Less than),> (Greater than), <= (Less than or equal), > = (Greater than or equal).

Logical operations in SQL are specified by keywords and not symbols. The following logical operations are used: AND, OR, and NOT operators.

The AND and OR operators are used to filter records based on more than one condition:

- the AND operator displays a record if all the conditions separated by AND is TRUE;
- the OR operator displays a record if any of the conditions separated by OR is TRUE;
- the NOT operator displays a record if the condition(s) is NOT TRUE.

AND Syntax:

SELECT *column1*, *column2*, ...
FROM *table_name*
WHERE *condition1* AND *condition2* AND *condition3* ...;

OR Syntax:

SELECT *column1, column2, ...*
FROM *table_name*
WHERE *condition1* OR *condition2* OR *condition3 ...*;

NOT Syntax:

SELECT *column1, column2, ...*
FROM *table_name*
WHERE NOT *condition*;

You can also combine the AND, OR and NOT operators.

Arithmetic operations are used in SQL in the same way as in most other languages. There are four operations: + (addition), * (multiplication), - (subtraction), / (division). Arithmetic operations can be combined.

The user can control the operations order in an expression only by using parentheses. A parenthesized expression means you need to consider the expression as a single unit, the order of operations (the priority of operations) specifies the order in which expressions are processed in mathematical expressions, and inline SQL functions.

3.4 Selection data from multiple tables

One of the most useful features of SQL is the ability to select data from multiple tables. To join the tables the key fields are used.

Table names for joining are specified in the FROM keyword list. The relationship is defined in terms of key words WHERE.

Natural joining. The tables are joined by common column which in each table is usually the key (INNER JOIN).

Syntax of the operator is as follows:

SELECT *table1.column1, table2.column2…*
FROM *table1, table 2 [, table3]*
WHERE *table1.name of the column = table2.name of the column*
[AND *table1.name of the column = table3.name of the column*]

Please note that the list of the SELECT statement along with the names of each column lists the names of the relevant tables. It is called a *detailed definition* of the column in the query. Detailed definitions are required only for those columns that are presented in several tables of the query. But the statement typically specifies a detailed definition for all the columns.

Recursive connection (using of aliases). Note that, in order to reduce the volume of printing tables may be assigned *aliases* - temporary renaming of tables. Very often aliases are used in the *recursive* connections.

Recursive connection (SELF JOIN) suggests joining the table with itself, using a temporary table has been renamed in the SQL statement.

The statement syntax:

SELECT *A.name of the column, B.name of the column, [,C.name of the column]*

FROM *table1 A, table 2 B [, table3 C*]

WHERE *A.name of the column = B.name of the column*

[AND *A.name of the column = C.name of the column*]

Here A, B, C are the aliases of the table.

Connection on several keys. Depending on the structure of the database the tables can have simple or *composite* keys, i.e. consisting of multiple columns. In this case the WHERE clause should specify a comparison of each key component.

For examples see *Appendix C*.

3.5 The order of execution of the laboratory work

3.5.1 Run MS SQL Server.

Determine the current database with the following command:

USE education

Now all subsequent commands will be carried out exactly in this database. Use examples of *Appendix C* as help.

3.5.2 Search information in separate tables; obtain the following information

- the list of teachers indicating their position;
- the names of the departments with the names of the heads;
- the list of students of the first group with different names (assuming that there are namesakes in this group);
- the list of students whose scholarship is more than 2000;
- the list of students living in Astana and Karaganda;
- the list of students of the second group who do not have scholarships.
- the list of students in the third group, whose names start with the letter "A";
- the list of students who were born in 1998;
- the list of students who do not reside in Almaty;

3.5.3 Search for information in several database tables:

- the list of students who got an excellent mark in a certain discipline;
- the list of marks of a specific student with the names of disciplines;
- the list of all students and their marks;
- the list of lectures/classes with the name of teachers who provide this classes;
- the list of students who failed the exam in a specific discipline;
- the list of students who failed the exam at least in one discipline;
- the list of students who have "A" and "B+" grades in the "Math" discipline;
- the list of teachers who work at the "History" department and teaches the groups of "Automation and Control" specialty;
- the list of disciplines which are learned by the groups of "Automation and Control" specialty;
- the list of teachers and the names of their departments;

- the list of progress on "Physic" of the students of "Automation and Control" specialty.

3.6 The report on the laboratory work

The report on the laboratory work contains listings of commands and the results of execution of queries.

3.7 Control questions

3.7.1 Which section of SQL is the SELECT statement?

3.7.2 What are the obligatory components of the SELECT statement?

3.7.3 Do all the data in the expression keyword WHERE require using quotation marks?

3.7.4 Is it possible to specify several conditions for keyword WHERE in the expression?

3.7.5 Are quotation marks permissible for numeric fields?

3.7.5 Are the quotes valid for numeric field values?

3.7.6 Can you use SELECT statement without the FROM keyword?

3.7.7 What is an alias of a table?

3.7.8 When you link tables, should they be linked in the same order in which they appear in the expression keyword FROM?

3.7.9 Can you link not one but more than one table columns in the query?

3.7.10Whichpart of the SQL statement specifies the conditions for linking tables?

3.7.11What would happen if in a query the selection from two tables is specified, but they are not linked?

3.7.12What is a recursive connection?

4 Laboratory work №4. Using aggregate functions and special operators in queries

The purpose of the work: study of the aggregate functions and special type operators in queries.

4.1 The task for the laboratory work

In the process of carrying out the laboratory work the student must:

- learn the assignment of aggregate functions in the SELECT operator;
- study the appointment of the special operators in a SELECT operator;
-fulfill operations on the data selection from the database using the aggregate functions and special operators.

4.2 Special operators

Special operators use different keywords in the query operators:

- LIKE operator is used in a WHERE clause to search for a specified pattern in a column. The LIKE operator is applicable only to character fields, with whom it is used to find substring. That is, it searches the symbol field to see whether it's part matches the condition. As the condition it uses special characters:

- the % sign replaces a sequence of any number of characters;
- the _ underscore sign replaces any single character.

LIKE is convenient when searching the values, you can use that part of values which you remember;

- IN operator allows you to specify multiple values in a WHERE clause. The IN operator is a shorthand for multiple OR conditions.

IN syntax:

SELECT *column_name(s)*
FROM *table_name*
WHERE *column_name* IN (*value1*, *value2*, ...);

or:

SELECT *column_name(s)*
FROM *table_name*
WHERE *column_name* IN (*select statement*);

the set of values for the IN operator is enclosed in parentheses, values are separated by commas;

- BETWEEN operators select values within a given range. The values can be numbers, text, or dates. The word BETWEEN, then the initial value, AND keyword and the final value must be indicated in the query. The first value must be less than the second.

BETWEEN syntax:

SELECT *column_name(s)*
FROM *table_name*
WHERE *column_name* BETWEEN *value1* AND *value2;*

- NULL value is a field with no value. If a field in a table is optional, it is possible to insert a new record or update the record without adding a value to this field. Then, the field will be saved with a NULL value.

Note: It is very important to understand that a NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation!

It is not possible to test for NULL values with comparison operators, such as =, <, or <>.We will have to use the IS NULL and IS NOT NULL operators instead.

IS NULL Syntax:

SELECT *column_names*
FROM *table_name*
WHERE *column_name* IS NULL;

For all these operations it is possible to build their negation (the NOT keyword) to consider the opposite condition. The NOT keyword is used with the operations as follows: NOT BETWEEN, IS NOT NULL, NOT IN, NOT LIKE.

4.3 Sorting the output

Usually it is required that the output should be somehow ordered. The ORDER BY keyword is used to sort the result set in ascending or descending order. The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

The syntax of the SELECT statement using the ORDER BY clause is as follows:

SELECT *column1, column2,* ...
FROM *table_name*
WHERE [*condition1* | *expression1*]
ORDER BY *column1, column2,* ... ASC|DESC;

Because ascending order is the default, there is no need to specify ASC in general.

Some cuts are available in SQL. The column specified in the key words list ORDER BY can be replaced by the number corresponding to the column order in the list.

4.4 Aggregate functions

Aggregate functions are used to summarize the data. SQL Server provides several aggregate functions:

- the COUNT() function counts the number of rows that satisfy the query condition.

COUNT syntax:

SELECT COUNT(*column_name*)
FROM *table_name*
WHERE *condition*;

- the SUM() function computes the arithmetic sum of all values of column.

SUM() syntax:

SELECT SUM(*column_name*)
FROM *table_name*
WHERE *condition*;

- the AVG() function returns the average value of a numeric column.

AVG() syntax:

SELECT AVG(*column_name*)
FROM *table_name*
WHERE *condition*;

- the MAX() function returns the largest value of the selected column.

MAX() syntax:

SELECT MAX(*column_name*)
FROM *table_name*
WHERE *condition*;

- the MIN() function returns the smallest value of the selected column.

MIN() syntax:

SELECT MIN(*column_name*)
FROM *table_name*
WHERE *condition*;

SUM and AVG functions are only applicable to numeric fields. With functions of COUNT, MAX, MIN can be used the numeric or symbolic fields. MAX, MIN values are compared alphabetically when applied to the character fields. Aggregate functions at their work ignore the NULL values. COUNT function is slightly different from the others. It counts the number of values in the column or the number of rows in the table.

The GROUP BY keyword in the WHERE clause allows you to specify a subset of values for which you apply the aggregate function and sort the result of the query output. GROUP BY applies the aggregate functions for groups of records. The condition of group formation is the same field value.

GROUP BY can be used with multiple fields.

The keyword HAVING is used in SELECT statement with GROUP BY clause to specify which groups should be represented in the output. For GROUP BY keyword HAVING plays the same role as the WHERE for ORDER BY (*Appendix D*).

4.5 Using subqueries

Queries can control other queries. This is done by putting the query inside the other query conditions and uses the output of the inner query in the right or the wrong condition. Usually the inner query generates the value that is tested at the condition of the outer query that defines whether it's right or not. To complete the outer (main request), it first runs the inner query (the subquery) in the WHERE clause (see example 4.1 in Appendix 4).

When you use subqueries in conditions based on comparison operations (greater, less, equal, not equal, etc.), you must ensure that the subquery will return

one and only one meaning. If your subquery does not return any value, then the main query will not display any values.

If you want to use a subquery that returns multiple rows, you must use the IN operator (*Appendix D*).

Commands like SELECT * are prohibited in subqueries.

Subqueries can also be used in the HAVING clause. These subqueries can use GROUP BY or HAVING clause (see example 4.3 in *Appendix* 4).

4.6The order of execution of the laboratory work

4.6.1 Run the MS SQL Server.

Determine the current database with the following command:

```
USE Education
```

Now all subsequent commands will be carried out exactly in this database.

4.6.2 Fulfill the exercises in *Appendix D*.

4.6.3 Search for information in separate tables, get the following information:

- identify how many students are entered into the database;
- count the total scholarship of students in a particular group;
- count the number of students of a certain group of scholarship recipients;
- count the number of teachers in each department;
- count the number of subjects in the database;
- count the number of lecture and laboratory hours in all subjects;
- count how many subjects are studied by each group;
- count the number of students with excellent marks in the DB;
- count the number of students with excellent marks in each group;
- get names of students with the minimum rating in ascending order;
- get names of students with the maximum rating in descending order;
- count the number of students living in the dormitory;
- calculate the average rating of DB's students;
- calculate the percentage of students with excellent marks;
- calculate the percentage of underachieving students;
- calculate the percentage of teachers with advanced degrees;
- calculate percentage of senior teachers;
- calculate the percentage of assistants.

4.7 The report on the laboratory work

Report contains listings of commands for using the aggregate functions and query results.

4.8 Control questions

4.8.1 What operators of a special kind do you know? The purpose of each operation?

4.8.2 What are the relational operations used in the data selection?
4.8.3 What logical operations are used in the data selection operations?
4.8.4 What is an aggregate function?
4.8.5 Does the type of data play role when using the COUNT function?
4.8.6 In order to group data by column, is it necessary to have this column specified in the SELECT keyword list?
4.8.7 What are the table aliases used for?

5 Laboratory work №5. Creating and using views and stored procedures

The goal of the work: training the use of views for data protection; development of stored procedures.

5.1 The task for the laboratory work

In the process of fulfilling the laboratory work the student must~~to~~:
- examine the purpose and creation of views;
- examine the purpose and creation of stored procedures;
- create views for database "Education";
- develop stored procedures for database "Education".

5.2 Using views for data protection

A view is a virtual table, which is a combination of tables in the form of a pre-specified query. The main difference between a view and a table is that data tables require the physical memory for its storage, and the view simply references the data table and therefore their data is not required.

The view is needed to limit user access to specific columns and rows of tables, depending on the conditions defined by the expression WHERE keyword in the view definition.

The view is created with the command CREATE VIEW:

1) Creating a view for the data in one table:

CREATE VIEW *viewName*
SELECT * | *column1*[,*column2*]
FROM *tableName*
[WHER E*expression1*[,*expression2*]]

2) Creating a view for the data in several tables:

CREATE VIEW *viewName*
SELECT * | *column1*[,*column2*]
FROM *tableName1, tableName2[,tableName3]*
[WHERE *expression1*[,*expression2*]]

3) Creating a view based on other view:

CREATE VIEW *viewName 2*
SELECT *
FROM *viewName 1*

5.3 Use of the stored procedure

A set of SQL operators, created for the convenience of use in programs is called the *stored procedure*. A stored procedure is a prepared SQL code that you save so you can use the code over and over again.

Some of the advantages of using the stored procedures:

- the operators of the procedure are already stored in the database;
- operators of the procedures have already been tested and are in the ready to use form;
- the procedures result is faster;
- the ability to maintain the procedures allows the use of modular programming;
- stored procedures can call other procedures;
- stored procedures can be called by other programs.

In SQL Server the procedures are created with the help of the operator of the following form:

CREATE PROCEDURE *name of the procedure*
[[( ] *@name of the parameter*
TYPE OF DATA [(LENGTH) | (ACCURACY) [, SCALE])
[=DEFAULT][COMING]
[, @NAME OF THE PARAMETER
TYPE OF DATA [(LENGTH) | (ACCURACY) [, SCALE])
[=DEFAULT][OUTCOMING]]
[WITH RECOMPILE]
AS *operators SQL*

The stored procedures are used as follows:

EXECUTE [@ =] *name of the procedure*
[[*@ name of the parameter* =] *value* |
[*@name of the parameter* =] *@ variable* [OUTCOMING]]
[WITH RECOMPILE]

There are various options that can be used to create stored procedures:

1) *Creating a simple stored procedure.*

CREATE PROCEDURE *procedureName*
AS
SQL operators

For example:

CREATE PROCEDURE *procedureName*
AS

SELECT* FROM *tableName*

To run the procedure to return the contents from the table specified, the code would be:

EXEC *procedureName*

or just simply

procedureName

2) *Creating stored procedure with parameters.* The real advantages of stored procedures are the ability to use parameters:

- *one parameter*. Consider the example: We will query the "studentsAddress" table from the "University" DB, and we want to get the list of students from a particular city:

```
CREATE PROCEDURE University
@City nvarchar(30)
AS
SELECT *
FROM studentsAdress
WHERE City = @City
```

Execution of this stored procedure:

```
EXEC University @City = 'Astana'
```

Here we can change the "=" to a LIKE and use the "%" template;

- *multiple parameters.* You need to list each parameter and the data type separated by a comma:

```
CREATE PROCEDURE University
@City nvarchar(30) ,
@AddressLine1 nvarchar(60)
AS
SELECT *
FROM studentsAddress
WHERE City = @City
AND AddressLine1 LIKE 'A' + '%'
```

To execute this you could do any of the following:

```
EXEX University
@City ='Almaty',
@AddressLine1='A'
```

3) *Using comments in a stored procedure*. To create comments you just use two dashes "--" in front of the code you want to comment. To create block comments the block is started with "/*" and ends with "*/". Anything within that block will be a comment section. You can also use both types of comments within a stored procedure(combining *Line* and *Block Comments*).

4) *To drop a stored procedure* you use the DROP PROCEDURE or DROP PROC command.

5) *To modify an existing stored procedure* is used ALTER PROCEDURE or ALTER PROC command.

5.4 The order of execution of the laboratory work

5.4.1Create views of various kinds for the database "Education":

- a view that contains only the description of the subject;
- a view that contains the names of the teachers and their positions;
- a view containing the student's ID, name of the student, course, name of lecturer, assessment in the discipline;

5.4.2 Create the stored procedures:

- to insert the new rows to the table Students;
- to search for the names of students whose GPA is less than required;
- to increase the scholarship amount by 12%;
- to generate a list of teachers with a list of disciplines, which they teach;
- to form a group name and a list of students in these groups;
- check the operation of all created procedures.

5.5 The report on the laboratory work

The report on the work contains:

- the listings of commands on creation and work of the view and procedures;
- the results of the created procedures;
- the results of fulfilled queries to the *Education* database.

5.6 Control questions

5.6.1 What is a view?

5.6.2 What happens if the table on which base the view is built will be deleted?

5.6.3 How can the view be used to protect data?

5.6.4 How can the view be deleted?

5.6.5 What is a stored procedure?

5.6.6 What are the benefits of using procedures?

5.6.7 Does a stored procedure call another stored procedure?

5.6.8 Does a stored procedure perform calculations?

5.6.9 Does a stored procedure perform comparison of the user's input with the pre-established conditions?

5.6.10 Where is the procedure stored?

6 Laboratory work №6 Using triggers

The purpose of the work: practicing development of the mechanism of triggers to maintain data integrity in the database.

6.1 The task for the laboratory work

In the process of carrying out the laboratory work the student must:
- study the purpose of triggers;
- study the syntax for creating a trigger;
- develop triggers of different categories for database "Education".

6.2 Definition of the trigger

Triggers are special stored procedures, which are automatically executed when certain events occur. The trigger is mainly used for maintaining the integrity of the information in the database. These procedures are called triggers because they interrupt execution of an event (such as adding a record to a table). The triggers are divided into three categories: UPDATE, INSERT and DELETE triggers. All three types can be combined in one trigger. Obviously, the inability to trigger on a SELECT operation is due to the lack of data modification of this operation.

A trigger is *attached* to a table. Unlike a stored procedure, a trigger cannot be started manually, to have parameters and return values. Triggers have many uses, but are most often used to implement business rules.

Action of triggers is not limited to databases in which they are installed. They can be used to modify data in tables in other databases, even on other servers.

The syntax of a trigger:

```
/*The title of the trigger*/
CREATETRIGGERtriggerName                 /*name of the trigger*/
ONtableName                              /*the name of the table for which
                                         the  trigger was written */
FORINSERT, UPDATE, DELETE                /*for what events reacts*/
AS                                       /* key word */
/*Determination of the trigger data*/
BEGIN                                    /*start of trigger body*/
DECLARE @varName type                    /*var_name – name of the variable,
                                         @ - obligatory symbol,
                                         type- type of variable data*/
SELECT @varName=tableName.columnName     /*assignement of variable
                                         the values of a table column */
FROM tableNameT, inserted Q              /*from what table will be selected
                                         column;  T,Q –aliases; inserted –
                                         symbol  of inserted data*/
```

```
WHERE T.key=Q.key                    /* compares the key fields
                                     of the source table  and inserted data */
IF condition                         /*the condition for the trigger execution*/
BEGIN
ROLLBACK TRAN                        /*decline of transaction*/
RAISERROR ('Notification')           /*error message */
END
END                                  /*the end of the trigger*/
```

Here:

- the CREATE TRIGGER statement is used to create the trigger;
- the ON clause specifies the table name on which the trigger is to be attached;
- the FOR INSERT specifies that this is an AFTER INSERT trigger. Instead of FOR INSERT, AFTER INSERT can be used. Both of them mean the same.

Example of a trigger creation. Define a trigger *tri_ins_progress* for the table "Progress", which will be run every time when a record is inserted into the table or updated. If exam is not passed in time (for example, after the 15th day of the month), the insertion is not accepted.

To create a trigger, select a table "Progress" in the list of database objects, and then in the context menu *All tasks - Manage Triggers*. Thedialog box of the trigger's properties is open. In this window, insert the following text:

```
CREATE TRIGGER tri_ins_progress
ON Progress
FORINSERT, UPDATE
AS
/*description of local variables*/
DECLARE @nDayOfMonthTINYINT
/* an information about insertion records*/
SELECT @nDayOfMonth = DATEPART (Day, I.Pr_DATE)
FROM Progress P, Inserted I
WHERE P.studentID = I.studentID
AND P.mark = I.mark
/*checking the insertion condition */
/*if it is need the error is reported */
IF@ nDayOfMonth>15
BEGIN
      /*ROLLBACK command is used to cancel the modification of data
in case the database is locked */
      ROLLBACK TRAN
      RAISERROR('Enter the marks received up to 15th', 16,10)
END
```

In the table "Progress", enter the data about the students' grades marked later than the 15th day of month.

The trigger can be removed as follows:

DROP TRIGGER *trigger_name*

or from the context menu *All tasks* - *ManageTriggers*, select the trigger's name from the list.

6.3 The order of execution of the laboratory work

6.3.1 Run MS SQL Server.

Determine the current database with the help of the following command:

USE Education

All tasks are executed by Transact-SQL commands (do not use the graphical user interface).

6.3.2 Make the following changes in the structure of database "Education":

- add the table "Finance" in which the data of tuition fees of student sis stored;
- add the fields "studentGPA","studentPaid" into the table "Students";
- add the field "transfer" into the table "Students"; it is a logical field, which states whether the student is transferred to the next course (depending on GPA);
- in table "Finance", provide the possibility of calculating the total amount for the semester, for the year.

6.3.3 Using the UPDATE command, fill in with the data the fields "studentsGPA", "studentsPaid" of the table "Finance" (the last field for some students becomes a value of NO).

6.3.4 Create triggers that allow you to control the following situations:

- not admit the student to the session in case of non-payment for the studying;
- nottransfer the student to another course if his/her GPA is less than the required value;
- not allowed for certain classes to set the number of hours greater than the specified value;
- create own triggers for specific situations.

6.4 The report on the laboratory work

The report contains:

- triggers listings;
- listings of the results of the trigger's work.

6.5 Control questions

6.5.1 What are the advantages of using procedures?

6.5.2 When are triggers executed?

6.5.3 Is it possible to change the trigger?

6.5.4 How can you enter the text of a trigger?

6.5.5 How is the work of the trigger checked?

7 Laboratory work №7. Preparing to create user applications

The purpose of the work: studying the procedure to create a client application to a database by MS Access.

7.1 The task for the laboratory work

In the process of carrying out the laboratory work the student should:
- study the purpose of the client application;
- import a database of "Education" from MS SQL Server in Access;
- create the required queries;
- create the necessary forms and reports.

7.2 The purpose of the client application

The aim of a client application is to enable the user to work with data in a convenient form. To date, there is no perfect database management system with the developed interface and the optimized structure. For example, SQL Server has no user interface in the usual sense. One of the technologies developed by the Microsoft Corporation is the technology ADO (Active Data Objects). This technology is focused primarily on creating client-server applications involving constant customer interaction with the database server.

DBMS MS Access is quite convenient as the client application when working with database systems. For some simple tasks you can use it as the primary database management system. In the laboratory work there will developed the interface through which you can perform the following tasks: view data in tables of a database; performing a variety of database queries; creating forms for a future interface; generation of reports.

7.3 Working in MS Access

7.3.1 Import database from MS SQL Server to MS Access.

In this laboratory work the user interface design will be performed for the database "Education", which had been previously imported into MS Access.

The process of import consists of two steps (*Appendix E*):

1) The connection to the database.
2) Import database in MS Access.

7.3.2 Short description of working in MS Access.

Now, in the following description it is assumed that all tables of the database are already available.

Tables. According to the chosen field type, several field properties appear on the bottom window of the table *design* mode.

If you add new data to database tables, keep in mind that unlike MS SQL Server the rule of referential integrity in MS Access is not controlled by the system, therefore, the user should remember it when filling in these fields.

Search data. In MS Access the graphical version of the relational language SQL is implemented. This version of the language is called QBE (*queries by example*). In this language the queries are performed by a graphical display of database tables. It is very convenient for non-professional users. Of course, some of the features of the SQL language are absent in the language QBE.

To search for linked data in MS Access object *Queries* is designed. In the section *Create* you select the *Create query* in design view, a window *QBE* will appear. Filling this form according to certain rules, obtain the necessary information. When you want to limit the results of a query based on the values in a field, you use *query criteria*.

When solving practical problems you must enter a field value in a dialogue with the user during execution of the query. In order to display a dialog box for entering a specific value fields in selection criteria, you need to define *query parameter*. The name of the query parameter can be specified in the *selection criteria* field in brackets. When the query is run the name will appear in the dialog box, the parameter value is entered in this window.

For all queries you can view the SQL statements: *Design– SQL mode*.

*Creating forms***.** MS Access has a means of developing a user-friendly graphical user interface. The basis for the development of dialog user applications for work with database is the *forms*. Typical procedures are automatically generated when creating form elements. Such elements, for example, are graphic buttons that can connect the events of different categories (navigation records, work with a form, report, run a query, macro, print table, etc.).

Design of reports. All created objects can be printed. But the document will be printed in the form as it looks on the screen. Usually the documents have a form that is used in daily work. To get them, use the object *Report*. Reports, like forms, can be obtained by using the *Wizard* mode, but it is more interesting to construct them yourself.

7.3 The order of execution of the laboratory work

7.3.1 Run MS Access.

7.3.2 Import the database "Education" from MS SQL Server in MS Access (see *Appendix E*).

7.3.3 Consider the table "Students". Change some table field *properties*(in the lower part of window):

- set the "*Mask of input* " for mobile phone the mask of number can be "0-000-000-00-00");
- for the field "studentAddress" set the properties: "*Default value*", "*Condition value*" – “Almaty” and "*Message about error*" – “Only from Almaty”.

7.3.4 Make some changes to the structure of the tables: open the "Students" table in the design mode, add a new field called "*gender*", the field type is *numeric*. In the column *Description field*, enter the text: 1 – female, 2 – male. Open the "Students" table, place the cursor on any row of the column "*gender*" and look at the status bar; fill the box.

7.3.5 Let's look at a simple example in which we will use criteria in a query. We need to get a list of students attending the classes of a certain teacher.

Select the tables which will fill the bottom part of the QBE window: choose the table's names and their fields; then on the string *Condition of selection* it is possible to set the condition. There can be several conditions: the conditions associated with the operation OR installed in multiple rows of one field; for operation AND conditions are established on single line multiple fields. In terms of selection it is possible to apply special operators (IN, BETWEEN, LIKE, etc.).

7.3.6 Define the *query parameter*. The name of the query parameter can be specified in *the selection criteria* in brackets. We want get information about students' marks on the specific subject. Create the query, on the QBE window in the field "dicsiplinesName" type: [input discipline name]. Check the work of the query.

7.3.7 *Create a calculated field in the query*. Suppose you want to find the records of the disciplines in which the total number of hours for the discipline is not the sum of lectures, labs and practices hours.

Create a query by the table "Disciplines". In the empty cell of the string *Field* choose (in context menu) *expression Builder;* selecting in its window the needed table and its field using click, create the expression: [totalHours]-[lecturesHours]-[practiceHours]-[labHours]. The criteria are<>0 (not equal 0). Save the query by the name "*Difference in hours on the discipline*".

7.3.8 Change the structure of the table: open the table in design mode, add the field "workingProgram" with a data type of *Memo*. Recall that if the data type of field is *Text*, you can enter up to 256 characters, if the amount of text is larger, use the *Memo* type.

7.3.9 *Creating forms*. Create a form for the table "Disciplines" using the Wizard. Open this form, fill in the new field.

Open this form in *design* mode. Add the *Title* using *Label* element; add the *date*, *number of page*. Change the same properties of the form (*context menu*, *Properties*). Save changes and look for the form (click to *Open*).

7.3.8 Now we will create a form consisting of several tables based on the tables "Departments" and "Teachers".

Create any form for the "Departments" table. Open this form in the *design* mode. On the *Toolbar* find the element *Subordinated form/report*. Click and set it on the form, follow *wizard* instructions. Close the design window; open it in the form mode. Check the work of form: when you look for the specific department, the list of teachers of this department appears in the subordinated window.

7.3.9 To the "Students" table, add the field of "zodiacSign", but do not fill in this field. Using the *Wizard* create this form by selecting fields to include: "studentName" and "dateOfBirth".

Open the form in design view. Choose the item *list Box* on *Toolbar*; install it in the form freespace. In the next window of the *Wizard* select the string "a fixed set of values will be entered". In the next window, enter the list of zodiac signs, then select a box to save this list of zodiac signs. After completing open the form, fill in the field "zodiacSign" by selecting fields from the list. Viewing the form in *Datasheet* view, you will see a filled box.

7.3.10 Consider the following problem: in the form to view the group information, the user must open the list of students of the current group. First, based on the tables "Groups" and "Students" forms with the same nameswill be created.

In the "Groups" form, clicking on*Button*item you want to open the form "Students". To do this, open the "Groups" form in design mode and add *Button* to open the form the "Students", using *Wizard*. In the being opened “Students” form we want to reflect only the connected records. This requires choosingthe string "*Open a form for selected records*" in the *Wizard*, and selecting the field "groupID", by which the forms and records of subform are connected. Give the name *List of students* to the button.

Go to form mode, check the button’s work.

7.3.11 Implement a data search of the students by their name in the table "Students". For this in the "Students" *form*, create a button to start the search procedure.

7.3.12 *Creating reports*. Suppose that you want to get a list of students in groups:

- create any report for the table "Students". Open it in design mode;
- for the fields "groupID" and "studentID" set the *ascending sorting;*
- the total list of students should be divided into groups, so you set *grouping* by the field "groupID";
- using item "Label" set the report name in the report header.

7.4 The report on the laboratory work

The report should contain: the listings of all executed tasks with detailed explanation.

7.5 Control questions

7.5.1 What is the user application?
7.5.2 How do you import your DB from MS SQL server to Ms Access?
7.5.3 What is QBE?
7.5.4 What is the query parameter?
7.5.5 How is the user dialogue organized when you run the query?
7.5.6 The purpose of forms. The types of forms.
7.5.7 How can you design the reports?

8 Laboratory work №8. Development of the user interface

The purpose of the work: to study the procedure of creating a client application to a database by MS Access.

8.1 The task for the laboratory work

In the process of carrying out the laboratory work the student should:
- understand the need of the user application;
- study the purpose of switchboards;
- develop user applications by two ways.

8.2 Creation of the user interface

A large number of objects, not grouped by application features, hinder executing of data processing tasks in the automating subject domain. Creation of a cohesive application of the given subject domain, all of whose components should be grouped on a functional basis is needed to organize effective user's work.

It is necessary to provide a convenient graphical user interface. *Forms* play a special role when creating applications since they are the primary interactive tool of the user experience.

To combine objects into a single dialog application *Switchboard* can be created. Switchboards in MS Access allow users to easily navigate between the different forms and reports within a database. The buttons of this form provide a challenge of the other switchboards, and here also the buttons to return to the forms of the previous levels, for exit the program are placed. Usually a button to change the switchboard is also provided. Upon downloading the main switchboard the user immediately starts work in the application environment and is prepared to perform tasks.

The first way of developing applications is using control items (such as *Button*). MS Access has also a facility of automatic development of application - *Switchboard Manager*.

8.3 The order of execution of the laboratory work

8.3.1 Consider the features of future application to the database "Education".

8.3.2 Let's create our first Switchboard. For this example we'll add several switchboard buttons:

- a button that opens the "StudentsList" form in "Edit" mode;
- a button that opens the "ProgressOfStudents" report;
- a button that opens the "Teachers&Subjects" report;
- a button to exit the program.

Firstly, you should create the forms on the basis of corresponding tables and queries.

8.3.3 Here's how we create and add items to our switchboard:

- choose menu *Database Tools- Switchboard Manager*; Access will inform you that no switchboard was found – do we want to create a new one? Click *Yes;*

- the *Switchboard Manager* will open with a new default switchboard highlighted. Click the *Edit* button;

- on the *Edit Switchboard* page, click *New;*

- in the *Edit Switchboard* item page, give your switchboard button a label; for the *Command* drop down, we'll pick *Open Form* in *Add* mode; under *Form* we'll choose the "StudentList"; click *OK* (Figure 8.1).

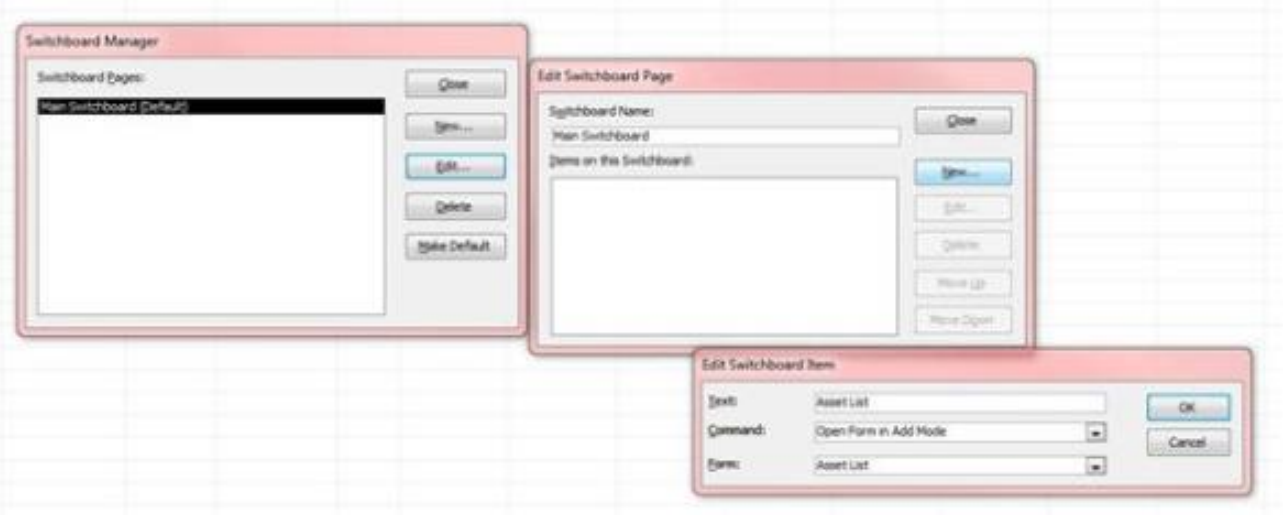

Figure 8.1 – Creating the first page for Switchboard

- back at the *Edit Switchboard* page, click *New* and repeat the process to add your reports. This time for the *Command*, you'll choose report; under *Report*, select the report you want to display;

- repeat the steps to add the second report;

- for the *Exit* button we'll add another item. This time for the *Command*, we'll choose *Exit Application;*

-your switchboard page should look like Figure 8.2;

- if you need to rearrange the order in which the buttons appear, you can highlight an item and use the *Move Up* and *Move Down* buttons;

- to switch to editing the switchboard form, create in a *Main switchboard form* the button *Application design*. This will allow to make the necessary changes;

- returning in the main application you can test the switchboard by navigating to *Forms* and opening the *Switchboard form*;

- if we want that MS Access automatically open our switchboard when the application is opened, you can do this by the following way: on the header of Access click the drop-down list button, the list "Customize the quick access toolbar" being opened, choose "Create"; click "Current Database" on the left side and under *Display Form*, select your switchboard, click *OK*.

Figure 8.2 – Created main switchboard page

Now when you close your application and re-open it, your switchboard will open automatically;

- open the switchboard form in design mode and using the *Properties* window, you can place the picture in the left part of the shape.

8.3.3 Create some forms to reflect all possibilities of the DB "Education":

- when you view the list of groups you must get access to the list of students of the specific group; the same for departments and teachers;

- the user needs to get information about the progress of the students, personal information about them in convenient forms;

- using different items create on your form the possibilities for answering any questions which can be addressed to the DB "Education".

8.3.4 Create a switchboard form for the DB "Education" using *Switchboard Manager*.

8.3.5 Create the application using a blank form and placing on it the buttons which realize some events (open reports, forms and so on).

8.4 The report on the laboratory work

The student demonstrates the developed applications.

8.5 Control questions

8.5.1 What is the user application?

8.5.2 How can you develop an application?

8.5.3 Explain the purpose of Switchboard Manager.

9 Laboratory work №9. Development of a client application in Delphi

The purpose of the work: to study the procedure of creating a client application using Delphi

9.1 The task for the laboratory work

In the process of carrying out the laboratory work the student must:

- familiarize with utility BDE Administrator*;* configure BDE to create an alias;
- create the forms, which are needed for application;
- create the procedures for using some Delphi components in the application form;
- check the application's work.

9.2 Access to data from Delphi applications (creating an alias)

In the lab work the access via interface created on the basis of a powerful programming library - *Borland Database Engine (BDE)* is used. Themain convenience of BDE is that all database-related settings are separated from the client application. It is enough to specify some conditional name, and the program will find and include its related settings. This name is called an *alias*.

So that the program can manipulate the data from the database, you first need to configure the BDE to create some alias and associate it with the database. The developers have made for this problem special utility - *BDE Administrator*.

This procedure is performed in the following order:

1) Select *Start-Programs - Borland Delphi7 - BDE Administrator*. In the left part of the BDE Administrator is a database panel, which shows all aliases defined at this moment in the BDE.

2) Select: *Object-ODBC Administrator*. In the appeared window click *Add*.

3) In the next window, select the database driver type.

4) In the next window, specify the name of the alias (for example, *My_Application*), description and type of the server (Figure 9.1).

5) In one of the following windows data source is selected from the drop-down list, for example, *Education* (Figure 9,2), the language used for messages.

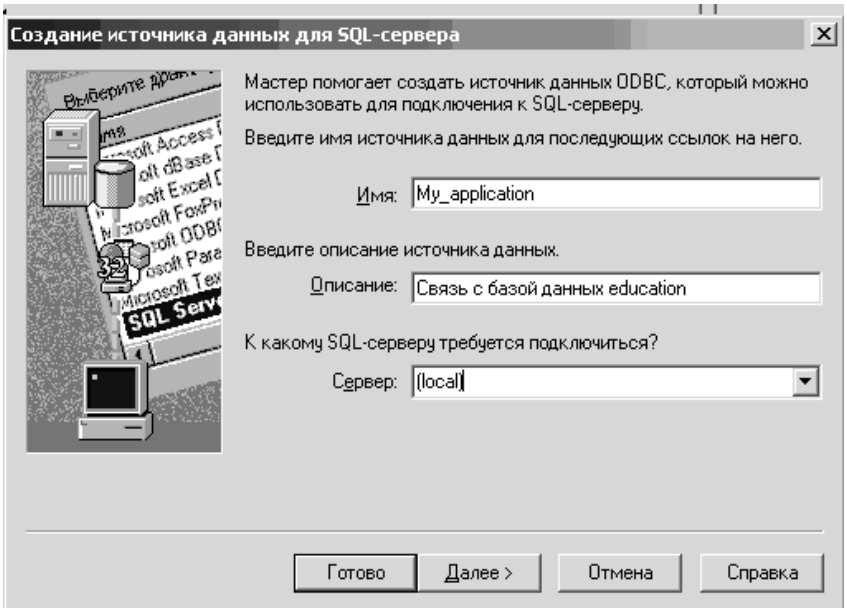

Figure 9.1 – Creation of aliases

6) In the end the name of your alias will appear in the list. Now, through it, you can access to the database.

Perform testing to ensure that the database connection is established.

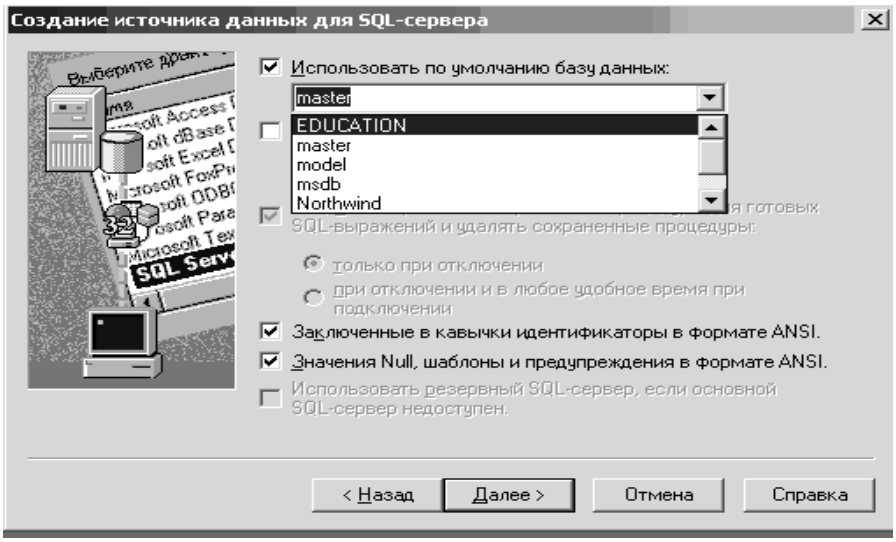

Figure 9.2 – Choosing the data source

9.3 Creating an application to the database "Education"

9.3.1Creating the initial form

9.3.1.1 Run Delphi. From the components we mainly need:

- DataAccess – data sources. These components can be used in all types of access to the database;

- BDE – components that provide data access through Borland Database Engine;

- DataControl – components that need to work with the data themselves (editing, navigation, display).

9.3.1.2 Place the following components of Query on the form: (section BDE), Data Source (Data Access section), DB Greed (Data Control section).

9.3.1.3 Set the following component's properties:

for Form1: *Caption* - Reference to the database Education;
Position - poScreenCenter;
for Query1: *DatabaseName* - MyApplication (by selecting from a list);
requesLive - true;
SQL - select * from Students
(the SQL property - text of the query. To enter it you must click on the button with ellipsis that appears in the right part of the field);
for Data Source1: *Data Set* - Query1;
for DBGreed: *Data Source* - DataSource1.

9.3.1.4 Set the property *Active := true* for Query1 component.

On the screen, a dialog window for query input will be displayed. Click on "OK". There will be a connection to the database and the records will be displayed in the table. Return the value *Active:= false* for the Query1 component.

9.3.1.5 Add two buttons (a Button from the Standard section) to the bottom of the form. Set the properties:

*Caption for Butto*n1 – Connect;

Caption for Button2 – Exit.

9.3.1.6 Write for the buttons handler methods of the OnClick event. Double-click on Button1 to edit the OnClick method:

```
procedure TForm1.Button1Click(Sender: TObject);
        begin Query1.Active:=true;
        Button1.Enabled:=false;
end;
```

Method for Button2 (this button closes the application):

```
procedure TForm1.Button1Click(Sender: TObject);
        begin if Query1.Active then Query1.Close;
        Application. terminate;
end;
```

9.3.1.7 Save the project. Run the project and click "Connect". Click "Exit".

The obtained table contains all columns of the database table and the names of columns are used as the column headings. Change the names of table headers for making them more user-friendly.

9.3.1.8 Select the table *DBGreed1* and from the context menu, select *Column Editor*. Press *Ins*. A new row will appear; each row of this table corresponds to a column of component *DBGreed*1.

9.3.1.9 Set the following values for the column properties:

*FieldNam*e - the field name of the table associated with the column (e.g. "Name of student" instead of studName").

Open the list *Title* by clicking on the cross next to the word *Title*, and select title properties:

Alignment - taCenter;

Caption - the name of the student;

Font - bold.

Similarly, create headings *Name*, *Date of birth*.

9.3.1.10 For *Query1* component turn on the property *Active = true* and edit the width of the form, tables and columns so that they all fit in the window.

Check the work of the application.

9.3.2 Working with data.

9.3.2.1 Run your application. To switch to edit mode, place the cursor in an edited position and press any key. Edit some records.

9.3.2.2 Implement deletion of records programmatically. On the form, place *Button3* labeled "Remove". Create a handler of the clicking this button:

```
procedureTForm1.Button3Click(Sender:TObject);
begin
    Query1.Delete;
end;
```

9.3.2.3 Before deleting a record, you should receive a warning message. So add an event handler *BeforeDelete* (it is the property of SQL component on the tab *EVENTS*):

```
procedureTForm1.Query1BeforeDelete(DataSet:DataSet);
    begin
        if not (MessageDlg('Delete record?',
        mtError, [mbYes,mbNo],0) = mrYes) then
    begin
    Abort;
    end;
end;
```

9.3.2.4 Place in the bottom of the form an element *TPageControl* (from section Win32). On one of the pages of this element we will place the searching elements, on the other – background information about the developer program. Edit the component sizes.

9.3.2.5 To create the component pages, click on it by the right button and select "New Page" from context menu.

Select the *Caption* property for TabSheet1 – "Search"; for TabSheet2 –"Help".

9.3.2.6 On the "Search" page, place components *TLabel, TEdit, TButton* with the following properties:

CaptionforLabel1 - Name;
CaptionforButton4 - Find;
Text for Edit1 - empty.

Create a handler for the Button4:

```
procedure TForm1.Button4Click(Sender: TObject);
begin
    Query1.Locate('studentName',Edit1.text,[loPartialkey, CaseInsensitive]);
end;
```

When you click this button, there will be a search for the record with last names beginning with the string entered in the Edit1.

9.3.2.7 Change behavior of search functionality. Add two components to the form: *TCheckBox* (switches) with the signatures of "Partial string" and "Case insensitive".

Change the handler for Button4 as follows:

```
procedure TForm1.Button4Click(Sender: TObject);
varLocOpts:TLocateOptions;
        begin
          LocOpts:=[ ];
          if CheckBox1.Checked then LocOpts:= LocOpts+[loPartialKey];
          if CheckBox2.Checked then LocOpts:= LocOpts+[loCaseInsensitive];
          Query1.Locate ('Stud_FAM', Edit1.Text, LocOpts);
        end;
```

Now the work of search function will vary depending on the user assignment.

9.3.2.8 Improve the program to search by multiple table fields, for example, and by name and surname. Add fields *TEdit, TLabel*. Change field labels and the button handler for Button4:

```
        procedure TForm1.Button4Click(Sender: TObject);
        var LocOpts:TLocateOptions;
                LocFields:string;
                LocValues:variant;
        begin
          LocOpts:=[ ];
          LocFields:='studentName';
          if Length(Edit2.Text)>0  then
                begin
                        LocFields:=LocFields+'; studentSurname;
                                LocValues:=VarArrayCreate([0,1],varVariant);
                                LocValues[0]:=Edit1.Text;
                                LocValues[1]:=Edit2.Text;
                         end
                                        else
                                LocValues:=Edit1.Text;
                    if CheckBox1.Checked then LocOpts:=LocOpts+[loPartialkey];
                    if CheckBox2.Checked then LocOpts:=LocOpts+[loCaseInsensitive];
                    Query1.Locate (LocFields, LocValues, LocOpts);
                end;
```

9.3.2.9 On the "Help" of the *PageControl1* component, place the background information.

9.4 The order of execution of the laboratory work

You must create an application for your database.

Set on the basic form *TPageContro*l component with three pages: *Tables, Queries, Evaluation.*

On the page "Tables" by selecting the table from the drop-down list (element TComboBox), and clicking "Browse table" (TButton) the desired table to view is

called, and the selected table is recorded in the TDBGrid. Using the DBNavigator you can navigate the table.

Similarly, create "Queries" and "Marks" pages.

You can develop your own applications.

9.5 Report on laboratory work

Students present their applications to the audience and the teacher.

9.6 Control questions

9.6.1 What is a user application?
9.6.2 What is ODBC?
9.6.3 How is called the software by which we get access to a database?
9.6.4 Can the database on one server be accessible from another server?

Appendices

Appendix A

The script for the creation of database "Education"

```
CREATE DATABASE education
ON PRIMARY
(NAME = education_data, FILENAME='C:\Program Files\Microsoft SQL
Server\MSSQL\Data\education_data.mdf',  size = 4, maxsize =25, filegrowth = 1 mb)
LOG ON
(NAME = education_log, FILENAME='C:\Program Files\Microsoft SQL
Server\MSSQL\Data\education_log.ldf', size = 4, maxsize = 20, filegrowth =1 mb)
```

Here:

- education is the name of the database;
- ON - specifies a list of files on the disk which will store the database information;
- PRIMARY - specifies the file containing the logical start of the database and system tables. A database can have only one primary (PRIMARY) file. If this parameter is omitted, the primary shall be the first file in the list;
- LOG ON - specifies the list of files on disk to store the transaction log. If this parameter is not specified, the transaction log size will be 25% of the total size of the data file;
- education_data - defines a logical name that SQL Server will use to access the file;
- FILENAME –specifies the parameters of the operating system file (the file name which must reside on the server where SQL Server is installed, the initial and maximum sizes of database and increment to increase the size of the database).

Appendix B

Transact-SQL scripts for creating tables of the database "Education"

```
CREATE TABLE Groups (
        groupID             integer  identity (1,1) not null  PRIMARY KEY,
        groupName          char (9) not null,
        course               int not null,
        studentsNumber    smallint  not null   )

CREATE TABLE  Departments (
        departmentID              integer not null PRIMARY KEY,
        departmentName          char(20)  not null,
        departmentHeadName    char(10)  null,
        departmentPhone          int   not null  )
```

```
CREATE  TABLE  Disciplines (
        disciplineID       integer   not null PRIMARY KEY ,
        disciplineName   char(20)  not null,
        totalHours         integer  not null,
        lecturesHours     integer  not null,
        practiceHours     integer  not null,
        laboratoryHours integer  not null )

CREATE  TABLE Students (
        studentID             int  not null,
        studentName         char(20)  not null,
        studentBirthday     datetime  not null,
        studentAddress     char(25)   null,
        groupID               integer   not null   FOREIGN KEY
                                        REFERENCES Group (groupID),
        groupLeader          bigint  not null,
CONSTRAINT  PK_Students  PRIMARY  KEY (studentID),
CONSTRAINT FK_Students_Students   FOREIGN KEY(groupLeader)
                        REFERENCES Students (studentID)

CREATE  TABLE Teachers (
        teacherIDint          not null PRIMARY KEY ,
        teacherName         char(10)  not null,
        teacherPosition     char(18)  not null,
        teacherDegree       char(12)  null ,
        departmentID        integer   not null FOREIGN KEY
                                REFERENCES Department (departmentID)

CREATE  TABLE Learning (
        groupID           integer not null  FOREIGN KEY
                                REFERENCES Groups (groupID),
        disciplineID      integer  not null  FOREIGN KEY
                                REFERENCES Disciplines (disciplineID),
        teacherID         int  not null FOREIGN KEY
                                REFERENCES Teachers (teacherID),
        typeOfClasses   char (20) not null,
        totalHours        integer  not null,
CONSTRAINT PK_Learning
        PRIMARY KEY (groupID, disciplineID, teacherID, typeOfClasses) )
```

```
CREATE  TABLE Progress (
        studentID               int not null  FOREIGN KEY
                                        REFERENCES  Students (studentID),
        groupID                 int not null,
        disciplineID            integer not null,
        teacherID               int  not null,
        typeOfClasses           char (20) not null,
        typeOfMonitoring        char (20) not null,
        dateOfExam              datetime  null,
        mark                    integer
    CONSTRAINTFK_Progress_Study
        FOREIGN KEY  (groupID, disciplineID, teacherID, typeOfClasses)
        REFERENCES Study (groupID, disciplineID, teacherID,
                                typeOfClasses),
    CONSTRAINT  PK_Progress
        PRIMARY  KEY  (groupID,  disciplineID,  teacherID,  typeOfClasses)  )
```

Examples of data manipulation

```
INSERT INTO Students
VALUES (02, 'Jhon', '02/04/2002','Almaty', 1, 1)

INSERT INTO Students
VALUES (03,'Jhonson', '11/09/2001', 'Aktau',1, 1)

INSERT INTO Students
VALUES (04, 'Smith',  ' 10/05/2001 ' , NULL,1,1)

 DELETE FROM Students
WHERE  studentsID = 03

UPDATE Students
SET studentAddress = 'Almaty'
WHERE studentsID = 03
```

Appendix C

Examples of simple queries

```
SELECT *
FROM Students
   WHERE  groupID = 1

SELECT   studentName, studentsBirthday, studentAddress
FROM  Students
  WHERE  studentAddress = ' Aktau '
```

```
SELECT *
FROM students
   WHERE studentName = 'Alex'
              AND
              studentAddress = 'Almaty'

SELECT *
FROM students
    WHERE studentName = 'Alex'
              OR
               studentName = 'Jhon'
```

Creation of complex queries

SQL allows us to access multiple tables in a single query. In such queries, the names of the relevant tables are provided with the names of each column. This is called a *detailed column definition* in a query. Detailed definitions are required only for those columns that are present in several of the tables specified in the query. However, typically detailed definitions specifies for all columns:

```
SELECT  Teachers.teacherName, Department.departmentName
FROM Teachers, Department
    WHERE Teacher. departmentID = Department.departmentID
```

Using of aliases

```
SELECT A.teacherID, B.departmentName
FROM Teacher  A,
         Department B
    WHERE A.departmentID= B.departmentID
```

Using multiple keys

The list of students with the names of their group's leaders:

```
SELECT  A.studentName, B.studentName
FROM Students A,
         Students B
    WHERE A.studentLeader = B.studentID
```

The list of students with their marks on the disciplines

```
SELECT  Students.studentName,  Disciplines.disciplineName,  Progress.mark
FROM Students, Disciplines, Progress
   WHERE Students.studentID = Progress.studentID
              AND
              Disciplines.disciplineID = Progress.disciplineID
```

The list of students assessed positively by the teacher Fedorenko:

```
SELECT Students.studentName, 'on discipline ', Disciplines.disciplineName,
                                    'got a score', Progress.mark,
                                    'at the teacher ', Teacher.teacher,Name
FROM Students, Disciplines, Progress, Teachers
    WHERE Students.studentName= Progress.studentID
    AND
            Teachers.teacherID = Progress.teacherID
   AND
            Disciplines.disciplinesID = Progress.disciplinesID
   AND
            Progress.mark<>2
   AND
           Teachers.teacher.Name= 'Fedorenko'
```

Appendix D

Sorting the output

```
SELECT studentID, studentName
FROM students
ORDER BY studentName

SELECT groupID, studentID, studentName
FROM students
ORDER BY 1, 3 DESC
```

Using special operators and aggregate functions

```
SELECT *
FROM Students
WHERE studentName LIKE 'O%'

SELECT *
FROM   Progress
WHERE mark BETWEEN 3 AND 5

SELECT COUNT (*)
FROM Students

SELECT COUNT( DISTINCT studentID)
FROM Progress
```

```
SELECT  studentID,  MIN(mark) AS  minMark
FROM Progress
GROUP BY studentID
```

```
SELECT studentID, dateOfExam, MAX(mark)
FROM Progress
GROUP BY studentID, dateOfExam
```

HAVING clause (defines the criterion used to remove groups from the query result, as does the WHERE clause for individual rows):

```
SELECT studentID, dateOfExam, MAX(mark)
FROM Progress
GROUP BY studentID, dateOfExam
HAVING MAX (mark) > 4
(we get a maximum score of each student, which is more than 4)
```

Using subqueries

We know the name of the student, but don't know his ID , and want to get all his marks from the table Progress:

```
SELECT *
FROM Progress
  WHERE  studentsID  = (
              SELECT  studentsID
              FROM Students
                 WHERE studentName = 'Willis')
```

There are several students with name Willis in DB, and all of them have marks:

```
SELECT *
FROM Progress
  WHERE  studentsID IN (
            SELECT  studentsID
            FROM Students
               WHERE studentsName = ' Willis')
```

Find all the estimates for the discipline of Computer Science:

```
SELECT *
FROM Progress
  WHERE disciplineID IN (
        SELECT  disciplineID
        FROM Disciplines
        WHERE disciplinesName = 'Computer Science')
```

This result can also be obtained by uniting:

```
SELECT Progress.*
FROM   Disciplines, Progress
   WHERE Disciplines.disciplineID = Progress.disciplineID
              AND
              Disciplines.disciplineName= 'Computer Science'
```

Subqueries can be used in the HAVING clause:

```
SELECT mark,  COUNT (DISTINCT studentID)
FROM Progress
GROUP BY mark
HAVING mark > (
        ELECT AVG(mark )
        FROM Progress
        WHERE dateOfExam>18/06/04)
```

(Counts students with a grade above average passed the exam after 04.06.18).

Find all students who passed the exams on May 3:

```
SELECT *
FROM Students C
  WHERE '2018-06-03' IN (
          SELECT dateOfExam
          FROM Progress O
        WHERE O.studentID   = C.studentID)
```

This result can also be obtained by another way:

```
SELECT C.*
FROM Students C, Progress O
WHERE C. studentID = O. studentID
AND
  O.dateOfExam = '2018.06.03'
```

To find all students with a mark above average, you can use the compare table with yourself:

```
SELECT *
FROM Progress O
WHERE mark > (
           SELECT AVG(mark )
           FROM Progress O1
           WHERE O1.studentID  =O.studentID)
```

Appendix E

Import from SQL Server to MS Access

Step#1 - Working with SQL Server

In the computer you should open the *Control Panel*, then choose *Administration*.

In the opened window choose the section *Source of Data ODBC*(depending on version of SQL Server it can be two types: 32d and 64d (see Figure 1).

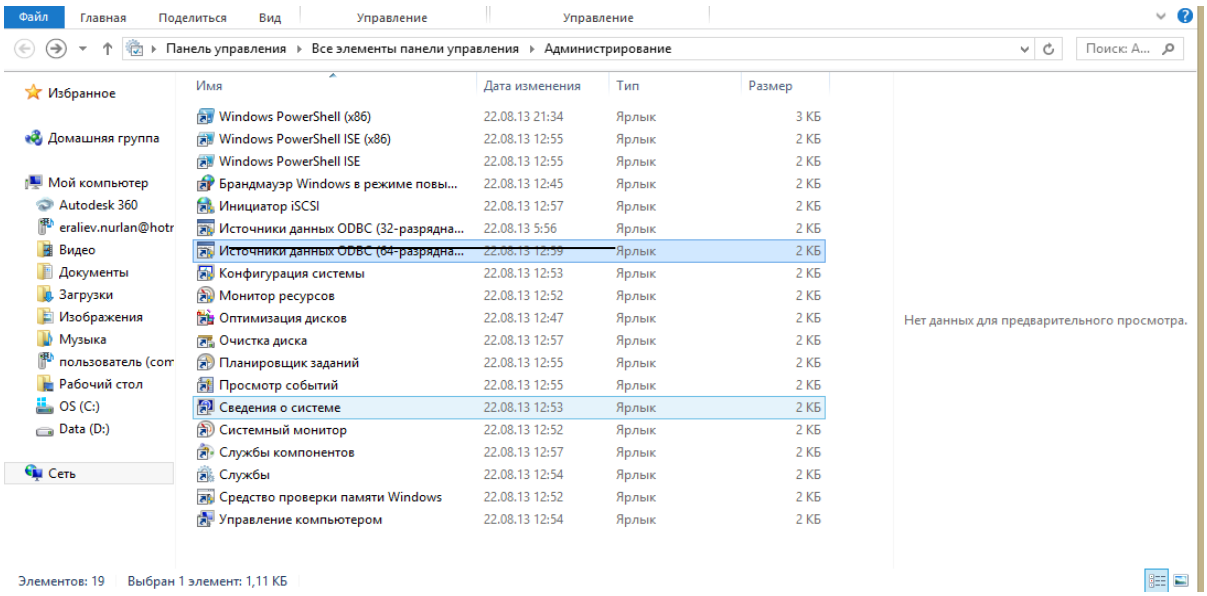

Figure E.1 - Source of Data ODBC

In the next window press to the button *Add*, then choose SQL Server (Figures E.2 and E.3).

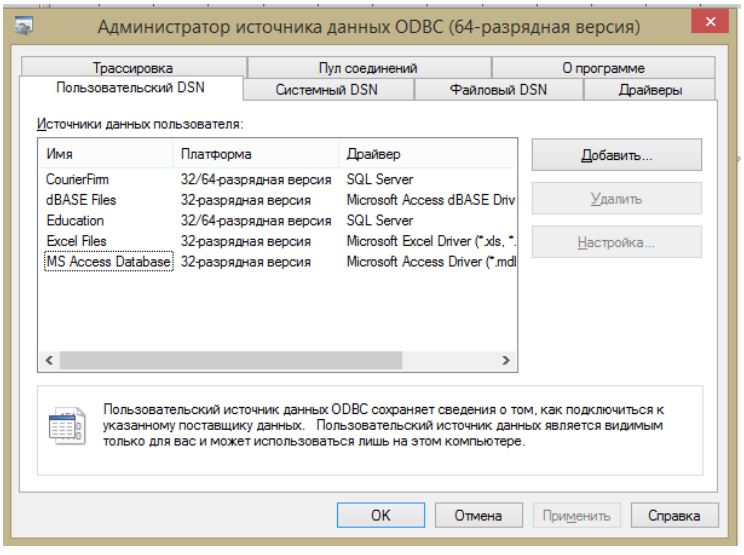

Figure E.2 – Adding the source

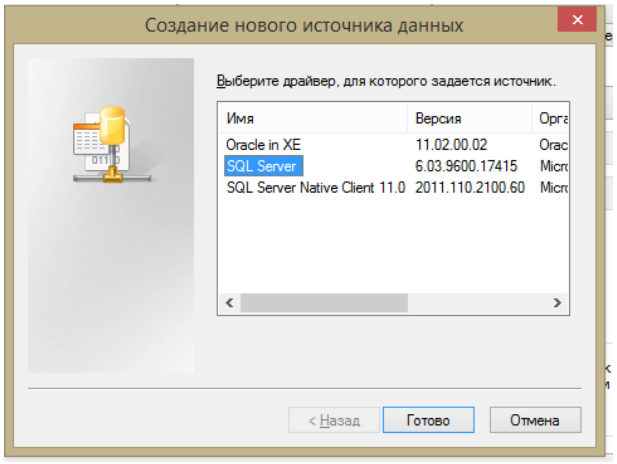

Figure E.3 – Choose SQL Server

In the next window it should fill the field *Name* of your future source, the field *Description* must be empty and in the field *Server* put the dot, then two times the button *Next* (Figure E.4).

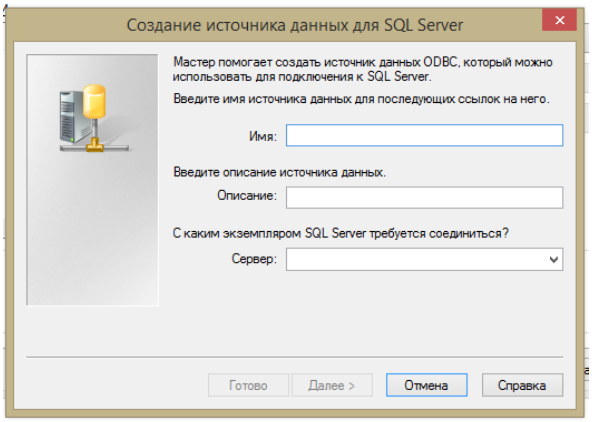

Figure E.4 – Give the name for source

In the next it should choose the database created in SQL Server (here the name of database is Nurdaulet) then press *Next* (Figure E.5).

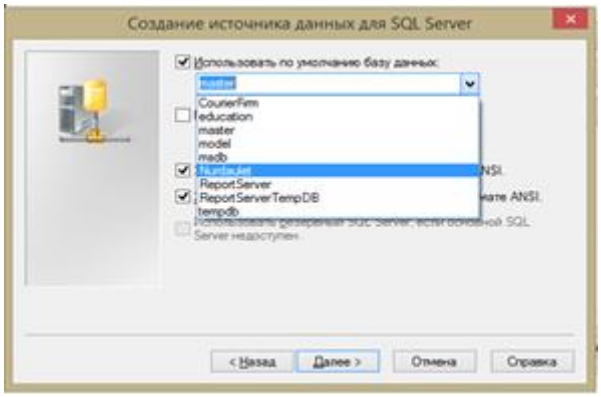

Figure E.5 – Choosing the DB

At the next window (Figure E.6) press *Ready*.

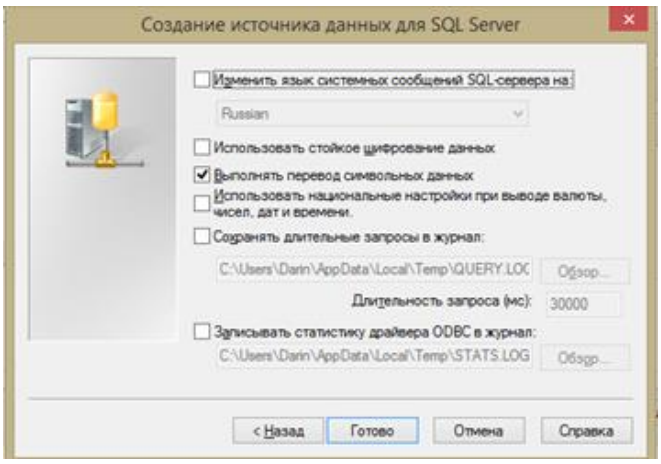

Figure E.6 – Next window

In the new window choose *Check* the source of data (Figure E.7).

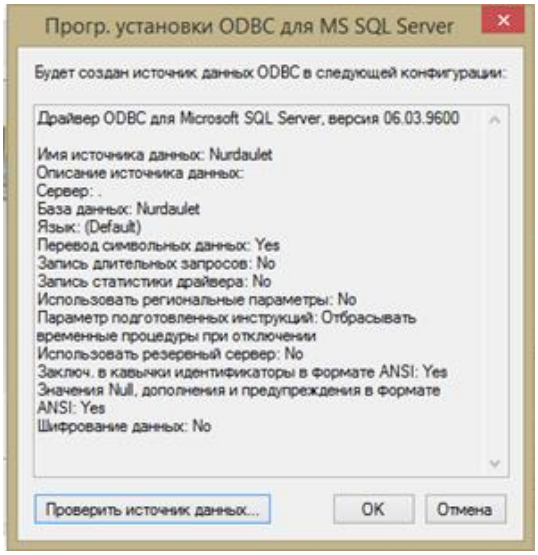

Figure E.7 – Checking the source

If you do all of this stages right it will be shown the message like in Figure E.8.

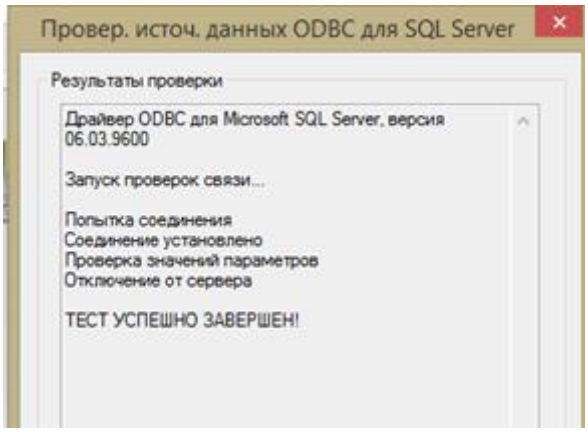

Figure E.8 – Message about correct test

And it will be added the new database (Figure E.9).

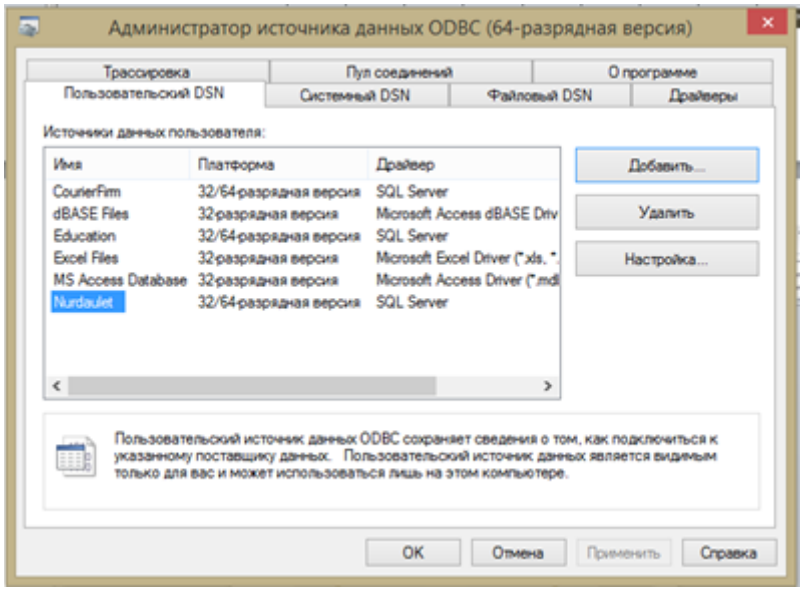

Figure E.9 – The DB is added

So the next step is setting of connection with DBMS MS Access.

Step #2 – Working with MS Access

In MS Access open the section of *External Data*, then choose the *Database ODBC* (result at Figure E.10).

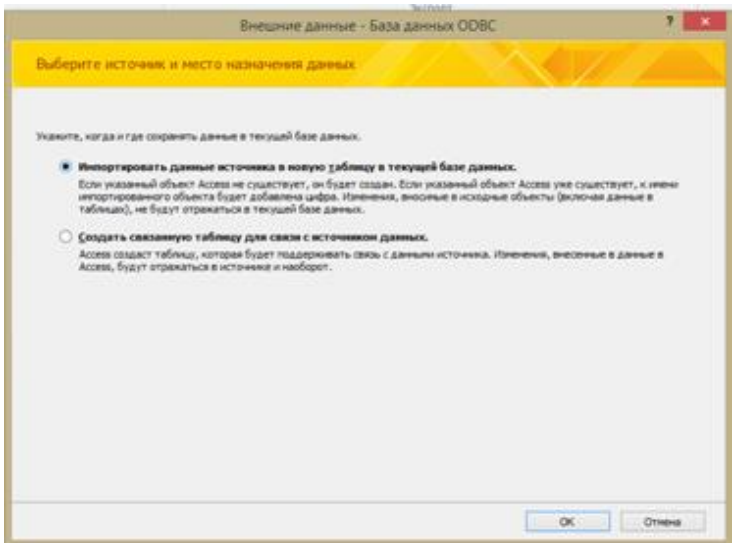

Figure E.10 – Importing data

In the next step choose the section *Source of Data* and your database is included in the existed list.

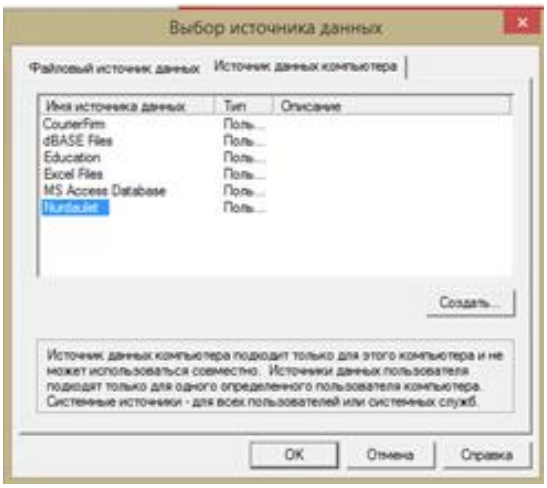

Figure E.11 – Choosing your DB

Then you have to select all of the DB objects (with *.dbo*).

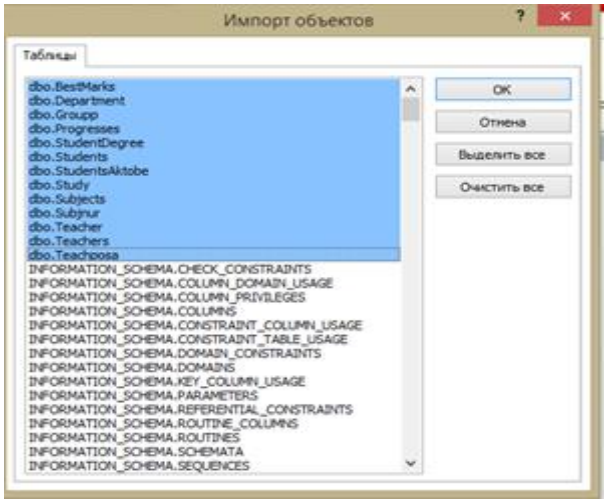

Figure E.12 – Selecting DB objects

So that’s all, and at the end there should be shown the result like in Figure 13.

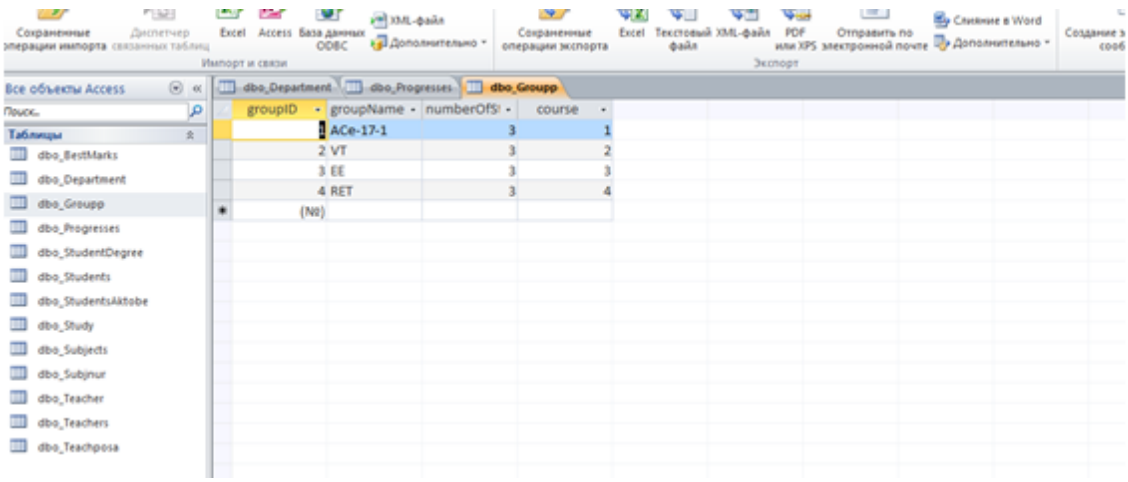

Figure E.13 – DB imported in MS Access

References

1. Ibrayeva L.K. Database design. Lecture notes for the specialty 5B070200 - "Automation and Control" - Almaty : AUPET, 2017. - 58p

2. Ramez Elmasr. Fundamentals of Database Systems. Sixth Edition - Great Britain: Pearson, 2014. - 1085p

3. Jan L. Harrington. Relational Database Design**.** Third Edition **-**Elsevier Inc., 2016.

4. T.Connolly, K.Begg. A Database Design. A Practical Approach to Design, Implementation, and Management. – www.booksites.net (A Companion Web site accompanies Database Systems).

5. Conceptual Data Modeling and Database Design: A Fully Algorithmic Approach, Volume 1: The Shortest Advisable Path - www.crcpress.com/

6. Relational Database Design and Implementation -2016, dl.acm.org/

Contents

	p
Introduction	3
1 Laboratory work №1. Development of a database project. Creation of a database……..…….	4
2 Laboratory work №2. Converting of a conceptual model into a relational one. Creation of database tables. Manipulation of data……..	13
3 Laboratory work №3. Data retrieval (Data Query Language)……..	23
4 Laboratory work №4. Using aggregate functions and special operators in queries……...	27
5 Laboratory work №5. Creating and using views and stored procedures……...	32
6 Laboratory work №6.Using triggers…….	36
7 Laboratory work №7. Preparing to create user applications……	39
8 Laboratory work №8. Development of the user interface……	43
9 Laboratory work №9. Development of a client application in Delphi……..	45
Appendices…….	51
References……	52

Composite plan 2018, pos.108

Lida Kuandykovna Ibrayeva

DATABASE DESIGN

Methodical guidelines to laboratory works for students of specialty
05070200 - Automation and Control

Editor L.Ya. Korobeinikova .
Specialist for standardization Moldabekova N.K.

Signed for printing __.__.__.
Circulation 50 copies
Volume 4.1 e.-p. sheets

Format60x84 1/16
Printing paper №1
Order . Price 2100 tg.

Copying Bureau of the Noncommercial Joint-Stock Company
"Almaty University of Power Engineering and Telecommunications"
050013 Almaty, 126, Baitursynov str.