Transportation Information Systems I.

Study-aid for practices in computer laboratory

Authors: Dr. Csaba Csiszár, Bálint Caesar, Bálint Csonka and Dávid Földes

Keywords: information, data, database, SQL, Microsoft Access

This study-aid summarizes the basic concepts, knowledge and terminology of the database management systems and Database Management. Commands and rules of SQL (‘words’ and ‘grammar’) are explained using a sample database.

During the lectures students learn about data modelling and the techniques of creating databases. They will be able to create own databases and to manage data according to different aspects.

Content

Foreword

Organizing transportation systems (organizations) and their processes, as well as the target-oriented operation require high quality information supply. The aim of the transportation information systems is the satisfaction of the demand for information, where efficiency depends on the managed database structure as resource. The information and the data have own value and therefore it is important, how we manage this value.

The most important 'soft' components of the transportation information systems are data. The relationships/connections between the components and processes are realized by streaming data. Database is a structured system of data which bridges the temporal gap between the recording and utilization processes (storage). In addition, the concept of the database has been enlarged with processing procedures in recent years. Accordingly, the knowledge related to the database management focuses preliminary on planning of storage structure and processing operations. As data map the components and processes of the operated system, therefore data modelling requires precise knowledge regarding structure and function of the transportation system and high level skills for abstraction.

Importance of this topic is enhanced by the so called information explosion that is remarkable recent years. The volume of the handled data is basically affected by the number of components taking part in the transportation, the attributes of the components and the sampling interval. Because of the two latter factors the amount of the handled data increased significantly recently, whereas only short time interval is tolerated for running of queries. Further challenges are merging and common handling of the information originating from different sources, with dissimilar format and meaning; what is basic precondition of the integration of the entire transportation system [1].

Courses regarding analyzing, planning transportation information systems including database planning have decades long tradition at the Department of Transport Technology and Economics, Faculty of Transportation Engineering and Vehicle Engineering of Budapest University of Technology and Economics [2]. Knowledge of relational data models and database planning is relatively stable. This knowledge is provided for the students in ever changing circumstances, especially with regards to software. Accordingly, it is focused mostly on the timeless knowledge, whereas students are getting familiar with the software too. Compilation and update of this study-aid according to didactical aspects are based on experiences gained in the last years [3].

The knowledge of the database management begins with the background and basics of the development. Main concepts and definitions, as well as their relationships are learnt on the lectures, what are deepened on practices through examples. Due to space limitations the study-aid contains only the framework of the learning material. The content may be changed year by year. Learning and practicing the commands take place in computer laboratory which requires the active participation of the audience. In order to make study easier the learning material contains practical exercises at the end of the chapters. Literature can be found at the end of the learning material which contains the most important sources. In case of demand for further details, students may gain adequate knowledge in the topic.

1. Basics of Database Management

Database: Database is a set of data elements, which are related to each other and stored together without overlapping and redundancy in a *complex, logical structure*. Beforehand database meant only a structured set of data, but nowadays a significant part of the processing operations are also included in the concept of database.

The structure of the database allows the access to the data and their modification. The aim of the database is reliable storage of the data and the fast retrieving. The database is not the same as the database management system, which is a computer program that provides basic functionalities including creation and maintenance of the databases, handling the system and the user processes.

Database planning begins with data modelling. Data model: mapping of the reality into data elements considering their relations, the circumstances and rules of the utilization - planning of the database structure.

Operations carried out with the database called Database Management, which aims at provision of information from the available and generated data (the volume of which is continuously growing along with the evolution of technology) with short retrieving time.

Storage of the data is considered on two levels:

- Logical level – what we store, which data and in which context;
- Physical level – how we store, how we access to the data on the physical storage system;

The tasks of computer data processing:

1. planning and implementing of the database structure,
   (defining the empty tables and their relations),
2. data input to the empty structure (to be user friendly), validation of the format and the content of the data,
3. update the data: new data entry, modification, deletion.
4. planning of the data query (we harmonize this with the data model)
   - simple query (e.g. searching, sorting),
   - complex query (e.g. classification and processing).

1.1. Evolution of Database Management Systems and software

The demand for fast, mechanized storage and retrieval of data has been arisen in the first half of the last century, when the first minicomputers were built, which used punched tape to ease the census. The first, so called, sequential files ('serial' access to the data according to the physical order, going through a lot of unnecessary data) were used in the late 1940s. The first non-sequential file system ('direct' access to the data using labels) has been developed at IBM in 1959. The current form of databases was emerging in the mid-1960s, when several new programming languages (e.g. Fortran, Basic) were developed. Shortly thereafter the first Database Management Systems (DBMSs) came into existence. The basics of the ‘retiary’ data

model was developed in 1961, and shortly after the hierarchical model appeared as well. Any entity in the 'retiary' model can be connected to any another entity (with different relationships too). For example, relationships between individuals (N:M relationship). The relationships use links and pointers in the database. It is disaggregated into 1: N connections. The basic of the hierarchical data model is the Parent-Child Relationship or Hierarchy which is illustrated in a graph (1: N, 1: 1 connections). The first 'databank' with computer network based accession was created by the IBM in 1965 and was called SABRE.

The conventional DBMSs collected and processed data which can be described by numbers and letters (alphanumeric characters). The advanced DBMSs, which were emerging later, allowed storing visual, sound, etc. data.

The aim of data modelling is to create an easily processable (by computer) data structure regarding the entities. The most common types of data models are: hierarchical, retiary, *relational*, object-oriented and multi-dimensional.

The retiary and hierarchical data models were used in the first commercial database systems in the late 1960s and early 1970s. Later these data models were excluded by the relational data models. The most common is the relational data model from the 1980s. The data are stored in the rows of the tables. In this model there are no pre-defined relationships between the single data elements. All data are stored with redundancy, which are required to create connections. Nowadays on the field of Database-Management new ways like object-orientated approach are getting widely extended. The role of Distributed Database Management Systems (DDBMSs) is also increasing significantly. The most commonly used database management software are summarized in Table 1.1.

Table 1.1. Most common DBMSs

Open source DBMSs Without any restrictions		**Free accessible DBMSs** Some DBMSs are restricted, but all of them are appropriate for studying purposes	
MySQL	MySQL (Linux/Windows)	Windows Server System	Microsoft SQL Server 2008 Express Edition Microsoft SQL Server 2008 Management Studio Express
PostgreSQL POWERED	PostgreSQL (Linux/Windows)	ORACLE	Oracle Database 11g Express Edition (Linux/Windows)
INGRES BUSINESS OPEN SOURCE	Ingres Community Edition		Microsoft Jet Database Engine 3
SQLite	SQLite (Linux/Windows)	IBM	Trial: Informix Dynamic Server Enterprise Edition V10.0
		MINI SQL	Mini SQL

At Faculty of Transportation Engineering and Vehicle Engineering in previous decades the learning material were dBase, Clipper, Oracle9.i, Access 2.0 and the advanced versions and the students learned the method of application development. Nowadays, Access DBMS is the most suitable for teaching purposes due to the wide availability. Self-sufficient work on database planning and development was always important part of our education, therefore *students prepare data model and then an application in a selected topic*, which is aided by consultations. (Selection of topic of homework).

1.2. Characteristics of SQL language

Evolvement of different DBMSs required development of a standard query language. As a consequence, the *SQL (Structured Query Language)* was created using similar 'linguistic elements' as in everyday speech and thinking. It was standardized in 1986.

SQL= (command) words + grammar (regulations regarding use of words).

SQL makes creation and query of relational database as well as retrieve of information stored in the 'relations' possible. There is no need to specify how to access the data, but only the properties of the necessary information should be determined.

SQL language contains the following elements:

- objects,
- NULL value,
- expressions,
- operations,
- commands,
- syntactical elements,
- procedures.

Formal rules of commands is based on using a unified coding system (Table 1.2). For instance:

- Uppercase: commands, logical operators, functions, etc.
- Lowercase: objects, column, etc.

Table 1.2. Most important formal rules of SQL commands

Indication	**Meaning**
UPPERCASE	Required, cannot be left out and must be typed correctly.
italic lowercase	User defined variables.
[]	List of optional items.
…	Repetition of the identical elements.
<>	Mandatory elements should be listed.
{}	At least one item must be listed.
|	Choosing between two or more possibilities.

Input of commands is facilitated by DBMSs in different ways. They recognize the mistyping and modify them according to syntactical rules.

The aim of creation of databases is to gain as many information as possible. Accordingly, the most important command is SELECT.

In case of large and network-based databases, setting user groups' permissions (e.g. commands GRANT, REVOKE) and logging their operations are especially important.

The basic SQL commands are summarized in Table 1.3.

Table 1.3. Basic SQL commands

DDL Data Definition Language	creates an object	CREATE
	modifies the structure of an existing object	ALTER
	deletes all data and structure of a table	DROP
DML Data Manipulation Language	adds records	INSERT INTO
	modifies records	UPDATE
	removes records	DELETE FROM
	query	SELECT

SQL is not procedure oriented, focusing on what we want receive, not how. The syntax is strict.

Languages involving SQL: e.g. Pascal, C, Cobol, dBase, Access

PL/SQL=Procedural Language – allows to prepare procedures.

- declare variables,
- assign values to variables,
- conditional structures,
- loop,
- functions, procedures.

1.3. Features of Microsoft Access

MS ACCESS= relational database management system, which has a user friendly graphical interface.

Applied object types and their relations are summarized in Fig. 1.1. Detailed description of object types is available in Table 1.4.

In many cases the user operations are supported by built-in applications, called wizards. Newer and newer versions are published (compatibility issues), but the knowledge in the background (relational data model) is unchanged.

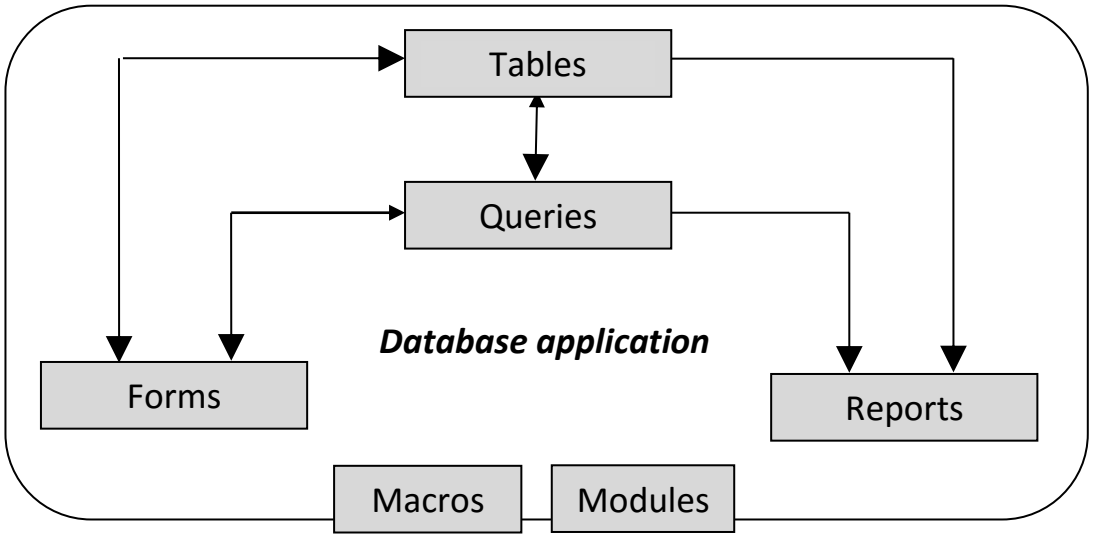


Fig. 1.1: Relations of object types in Access

Table 1.4. Properties of objects of the Access

Tables	• similar to the tables in their appearance • redundancy, duplication should be avoided (normalization)
Queries	• based on existing table(s) and/or query(ies) • applied for different tasks • two basic types: select (collect and visualize data) and action queries (modify records in a table) • data stored in different tables are listed together in one object • queries and reports are usually based on these source of records
Forms	• data entry and data visualization on screen • buttons can be placed on it (menu system can be built)
Reports	• data visualization and summing/grouping in printing format • the current data are reported • it can be sent by e-mail
Macros	• simple programming language • automation of event control is possible; incorporation of often repeated processes into one command, execution of operations is conditional
Modules	• new functions can be added to the database • own 'programs', procedures, functions, etc. compiled in VBA-visual basic language

Creating a new database, sample databases are not to be used. Only one file is created, which is portable. Fig. 2. illustrates a typical screenshot. The object type can be selected at the left side of the window. In general there are three possibilities to create database-objects:

- *Automatically*: Access automatically creates a predefined version of the object (rarely used).

- *Wizard*: Wizard guides you through the planning process.
- *Design view*: only manual planning; all the system setting opportunities are available.

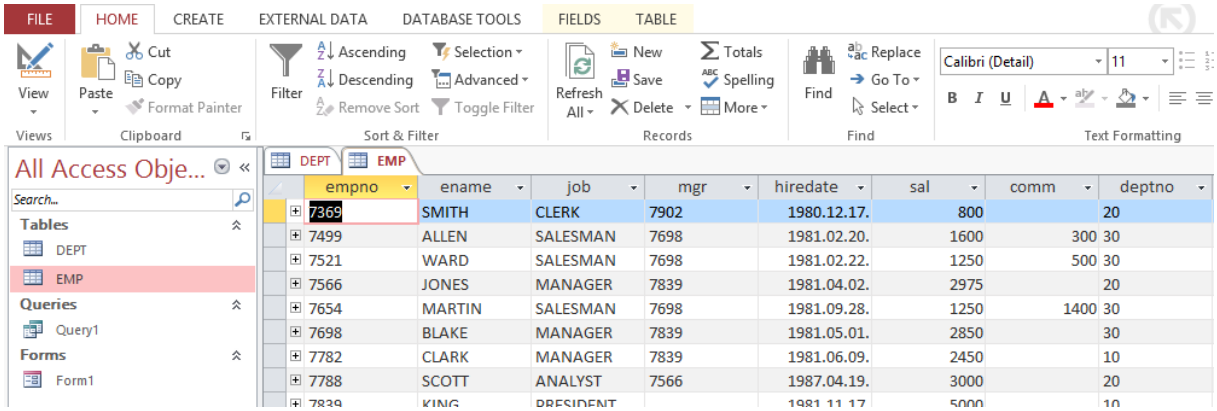


Fig. 1.2: A typical screenshot from the Access

The 'engine' of the Queries is the SQL language, which defines the operations. One SQL command is assigned to each query. Simple queries can be created in Design view using the 'grid'; whereas complex queries can be created by editing the SQL commands (Fig. 1.3.).

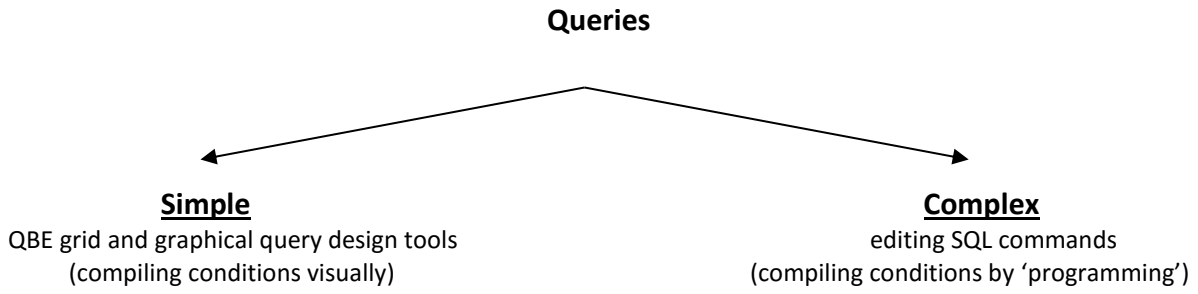


Fig. 1.3: Creation of query types

1.4. Types of data

Different versions of Access allow using app. 10 typical data types (Table 1.5.). There may be differences according to the versions. The data type of a field in a table (the attribute of the entity) should be set at creating the table (Design View).

The most simple data validation is also realized by setting the data type, as users can input only the data corresponding to the data type.

Search Wizard: we can select value from another table (foreign key), but we can also declare the set of applicable data. It is usually applied in case of foreign key, when we select data from the table on the '1' side.

Table 1.5. Types of data in Access

Types of data		Stored	Restrictions, requirements
character	text	Alphanumerical data (text and numbers) – fix length	• max. 255 character • data input with input mask
	memo	Alphanumerical data (text and numbers) – variable length	• searching, filtering, sorting is *NOT POSSIBLE in the case of memo* • manual data input with max. 65535 character
numerical data	number	Numerical data (when we use them in aritmetic operations; otherwise character type)	• adjust the size of the filed to 1, 2, 4, 8 or 16 byte • byte (0…255), integer (0…65535), long integer (not decimals) • single, double (decimals)
data derived from numerical data	date and time	In different formats	• as integer with double precision in 8 byte • data input with input mask
	currency	For specific purposes	• 8 byte numbers; 4 digits for decimals
	auto number (counter)	Unique values in ascending order	• 4-byte (long) integer • usually the primary key
	yes/no (logical)	Logical (Yes or No) data	• -1 → Yes • 0 → No
OLE-object		Picture, sound, Excel worksheet, Word document, and others from Office and Windows-based programs	• max. 2 GB data • one OLE service is required (specific program that supports the type of the file)
hyperlink		Web pages	• max. 1 GB data • links to files, webpages, on own computer, local network, etc.
attachment		Any supported file format	• images, spreadsheets, documents, graphs and other supported file types can be attached to the database records

Two types of the stored data:

- simple data storage unit: variable,
- complex data storage unit: record.

Data elements are logically corresponding in the records. The association principle is the entity, whose attributes are stored in the record. The entity can be a person, an object or a concept (e.g. serving one station by public transportation). The entities belong to each other (into one entity type), that are mapped by the same attributes.

2. Normalization

Database planning begins with identification of data to be stored. It is followed by *normalization* (operations for disaggregation of tables): when the 'appropriate' structure is created. Through the normalization the data to be stored are grouped into tables connected to each other, so that the database requires smaller storage capacity which is more transparent and complying with the rules of normalization. The normalization consists of several steps. As a result of each step, the structure of the database meets the conditions of one level higher normal form (NF). The NF levels are cumulative, namely, in order to meet higher level conditions, all the conditions of lower levels have to be met (stricter rules).

The successive steps of normalization process are introduced through examples.

2.1 Steps of normalization – Example 1

In our example car accessories are registered. Creation of a database structure begins with definition of necessary data and assigning them to tables (Table 2.1.):

- vehicle identification number (V_ID),
- product,
- type,
- door numbers,
- accessory name,
- accessory part number.

Table 2.1. 0NF

W0B578547	Opel	Astra	5	Airbag	10125	Air condition	98542		
GTH4724782	Fabia	Skoda	five	Airbag	10125	Air condition	32545	Security alert, CD player	10254, 65020
CCP4672134	Lada	Samara	3	Brake light	00001				
CCP4672134	Lada	Samara	3	Brake light	00001				

1 NF (first normal form)

Conditions of 1st NF are not met because:

- number and order of columns are not equal in each row,
- attribute of door numbers has not numerical value in each row,
- some attributes have more than one value,
- there are two equal lines (redundancy), thus it is impossible to create primary key.

The proper structure for 1NF is shown in table 2.2. One entity (record) corresponds to a specific accessory built into one vehicle. Because of the repetition of rows unambiguous identification can be realised only with multiple key. For instance, one record can be identified with V_ID and accessory part number. In the multiple primary key the accessory name is not

appropriate because the accessory name does not imply the part number, but on the contrary, the accessory part number implies its name. Namely in 1NF, the secondary attributes functionally depend on the primary key.

In the case, if we cannot define any multiple key or it had too many fields, unique identifier is used (e.g.: _ID as suffix in the name – counter type).

Table 2.2. 1NF

V_ID	Product	Type	Door numbers	Accessory name	Accessory Part Nr.
W0B578547	Opel	Astra	5	Airbag	10125
W0B578547	Opel	Astra	5	Air condition	98542
GTH4724782	Skoda	Fabia	5	Airbag	10125
GTH4724782	Skoda	Fabia	5	Air condition	32545
GTH4724782	Skoda	Fabia	5	Security alert	10254
GTH4724782	Skoda	Fabia	5	CD player	65020
CCP4672134	Lada	Samara	3	Brake light	00001

2 NF (second normal form)

Conditions of 2nd NF are not met because:

- there is a field which depends on only one part of the multiple primary key (e.g.: V_ID per se identifies the product).

In order to create the 2NF structure, it is necessary to decide: which components the non-key (secondary) attributes depend on? (Fig 2.1.)

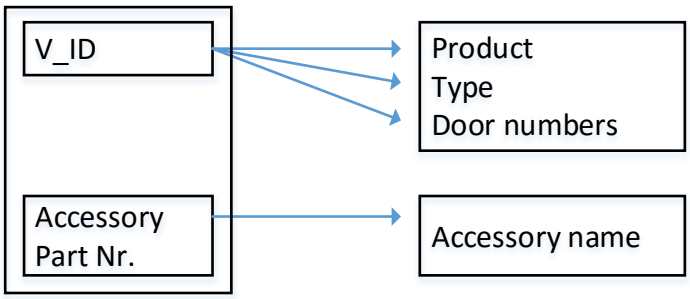
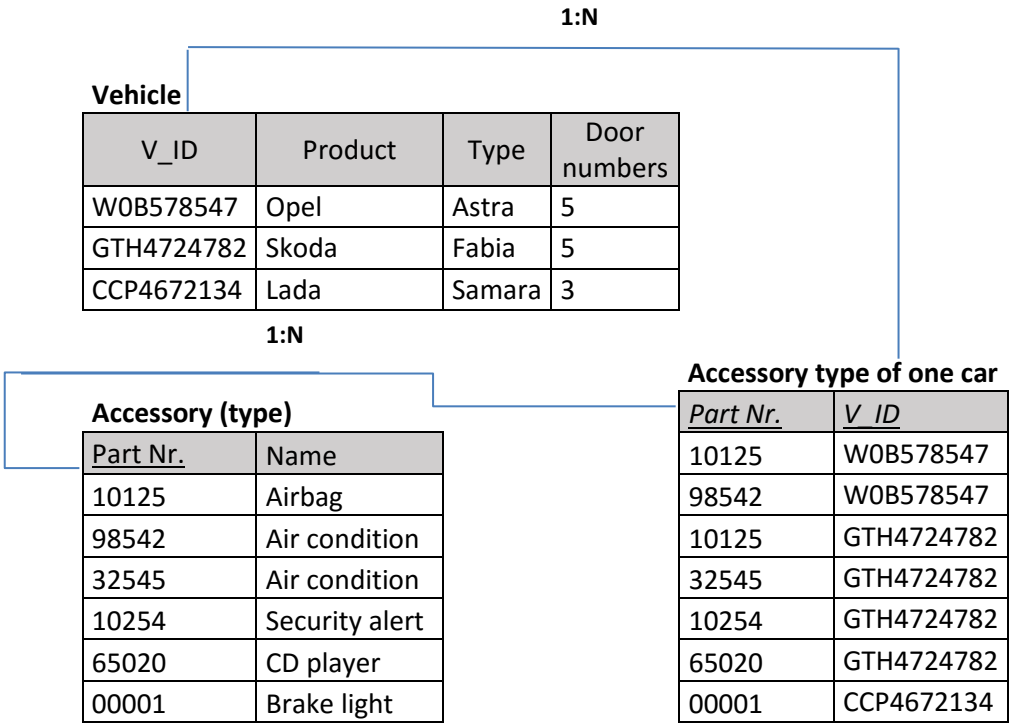


Fig. 2.1. Functional dependences between the attributes

The V_ID attribute identifies the product, the type and the door numbers of the vehicle. According to the disaggregation rules of 2NF (all fields depend on the multiple key or the key consists of one single field) the attributes are assigned to 3 different tables (Fig 2.2.), where the entity types can be identified yet. Based on the type of relationships between tables the primary and the foreign keys can be identified. The primary key consists of one field in the **Car** and the **Accessory (type)** table, whereas it consists of two fields in the **Accessory type of one car** table (the latter two fields are also foreign keys). At the more (N) side table (**Accessory type of one car**) the two fields constitute the multiple key together; this table is called as connector table.



V_ID	Product	Type	Door numbers
W0B578547	Opel	Astra	5
GTH4724782	Skoda	Fabia	5
CCP4672134	Lada	Samara	3

Part Nr.	Name
10125	Airbag
98542	Air condition
32545	Air condition
10254	Security alert
65020	CD player
00001	Brake light

Part Nr.	*V_ID*
10125	W0B578547
98542	W0B578547
10125	GTH4724782
32545	GTH4724782
10254	GTH4724782
65020	GTH4724782
00001	CCP4672134

Legend: primary key
foreign key

Fig. 2.2. 2NF

It can be observed, that the number of data elements has been decreased: 38 instead of 42, that is the level of redundancy became lower. (Weak logical redundancy is necessary to build relationships.)

3 NF

Conditions of 3rd NF are not met because:

- functional dependency is not only originated from the primary key, that is functional dependency between the secondary attributes also exists (transitive dependency).

In the 2NF **Car** table all the secondary attributes depend on the V_ID. However, the field Type also implies Product; it means there is transitive dependency between two secondary attributes. For instance, if this table contained more than one car with Opel Astra type, in each case the product and the type should be recorded (redundancy). Accordingly, the conditions of 3NF can be met by further disaggregation of the table. The result is shown in Fig. 2.3.

In the literature 4NF and 5NF are also introduced, but usually it is enough if database meets the requirements of the 3NF.

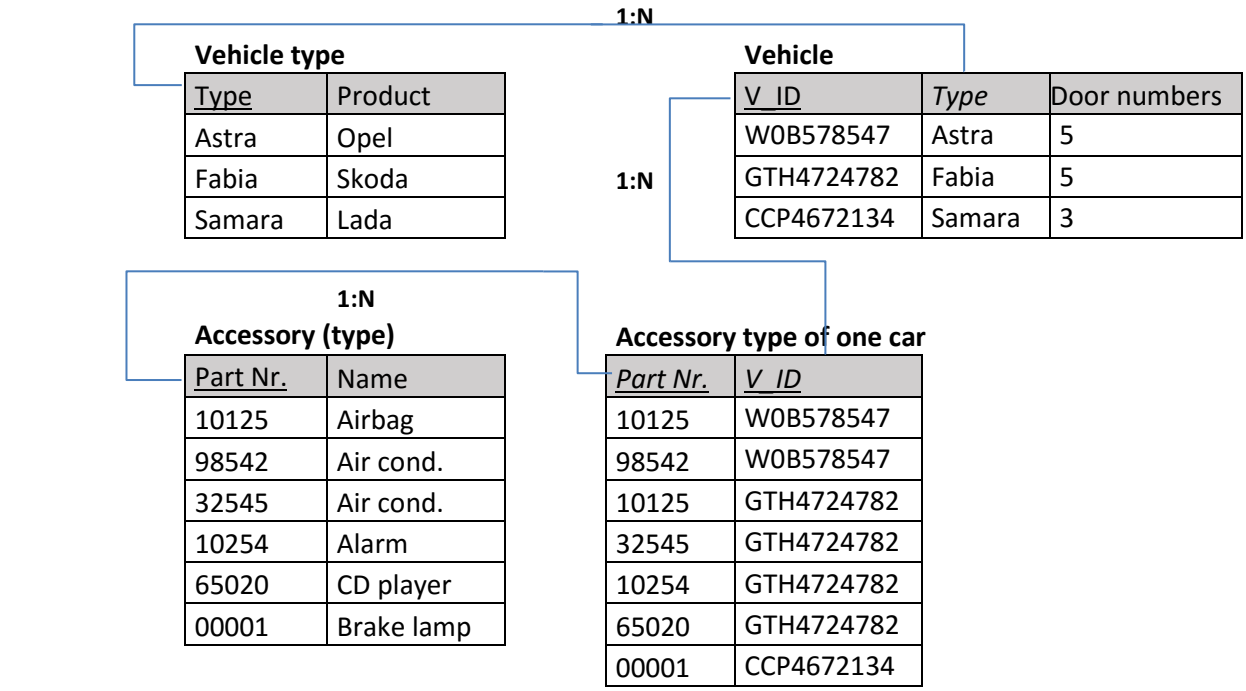


Fig. 2.3. 3NF

2.2 Steps of normalization – Example 2

In this example invoices are registered. The following data elements (attributes) are stored:

- INVOICE NR
- DATE
- CUSTOMER'S NAME
- CUSTOMER'S ADDRESS

- PRODUCT CODE
- PRODUCT NAME
- QUANTITY
- UNIT PRICE
- PRICE (values calculated from other fields are not stored)

- TOTAL (values calculated from other fields are not stored)

The proper structure according to the normalization is shown in Fig. 2.4., which is a simplified representation (name of the tables and the attributes, indication of the relationships with keys).

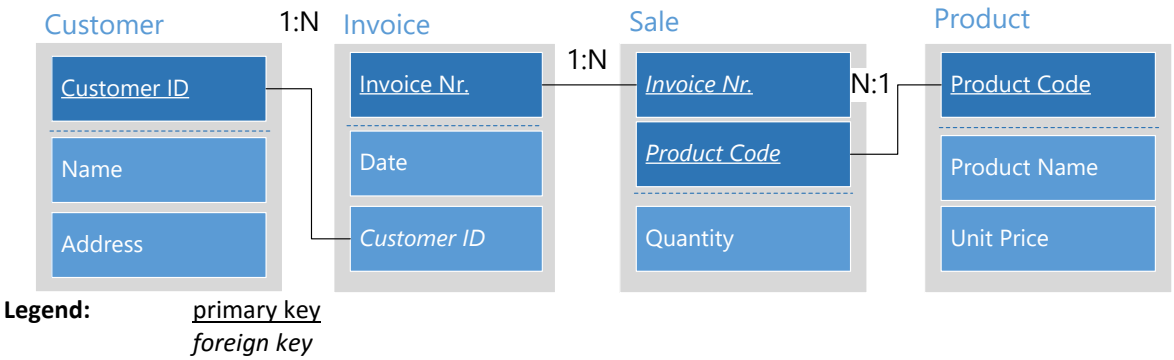


Fig. 2.4. Table structure in 3NF (example of invoice)

2.3. **Disaggregation of N : M relationships** – using connector table (conceptual entity type)

During creating table structure of the database, the N:M type, many-to-many relationships should be avoided. This kind of relationships can be disaggregated with using a third intermediate connector table. The new table mostly associated by a conceptual entity type. The process of disaggregation is presented by two examples. The result is two 1:N relationships being symmetric to each other.

Example 1: medical treatment (doctors and patients) registration – Fig. 2.5.

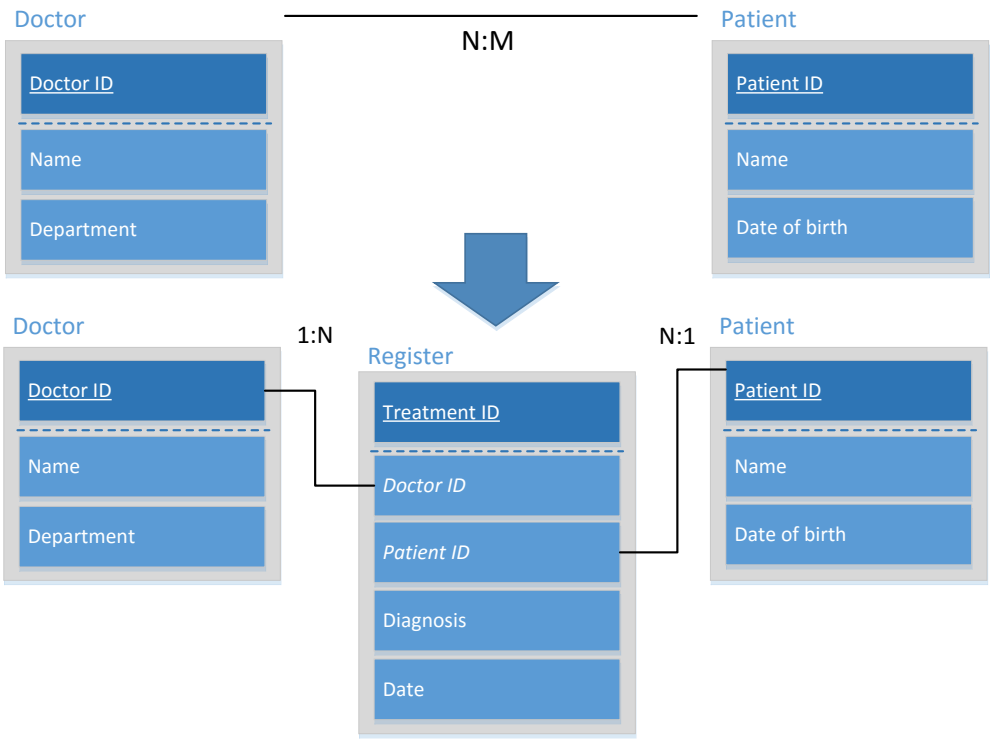
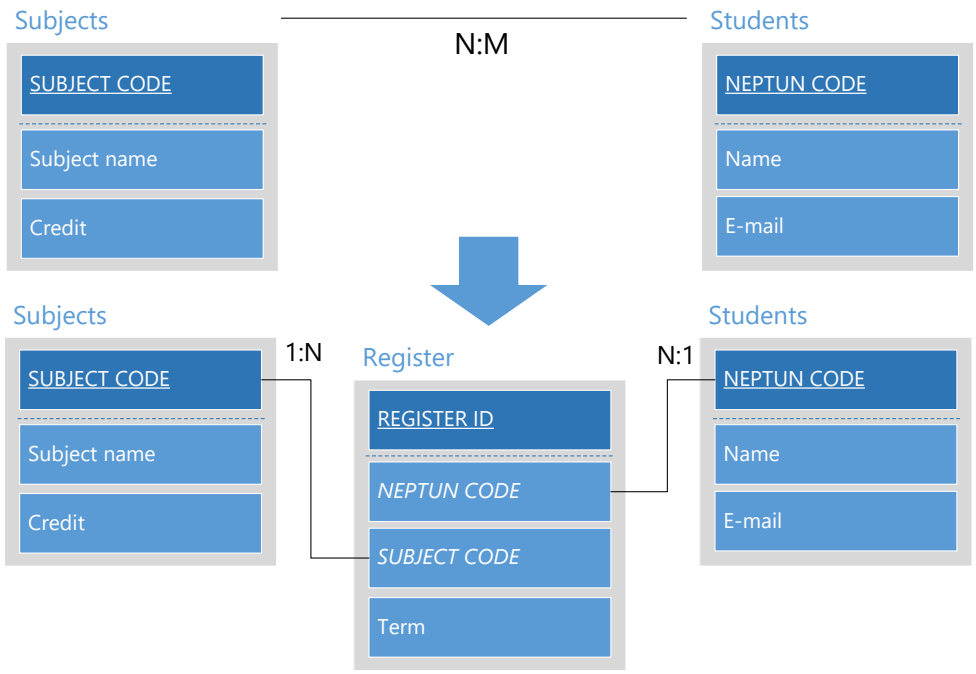


Fig. 2.5. Disaggregation of N:M relationships (example of medical treatment)

Example 2: Registration of students and subjects (the subjects may be offered in different terms) – Fig. 2.6.



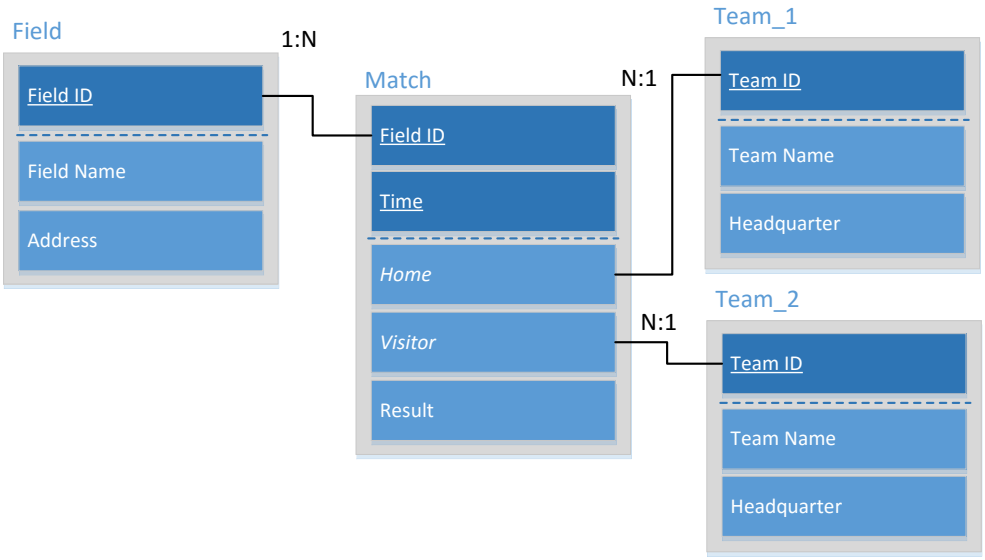
Legend: primary key
foreign key

Fig. 2.6. Disaggregation of N:M relationships (example of subject registration)

2.4. Building relationships using multiplied tables

In this example the most important data of football matches are registered. We don't do any calculation with the number of goals. The proper structure according to the normalization is shown in Fig. 2.7., which is a simplified representation (name of the tables and the attributes, indication of the relationships with keys).

The curiosity of the structure that the Team table is joined twice to the Match table; as the home team (Team_1) and the visitor team (Team_2); whereas the same data about the home and the visitor team are stored, so in reality only one table exists. (The teams can change their 'role', because once they are home teams, other times visiting teams.) This problem can be solved with using so called *virtual tables*, which are created by repeated adding in the Access (e.g.: in the field above the query planning grid).



Legend: primary key
foreign key

Fig. 2.7. Building relationships using multiplied tables (example of football matches)

2.5. Structure of sample database – relationship within table

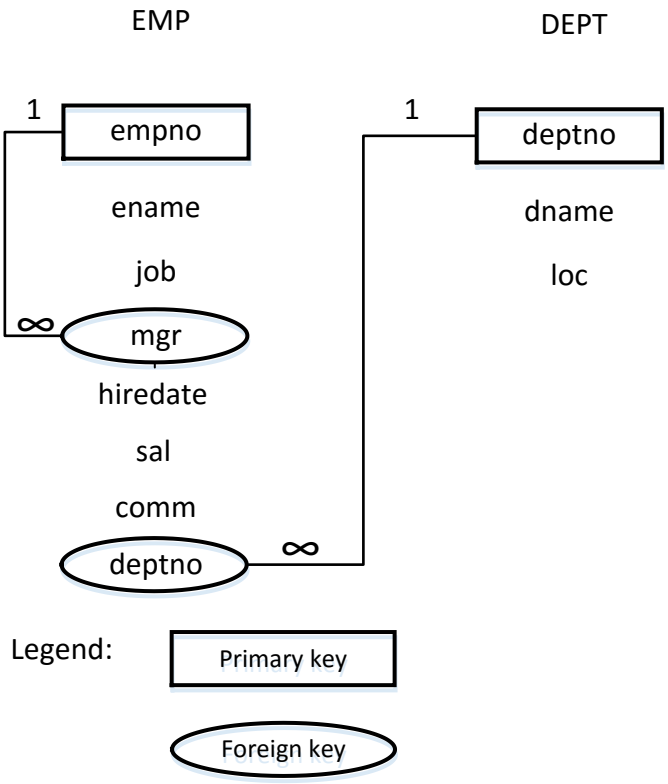


Fig. 2.8. Structure of the sample database

In this example data of employees and departments are registered in two tables. The EMP table contains the data of employees (e.g. ID, name, job, etc.), whereas in the DEPT table we can find the data of department (ID, name and location). Between the two tables the connection is realized by department number (deptno). The primary key in the EMP table is the Emloyee_ID (empno), but it appears also in the mgr field in the same table as a foreign key. The primary key in the DEPT table is the department number (deptno), which is a foreign key in the EMP table. Table 2.3 contains the short description of fields.

Table 2.3. Description of fieldnames of the sample database

Field name	Meaning	Description
empno	Employee ID	Each employee has an individual identifier
ename	Name of the employee	Surname of the employee
job	Job	Name of the job (e.g. CLERK)
mgr	Manager ID	Direct supervisors' ID of the employee
hiredate	Hire date	yyyy/mm/dd is the format
sal	Salary	Monthly salary (it is filled for everyone)
comm	Commission	It is not filled for everyone – NULL value (if it is filled, it can be also 0).
deptno	Department number	Each department has an individual identifier
dname	Department name	Name of the department (e.g. SALES)
loc	Location	Name of city, where the department is located (e.g. CHICAGO)

The curiosity of the structure is the ***relationship within table*** between empno and mgr fields; that means the empno value of the manager is stored in the mgr field of an employee. This structure is especially suitable in case of ***hierarchical organizations*** to map the relationships between the employees. This solution is regarded as a traditional relationship between the original and new 'virtual' EMP table. The theoretical application of the virtual table is similar to previous example. One employee can be both manager (he has a subordinate) and subordinate (he has a manager) at the same time. The CEO (ename: KING, job: PRESIDENT) is the only exception because he has no boss, so the value of his mgr field is NULL. In case of the other employees the value of the mgr field is filled. If someone doesn't have any employee (he is on the lowest level of the hierarchy) his empno doesn't appear in another employee's mgr field.

Data input order: as during data input always records of '1' side table are filled at first, in this structure the employee (manager/boss) standing on higher level of the hierarchy is registered first (or we return later to fill the mgr field as foreign key).

3. Create a sample database

The sample database that was shown in chapter 2.5. is created using **Microsoft Access** software, which supports the SQL programming language. The software can be downloaded for students from the szoftver.eik.bme.hu webpage for free as the part of the whole Office package.

When creating the sample database, open the Access and choose 'Blank desktop database' type, name the file and select the destination folder. Then the program creates a file with .accdb extension. During opened file status the program automatically prepares a temporary file with .laccdb extension using the same file name. Before copy close the entire application, and copy only the .accdb file. It should be noted that there are some differences despite of standardization in SQL language used by different software. Before using the program we should check whether the proper ANSI 92 type of syntax is used. (File menu -> Options -> Object designers -> SQL Server Compatible Syntax (ANSI 92) -> This database: ✓) Furthermore change of the display language is possible: File menu -> Options -> Language.

3.1. Creating the table structure

After the creation of a new database, a new table can be created by clicking on the „Create" tab and then „Table Design" icon (Fig.3.1.). A table is to be created, if all the foreign keys have already been created, as primary keys, in the related existing tables. Preference rule: using connected tables, the '1' side table should be created always as first and then the primary key should be set in the 'many' side table as a foreign key. During set up the datatype of the foreign key, use of the Lookup Wizard is a convenient opportunity, which creates the proper relationships between the tables automatically. If we don't use the Lookup Wizard, the relationships must be set up manually clicking on the 'Database Tools' tab and then the 'Relationships' icon. The preference rule should be applied during data input too; namely, first enter the data into the table on '1' side and only thereafter into table on 'many' side.

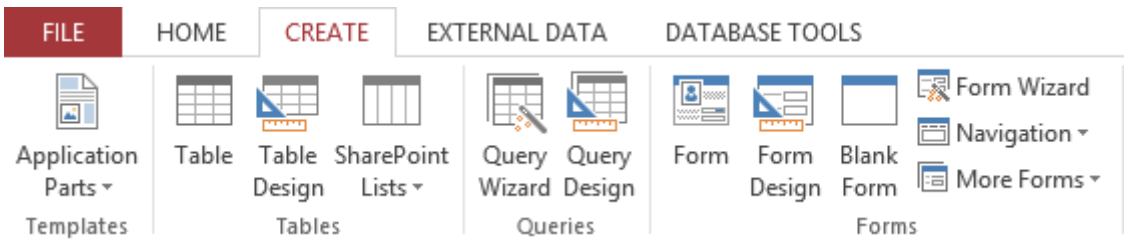


Fig. 3.1. Create and open a new table in design view

Table 3.1. Datatypes of the fields in sample database

Table	Field name	Datatype	Further options
EMP	empno	Short Text	Input mask: 0000
	ename	Short Text	NOT NULL; maximum 30 character
	job	Short Text	Maximum 30 character
	mgr	Short Text	Using Lookup Wizard
	hiredate	Date/Time	Short date
	sal	Number	>0
	comm	Number	
	deptno	Short Text	Lookup Wizard
DEPT	deptno	Short Text	Input mask: 00
	dname	Short Text	NOT NULL; maximum 30 character
	loc	Short Text	Alternative header: Location

Legend: primary key
foreign key

In the sample database the listed tables and fields (see Table 3.1.) should be created. Setting the primary key: select the appropriate field or fields (depending on whether the key is multiple) in design view and click on the 'Primary Key' icon on 'Design' page. The primary key can be deleted similarly from the row of the field. The most important attribute for each field is the datatype, which determines, what kind of data can be stored therein. Datatype of fields determines further important attributes:

- use of the fields in expressions,
- field size,
- possibility of indexing,
- possible formats.

The created structure of the database with relationships can be seen by clicking on the "Database Tools" tab then "Relationship" icon (Fig. 3.2.).

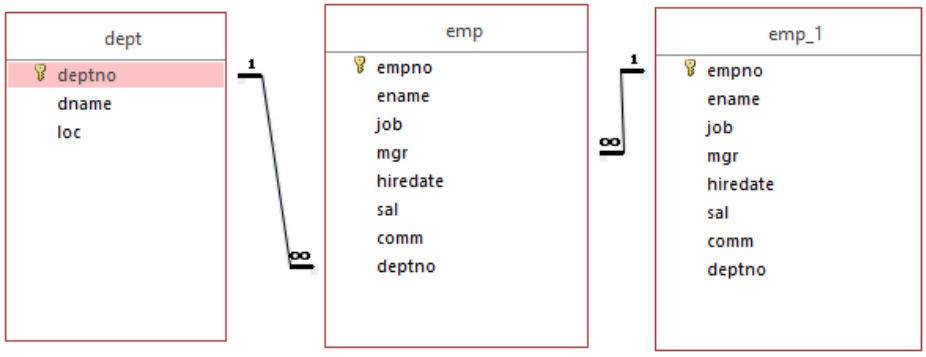


Fig. 3.2. Structure of the sample database in Access

There is no need in the Access to create the emp_1 table again in order to build the inner relationship. The emp_1 table appears as a virtual table to illustrate the inner relationship (using multiplied tables).

3.2. Input of sample data

Input of new data (records) into the certain table is possible after opening the Datasheet view. Data contained in the sample database are listed in the table 3.2. and 3.3.

Table 3.2. Records of DEPT table

DEPT		
deptno	dname	loc
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

Table 3.3. Records of EMP table

EMP							
empno	ename	job	mgr	hiredate	sal	comm	deptno
7369	SMITH	CLERK	7902	1980. 12. 17.	800		20
7499	ALLEN	SALESMAN	7698	1981. 02. 20.	1600	300	30
7521	WARD	SALESMAN	7698	1981. 02. 22.	1250	500	30
7566	JONES	MANAGER	7839	1981. 04. 02.	2975		20
7654	MARTIN	SALESMAN	7698	1981. 09. 28.	1250	1400	30
7698	BLAKE	MANAGER	7839	1981. 05. 01.	2850		30
7782	CLARK	MANAGER	7839	1981. 06. 09.	2450		10
7788	SCOTT	ANALYST	7566	1987. 04. 19.	3000		20
7839	KING	PRESIDENT		1981. 11. 17.	5000		10
7844	TURNER	SALESMAN	7698	1981. 09. 08.	1500	0	30
7876	ADAMS	CLERK	7788	1987. 05. 23.	1100		20
7900	JAMES	CLERK	7698	1981. 12. 03.	950		30
7902	FORD	ANALYST	7566	1981. 12. 03.	3000		20
7934	MILLER	CLERK	7782	1982. 01. 23.	1300		10

In Table View the data input is aided by the TAB button on the keyboard, which switches to the next field or the copy-paste keyboard shortcut. Input of huge amount of digitalized data is possible by clicking on the ‘External Data’ tab then ‘Import &Link’ group, where beside other possibilities excel is supported too.

If a new record is inputted in Datasheet View, during the data input a pen icon is visible at the first column. The icon remains there during edition, and then disappears only if the record has been saved. The practical advantage of the icon is that a user working in network knows when the new record is available for the other users (after the pen icon disappeared).

4. Types of SQL commands, SELECT command

The commands used in the SQL language can be classified in the following groups:

- **DDL** (Data Definition Language): commands to define the database (table) structure (logical structure).
- **DML** (Data Manipulation Language): commands to manage/maintain data
- **DCL** (Data Control Language): commands to handle rights/privileges in database
- **TCL** (Transaction Control Language): commands to manage the changes made by DML commands.

During the seminars only the first two groups, the DDL and DML commands are discussed.

One command is to be executed in Access as follows (Fig. 4.1.): type the expression into the SQL view of a query, then run it.

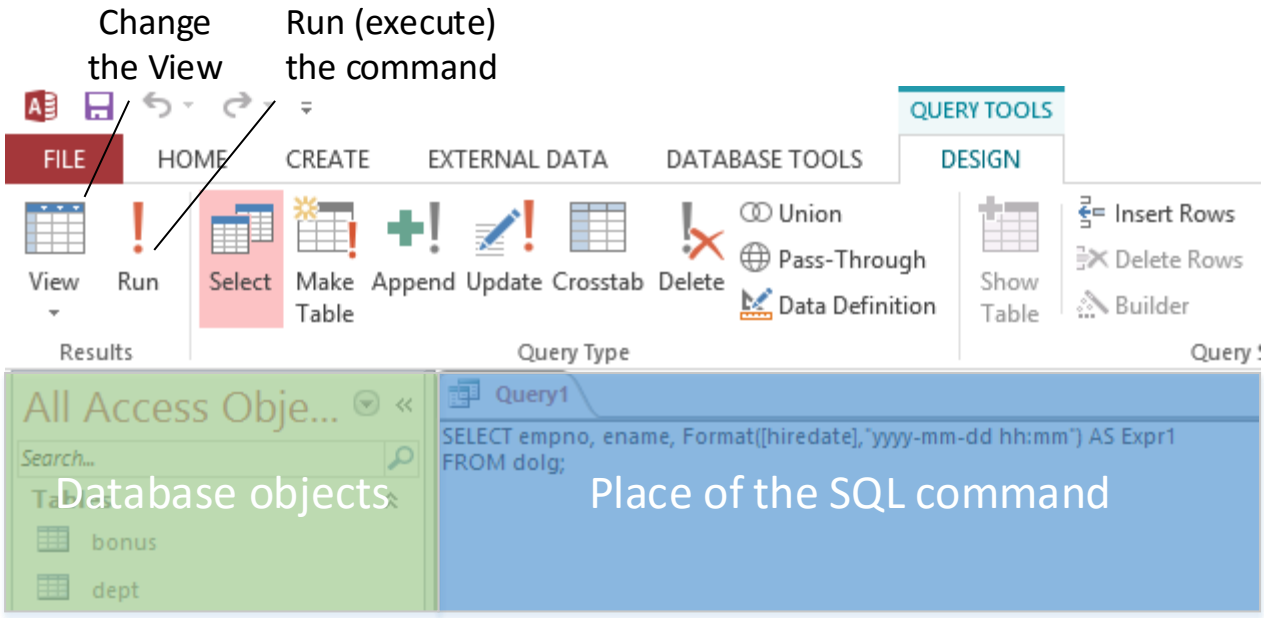


Fig. 4.1. Execute SQL commands in Access

4.1. DDL commands

Data definition language, which contains the commands to create and maintain the data structure (tables, indexes, constraints, etc.). The most important DDL commands:

- **CREATE**: creation of structure of a new object (e.g. table). In case of tables, its structure is defined, taking also the relationships into account.
- **ALTER**: modification of structure of an existing object. For instance a new column can be added to the table.
- **DROP**: *complete* deletion of an object (both the structure and the content is deleted).

Other DDL commands are also available in other database management systems, like TRUNCATE (removes all records from the table and 'reset', the counter starts from 1 in case of a new record) or RENAME. However, the most used commands in Access are listed above. As Access is especially user-friendly, many operations can be performed in other ways, without use of the DDL commands.

4.2. DML commands

DML commands are necessary to handle the data stored in the database. Database structure created by DDL commands is filled in by data using DML commands. DML commands can be used in order to modify the content too. The most important ones:

- **INSERT INTO**: insert data (record) into a table.
- **UPDATE**: update/modify existing data (record or records).
- **DELETE FROM**: delete one/more/all records, but the structure remains.
- **SELECT**: retrieve data from the database (queries with raw or processed data).

4.3. Structure of SELECT command

Using SELECT command record(s) can be shown from one or more tables (queries) or groups can be created. The structure of the SELECT command:

SELECT [DISTINCT] *select-list*
[FROM <*table name, query name*>
WHERE <*row-condition*>
GROUP BY <*grouping aspects*>
HAVING <*group-condition*>
ORDER BY <*ordering aspects*>];

The sequence of command clauses within SELECT command is always bound; however, only the SELECT clause is inevitable for execution. Explanation of the clauses:

- **SELECT**: obligatory, determining the content of the select-list (column names, expressions).
- **FROM**: the sources of data (name of tables and queries) are identified.
- **WHERE**: the records are selected, that meet the row-condition. If several row-conditions are applied, they are concatenated by logical operators.
- **GROUP BY**: records selected by row-condition (if there is no row-condition, all the records of the table) are grouped according to the grouping aspects. The entities (records), where the content of grouping column is the same, are assigned to one group.

- **HAVING**: only if GROUP BY is applied. The groups are listed, where groups meet the group-condition. If several group-conditions are applied, they are concatenated by logical operators.
- **ORDER BY**: the results (records, groups) are listed according to the sorting aspects, considering the ordering directions (ascending or descending).

4.4. SELECT list, alternative column name

The **select-list** may contain:

- *: all columns are displayed.
- column list: column1, column2, …
- calculated expression: 3*column1 (if the column1 has number datatype), 47/3, etc.
- character-chain as constant: e.g. ‘HUF’. Beware: both quotation marks should be on the top in SQL.
- concatenation: e.g. column1&column2. Two columns are ‘merged’.
- alternative column header: column1 as [alternative column header]. Alternative column header is displayed only in the result of the query.

Examples:

- SELECT * FROM EMP;
- SELECT empno, ename FROM EMP;
- SELECT 6*30;
- SELECT 6*sal FROM EMP;
- SELECT 'TOM SELLECK';
- SELECT empno & 'identification' FROM EMP;
- SELECT empno as [identification] FROM EMP;

5. SELECT command

Structure of the SELECT command:

SELECT [DISTINCT] *<field list>*
[FROM *<table, query name>*
WHERE *<row-condition>*
GROUP BY *< grouping aspects >*
HAVING *< group-condition >*
ORDER BY *<column1 [ASC | DESC][, column2 [ASC | DESC]][, ...]] >*];

5.1. ORDER BY clause

ORDER BY: records/groups are sorted according to the given column or expression in ascending or descending order. The default order is ascending (A to Z, 0 to 9).

Type of orders:

- **ascending or alphabetical order**: ORDER BY column1 [asc];
- **descending or inverse alphabetical order**: ORDER BY column1 desc;
- **mixed**: ORDER BY column1 desc, column2;

Several columns can be applied in ORDER BY clause. Records having the same value in the first column are sorted by the value in the second column. All the further columns are considered, if the records contain the same values in the previous columns. The column (or expression) used after ORDER BY is not necessarily included in the select list.

5.2. Anomaly of missing data (NULL value)

Problem with NULL value: if a field is empty (not filled out), the value of the field is NULL. The content of information: undefined (unknown). The result of the operations with fields containing NULL value can also be NULL value. This may cause problems. For instance, the result of sum is NULL, if any of the components is NULL. To obtain practical result, the NULL value can be replaced by the built-in *iif* function. The structure of the *iif* function:

iif(expression; if true; if not true)

For example, an undefined NULL value can be converted to a definite 0 for calculations. But the conversion causes distortion in the information:

iif(comm is NULL; 0; comm)

5.3. Omit duplicated data (DISTINCT)

DISTINCT: duplicated rows are displayed only once. For example:

SELECT job FROM emp;	SELECT DISTINCT job FROM emp;
CLERK	ANALYST
SALESMAN	CLERK
SALESMAN	MANAGER
MANAGER	PRESIDENT
SALESMAN	SALESMAN
MANAGER	
MANAGER	
ANALYST	
PRESIDENT	
SALESMAN	
CLERK	
CLERK	
ANALYST	
CLERK	

5.4. WHERE clause

Row-condition: specifies filter conditions that determine the rows that the query returns.

Elements and their orders after WHERE clause (at simple condition):

I. expression (*what do we compare?*)

II. comparison operator (*what is the comparison like?*)

III. expression, value, list of value, interval, schema (*what we compare to?*)

Table 5.1 contains the comparison operators used in row-conditions. Both sides of operators must be the same type of data.

Table 5.1. Comparison operators

Operator	Purpose
=	Equal, it can be used with character data type too (expressions on both sides must be exactly the same – upper and lower case is important)
<>	Not equal, can be also used with character data type. Note: in some versions of SQL this operator may be written as !=
<	Less than
>	Greater than
>=	Greater than or equal
<=	Less than or equal
[NOT] BETWEEN	Between two values, equality is allowed (closed interval)
[NOT] IN	Equals to any value of the list (IN statement creates a series of OR statements)
[NOT] ALIKE	Matches the looked for pattern (fits to the template)*
=[]	Parametric query

*: In SQL ANSI92 language only ALIKE operator can be used. Access applies SQL ANSI89 language as default, which recognizes both LIKE and ALIKE operators, however wildcards are different.

Table 5.2 contains wild-card characters used for ALIKE operator.

Table 5.2. Wild-card characters

ANSI89	ANSI92	Wildcard description
? or _	_	A substitute for a single character
#	[1-9]	A substitute for a single digit
*	%	A substitute for zero or several characters
[]	[]	Any character, what is contained by the given range E.g. [a-m] or [1-5] or [abcdef]
[!]	[!]	Any character, what is not contained by the given range E.g. [!a-m]

The date datatype is stored as a number by Access. This number indicates, how many days went by since 1899. December 30 0:00. For example, 1899. December 31. 12:00 is resulted in 1.5 and vice versa. Accordingly, a date type can be compared to constant in conditions in two ways: either the number of days went by since the basis date or the date written in ISO form: #yyyy-mm-dd# is used (this date format works properly in all cases irrespectively the regional settings).

Examples:

- SELECT ename FROM emp WHERE sal=1250;
- SELECT ename FROM emp WHERE ename= 'CLARK';
- SELECT ename FROM emp WHERE sal>comm;
- SELECT ename FROM emp WHERE sal NOT BETWEEN 500 and comm;
- SELECT ename FROM emp WHERE empno IN('7900','1234');
- SELECT ename FROM emp WHERE empno ALIKE'7900' OR empno ALIKE'1234';
- SELECT empno, ename FROM emp WHERE empno ALIKE'_[5-9]0%';
- SELECT ename FROM emp WHERE empno=[Identification number?];
- SELECT ename, hiredate from emp where hiredate IN(#28-09-1981#,29637);

5.5. Arithmetical, logical and concatenation operators

Table 5.3 contains the arithmetical operators.

Table 5.3. Arithmetical operators

Operator	Purpose
+	addition (sum two numbers), 5+2=7
-	subtraction (find the difference between two numbers), 5-2=3
*	multiplication (multiply two numbers), 5*2=10
/	division (divide the first number by the second number), 5/2=2,5
\	divide the first number by the second number, and then truncate the result to an integer, 5\2=2
^	raise a number to the power of an exponent, 5^2=25
MOD	divide the first number by the second number, and then return only the remainder: 5 MOD 2 =1

A logical operator combines the results of two component conditions to produce a single result based on them or to invert the result of a single condition (building complex conditions). Table 5.4 contains the logical operators.

Table 5.4. Logical operators

Inputs		Outputs of logical operators					
A	B	AND	OR	EQV	IMP	NOT(A)	XOR
0	0	0	0	1	1	1	0
0	1	0	1	0	1	1	1
1	0	0	1	0	0	0	1
1	1	1	1	1	1	0	0
0	NULL	0	NULL	NULL	NULL	1	1
1	NULL	NULL	1	NULL	NULL	NULL	0
NULL	0	0	NULL	NULL	NULL	NULL	NULL
NULL	1	NULL	1	NULL	NULL	1	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Table 5.5 summarizes the concatenation operators.

Table 5.5. Concatenation operators

Operator	Purpose
+	Combines two strings to form one string with propagation of NULL values (if one value is NULL, the entire expression becomes to NULL).
&	Combines two strings to form one string.

6. SELECT command

Structure of the SELECT command:

SELECT [DISTINCT] *<field list>*
[FROM *<table, query name>*
WHERE *<row-condition (sub query)>*
GROUP BY *< grouping aspects (sub groups)>*
HAVING *< group-condition >*
ORDER BY *<column1 [ASC | DESC][, column2 [ASC | DESC]][, ...]] >*];

6.1. Functions

The built-in functions in the MS Access execute operations to transform and process data. The functions can be divided into two types according to the number of input records (Fig. 6.1.).

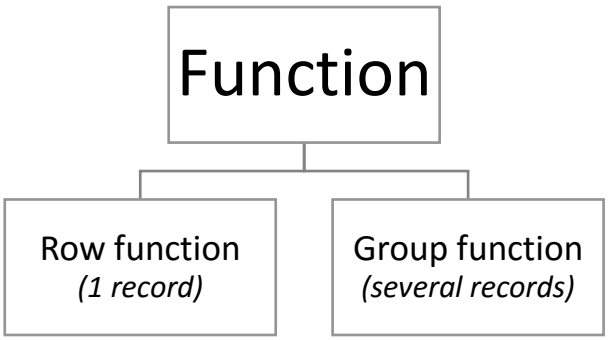


Fig. 6.1. Classification of functions according to the number of input records

The characteristics of functions:

- the input and output types of data can be different,
- the functions can be embedded into each other.

The main characteristics of the **row functions**:

- the input is one argument in all cases (column name, constant, expression),
- one return value belongs to each record,
- the function can be applied in select-list, WHERE, ORDER BY or HAVING clauses.

According to the type of in- and output, the row functions can be classified into further groups. The most relevant ones are introduced. Table 6.1 contains the most typical row functions. (The user can also define row functions). Some built-in functions belong to several groups, such as Day function, which can be assigned to the Conversion and Date/time groups too.

Table 6.1. Typical built-in row functions in Access

Group	Function	Description
Date/time	CDate(number)	Number is converted into Date type
	Day(date)	Returns the day of the month (integer 0-31)
	WeekDay(date, [firstdayofweek])	Takes date as a parameter and returns a number (integer) between 1 and 7 indicating day of week; firstdayofweek: 1 if Sunday, 2 if Monday, etc. By default 1.
	Date()	System date
	Time()	System time
	Now()	System date and time
	WeekDayName(weekday, [abbreviation], [firstdayofweek])	Takes numeric value as a parameter and returns a week day name. ▪ weekday: number of weekday 1-7 ▪ abbreviation: o 0: full name is the result o -1: abbreviated name is the result ▪ firstdayofweek: see WeekDay function
Mathematic	Abs(number)	Returns the absolute value of a number
	Int(number) or Fix(number)	Returns the integer portion of a number
	Round (number, [decimal_places])	Returns a number rounded to a specified number of decimal places
	Sqr(number)	Returns a Double specifying the square root of a number
Text	LCase(string)	Returns a String that has been converted to lowercase
	UCase(string)	Returns a Variant containing the specified string, converted to uppercase
	StrConv(string, n)	Returns a Variant converted as specified If n: ▪ 1: converts the string to uppercase characters ▪ 2: converts the string to lowercase characters ▪ 3: converts the first letter of every word in string to uppercase
	Len(string)	Returns a long containing the number of characters in a string

Note: Different first day of week settings can cause error in result if WeekDayName and WeekDay functions are used at once. Iif function, what was introduced in chapter 5.2. belongs to the advanced/logical functions.

Examples:

- SELECT CDate(40235);

- SELECT Weekday(hiredate, 2) FROM EMP;
- SELECT Date();
- SELECT Round(Sqr(13), 2);

The main characteristics of the **group functions** (called as aggregate functions):

- return a single value per groups which is calculated from values in any column considering all records of the group,
- records containing NULL value are not taken into consideration,
- with use of DISTINCT the recurring values are taken into account only once,
- can be applied in the SELECT list, in HAVING or ORDER BY clauses.

Table 6.2. contains the group-functions.

Table 6.2. Group functions in the Access

Function	Description
Sum()	Calculates the total value of all records in the group.
Avg()	Calculates the average value of all records in the group.
Min()	Returns the lowest value in the group.
Max()	Returns the highest value in the group.
Count()	Simply counts the number of entries in the group. (Number of records in the group can be calculated by this function.)
StDev()	Calculates the standard deviation across all records in the group.
Var()	Calculates the variance across all records in the group.
First()	Returns the value of the first record in the group.
Last()	Returns the value of the last record in the group.

It is not necessary to create groups with GROUP BY expression in order to use the group functions. The operation without GROUP BY expression concerns about the entire table (all records are assigned to one group). Example:

- SELECT Avg(sal) FROM EMP;

6.2. GROUP BY clause

The records can be assigned to groups. The column name(s) being the basis of grouping can be given after the GROUP BY expression. The records containing identical value in the specified column(s) are assigned to one group. It is recommended to choose such a column which contains repeated values. This column should be displayed also in the select list. It may contain only group-specific values (grouping column(s) or expressions created with group functions).

If more than one column is defined as grouping aspect, the query takes into consideration the grouping aspects from left to right. Different order of grouping aspects is resulted in different operation. If more than one grouping aspect is applied, they can be either independent or embedded.

After ‘order by’ the values also can be used that are not appeared in the select-list. An alternative column name may be used after the ‘order by’; but it must not be used after ‘group by’ and ‘having’.

GROUP BY <column1>, <column2>, …, <columnn>

Fig. 6.2. shows an example for embedded grouping. (E.g. classification of employees according to their residence).

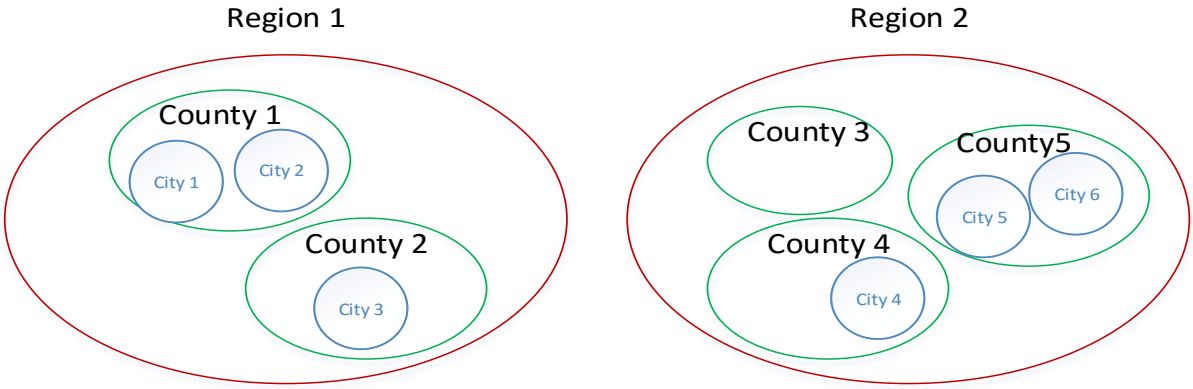


Figure 6.2. Grouping according to region – county – city

Fig. 6.3. shows an example for grouping by independent aspects (deptno, job). The order of grouping aspects influences the order in the result list (Table 6.3.). (The employees as entities are indicated by circles with numbers.)

Table 6.3. Result lists of different sequence of grouping aspects

GROUP BY deptno, job

deptno	job	Count
10	CLERK	1
10	MANAGER	1
10	PRESIDENT	1
20	ANALYST	2
20	CLERK	2
20	MANAGER	1
30	CLERK	1
30	MANAGER	1
30	SALESMAN	4

GROUP BY job, deptno

job	deptno	Count
ANALYST	20	2
CLERK	10	1
CLERK	20	2
CLERK	30	1
MANAGER	10	1
MANAGER	20	1
MANAGER	30	1
PRESIDENT	10	1
SALESMAN	30	4

Example: What is the average salary of superiors’ employees?

- SELECT mgr, Avg(sal) FROM EMP GROUP BY mgr;

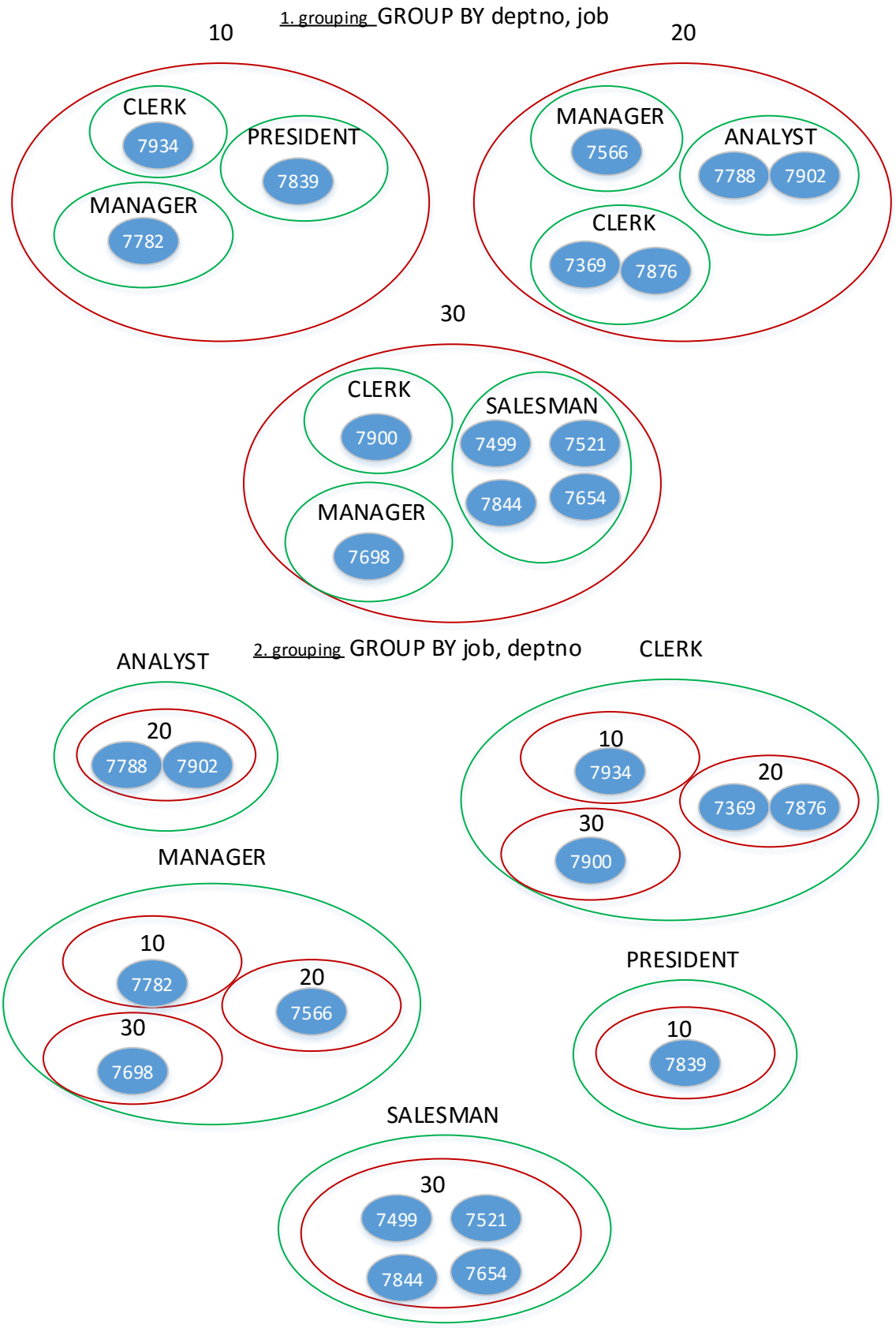


Figure 6.3. Grouping according to department (deptno) and job

6.3. HAVING clause

The created groups can be filtered by simple or complex group-conditions. The structure of the group condition is similar to the row condition. The only difference is that on the left side of the condition should be a group-specific value (alternative column name cannot be used in the condition). Group-condition is applied after HAVING.

Example: What is the highest salary among the employees in superiors' groups, where the superior's ID is not equal with 7452 or 7899?

- SELECT Max(sal) FROM EMP GROUP BY mgr HAVING mgr<>'7452' OR mgr<>'7899';

The group condition excludes whole groups, whereas row condition just records. Fig. 6.4. shows the differences between the row and group condition.

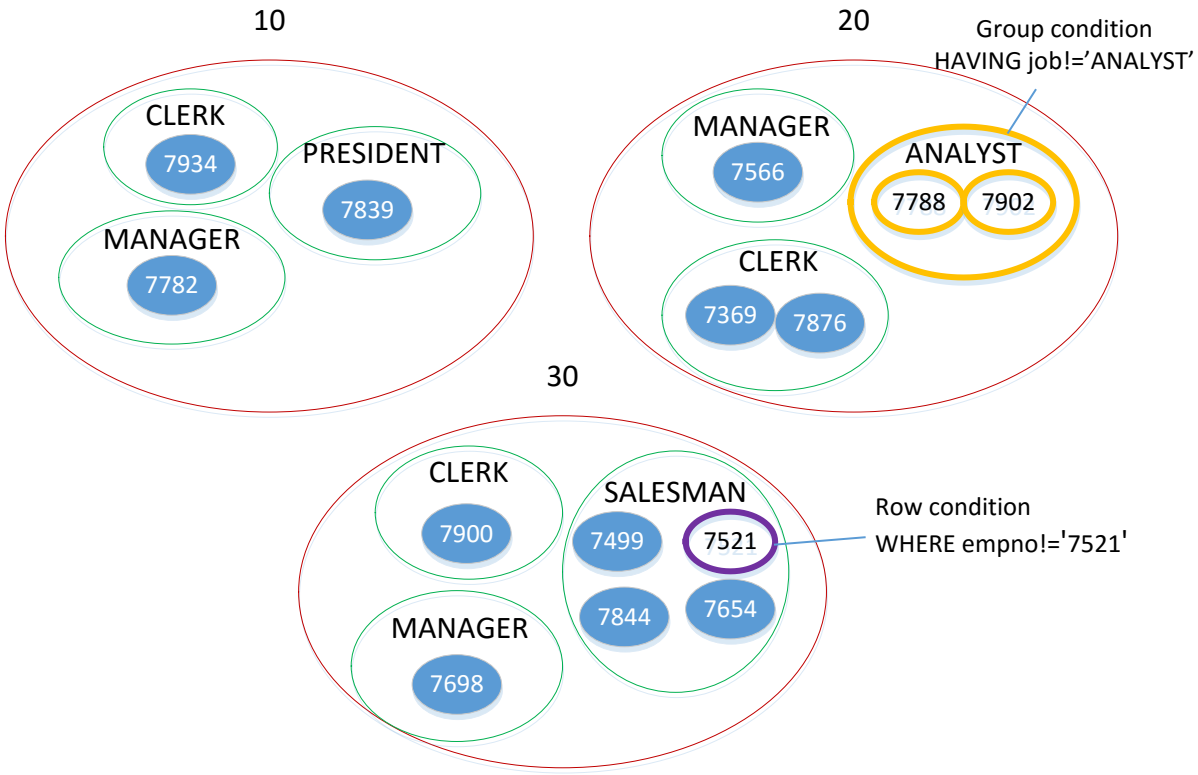


Figure 6.4. Differences between row and group condition

Row and group condition are applicable at the same time. In this case, the grouping method is based on the result list filtered by the row condition. The groups can be filtered further by the grouping condition.

7. Complex queries

The already studied simple queries based on one table. However, the queries can be built on other queries or also several objects (mixed tables and queries) at the same time. Queries may be cumulative, namely, they can be built on each other in many levels.

7.1. Queries based on many (related) tables

Columns in the select list of the SELECT command may be originated from different tables in the SQL language. In this case, all source tables should be listed after the expression FROM. If data are listed from different tables attention needs to be paid to the following:

- If the column names in the source tables are equal, it should be noted in the select list, which table's data should be displayed.
- If the source tables are connected, it should be indicated in the row condition of SELECT command (the equality of the connector fields should be prescribed as condition after WHERE command); otherwise the query is resulted in all permutations of the records.

Example:

- SELECT ename, job, emp.deptno, loc
  FROM EMP, DEPT
  WHERE emp.deptno=dept.deptno;

7.2. Cumulative queries

After saving a query, it can be applied as a table in the further processing. Queries derived from another query are called queries based on each other (cumulative queries). Among the column names at least one item is originated from another query. Cumulative queries are to be applied, if not only a constant, but a value depending on the content of another source object (table, query) is considered in the SELECT command. This value defined by the so-called subquery may be used

- in the select list,
- on the right side of the row condition,
- on the right side of the group condition.

The subquery may return with one single value, several (finite number) values or value pair (triple value…) lists. When a subquery is applied in a condition, the number of the columns and the used operators should be modified according to the returned set of results. In the first step always the subquery should be created. The so called external query is built on this. Several subqueries can be included by an external query at the same time.

Processing time can be increased in the case if a query based on another queries. If the values of the basis queries are constant, it is worth to record the constant values into a table in order to avoid the increasing processing time.

Example: Whose salary is the highest?

First, it is necessary to determine how much the highest salary is. In the second step, the salaries of the employees are compared to the highest salary determined in the subquery.

1. SELECT max(sal) as max_sal FROM EMP;
2. The query is saved as query_1.
3. SELECT ename, sal FROM EMP, query_1 WHERE sal=max_sal;

7.3. Embedded queries (subqueries)

In addition to the queries based on each other, the embedded query gives also the opportunity to take into consideration not just constants but any value depending on the content of another table. In this case, the calculated value or values can be given with SELECT command in brackets. This value defined by the embedded query (similar to the queries based on each other) can be applied

- in the select list: if the result of the embedded query is a single value,
- on right side of the row condition,
- on right side of the group condition.

Advantage of the embedded query compared to queries based on each other is that the whole query can be defined within a single SQL command. Table 7.1 contains the list of operators, which can be applied if embedded query is resulted in several values at the same time.

Table 7.1. Comparison operators used with subqueries

Operator	Description
ANY or SOME	Used in row and group conditions. Must be preceded by =, !=, >, <, <=, >=. It can be followed by any expression or subquery that returns one or several values. The records meet the condition, where the compared value meets the condition in case of **any** value in a list or returned by a query. The SOME and ANY comparison operators do exactly the same and are completely interchangeable.
ALL	Used in row and group conditions. Must be preceded by =, !=, >, <, <=, >=. It can be followed by any expression or subquery that returns one or several values. The records meet the condition, where the compared value meets the condition in case of **all** values in a list or returned by a query.
[NOT] IN	Used in row and group conditions. It compares a value to **each** value in a list or returned by a query. The records meet the condition, where the compared value equals any value in a list or returned by a query.
[NOT] EXISTS	The result of the subquery is true if the result list of the embedded query is not empty. Otherwise it is false.

Examples:

1. Whose salary is the highest?

```
SELECT ename FROM EMP WHERE sal=(SELECT max(sal) FROM EMP);
```

2. Whose salary is higher than the lowest salary of SALESMEN?

```
SELECT ename FROM EMP WHERE sal> ANY(SELECT sal FROM EMP WHERE job='SALESMAN');
```

7.4. Crosstab queries

Queries being introduced until this chapter listed the results in rows below each other. There is an opportunity in the Access to display data grouping both horizontally and vertically as a matrix. This is called crosstab query, where number of columns and rows depends on content of the table. A crosstab query aggregates the results by two independent sets of values (columns in the original table) - one set down the side (row headings), and the other across the top (column headings). Accordingly, the elements of the query (matrix) are created by group (aggregate) functions as e.g. sum, average.

The advantage of the cross-table query is that the aggregated data are displayed in a more interpretable way. The differences are shown in Table 7.2. and Table 7.3.

Table 7.2. Traditional query with double grouping (aggregation)

GROUP BY job, deptno

job	deptno	Count
ANALYST	20	2
CLERK	10	1
CLERK	20	2
CLERK	30	1
MANAGER	10	1
MANAGER	20	1
MANAGER	30	1
PRESIDENT	10	1
SALESMAN	30	4

Table 7.3. Cross-table query

	ANALYST	CLERK	MANAGER	PRESIDENT	SALESMAN
10		1	1	1	
20	2	2	1		
30		1	1		4

The SQL structure of the cross-table query:

```
TRANSFORM group function
SELECT row header, [group function]
FROM table_1
```

[WHERE row-condition]
GROUP BY *row header*
PIVOT *pivotfield (column header)* [*the condition or method of grouping/aggregation*];

Where:

- TRANSFORM – after the headword the group (aggregate) function is indicated, whose result is displayed in matrix cells.
- SELECT – after the headword the grouping aspect of the row header is indicated. Here is possible to insert the group (aggregate) function once again in order to add a column to the result matrix, which contains the sum of rows.
- FROM – after the headword the source of the select list is given.
- WHERE – after the headword any additional selection criteria can be given concerning the result list.
- GROUP BY – after the headword the grouping aspect of the row header is indicated.
- PIVOT – after the headword the grouping aspect of the column header is indicated. Here can be specified any grouping condition or method regarding column header.

Example: display in cross-table query the number of employees having the same superior and the same job, in the case when job is ANALYST, CLERK, MANAGER or SALESMAN.

```
TRANSFORM Count(empno)
SELECT mgr, Count(empno)
FROM EMP
GROUP BY mgr
PIVOT job IN('ANALYST', 'CLERK', 'MANAGER', 'SALESMAN');
```

Table 7.4 contains the result.

Remark: a new column has been added containing the sum of rows (Expr1003), which is calculated from the values of cells as the results of Count(empno) expression in the select list.

Table 7.4. Cross-table query in the sample database

mgr	Expr1003	ANALYST	CLERK	MANAGER	SALESMAN
7566	2	2			
7698	5		1		4
7782	1		1		
7788	1		1		
7839	3			3	
7902	1		1		

8. DDL commands

The data definition commands include the commands to create and manage the structure of database (tables, validity rules, etc.).

8.1. CREATE command

New database object can be created with the CREATE command (e.g. table, procedure).

Create table:

The common structure of the CREATE command to define a table:

CREATE TABLE <table name>
(<column1 name> type(length) [constraint list] [DEFAULT value],
<column2 name> type(length) [constraint list] [DEFAULT value],
…
<column name> type(length) [constraint list] [DEFAULT value]);

Table 8.1 contains the types of data.

Table 8.1. Types of data used in Access SQL

Types of data	Size	Description
autoincrement	4 bytes	Counter, starts from 1
identity(x,y)	4 bytes	Counter, starts from x, difference is y
byte	1 byte	Integers 0 to 255
single	4 bytes	Floating-point number, max 7 numerical digit
number	8 bytes	Floating-point number, max 15 numerical digit
integer	2 bytes	Long integer, from -2 147 483 648 to +2 147 483 647
currency	8 bytes	Currency
char(n)	<= 255 characters	Shorts text; fixed-length text strings; n: max. number of characters
text		Long text; variable-length text strings
datetime	8 bytes	Date
yesno	1 bit	Yes/No
longbinary	<= 2 GB	OLE object, e.g. excel file can be attached to a record

8.2. Validation rules (CONSTRAINTs)

Validation rules (called constraints) are defined during creating a table. The data elements are checked in case of data input or modification, whether they satisfy the condition of constraint or not. If yes, they can be inserted/modified. If the validation rule has a name, the error is indicated through this name at data input/modification. The validation rules have two types:

- **Field Validation Rule:** the name of the column is included in the condition.
- **Record or Table Validation Rule:** several columns from the same table are included in the condition.

Table 8.2 contains the often used validation rules.

Table 8.2. Often used validation rules in the Access SQL

Validation rule	Description
Not null	It must be filled
Unique	Data repetition is not allowed in the column (must be unique)
Primary key	Not null + Unique
References	Foreign key - type, length have to be equal with primary key
Check	Optional (user-defined) condition

Example:

```
CREATE TABLE service1 (
S_ID autoincrement primary key,
Plate_Nr char(7) not null,
Type char(25),
Odometer number,
Production_Date Datetime,
Date_of_service Datetime not null,
Casco yesno,
Admin_ID integer references EMP(empno),
constraint [Odometer cannot be 0 or minus] check (Odometer > 0),
constraint Production_Date check (Production_Date < Date_of_service) );
```

Note: the references constraint implies constraint regarding the data type. The data type of the foreign key must be the same (when creating the relationship) as the type of the primary key.

The Validation Text is an 'error message' to alert the user about any mistakes when the entered value does not meet the condition defined by the user in the validation rule.

8.3. Default value

Default value can be defined to columns during creating a table. The default value is inserted automatically if a new record is created, but its modification is also possible. It is applied when the majority of the entities (records) have the same value regarding one attribute. The default value is defined after the validation rule regarding the column using the DEFAULT headword. The syntactic rules required by the data type should be considered. Default value can be not just constant but any value returned by a function.

Examples:

```
…
column1 number default 5,
column 2 char(25) default 'BME',
column 3 datetime default now(),
…
```

In the first case the number 5, in the second case the text BME and in the third case the current time is the default value.

8.4. ALTER command

The ALTER command is suitable for modifying the existing structure of the database object. Modifying the table structure can be particularly hazardous in case of tables with stored data, therefore it requires special attention. The following properties of a table can be modified by using the ALTER command:

- **Add new column:**
  ALTER TABLE table1 ADD columnn type [constraint DEFAULT value];

  Example: ALTER TABLE service1 ADD manufacturer char(25) not null;

- **Modify existing column:**
  ALTER TABLE table1 ALTER COLUMN columnn type [constraint DEFAULT value];

  Example: Alter table service1 ALTER COLUMN manufacturer char(25) unique;

- **Delete column:**
  ALTER TABLE table1 DROP COLUMN columnn;

  Example: Alter table service1 DROP COLUMN Casco;

- **Add validation rule:**
  ALTER TABLE table1 ADD CONSTRAINT constraint1 CHECK (logical condition);

  Example: Alter table service1 ADD CONSTRAINT [Not allowed to create a new service record in the past] CHECK (Date_of_service > (Now()-3));

- **Remove validation rule:**
  ALTER TABLE table1 DROP CONSTRAINT constraint1;

  Example: Alter table service1 DROP CONSTRAINT [Odometer cannot be 0 or minus];

8.5. DROP command

DROP command is used to delete an existing database object ENTIRELY (both the structure and the content are deleted). Components of a table can also be deleted separately by drop command, which was presented at the ALTER command in chapter 8.4 (deletion of the column and validation rule).

The structure of command to delete the entire table:

DROP TABLE <table name>;

Example: DROP TABLE service1;

Note: if a table has valid constraint with CHECK or is in a relationship, it cannot be removed from Access in general.

9. DML commands

Data Manipulation Language (DML) commands are used to manage data in a database. DML commands are not executed irrevocably, they can be rolled back until 'commit work'. The commands are used to retrieve and manipulate data in a relational database. They modify the stored data but not the structure. The SELECT command was detailed in the previous chapters, so in this chapter the most common data input and modification commands are introduced.

9.1. INSERT INTO command

INSERT INTO command is used to insert data into a table. Ways of data input are the following:

- **Input a new record**: values of the record are given by the user. The common structure of the command:

  INSERT INTO <table name> [(*list of columns*)]
  VALUES (list of values);

  If value for all the columns of the table is defined, the list of columns can be omitted. Otherwise, the list of columns defines which fields (and in which order) are filled with data.

  Example:

  INSERT INTO EMP (empno, ename, job, hiredate, sal) VALUES ('1001', 'FOG', 'ANALYST', '1989.03.21.', 1500);

  Note: the default value is filled by the Access in all cases; even if the user doesn't define it during data input.

- **Input/copy of data from other table or query:** it is applicable, when the values of new records are stored in another table or produced by a query. If the structure of 'source' and 'destination' tables (queries) are exactly the same and all columns of records are copied, the column list can be omitted and select list contains * sign. In other cases, coordination is needed between the content of the column list and the select list in order to insert the data elements into the proper field. The sequence of column list and select list should be the same. Common structure of the command:

  INSERT INTO *<table name_1> [(list of columns)]*
  SELECT *<select-list>*
  FROM *<table name_2> or <query_1>*
  [WHERE *row condition*]*;*

  Example:

```
INSERT INTO DOLG
SELECT *
FROM EMP
WHERE empno='7899';
```

Note: calculated values are not stored in table in general because of redundancy. Except when further queries use it to reduce the calculation time.

9.2. UPDATE command

The command is appropriate for modification of data of existing records. The common structure of the command:

```
UPDATE <table name>
SET <column1>=<exp1>, <column2>=<exp2>,…
[WHERE <condition>];
```

If the WHERE condition is not given, the data are modified in all records.

Example: increase the salary with 500 for the employee, who has ID 7902

```
UPDATE EMP
SET [sal] = [sal]+500
WHERE empno='7902';
```

Note: one data element of a record can be deleted with the UPDATE command, if the NULL value is set up as a new value.

9.3. DELETE FROM command (record, table content)

The command is appropriate for deletion of records from tables. The common structure of the command:

```
DELETE FROM <table name>
[WHERE <condition>];
```

If there is no row condition in the command, all the records are deleted. (But the structure of the empty table remains.)

9.4. UNION of records

The union query merges the results of two or more independent queries or tables having the entirely same structure into one result set. The command connects two SELECT queries with the UNION word. The result set contains records selected from both queries (without repetition). The common structure of the command:

```
SELECT command 1
UNION
SELECT command 2;
```

Example:

```
SELECT * FROM EMP
UNION
SELECT * FROM DOLG;
```

9.5. MAKE TABLE query

The result set of a query can be saved into a table using 'make table' query. The existing table with the same name is deleted before execution of the query; then it is re-created and uploaded with data. This command belongs to DML command group as the SELECT command is applied. The common structure of the command, **highlighting only the differences**:

SELECT *<select-list>*
INTO new_table
FROM *<table name_1> or <query_1>;*

Note: calculated values are not stored in table in general because of redundancy. Except when further queries use it to reduce the calculation time.

10. Practice assignments

You may find in this chapter some typical assignments to prepare for the midterms.

10.1. Exemplary assignments for the 1st midterm

Queries based on the sample database:

1. List jobs at the company! Each job is displayed only once.
2. List employees in descending order by their manager's ID!
3. List data of departments with new English header.
4. List name, job and department ID of employees. Insert a new column containing the "Department:" expression before the ID.
5. How much is the yearly income of the employees?
6. List name of the employees in ascending alphabetical order, who have undefined commission value.
7. List name and job of the employees, who work not at department 20.
8. Who are the employees whose name contains A in the second position?
9. Who are the employees whose department ID ends with 9?
10. List name of the employees, whose boss is not the employee with ID 7788 or 7839?
11. Who are the employees, whose commission is between 200 and 600?
12. Who were hired in December 1981?
13. List the employees, whose name begins with not M or not A.

Conceptual models: create the table structures taking also into account all parts of the assignment. Record some data in order to exemplify the results of the queries.

1. Charging points and their monthly checks are registered at an EV charging station. There are 4 charging points: 1 normal, 1 fast and 2 ultra-fast chargers. Charging capacity and type of connector are registered. The counter of the electricity meter and occurrence of failures are also registered during monthly checks.

    - List the charging points in ascending order by charging capacity.
    - List the monthly checks of ultra-fast chargers in descending order by date.

2. Airplanes and their bases of an air company are registered. There are 5 planes; 3 Boeing have base in Budapest and 2 Airbus have base in Rome. Type, year of manufacture, date of launch, and the capacity of the planes are registered.

    - List planes based in Budapest in descending order by date of launch.
    - How old will the airplanes be on 17.05.2020?

10.2. Exemplary assignments for the 2nd midterm

1. How many days passed since the revolution of 23.10.1956?
2. Calculate the value of √7 with 3 decimals.
3. What is the distance between points A(-4; 12) and B(7;2)?

Queries based on the sample database:

4. On which day was FORD hired (name of the day)?
5. Name of the employee, who was hired earliest.
6. What is the average salary in the SALESMAN and CLERK jobs?
7. How much extra salary in % should be paid off in order to attain at 2500 as average salary?
8. Who are the employees, who earn more than BLAKE and how much is the difference?
9. Whose commission is the highest and what is the proportion (in %) of his commission and income?
10. Who work in New York?
11. How many employees work in the different cities?
12. How many employees work at the departments in the different jobs (let' s use crosstab).
13. How much higher is the salary for the best paid employee than JONES's salary?

Conceptual models: create the table structures using SQL **DDL commands** taking also into account all parts of the assignment. Record some data using **DML commands** in order to exemplify the results of the queries

1. Charging points and their monthly checks are registered at an EV charging station. There are 4 charging points: 1 normal, 1 fast and 2 ultra-fast chargers. Charging capacity and type of connector are registered. The counter of the electricity meter and occurrence of failures are also registered during monthly checks.

    - Delete the normal charger from the database and register another fast one.
    - How much is the total charging capacity in the charging station?

2. Airplanes and their bases of an air company are registered. There are 5 planes; 3 Boeing have base in Budapest and 2 Airbus have base in Rome. Type, year of manufacture, date of launch, and the capacity of the planes are registered.

    - Modify one Boeing airplane's base to Rome.
    - How much is the average age of the airplanes in the different bases?

REFERENCES

(in Hungarian)

[1] Csiszár, Cs. – Sándor, Zs.: *Közlekedési informatika* (jegyzet) 2014.

[2] Csiszár, Cs. – Westsik, Gy.: *A közlekedési informatika kutatása és oktatása a BME Közlekedésüzemi és Közlekedésgazdasági Tanszékén*. Közlekedéstudományi Szemle LXIV. évf. 2. szám 44-52.o. Budapest, 2014.

[3] Munkácsiné Lengyel, E. – Tóth, J. – Csiszár, Cs. – Juhász, J.: *Közlekedési informatika* (jegyzet) 2004.

Database-theme literature collection

(in Hungarian)

Szelezsán János: Adatbázisok (példatár is)

Bálint Dezső: Adatbázis-kezelés (Talentum kiadó)

Jeffrey D. Ullmann – Jennifer Widom: Adatbázis-rendszerek

Kende Mária - Kotsis Domokos - Nagy István: Adatbázis-kezelés az ORACLE rendszerben

Michael Abbey - Michael J. Corey - Ian Abramson: Oracle 8i kezdőknek

Bodnár Ibolya - Nagy Zoltán: Adatbázis-kezelés

Stolniczki Gyula: SQL kézikönyv

More information can be found:

https://docs.oracle.com/cd/A87860_01/doc/server.817/a85397/operator.htm

https://support.office.com/en-us/article/Table-of-operators-e1bc04d5-8b76-429f-a252-e9223117d6bd

https://support.office.com/en-US/article/ORDER-BY-Clause-E8EA47F7-5388-460A-BEC8-DCC81792D762

http://www.w3schools.com/sql/sql_like.asp

http://freemarker.incubator.apache.org/docs/dgui_template_exp.html

https://developer.mozilla.org/hu/docs/Web/JavaScript/Guide/Expressions_and_Operators

http://www.w3schools.com/sql/sql_wildcards.asp

http://www.w3schools.com/sql/sql_functions.asp

https://oracle-base.com/articles/misc/all-any-some-comparison-conditions-in-sql

APPENDIX

Appendix 1: Data modelling assignment

Prepare a logic data model in 3NF identifying the appropriate entity types.
Define the data types and length of attributes.

Topic: Register of an automatic parking garage

Stored attributes:

IN_TIME	(time of entering the parking garage)
PLATE_NUMBER	
PLACE_ID	(identifier of the parking place)
OUT_TIME	(time of leaving the parking garage)
BRAND	(brand of the vehicle)
PARKING_FEE	(parking fee per hour, depends on the entering time)
PARKING_ID	(identifier of parking)
OBSERVED	(is there camera surveillance?)
PARKING_PROPERTIES	(description of irregular parking)
SEGMENT	(in which segment is the parking place?)
COLOUR	
LEVEL	(on which level is the parking place?)
TYPE	

Appendix 2: Data modelling assignment

Prepare a logic data model in 3NF identifying the appropriate entity types.
Define the data types and length of attributes.

Topic: Register of cargo traffic at a haulier company

Stored attributes:

V_ID	(vehicle identification number)
DRIVER_NAME	
DEP_TIME	(departure time)
DISTANCE	
PLATE_NUMBER	
INJURY	
ACCIDENT_REP_NR	(accident report number)
BIRTHDAY	
LAST_SERVICE_DATE	
ACCIDENT_TIME	(accident date and time)
CARGO_WEIGHT	
MILEAGE	(kms have been run before start)
ARR_TIME	(arrival time)
ACCIDENT_CIRCUM	(circumstances of the accident)
WAYBILL_NR	(number of waybill)
GUILTY	(is the driver guilty?)
VEHICLE_TYPE	
DRIVER_PROPERTIES	
CARGO_VALUE	(value of transported cargo)
LICENCE_NR	(driver's licence number)
DESTINATION	(city name of destination)
DEPLOYMENT_DATE	(date of first deployment)
DAMAGE_VALUE	(value of damage)