Noncommercial Joint-Stock Company

ALMATY UNIVERSITY OF POWER ENGENEERING AND TELECOMMUNICATIONS

Engineering cybernetic department

DATABASE DESIGN

Lecture notes for the specialty 5B070200 - "Automation and Control"

Almaty 2017

Prepared by: L.K. Ibrayeva. Database design. Lecture notes for specialty 5B070200 – Automation and Control. – Almaty: AUPET, 2017. – 60 p.

Lecture noteshave been prepared in accordance with the working program of discipline "Database Design" for undergraduate students majoring in 5B070200 – Automation and Control. The issue contains 15 lectures.

The abstract of lectures is intended to help students in studying theoretical material and for exam preparation, to be used in practical and laboratory classes.

Ill. -15, tables – 7, bibliography – 9.

Reviewer: PhD, ass.prof. E.G. Satimova

Printed according to the additional plan for publications of NC JSC "Almaty University of Power Engineering and Telecommunications" for 2017 y.

Introduction

Development of computer technology has had a significant impact on many aspects of society. Over the past decade the scope of application of computing machinery has significantly expanded, and, correspondingly, increased its influence on everyday life. An automated system that organizes data and produces information is an information system. An important part of any information system is a database. Databases today are essential to every business. The power of databases comes from a body of knowledge and technology that has developed over several decades and is embodied in specialized software called a database management system, or DBMS.DBMS is a powerful tool for creating and managing large amounts of data efficiently and allowing it to persist safely over long periods of time. These systems are among the most complex types of software available.

One of the most difficult and crucial problems associated with the creation of an information system is a database design problem. The database content, effective for all its future users, a way of organizing data and tools of data management should be determined as a result of database design. The process of database design is a sequence of transitions from informal verbal descriptions of information structures of a subject domain to formalized description of domain objects in terms of some model.

Generally speaking, it is possible to distinguish the following stages of design:

- system analysis and verbal description of information objects of the subject domain;
- designing a conceptual domain model; it is a partially formalized description of the domain objects in terms of some semantic model;
- the choice of a specific database management system;
- a description of the database in terms of the selected system,
- and finally, physical database design, i.e. the choice of efficient placement of database on external storage to ensure the most efficient operation of the application.

The information technologies of data management are based on the use of relational databases management systems. All languages of data manipulation created before the advent of relational databases are focused on operations with the data presented as logical file records. Previously it was required that the user should be aware of detailed knowledge of the organization of data storage and apply sufficient efforts to specify not only what data is needed, but also where they are placed, and how step by step to get them. Now databases languages are more heavily focused on the final result of data processing than on the procedure of this processing.

These lecture notes present an overview of the development of database technology; three-level architecture of database systems; basic concepts of conceptual and relational approach to database design; principles of normalization of the database.

1 Lecture #1. Development of data processing methods

The content of the lecture: basic concepts of information systems.

The goal of the lecture: to learn the history of data processing methods andthe limitations of the file-based system.

1.1 Information systems

The database system becomes the most important development in the field of software engineering. The database is now the basis of the information system, and has fundamentally changed the way many organizations operate.

Information system is complex based on computers and other technical tools for collection, storage, updating and processing of information to support any activity.

A computer-based information system includes computer hardware, software tools, computers or network of computers, a database, database software, application software, personnel using and developing the system and procedures. The hardware, software, and telecommunications constitute information technology (IT), which is now necessarily used in the operations and management of organizations. The database is a fundamental component of an information system.

Many organizations work with large amounts of data. In this context, we distinguish the concepts of *data* and *information*. Data are the *separated* facts. Many people think of data as synonymous with information; however, information actually consists of data that has been organized to help to find answers to questions and to solve problems. Information is the *processed* data. Information is useful to managers in *making decisions*.

The seriousness of the information impact on planning and decision making, led to the understanding, that information is a resource with a certain value, and, therefore, needs to be *streamlined* and *managed*. If managers are well informed they are more likely to make timely decisions. The information systems that use databases have become a fundamental tool for logistics managers, with accurate and timely information.

An information system is defined as the software that helps organize and analyze data. So, the purpose of an information system is to turn raw data into useful information that can be used for decision making in an organization.

Since the 1970s, database systems have been gradually replacing traditional file-based systems as part of an organization’s information systems infrastructure. At the same time there has been a growing recognition that data is an important corporate resource that should be treated with respect, like all other organizational resources.

We will start considering the database concepts with a review of its predecessor, the *file-based* system. Although the file-based approach is largely obsolete, there are good reasons for studying it.

1.2 File-based systems

In the history of the computer, development of two main domains of its use can be noted.

The first sphere is the use of computer technology to fulfill numerical calculations that are too long or impossible to perform manually. Computers were usually used for numerical solution of complex mathematical problems, writing the codes of algorithms of such tasks. These algorithms are applied to simple data, the volume of which is relatively small.

The second domain that directly relates to our topic is the use of computer technology in information systems. Typically, such systems work with the large volumes of information having a complex structure. The second domain of using the computers came later, after the first one. This is because at the dawn of the computing the capabilities of computers for the storage of information were very limited.

The development of database technology was the result of development of data processing and information management. This process continued over several decades. Before 1950s (early manual system) data was stored as paper records, lot of man power involved, lot of time was wasted e.g. when searching; therefore such a system was inefficient.

The first computer system (1950s and early 1960s) performed clerical work, reducing the number of paper. The first commercial systems were used primarily for accounting purposes. The costs of manual labor on this job are so great that the cost of computer systems is quickly recouped. Since these systems were performing regular work with the documents, they were called *data processing systems*.

Computer files met the folders for papers. These files allow only *sequential* access. This means that each entry in the file can only be read after you read all the preceding records. In 60-ies, when the storage of data on disk was relatively expensive, most of the files were stored on magnetic tape, and recordings were extracted and processed sequentially.

However, in information systems a set of interrelated information objects in fact reflects the model of real-life objects. And the demand of users in the information adequately reflecting the status of real objects requires a relatively fast response of the system to their needs.

It can be assumed, that the requirements of *non-numeric* applications has led to the emergence of hard disks, which was a revolution in the history of computing. These devices of external memory had a significantly higher capacity than magnetic tapes, provided satisfactory speed of access to data in the arbitrary order, and the possibility to change the disk on the device allowed us to have a practically unlimited data archive.

With the appearance of hard disks (late 1960s and 1970s) began the history of data management systems in the external memory. There appeared the necessity of random access to records in the file.

Arbitrary access to data is the ability to directly access to a specific record without prior sorting or sequential reading of all records. Hard disks allow direct access to data; data stored in files; these systems are known as *file processing systems*.

The file-based system is a collection of application programs that perform services for the end-users. Each program defines and manages its own data. File-based systems were an early attempt to computerize the manual filing system that we are all familiar with. For example, in an organization a manual file is set up to hold all external and internal correspondence relating to a project, product, task, client, or employee.

Typically, there are many such files, and for safety they are labeled and stored in one or more cabinets. For security, the cabinets may have locks or may be located in secure areas of the building. In our own home, we probably have some sort of filing system which contains receipts, guarantees, invoices, bank statements, and such like. When we need to look something up, we go to the filing system and search through the system starting from the first entry until we find what we want.

Alternatively, we may have an indexing system that helps locate what we want more quickly. For example, we may have divisions in the filing system or separate folders for different types of item that are in some way logically related. In the sixties of the last century, index files were widely spread. A good index file helps users to locate the information they need. These files allow you to select one or more fields to accurately define which record to extract. These fields were named keys. The key is the data fields that uniquely identify a record in the file. For example, our ID can be a key. But in a student group you cannot use the surnames as key because there can be namesakes.

The manual filing system works well while the number of items to be stored is small. It even works quite adequately when there are large numbers of items and we have only to *store* and *retrieve* them. However, the manual filing system breaks down when we have to cross-reference or process the information in the files.

1.3 Drawbacks of the File-Based Approach

Despite the appearance of files with random access, it quickly became evident that the file systems of any type have some limitations.

These limitations are the following:

1. *Separation and isolation of data.* When data is isolated in separate files, it is more difficult to access data that should be available. Each program maintains its own set of data. Users of one program may be unaware of potentially useful data held by other programs.

2.*The data redundancy (duplication of data)*. Many applications use their own data files, and therefore, the same unit of data is repeated in different files. So, if the applications are based on a file system we get data redundancy.

Uncontrolled duplication of data is undesirable for several reasons, including:

- duplication is wasteful; it costs time and money to enter the data more than once;
- it takes up additional storage space, which can be more costly;
- what, perhaps, is more important, duplication can lead to loss of data integrity.

Data *integrity* is the accuracy and consistency of data.

3. *Data dependence.* The physical structure and storage of the data files and records are defined in the application code. This means that changes to an existing structure are difficult to make. This characteristic of file-based systems is known as *program-data dependence*.

4. *The high cost of programmer's labor*. In the file system anew application program requires a new set of files. Even if the existing file contains the required information, an application requires another set of data. As a result, the programmer needs to write another program. That is, between data and programs there exists a stable dependency.

5. *Incompatible file formats.* Because the structure of files is embedded in the application programs, the structures are dependent on the application programming language.

6. *Poor control of data*. In the file system there is no centralized control. The same data item may have different names; terminology in different departments may be different. For example, the Bank may imply one meaning in the term *account t*if it is related to savings and quite a different one when it is referred to loans. Different meanings of the same term are called *homonyms*. And, conversely, different words can have the same meaning. For example the Bank can talk about the *account holder* or *customer*, meaning in this term the same meaning. Terms having the same meaning are called synonyms. Because the homonyms and synonyms can be in the system there can be misunderstanding. And it needs much time and lot of work for coordination between departments.

7. *Insufficient data management capabilities*. The index files make it possible to access a specific record by key. This is enough to until we need a separate record. If we need a set of related data, such information is difficult or even impossible to extract from the file system. This is because the file system does not allow establishing a link between data in different files.

8. The users *need access to data in the same time*. In general, the files systems provide concurrent access. If all users are going to only read the file, nothing bad will happen. But if at least one of them will change the file, for correct work of these users is required mutual synchronization of their actions in relation to the file.

These deficiencies served as the impetus that forced the developers of information systems to propose a new approach to information management. This approach was implemented within the framework of new software systems, later subsequently called Database Management Systems (DBMS) and information storage.

2 Lecture #2. Information system that uses database

The content of the lecture: the necessity of databases appearance.
The goal of the lecture: to learn the main terms and components of DB.

2.1 Database Approach

All the limitations of the file-based approach pointed out in the previous lecture, can be explained by two factors:

1) the definition of the data is embedded in the application programs, rather than being stored separately and independently;

2) there is no control over the access and manipulation of data beyond that imposed by the application programs.

To become more effective, a new approach was required.

Database (DB) is an ordered set of stored data associated by a common theme or purpose. The main purpose of the database is a quick search of the information contained therein. The database concept is wider than just a set of data. Besides the actual database, there is a set of application programs that work with these data, processing them in the usual way, as well as the appropriate equipment and people.

The database is a single, possibly large repository of data that can be used simultaneously by many departments and users. Instead of disconnected files with redundant data, all data items are integrated with a minimum amount of duplication. The database is no longer owned by one department but is a shared corporate resource.

The database holds not only the organization's operational data but also a description of this data. For this reason, a database is also defined as a *self-describing collection of integrated records*. The description of the data is known as the *system catalog* (or *data dictionary* or *metadata*– the 'data about data'). It is the self-describing nature of a database that provides *program–data independence.*

The database management system (DBMS) is software that manages a database. It is a software system that enables users to define, create, maintain, and control access to the database. The DBMS is the software that interacts with the users' application programs and the database.

Users interact with the database through a number of *application programs* that are used to create and maintain the database and to generate information. These programs can be conventional batch applications or, more typically nowadays, they

will be online applications. The application programs may be written in some programming language or in some higher-level fourth-generation language.

The program that performs a specific practical problem in business is called an *application program* or *application.* A set of programs that jointly perform a common associated task, is called an *application system*. In our context the programs with which users work with the database will be called applications that is a computer program that interacts with the database by using appropriate request (typically an SQL statement) to the DBMS.

In general, many different applications can work with asingle database. For example, if a database models some entity, then for the work with it, there can be created an application that services the subsystem of accounting personnel, another application may be devoted to the work of a subsystem of the payroll employees, the third application runs as a subsystem of inventory, the fourth application is dedicated to the planning of the production process. When considering applications working with one database, it is assumed that they can work in parallel and independently from each other, and that the DBMS is designed to provide multiple applications with a single database so that each of them was executed correctly, but take into account all changes in the database made by other applications.

2.2 Components of the DBMS Environment

An information system that uses the databases or a *database system* consists of five major components: the hardware, software (DBMS), data (DB), procedures and people (users, administration, staff).

Hardware. The hardware can range from a single personal computer, to a single mainframe, to a network of computers. The particular hardware depends on the organization's requirements and the DBMS used. Some DBMSs run only on particular hardware or operating systems, while others run on a wide variety of hardware and operating systems. A DBMS requires a minimum amount of main memory and disk space to run, but this minimum configuration may not necessarily give acceptable performance.

Software. The software component comprises the DBMS software itself and the application programs, together with the operating system, including network software if the DBMS is being used over a network. Typically, application programs are written in a third-generation programming language (3GL) or using a fourth-generation language (4GL)

Data. Perhaps the most important component of the DBMS environment, certainly from the end-users' point of view, is the data. The data acts as a bridge between the machine components and the human components. The database contains the operational data and the metadata, the 'data about data'.

The structure of the database is called the *schema*.

Procedures. Procedures refer to the instructions and rules that govern the design and use of the database. The users of the system and the staff that manage the database require documented procedures on how to use or run the system. These

may consist of instructions on how to: log on to the DBMS; use a particular DBMS facility or application program; start and stop the DBMS; make backup copies of the database; handle hardware or software failures. This may include procedures on how to identify the failed component, how to fix the failed component and, following the repair of the fault, how to recover the database; change the structure of a table, reorganize the database across multiple disks, improve performance, or archive data to secondary storage.

People. The final component is the people involved with the system. They are data and database administrators, database designers, application developers, and the end-users. The database and the DBMS are corporate resources that must be managed like any other resource. Data and database administration are the roles generally associated with the management and control of a DBMS and its data.

The *Data Administrator* (DA) is responsible for the management of the data resource including database planning, development and maintenance of standards, policies and procedures, and conceptual/logical database design. The DA consults with and advises senior managers, ensuring that the direction of database development will ultimately support corporate objectives.

The *Database Administrator* (DBA) is responsible for the physical realization of the database, including physical database design and implementation, security and integrity control, maintenance of the operational system, and ensuring satisfactory performance of the applications for users. The role of the DBA is more technically oriented than the role of the DA, requiring detailed knowledge of the target DBMS and the system environment. In some organizations there is no distinction between these two roles; in others, the importance of the corporate resources is reflected in the allocation of teams of staff dedicated to each of these roles.

Database Designers. In large database design projects, we can distinguish between two types of designer: logical database designers and physical database designers. The *logical database designer* is concerned with identifying the data, the relationships between the data, and the constraints on the data that is to be stored in the database. The logical database designer must have a thorough and complete understanding of the organization's data and any constraints on this data (the constraints are sometimes called *business rules*). These constraints describe the main characteristics of the data as viewed by the organization. The *physical database designer* decides how the logical database design is to be physically realized. This involves: mapping the logical database design into a set of tables and integrity constraints; selecting specific storage structures and access methods for the data to achieve good performance; designing any security measures required on the data. Many parts of physical database design are highly dependent on the target DBMS, and there may be more than one way of implementing a mechanism. Consequently, the physical database designer must be fully aware of the functionality of the target DBMS and must understand the advantages and disadvantages of each alternative for a particular implementation. The physical database designer must be capable of selecting a suitable storage strategy that takes

account of usage. Whereas conceptual and logical database designs are concerned with the *what*, physical database design is concerned with the *how*. It requires different skills, which are often found in different people.

Application Developers. Once the database has been implemented, the application programs that provide the required functionality for the end-users must be implemented. This is the responsibility of the *application developers*. Typically, the application developers work from a specification produced by systems analysts. Each program contains statements that request the DBMS to perform some operation on the database. This includes retrieving data, inserting, updating, and deleting data. The programs may be written in a third-generation programming language or a fourth-generation language, as discussed in the previous section.

End-Users. The end-users are the 'clients' for the database, which has been designed and implemented, and is being maintained to serve their information needs. End-users can be classified according to the way they use the system:

- *naive users* are typically unaware of the DBMS. They access the database through specially written application programs that attempt to make the operations as simple as possible. They invoke database operations by entering simple commands or choosing options from a menu. This means that they do not need to know anything about the database or the DBMS. For example, the checkout assistant at the local supermarket uses a bar code reader to find out the price of the item. However, there is an application program present that reads the bar code, looks up the price of the item in the database, reduces the database field containing the number of such items in stock, and displays the price on the till.

- *sophisticated users*. At the other end of the spectrum, the sophisticated end-user is familiar with the structure of the database and the facilities offered by the DBMS. Sophisticated end-users may use a high-level query language such as SQL to perform the required operations. Some sophisticated end-users may even write application programs for their own use.

See the information system that uses database enabled to overcome the shortcomings of file systems (table 2.1). One of the main objectives of database systems is ensuring data independence, i.e. independence of the application from changes in the storage structure and access strategies. Within a database, maintain data *integrity*, i.e. accuracy and consistency.

Table 2.1 - Database System vs. File System

File systems	**Information systems that use databases**
1. Data redundancy.	All applications use the same (agreed) data set. Significant information is recorded*once.*
2. Poor control of data.	The database management system provides the *centralized* control of data.
3. Insufficient data management capabilities.	The DBMS are designed*to provide and manage data relationships.*

4. High cost of programmer's labor.	The DB allows *to separate* programs and data
5. File systems manages sharing of data through its code	A DBMS manages *concurrent access* to data

3 Lecture#3. The evolution of the database systems

The content of the lecture: questions of the development of database management systems

The goal of the lecture: to learn the history of the development of DBMS.

It is possible to distinguish four stages in the development of DBMS. However, it should be noted that there is no strict time restriction in these stages.

Databases on a mainframe computer. Originally, DBMS's were large, expensive software systems running on large computers. The size was necessary, because to store a gigabyte of data required a large computer system. Databases were stored in the external memory of the *mainframe computer*, the users tasks were run mostly in batch mode. Interactive access was provided through a console terminal, which did not have its own computing resources and was only used to device I/O to the mainframe. Program database access was written in different languages and run as a regular numeric program. Powerful operating systems enabled parallel execution of the entire set of tasks.

The features of this stage of development are expressed in the following:

- all DBMS are based on a powerful multiprogramming operating systems, so basically, it supports the centralized database in a distributed access;
- control functions resource allocation is mainly carried out by the operating system;
- supported languages are low-level manipulation of data, navigation-oriented data access methods;
- a significant role for the administration of data;
- carried out extensive work on the study and formalization of the relational data model, and created the first system (System R), realizing the ideology of the relational data model;
- research results are openly discussed in print, comes a powerful flow of publicly available publications covering all aspects of the theory and practice of databases, and the results of theoretical research are being actively implemented in commercial DBMSs;
- the first high-level languages appeared for working with the relational data model. However, there were no standards for these first languages.

The appearance of personal computers. Personal computers have rapidly changed the concept of the place and role of computers in society. There are plenty of programs designed for untrained users. The system programmers were sidelined. And of course this had an impact on working with databases. There appeared

programs, called database management systems and allowed you to store large amounts of information, they had a convenient interface for entering the data, the built-in tools to generate various reports. These programs have helped to automate many accounting functions that were previously carried out manually. Steady decline in prices for personal computers have made them available not only for companies but for individual users. Computers have become a tool for documentation and internal accounting functions. It's all played both a positive and a negative role in the development of databases. The apparent simplicity and accessibility of personal computers and their software have generated a lot of amateurs. However, the availability of personal computers has made users of many areas of knowledge that were not previously used computers in their activities, to contact them. And the demand for developed convenient programs of data processing has forced software vendors to put all of the new systems, which are called *desktop* database. Features of this phase are the following:

- all DBMS were designed to create databases, mainly with exclusive access and that's understandable. The personal computer, it was not connected to the network and the database; it was created for a single user. In rare cases, it assumed sequential operation of multiple users, for example, the first operator that introduced accounting documents, and then the accountant, who defined transaction, the relevant primary documents;

- most DBMS had a developed and friendly user interface. In most of them interactive mode with the database as part of its description, and in the query design existed. In addition, most DBMSs were developed and offered convenient tools for developing complete applications without programming. Tool medium consisted of ready-made elements of the application as templates, screen forms, reports, labels, graphical query designers that simply could be assembled into a single complex;

- desktop DBMS has no means of supporting structural and referential integrity of the database. These functions were to execute the application, but the scarcity of application development tools are sometimes not allowed to do it, and in this case these functions were to be carried out by the user, requiring him for additional control when entering and modifying information stored in the database;

- the presence of the monopolistic mode of operation actually led to the degeneration of the functions of the administration and in this regard, the lack of tools for database administration.

- a positive feature is the relatively modest hardware requirements from the desktop DBMS. Quite efficient applications developed, for example, on Clipper, worked on IBM286.

The distributed DBMS. Database technology has taken us from methods of data processing in which each application defined and maintained its own data, to one in which data is defined and administered centrally. It is well known that history develops in a spiral, so after the process of "personalizing" the reverse process is integration. Multiplying the number of local networks, more and more information is passed between computers, the acute problem of consistency of data stored and processed in different locations but are logically connected to each other,

there are tasks related to parallel processing of transaction sequence of operations on the database, transferring it from one consistent state to another consistent state. Multiplying the number of local networks, more and more information is passed between computers, states the problem of consistency of data stored and processed in different places, but logically linked with each other, appear the tasks connected to parallel processing of *transactions* – the sequence of operations on the database, transferring it from one consistent state to another consistent state. The successful solution of these tasks leads to the appearance of distributed databases that preserve all the advantages of a desktop database and at the same time allows organizing parallel processing of information and supporting the integrity of the database.

Distributed database is a logically interrelated collection of shared data (and a description of this data) physically distributed over a computer network. Distributed DBMS is the software system that permits the management of the distributed database and makes the distribution transparent to users. Features of this stage:

- almost all modern DBMS support a full relational model;
- most modern DBMS is designed for multi-platform architecture, that is they can operate on computers with different architectures and on different operating systems with user access to data managed by DBMS on different platforms, almost indistinguishable;
- the need to support multi-user database and the possibility of decentralized data storage has required the development of database administration tools, with the implementation of the General concept of data protection;
- the need for new implementations have caused a serious theoretical works on optimization of implementations of distributed databases and distributed transactions and queries implement the results in commercial DBMSs;
- all modern DBMS have the means to connect the client applications that are developed using a desktop DBMS, and tools to export data out of desktop DBMS formats;
- a number of standards have been developed in the framework of languages of description and manipulation since SQL89, SQL92, SQL99, and technologies for data exchange between different DBMS, which can be attributed to the ODBC (Open DataBase Connectivity) protocol, proposed by Microsoft Corporation.;

Object-Oriented DBMSs. We consider only some definitions. *Object-Oriented database model* is a (logical) data model that captures the semantics of objects supported in object-oriented programming. *Object-Oriented DB* is a persistent and sharable collection of objects defined by an object-oriented database model. *Object-Oriented DBMS* is the manager of an object-oriented DB.

Web Technology and DBMSs. The Web is a compelling platform for the delivery and dissemination of data-centric, interactive applications The Web's ubiquity provides global application availability to both users and organizations. As the architecture of the Web has been designed to be platform-independent, it has the potential to significantly lower deployment and training costs. Organizations are now rapidly building new database applications or reengineering existing ones to take full advantage of the Web as a strategic platform for implementing innovative

business solutions, in effect becoming Web-centric organizations. There are some of the different approaches to integrating databases into the Web environment.

The Web as a platform for database systems can deliver innovative solutions for both inter- and intra-company business operations. Unfortunately, there are also disadvantages associated with this approach.

4 Lecture #4. System analysis of the subject domain

The content of the lecture: the questions of the analysis of the modeled part of the real world.

The goal of the lecture: to learn the approaches of the system analysis of the modeled subject domain.

4.1 The subject domain of the information system

There is a stage that *precedes* the stage of database design. There are many questions that must be answered before a data warehouse can be built. Data warehouse is subject oriented; that is, it is oriented to specific selected subject areas in the organization. The DB should be structured and oriented to subject domain. Each subject domain contains objects which are relevant to that subject domain.

Subject domains are roughly classified by the topics of interest to the business (users). To extract a candidate list of potential subject domains, you should first consider what your business interests are. To help in determining the subject domain, you could use a technique that has been successful for many organizations, namely, the 5WH rule: that is, when, where, who, what, why, and how of your business interests. For example, for answering the "who" question, your business interests might be in customer, employee, manager, supplier, business partner, and competitor.

After you extract a list of candidate subject domains, you decompose, rearrange, select, and redefine them more clearly. As a result, you can get a list of subject domains that best represent your organization. We suggest that you make a hierarchy or grouping with them to provide a clear definition of what they are and how they relate to each other.

The model of this stage must express the information about the subject domains in a form independent from the used DBMS.

So, at the first stage of design it is necessary to fulfill the *system analysis of the subject domain.* From the point of view of database design within the frame of systems analysis, it is necessary to conduct a detailed verbal description of the domain objects and the real relations that exist between the described objects. It is desirable that this description let to correctly identify all the relationships between the domain objects.

In general, there are two approaches to the selection of the composition and structure of the subject domain:

- the *functional approach* is applied when the DB is created to serve the information needs of some groups of users and complexes of tasks, and functions of these groups and tasks are known in advance. In this case we can clearly distinguish the minimal necessary set of domain objects, that must be described;

- the *subject approach* is applied when the information needs of future users of the database are not fixed. They can be multidimensional and very dynamic. We cannot precisely select the minimum set of domain objects that need to be described. In the description of the subject domain in this case are included such objects and relations which are most characteristic and the most significant for it.

In practice it is recommended to use some compromise variant that, on the one hand, is focused on a specific task or functional needs of users, on the other hand, takes into account the ability to add new applications.

So, first steps of description of the subject domain are:

1) Determining the purpose of your database. It is a good idea to write down the purpose of the database on paper, how you expect to use it, and who will use it. If the database is more complex or is used by many people, as often occurs in a corporate setting, the purpose could easily be a paragraph or more and should include when and how each person will use the database. The idea is to have a well developed mission statement that can be referred to throughout the design process. Having such a statement helps you focus on your goals when you make decisions.

2) Finding and organizing the required information. To find and organize the information required, start with your existing information. Identify and list each of these items. As you prepare this list, don't worry about getting it perfect at first. If someone else will be using the database, ask for their ideas, too.

3) Next, consider the types of reports or mailings you might want to produce from the database. What information would you place on the report? List each item. Do the same for the form letter and for any other report you anticipate creating. By this way you identify items you will need in your database.

4) Think about the questions you might want the database to answer. Such questions help you define an additional item of data.

System analysis should end with:

- a detailed description of information about the domain objects, which is required for solving the specific tasks and which should be stored in a database;

-formulation of specific tasks that will be solved with the use of this database;

- description of incoming documents, which serve as the basis for filling in the database data;

- a brief description of algorithms of solving the tasks;

- description of outcome documents to be generated in the system.

4.2 Examples of description of the subject domain

Example 1. Let it be required to develop an information system for a company which is engaged in publishing activities. The database is created for information support of the editors, managers and other employees of the company

and must contain data about the employees, books, authors, the financial condition of the company and provide the opportunity to receive various reports.

In accordance with the subject domain the system is built with the consideration of the following features: each book is published as part of the contract; the book can be written by several authors; the contract must be signed by one Manager and by all the authors; each author can write multiple books (different contracts); the order in which the authors are on the cover affects the amount of the salary; if the employee is the editor, he can work simultaneously on several books; each book can have several editors, one is the Executive editor; each order is made per customer; in the purchase order may be listed several books.

Each published book has the following characteristics: authors, title, edition, date released, price of one copy, the total cost of publishing, and the publication royalties.

About the authors of the books the following information is needed: surname, first name, middle name, TRN, passport details, home address, phone numbers. For the authors it is necessary to store information about written books.

About each employee of the publisher the following information is stored: surname, first name, middle name, personnel number, sex, date of birth, passport details, TRN number, position, salary, home address and phone numbers. For editors it is necessary to store the information about the edited books; for managers – the information on signed contracts.

To reflect the financial position of the company in the system you need to consider the orders on the books. To order you must store the order number, customer, address of customer, date of order, date of fulfillment, list of books ordered by specifying the number of copies.

The system is created to service the following groups of users: administration (management); managers; editors; employees serving orders.

Users must have the possibility of information support of the database: maintaining the database (writing, reading, modifying, deleting, archiving); provision of the logical consistency of the database; protection of data from unauthorized or casual access (definition of access rights).

Also, the database users must be able to: get a list of all current projects (books, in print and on sale); get a list of all editors working on the books; to get the full information about the book (the project); to get the information about a specific author (with the list of all books); to receive information about sales (one or all projects); to determine the total profit from sales on current projects; to determine the fees of the author on a particular project.

Example 2. Let it be required to work out the information system for the automation of accounting of receipt and issuance of books in the library. The system must provide the regimes of operation of the system catalog describing the list of areas of knowledge for which there are books in the library. Inside the library the spheres of knowledge in the systematic catalogue can have a unique extension number and full name. Each book may contain information from multiple areas of knowledge. Each book in the library may be present in several copies.

Each book stored in the library, is characterized by the following parameters: a unique code; name; names of authors (can be absent); place of publication (city); publisher; year of publication; number of pages; cost of book; number of copies of the book in the library. The books can have the same name, but are differentiated by a unique code (ISBN).

In the library there is the card index of readers. At each reader in the file is entered the following information: surname, name, patronymic name; home address; phone (we assume that we have two phones - work and home); date of birth. Each reader is assigned a unique number of library cards. Each reader may simultaneously hold on hand no more than 5 books. The reader should not simultaneously hold more than one copy of the book of the same name.

Each book in the library may be present in multiple copies. Each instance has the following characteristics: a unique inventory number; the call number of a book, which coincides with the unique code from the description books; placement at the library.

In the case of the issuance of the copy of the book to the reader the library keeps a special liner, which should have the following information recorded: ticket number of the reader, who has taken the book; date of issuance of the book; the return date.

There are the following limitations on the information in the system: the book may not have a single author; every reader can hold no more than 5 books; each reader during registration in the library should give the phone for communication; each area of knowledge may contain links to many books, and each book can relate to different areas of knowledge.

With this information system the following user groups work: the librarians; the readers; the administration of the library.

When a librarian works with the system he/she must be able to solve the following tasks: take the new books and record them in the library; to perform the cataloguing of the books; to write-off old and unpopular books; keep a record of the books issued to the readers; to write-off the books lost by the reader; conduct the closing of the subscription of the reader,

The reader must be able to perform the following tasks: to view the system catalog; to obtain a complete list of books in the library for the selected area of knowledge; to get a list of books of the selected author available in the library.

The administration of the library should be able to get the information about the debtors - the readers of the library; information about books that are not popular, i.e. no instance which is not in the hands of readers; the information on the cost of a particular book, in order to establish the possibility of reimbursement of the cost of the lost books or the possibility of replacing it with another book; information on the most popular books, i.e., all instances which are in the hands of readers.

These examples show that before beginning of the development, it is necessary to have accurate representation about what should be done in our system, which users will work with it, what tasks each user will solve.

Unfortunately, often in relation to databases it is considered that all details can be determined later, when the project is already created. The lack of clear goals of the database creation can negate all the efforts of developers, and the project will turn out bad, uncomfortable, neither appropriate for the realistically modeled object, nor for the tasks that need to be addressed with use of this database.

5 Lecture#5. The principles of database design

The content of the lecture: the three-level architecture of databases.

The goal of the lecture: examine external, conceptual and internal levels of DB and the concepts of data models.

5.1 The architecture of the database. The physical and logical independence

A major aim of a database system is to provide users with an abstract view of data, hiding certain details of how data is stored and manipulated. Therefore, the starting point for the design of a database must be an abstract and general description of the information requirements of the organization that is to be represented in the database.

An important aspect of the development of data access methods was the idea of separating the logical structure and data manipulation, as they are understood by the users from the physical imagination of required computer equipment.

In the process of scientific research devoted to on how DBMS should be arranged, there have been proposed different methods of implementation. The most viable of them was proposed by the American Committee for the standardization of ANSI (American National Standards Institute) the three-level system of organization of a database. The levels form a *three-level architecture* comprising an *external*, a *conceptual*, and an *internal* level.

The way users perceive the data is called the *external level.* The level of external models is the top-most level, where each model has its own "vision" data. Each application sees and works out only those data that are necessary to this application. *External level* is the structural level of the database, defining user views of the database. This level is the closest to the users and is linked with individual users to imagine data in a database. The combination of all these custom views is the external level.

The middle level in the three-level architecture is the conceptual level. Conceptual level is the community view of the database. This level describes *what* data is stored in the database and the relationships among the data. In fact the conceptual level reflects a generalized domain model (real world objects), for which the database was created. Like any model, a conceptual model reflects only the essential, from the point of view of processing, especially real-world objects.

Conceptual level is the structural level of the database that defines the logical database schema. Conceptual database design performed at the conceptual level includes the analysis of information needs of users and defining the required data elements. The result of this design is a conceptual schema, or a single logical description of all data elements and relationships between them.

The way the DBMS and the operating system perceive the data is the *internal level*, where the data is actually stored using the data structures and file. *The internal level* is a structural level of the database that defines the physical view of the database. It is associated with the physical data storage: disk drives, physical addresses, indexes, pointers, etc. No user (as a user) does touch this level.

Physical level is the actual data located in files or in page structures, located on external media. It is the physical representation of the database on the computer. This level describes how the data is stored in the database.

A major objective for the three-level architecture is to provide data independence, which means that upper levels are unaffected by changes to lower levels. There are two kinds of data independence: *logical* and *physical*.

Logical data independence refers to the immunity of the external schemas to changes in the conceptual schema. And it implies the possibility to change one application without adjustment of other applications running with the same database.

Changes to the conceptual schema should be possible without having to change existing external schemas or having to rewrite application programs.Clearly, the users for whom the changes have been made need to be aware of them, but what is important is that other users should not be.

Physical data independence refers to the immunity of the conceptual schema to changes in the internal schema. It implies the possibility to transfer the stored information from one media to the other while maintaining operability of all the applications working with this database. This is exactly what was missing when using file systems.

Changes to the internal schema, such as using different file organizations or storage structures, using different storage devices, modifying indexes, or hashing algorithms, should be possible without having to change the conceptual or external schemas. From the users' point of view, the only effect that may be noticed is a change in performance. In fact, deterioration in performance is the most common reason for internal schema changes.

The selection of the conceptual level helped to develop an apparatus of the centralized management database. If the external level is associated with the "private" user views, the conceptual level, one can imagine determining a *generalized* representation of users. This view is closer to the data "as is", than what their users see. That is, there can be many "external views", each of which consists of a representation of parts of the database, and can only be "conceptual view", consisting of an abstract view of the database as a whole. Also there is only "internal" performance, reflecting the entire database as really stored.

5.2 Conceptual data model

We need to describe the data requirements of an organization in a way that is readily understandable by a variety of users. What we require is a higher-level description of the schema: that is, a *data model.*

As we already said the concept of "data" in the concept database is a set of specific values, parameters characterizing the object, condition, situation or any other factors. The data does not have a certain structure; the data becomes information when a user specifies a certain structure to them that is aware of their semantic content.

Therefore, the central concept in the database area is the notion of model. There is no single definition of the term, different authors define this abstraction with some differences, but, nevertheless, it is possible to point out what is common in these definitions.

Recall that a database is a collection of data structured in a certain way. The structure of the database captures the structure of the subject domain.

Model is the representation of reality, reflecting only the chosen parts. In the context of databases the *data model* is the methods of structuring data.

In accordance with the previously discussed three-level architecture we are faced with the notion of data models in relation to each level.

The core of any database system is the conceptual model. At the conceptual level the conceptual design is fulfilled, i.e., a conceptual database schema is created. At this step detailed models of the custom data views are worked out, and then they are integrated into a conceptual model fixing all the data elements of data that the database will contain.

A conceptual model is developed after a verbal description of the subject domain. This model should include such formalized description of the subject domain that will be easy to "read" not only for the database experts, and it should not be tied to a specific DBMS.

The choice of the system database management is a separate task, for proper solution, we need to have a project that is not tied to any particular DBMS.

Design of DB is primarily associated with the attempt to represent the semantics (meaning) of the subject domain in the database model. The problem of representing the semantics had taken developers' interest for a long period of time, and in the seventies several data models were proposed, which were called *semantic* models.

Semantic modeling is modeling of data structure, based on the meaning of these data. Such patterns include semantic data model proposed by Hammer and MC-Leon in 1981, the functional data model of Shipman, also established in 1981, the model "Entity—Relationship" proposed by Peter Pin-Shan Chen in 1976, and a number of other models.

All models had their positive and negative sides, but only the last one could stand the test of time. Later many authors developed their own variants may be

better “options” of such models (Martin notation, IDEF1X notation, notation of Barker and others).

Here as a semantic modeling tool are used different versions of the *entity-relationship* (ER) *diagrams*. At the moment, the Chen model has become the de facto standard in conceptual modeling of databases. Was accepted the abbreviation "ER model". The diagrams "Entity-Relationship" come from a single idea: the picture is always clearer than textual description.

All these diagrams use a graphical image of domain entities, their properties, and relationships between entities.

6 Lecture#6. Basic definitions of conceptual model of data

The content of the lecture: the important concepts of conceptual model of data.

The goal of the lecture: to learn the composite parts of a conceptual model of data.

6.1 Entities and relationships

We need to have a model for communication that is non-technical and free of ambiguities. The conceptual model is one such example. Conceptual modeling is a top-down approach to database design.

We introduce the basic concepts of the conceptual model. Although there is general agreement about what each concept means, there are a number of different notations that can be used to represent each concept diagrammatically.

The conceptual model begins by identifying the important data called *entities* and *relationships* between the data that must be represented in the model.

Entities are real-world objects about which we collect data.

The next basic concept is the *entity type*, which represents a group of ‘objects’ in the ‘real world’ with the same properties. An entity type has an independent existence and can be objects with a physical (or ‘real’) existence or objects with a conceptual (or ‘abstract’) existence.

Entity set is a set of entities of the same type. The *entity-element* is the specific element of entity set. Schematically each entity is shown as a rectangle labeled with the name of the entity, which is normally a singular noun (the first letter of each word in the entity name is upper case). If the entity name includes several words, first letter of each word in the entity name is upper case (for example, HouseOwner). Also we use the bold font for an entity name.

The next step is to determine the relationships between the entities and to determine the cardinality of each relationship. The relationship is the connection between the entities, just like in the real world: what does one entity do with the other, how do they relate to each other? For example, customers buy products, products are sold to customers, a sale comprises products, a sale happens in a shop.

Schematically each relationship is shown as a line connecting the associated entity types, labeled with the name of the relationship. Normally, a relationship is

named using a *verb.* Whenever possible, a relationship name should be unique for a given model. We will use italic font for relationship name. The name of a relationship is displayed with the first letter in lower case and, if the name has more than one word, with the first letter of each subsequent word in upper case (for example, *supervises* or *manages*) or a short phrase including a verb (for example, *leasedBy*). The entities involved in a particular relationship are referred to as participants in that relationship.

The number of participants in a relationship is called the *degree* of that relationship. A relationship of degree two is called *binary*. In fact the most common degree for a relationship is binary. A relationship of degree three is called *ternary*. The relationships between *n*entities (relationship of high order) are called *n-ary relationship.* The term 'complex relationship' is used to describe relationships with degrees higher than binary.

If the same entity participates more than once in different roles a relationship is called a *recursive* relationship. Consider a recursive relationship called '*supervises'*, which represents an association of staff with a supervisor where the 'Supervisor' is also a member of staff. In other words, the 'Staff 'entity element participates twice in the '*supervises'* relationship; the first participation as a supervisor, and the second participation as a member of staff who is supervised ('supervisee'). Recursive relationships are sometimes called *unary* relationships.

Relationships may be given *role* names to indicate the purpose that each participating entity plays in a relationship. Role names can be important for recursive relationships to determine the function of each participant. The use of role names to describe the 'supervises' recursive relationship: for example, the first participation of the 'Staff' entity in the '*supervises'* relationship is given the role name 'supervisor' and the second participation is given the role name 'supervisee'. Role names are usually not required if the function of the participating entities in a relationships unambiguous.

The next step is to determine the cardinality of each relationship. Cardinality describes the maximum number of possible relationship occurrences for an entity participating in a given relationship. Cardinality exists in four types: one-to-one, one-to-many, many-to-one, and many-to-many.

If the maximum cardinality of relationship in both directions is equal to one, the relationship is called the relationship one-to-one (written 1:1 or 1-1).

If the maximum cardinality of the relationship in one direction equals one, and in another direction equals many, relationship is called one-to-many (denoted 1:* or *1:M).

If the maximum cardinality of the relationship in both directions are equal to many, the relationship is called relationship many-to-many (denoted by a *:* or M:M).

6.2 Attributes

The particular properties of entity are called *attributes*. The set of allowable values for one or more attributes is called *attribute domain*. Attributes may share a domain. For example, the address attributes share the same domain of all possible addresses. Domains can also be composed of domains. For example, the domain for the address attribute is made up of subdomains: street, city, and postcode. The domain of the name attribute is more difficult to define, as it consists of all possible names. It is certainly a character string, but it might consist not only of letters but also of hyphens or other special characters.

Attributes can be classified as being: simple or composite; single-valued or multi-valued; or derived. An attribute composed of a single component with an independent existence is a *simple* attribute. Simple attributes are sometimes called *atomic* attributes.

An attribute composed of multiple components, each with an independent existence is called a *composite* attribute. For example, the 'address' attribute can be subdivided into 'street', 'city' and 'postcode' attributes. The decision to model the address attribute as a simple attribute or to subdivide the attribute into street, city, and postcode is dependent on whether the user view of the data refers to the address attribute as a single unit or as individual components.

The majority of attributes are *single-valued*, that is, an attribute holds a single value. Some attributes have multiple values. For example, we can have multiple values for the 'telephone Number' attribute. Such attribute is called *multi-valued* attribute. A multi-valued attribute may have a set of numbers with upper and lower limits. For example, the 'telNo' attribute of the 'Branch' entity has between one and three values. In other words, a branch may have a minimum of a single telephone number to a maximum of three telephone numbers.

Database can have derived data. Derived data is data that is derived from the other data that you have already saved. In this case the 'sum total' is a classical case of derived data. Such data are represented by derived attributes. So, the derived attribute is an attribute that represents a value that is derivable from the value of a related attribute or set of attributes, not necessarily in the same entity.

If for some element of entity sets the value of some attribute is not defined, then we say that this attribute has an empty value for element entity of the set. Attributes must be separated from the entities as attribute values may change, while they describe the entity remains the same. This does not mean that the values of all attributes are changed.

You often need to find the attributes whose values do not change because they can be used as keys. The *key* or *identifier* is the attributes whose values uniquely identify the element of the entity set. The absence of a key has no effect.

A *candidate* key is the minimal number of attributes, whose value(s) uniquely identify each entity occurrence. An entity may have more than one candidate key. The candidate key that is selected to uniquely identify each occurrence of an entity is called *primary* key. In some cases, the key of an entity is composed of several attributes, whose values together are unique for each entity occurrence but not separately. Such key will be a *composite* key.

Schematic representation of attributes. If an entity type is to be displayed with its attributes, we divide the rectangle representing the entity in two. The upper part of the rectangle displays the name of the entity and the lower part lists the names of the attributes. The first attribute(s) to be listed is the primary key for the entity, if known. The name(s) of the primary key attribute(s) can be underlined or bold (or labeled with the tag {PK}) (Figure 6.1).

The name of an attribute is displayed with the first letter in lower case and, if the name has more than one word, with the first letter of each subsequent word in upper case (for example, address and telNo).

For composite attributes, we list the name of the composite attribute followed below and indented to the right by the names of its simple component attributes. For derived attributes, we can use prefix the attribute name with a '/'.

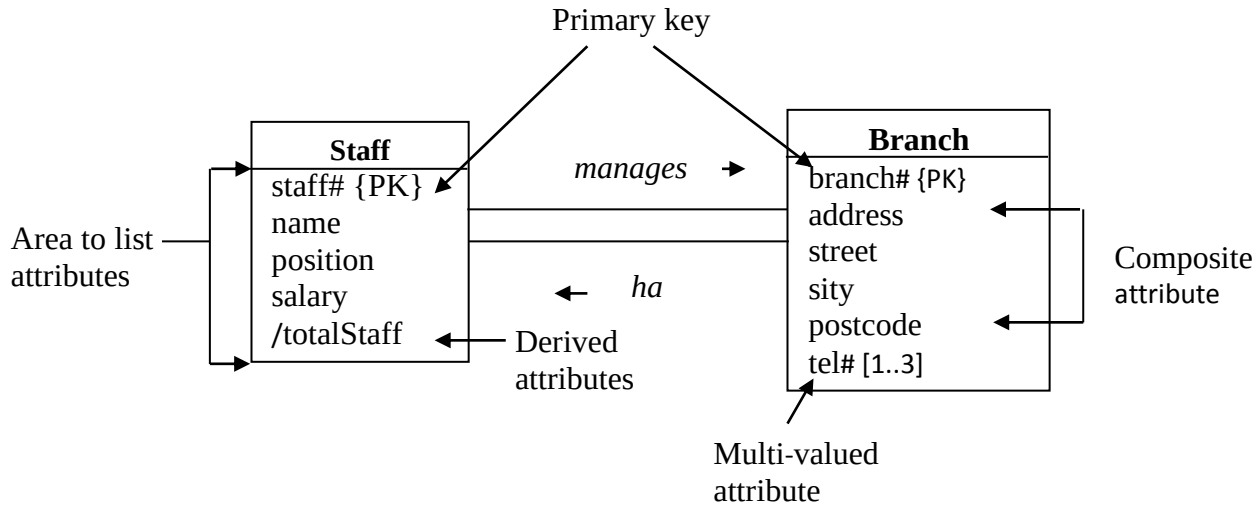

Figure 6.1 – Diagramed representation of entities and their attributes

Attributes on relationships. Attributes can also be assigned to relationships. For example, consider the relationship Advertises, which associates the Newspaper and PropertyForRent entities types as shown in Figure 11.1. To record the date the property was advertised and the cost, we associate this information with the Advertises relationship as attributes called DateAdvert and Cost, rather than with the Newspaper or the PropertyForRent entities.

Schematic representation of attributes on relationships. We represent attributes associated with a relationship type using the same symbol as an entity type. However, to distinguish between a relationship with an attribute and an entity, the rectangle representing the attribute(s) is associated with the relationship using a dashed line. For example, Figure 6. shows the *advertises* relationship with the attributes 'dateAdvert' and 'cost'.

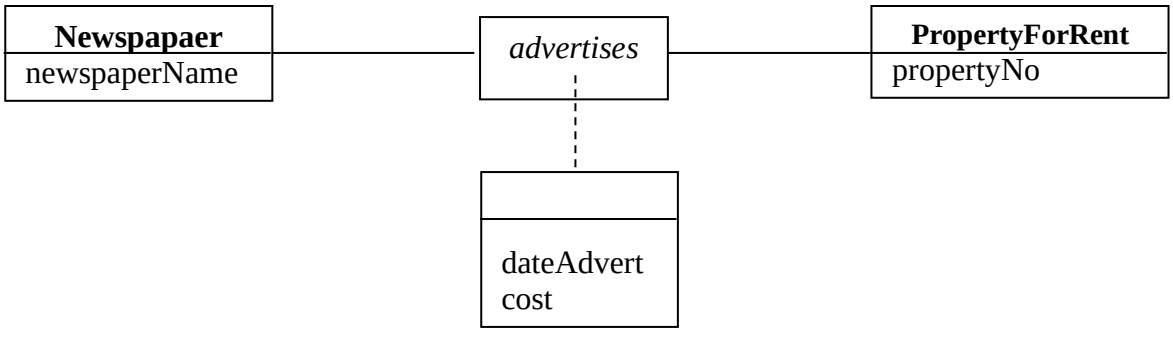

Figure 6.2 – An example of a relationship called *advertises* with attributes ‘dateAdvert’ and ‘cost’

Later we’ll consider more closely the question about the attributes of relationships.

7 Lecture #7 Creating Entity Relationship Diagram

The content of the lecture: principles of creating Entity Relationship Diagram (ERD).

The goal of the lecture: to learn additional definitions of conceptual modeling.

We'll consider the using of Entity-Relationship model that represents the database as entities and relationships between them. Currently there is no single generally accepted system of notation for ER-model, it uses a different graphical notation, but understandable for one, and easy to understand for the others.

As we mentioned above a much-used notation of ERD is the notation, where entities are represented as rectangles and the relationships between the entities are represented as lines between the entities. The signs at the end of the lines indicate the type of relationship. You can indicate the side of the relationship that is mandatory for the other to exist through a dash on the line. Not mandatory entities are indicated through a circle. "Many" is indicated so: relationship-line splits up in three lines. At the figure 7.1 ER-diagram shows the connection between entities ‘Students’ and the ‘Teachers’, where the relationship is the supervision of graduation projects: every student has only one supervisor, but the same teacher may lead several graduate students, relationship is "one-to-many". Note: in both entities some attributes (*id, name*) have clarification (*students* or *teacher*). It’s one of the rules of attributes naming (in the future it will help to distinguish between equally named attributes of different entities).

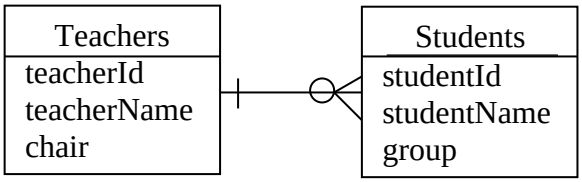

Figure 7.1 – The example of presenting entities and relationship

Most of the business problems require the use of compound entities. However the relationship can associate three or more object sets (*n-ary* relations). We illustrate this concept by example.

Let’s consider the experience of the company engaged in international trade of goods. The concept of the company is as follows: to find in different countries manufacturers, whose products always satisfy the highest quality standards; to find firms that sell goods; to establish business contacts between manufacturers and sellers, supplying the last merchandise purchased from selected manufacturers. The company has its offices in different countries; the staff of each office consists of sellers and buyers. Let’s create the conceptual data model of these companies. The company has its customers, sellers, the products, the manufacturers of these products. We select the following entities: ‘product’, ‘customer’, ‘seller’, and ‘manufacturer’. Specify appropriate relations between them with their cardinality, as shown in figure 7.2.

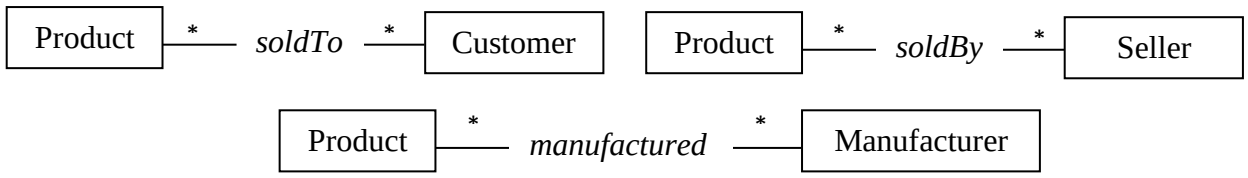

Figure 7.2 – The relationships between the entities

Solving ‘many-to-many’ relationships. So, we have three 'many-to-many' relationships. Many-to-many relationships are not directly possible in a database. There are a number of records from one *entity* belongs to a number of records from another *entity*. Somewhere you need to save which records these are and the solution is to split the relationship up in two one-to-many relationships. The relationship “many-to-many” -the termites lf is the entity consisting of pairs of elements from two entities connected by the relationship. The relationship is treated as an entity, called a *composite entity*. Composite *entity* can be given names and include them in relationships, just like a simple *entity*.

Suppose we want to know the quantity of each product sold to each client. We need to assign an attribute ‘*number’* one of the entities. If it is the attribute of the entity ‘product’ we will not be able to distinguish the quantity of the goods sold to different customers, if the attribute to be assigned to the ‘client’, we will not be able to distinguish between goods sold to the customer. Therefore, the ‘quantity’ is an attribute of *relationship* between ‘product’ and ‘customer’, not ‘product’ or ‘customer’ individually. This relationship forms a composite entity. Graphically the composite entity is indicated by a rectangle drawn around the relationship and the participating entities (figure 3). Let’s call this entity of ‘Sales’. So, the attitude "*soldto*" itself is an entity (‘Sales’) or a composite entity, which is assigned to the attribute ‘quantity’. For best performance and convenience, the attributes of composite entities are shown in an oval (figure 7.3).

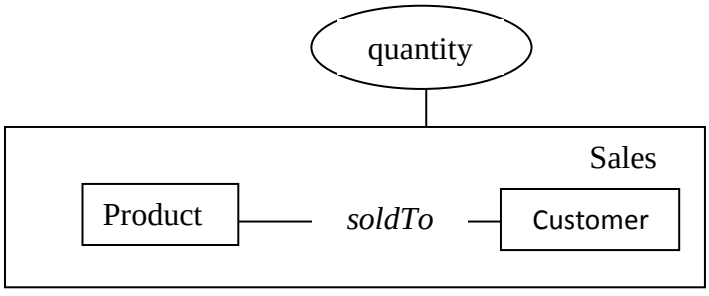

Figure 7.3 – The model shows the quantity of sales correctly

Assume that you want to record the number of sales of each product sold to each customer by a *specific* seller. Then we connect the relationship "*soldTo*" with the 'Seller' entity and assigned attribute 'quantity' to this new relation. The model is more conveniently presented in one ternary relationship (figure 7.4). Among the entities of our model, there are producers of goods. Display this entity and the cardinalities of the relationships on the diagram. Of course, in this diagram you must add the attributes of the entities.

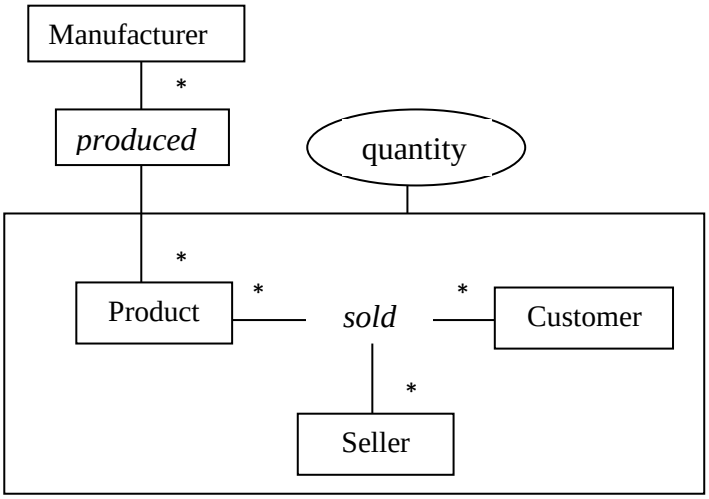

Figure 7.4 – The full model of sales

To illustrate another concept in conceptual modeling, consider the following example. A building company builds a variety of buildings. All buildings require a variety of materials in various quantities. We present ER-diagram for the relationship between the buildings and materials (figure 7.5).

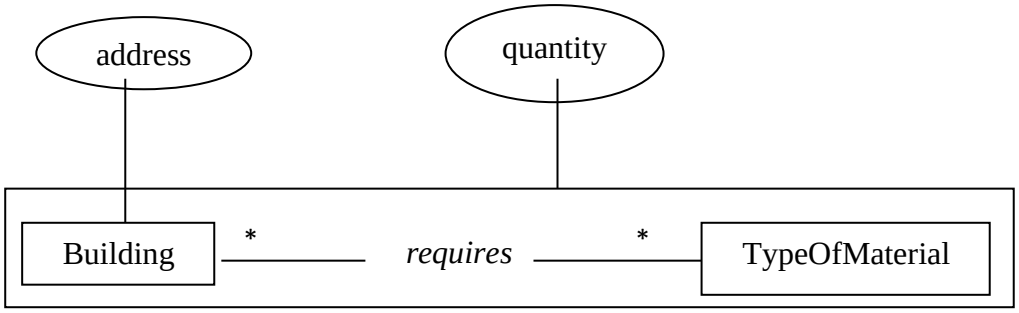

Figure 7.5 – Relationship between the buildings and materials

The cardinality of the relationship between the entity 'Building' and 'TypeOf Material' is'many-to-many;. The 'address; attribute applies only to the 'Building'. The rectangular around the relationship'*requires'* shows that we consider it as a composite entity. This entity has the attribute 'quantity, the elements of this entity are pairs: the building and type of material.It is important to note that in this

example entity ‘TypeOfMaterial’ is a *conceptual*, not a *physical* entity. The *conceptual*entity is the entity denoting type of things, that is, the elements of this entity are abstract concepts. That is, each element of this entity identifies the *type* of material, not a physical "piece" of material. A *physical*entity is the entity whose elements are the physical objects.

Such notion of the conceptual entity as opposed to the physical entity is often used in conceptual data modeling. For example in model of the library DB in the entity‘Books’ information about the *conceptual* booksis stored, *physical* books is*instances* of books. Although the reader may not distinguish between these concepts, in order to decide how to model the data, you need to figure out what information is needed to the user of the database.

We'll continue the developing of a conceptual model. When scheduling work, the firm varies the composition of the teams. Workers are assigned to various teams in accordance with their qualification. A worker can lead one team and work in the other an ordinary worker. Now we show the formation of teams and assignment of workers and supervisors. Another example of a conceptual entity – ‘TypeOfTeam‘ is not *specific* teams but*types* of teams (team of fitter, masons, etc.). The relationship between the building and the type of team is a specific team assigned to perform on this building this type of work. This relationship (‘many-to-many’) will be considered as an entity, let's call it the ‘Team’.Now we’ll represent the assignment of workers and foremen on the team: the cardinality of the relationship ‘a*ssignedInTeam*’ will be ‘many-to-many’ (each employee can work in different teams); the relationship of "*isTheForeman*" has the cardinality "one-to-many" because the team has one foreman, but one person can lead multiple teams (figure 7.6).

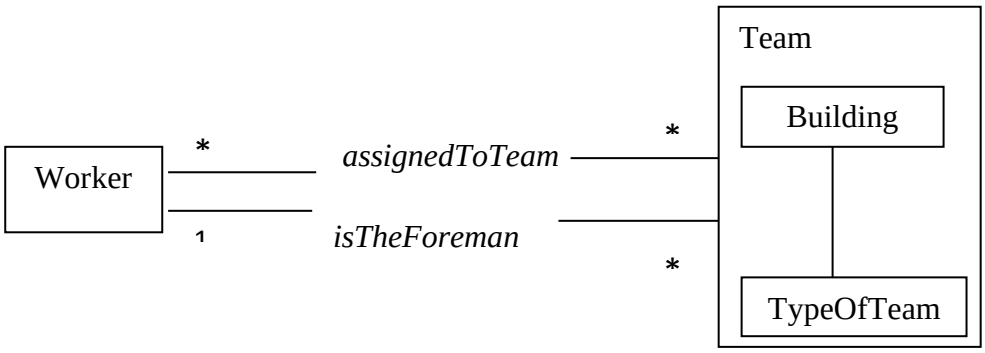

Figure 7.6 – The model of formation of teams

Figure 7.7 presents the combined chart, showing the complete data model for a building company.

8 Lecture #8 Methods of data modeling

The content of the lecture: historical development of methods of data modeling.

The goal of the lecture:to learn the methods of structuring the data.

8.1 Hierarchical and network data models

As we said in the previous lecture, the database system must be able to represent two types of objects: entities and their relationships, and between them there are the fundamental differences: the relationships are a special kind of objects. Three approaches to data modeling (hierarchical, network and relational) differ in

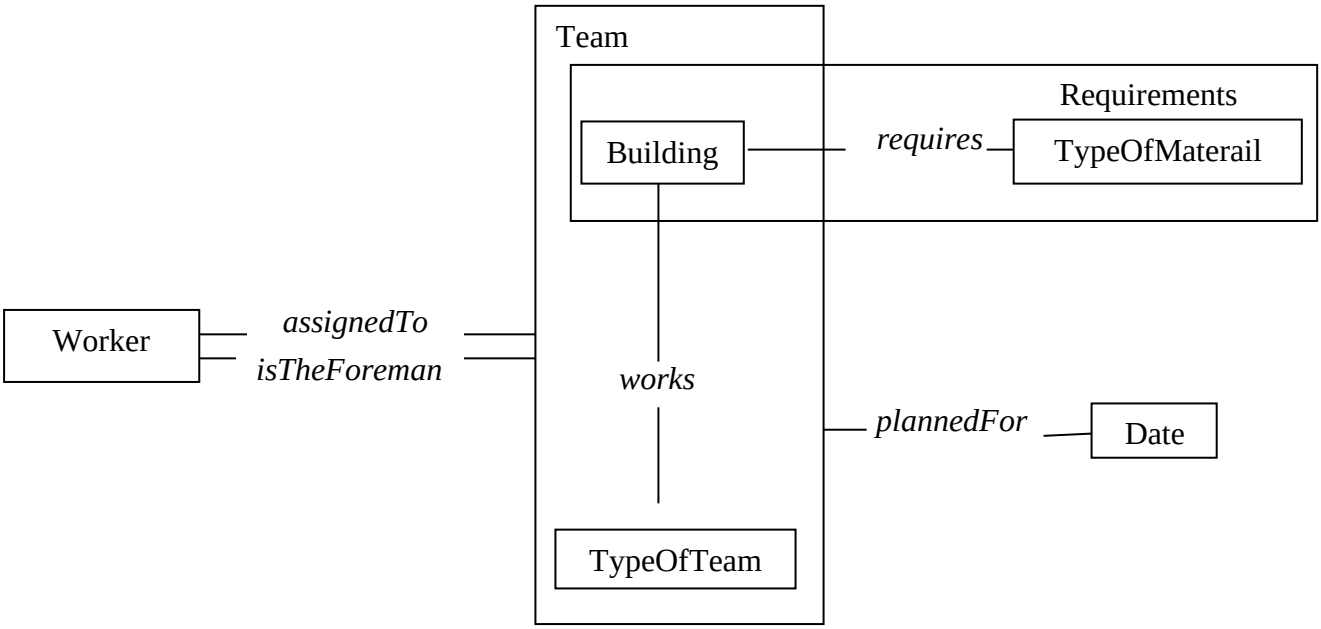

Figure 7.7 – The data model for a building company

the way they allow the user to imagine and process the *relationships*. These are hierarchical, network and relational models.

The Hierarchical model of data. The first information system using a database that appeared in the mid-60s of the 20th century was based on a hierarchical model. The *hierarchical* model is a data model in which relationships between the data have the form of hierarchies. Historically these models have appeared before and currently they exist today, but typically in legacy applications. In real world many connections correspond to the hierarchy, where one entity acts as the parent, and can be relate to any number of subordinate entities. The hierarchy is simple and natural in displaying the relationship between classes of entities.

In a hierarchical DB a user defined interpretation of a string of bytes is called a *record type*. A byte string of the appropriate length interpreted in the appropriate way is called an *instance* of a record type. A collection of record type which forms a *tree* is a hierarchy.

A hierarchical data base is *a collection of instances of records* such that each instance (except instances of the record type at the root of the hierarchy) has exactly

one parent. Moreover, its parent is of the same type as its parent record type in the hierarchy.

The hierarchical data model organizes data in a tree structure. There is a hierarchy of parent and child data segments. For example, in a sales order processing system, a customer may have many invoices raised to him and each invoice may have different data elements. Thus, the root level of data is customer, the second level is invoice and the last level is contents of invoice, such as invoice number, date, product, quantity, etc. (figure 8.1).

However, the lower levels are owned by higher level data elements, and elements at the same level have no connection at all. As a result, the query such as what products are purchased by which customer, in the above example, shall be difficult to carry out in the hierarchical structure.

Thus, where there are many-to-many relationships between two entities, this model would not be appropriate.

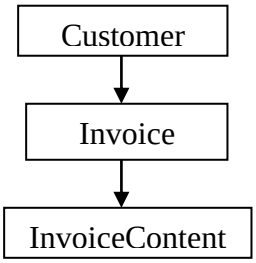

Figure 8.1 - A hierarchical model of relationships

Suppose we want to add to our hierarchical database of customer information. For example, if customers are the trading companies, the list of stores may be needed for each company. In this case, the chart can take the form as shown in the figure 2.

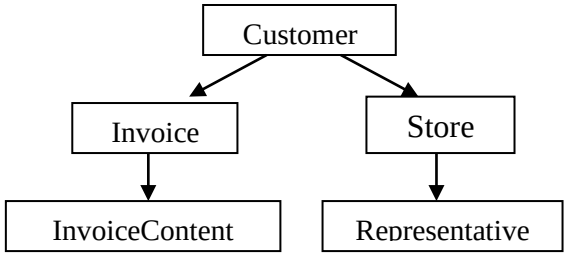

Figure 8.2 – The expanded hierarchical model of relationships

Another example: a company might store information about an employee, such as name, employee number, department, salary. The organization might also store information about an employee's children, such as name and date of birth. The employee and children data forms a hierarchy, where the employee data represents the parent segment and the children data represents the child segment (figure 8.3). If an employee has three children, then there would be three child segments associated with one employee segment. In a hierarchical database the parent-child relationship is one-to- many. This restricts a child segment to having only one parent segment.

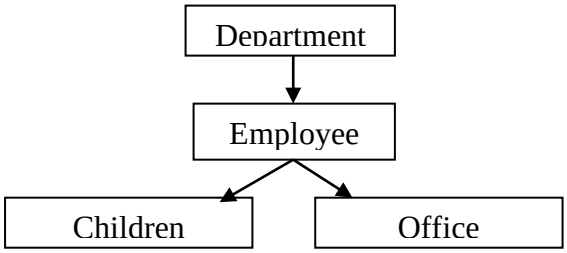

Figure 8.3 – Three-level hierarchy

When working with data having the structure of figure 3 we can observe a problem with hierarchical systems: office information must be repeated for each employee that works in a given office. This wasteful repetition is appeared on by the requirement that each instance of a record have exactly one parent.

Hierarchical model has a drawback, because not all relationships can be represented in a hierarchy. For example, you must have a list of all accounts in the sales made to certain sellers, in order to calculate the bonuses to him. The relationships are shown in the figure 8.4. This chart is not a hierarchy, as one subordinate (child) file has two *ancestors*.

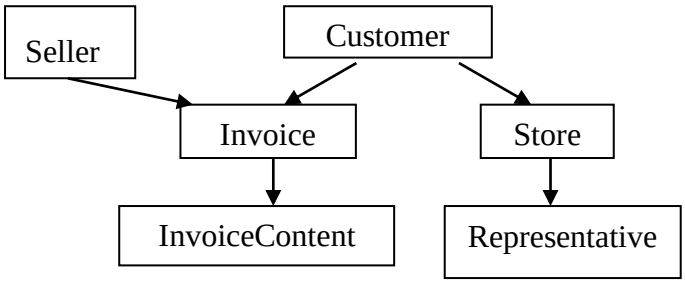

Figure 8.4 - Network model of relationships

In the hierarchical database the files are connected between each other by the *physical index*. The *index* is a physical address that indicates the storage location of the record on the disk.

The Network model of data. The diagrams as in the figure 1 are called *network*. In the network model of database, there are no levels and a record can have any number of owners and also can have ownership of several records. Thus, the problem raised above in the sales order processing will not arise in the network model.

In a network system we use the same record *type* and *instance* of record type definitions. A *set type* is an association between one owner record type and one or more member record types. An *instance of a set type* is binary relation between one instance of an owner record type and instances of member record type can be related to at most one instance of the owner record type. A *network database* is a collection of record instances and set instances.

Above we mentioned the problem of the scheme in the figure 3. The network approach shown in the figure 8.5 avoids the restrictions of this scheme.

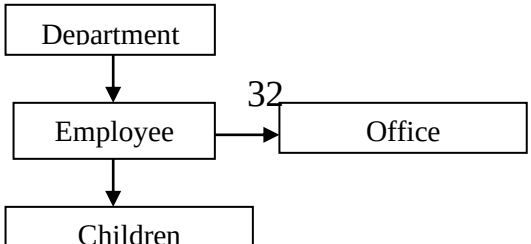

Figure 8.5 – A network structure

The network models support more complex relationships between records from different files. As in the hierarchical systems in the networked database systems to connection the data was used *predefined physical index*.

To understand what the disadvantages are inherent in systems based on physical *indexes*, consider figure 8.6. Here, 'Customer', 'Invoice' and 'InvoiceContent' are connected using physical indexes. Add to the schema entities 'Manufacturer', 'Products' with information about goods and their manufacturers. These entities are also connected (this is shown by the line). The dotted line between 'InvoiceContent' and 'Product' means that between them there is a *logical* connection, as each row of the accounts relates to specific products. If between these files is not *established* connection using of a physical index, then, for example, get information about the producers of goods purchased by customers directly, we are unable. This will require extensive programming.

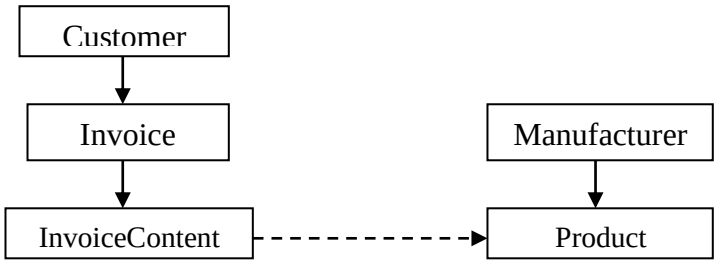

Figure 8.6 - The logical connection not supported by physical index

8.2 Transition to a relational model

In 1970, Dr.EdvardF. Codd (at that time employee at IBM) published a revolutionary article that has seriously shaken the traditional view of databases. He put forward the idea that data should be connected according to their internal *logical relationship*, *not* physical indexes.

In his articleCodd proposed a simple model of the data, according to which all the data are summarized in tables consisting of rows and columns. These tables are called *relations* and the model became known as *relational*.

The relational model defines the data structure), methods of data protection (data integrity), as well as operations performed with the data (manipulation of data). Common misconception that the relational model is so called because it defines relationships between tables. Actually the name of this model derives from the relations underlying.

In relational database systems entire data files can be processed with one command, whereas in traditional systems at a time is processed by only one entry. Method of separation of conceptual and logical levels has brought about a Revolution in the field of database programming. The Codd approach is tremendously improved the efficiency of programming in data bases. Previously the database programming was limited to writing code for physical control of devices intended for data storage. A logical approach to data also made possible the creation of query languages more accessible to users who are not specialists in computer technologies. Relational query languages made the databases available to a broader range of users than before.

The result of activity in creating a relational database management system was the establishment in the second half of the seventies relational systems that support languages such as Structured Query Language (SQL structured query language), Query Language and Query-by-Example (QBE). Today Relational databases are considered as standard for modern commercial systems work with data.

9 Lecture #9. The relational model of data

The content of the lecture: the basic concepts of relational models of data..

The goal of the lecture:to learn the basic definitions of relational models of data.

The relational model is based on mathematical principles derived directly from algebraic set theory and predicate logic. These principles were first applied in the field of modeling data by Dr. Edward F. Codd (at that time working for IBM). He put forward the idea that the data must be related according to *their internal logical relationship but not the physical indexes.* Thus users will be able to combine data from different sources, if the logical information required for such combination, is present in the source data.

Codd proposed a simple model of the data, according to which all the data are summarized in tables consisting of rows and columns. These tables are called *relations* and the model became known as *relational*. This concept of the table relates *only* to the *presentation* of data. No connection with the physical implementation of the records on the disk or in the memory it has.

Common misconception, that the relational model is so called because it defines relationships between tables. Actually (on the contrary) the name of this model derives from the relationships.

In relational model data is represented as relations at the conceptual level, however, it does not give any indication about how they will be implemented at the physical level. Looking at the data from a *conceptual not a physical* point of view Codd proposed another idea: in relational database systems entire data files can be processed *with one command*, whereas in traditional systems they are processed by *only one record* at a time.

The relational model defines:

- the way data are presented (data structure),
- methods of data protection (data integrity),
- how operations are performed with the data (manipulation of data).

In the relational DBMS are at least two conditions:

1) the data are perceived by the users as tables;
2) there are operators that generate new tables from old.

It is important that the result of the operation above the table is also at able that is the result of the operation is the object of the same type as the object on which the action is performed.

In the relational model there are the following concepts: *datatype, attribute, domain, tuple, primary key* and *relation.*

The notion of the *data type* in the relational data model completely corresponds with this notion in programming languages. Usually *character, integer, float, date, Boolean, string* types are selected.

For the storage of information *attribute* is used. Attribute in the relational model has the same meaning as the property of an entity in ER-model. The attribute has name and type. In the relational model attribute is the columns of a relation. Attributes appear at the tops of the columns. Usually, an attribute describes the meaning of entries in the column below.

The structure of the relationship consists of a set of attributes and their types. The number of attributes of the relation is called the *degree* of the relation.

An attribute can take a special *null* value. Null value represents either attributes whose value is *not known*, or do *not exist.*

The order of the attributes is considered insignificant. Therefore, no two attributes of the relationship can have the same names.

A domain is specified with each attribute of a relationship that is, a particular elementary type. The set of all possible values that can take the attribute is called*domain* of the attribute. So, domain is set of atomic values. Composite and multi-valued attributes not allowed.

The element of the domain can be a number, character string, date, but not the complex structure of the type array, list, etc. Two domains of the same attribute match only if they have the same meaning. Two attributes of the same domain do not necessarily have the same name. The values of each attribute belong to the same domain i.e. an attribute can take values from a given set. An attribute has a name and a value, often the name of the attribute matches the domain name.

The name of a relation and the set of attributes for a relation is called the *schema* for that relation. We show the schema for the relation *R* with the relation name followed by a parenthesized list of its attributes:

$$R\ (A_1, A_2, \ldots A_k),$$

WhereA_i- are domain names.

The attributes in a relation schema are a set, not a list. In the relational model, a database consists of one or more relations. The set of schemas for the relations of a database is called a *relational database schema*, or just a database schema.

The rows of a relation, other than the header row containing the attribute names, are called *tuples*. A tuple has one component for each attribute of the relation. The relational model requires that each component of each tuple be *atomic*: the values are not divisible. That is, we cannot refer to or directly see a *subpart* of the value.

A *tuple* is a set of "attribute name, attribute value", and each attribute of a relationship once and once only included in the tuple. It consists of a set of filled attributes. The tuple carries information about the object described by the relation. If the attributes and domains speak about the structure, the tuple is filling with information.

In practice, the relationship can be conveniently represented in the form of a table, the title of which is the scheme of relationships and strings are the tuples of relationships; in this case, the attribute names become the column names of this table.

The order of the tuples *doesn't matter* (tuples may be stored in an arbitrary order). Relationship defined as a set of tuple elements has no order among them, so the value of each record cannot *fully* be repeated.

. Any set of attributes uniquely identifying each tuple of a relational table is called a *super key*. A super key uniquely identifies each tuple within a relation. However, a super key may contain additional attributes that are not necessary for unique identification, and we are interested in identifying super keys that contain only the minimum number of attributes necessary for unique identification.

The *key* of the relation is the minimum set of attributes, that is, a minimal super key.

A key consisting of more than one attribute is called a *composite* key.

Key satisfies two properties:

- two distinct tuples cannot have identical values for (all) attributes in key;
- cannot remove any attributes and still have uniqueness constraint in the above condition hold (minimal super key).

Sometimes tuples ID or sequential numbers are assigned as key to identify the tuple in a table, sucha key is called *artificial* or *surrogate key.*

There may be several candidate keys for a relation. The candidate key that is selected to identify tuples uniquely within the relation is called *primary* key (PK).

The primary key value is used to *uniquely identify each tuple* in a relation and provides the tuple identity. Usually a candidate key that is easiest to use in daily work to data entry is chosen as a primary key.

A relationship between tuples of the two relations is represented by the foreign key. *Foreign key (FK)* is the set of attributes in one table, which is a key in another (or the same) table. Attributes of foreign keys are not required to have the same names as the attributes of the key they correspond to.

Foreign key referencing its own relational table is called a *recursive foreign key*.

The rules of foreign key:

- the attributes in *FK* have the same domain(s) as the primary key (*PK)* attributes:

- value of *FK* in a tuple either occurs as a value of *PK* for some tuple or is *NULL.*

The information about foreign keys is important, so it is necessary to reflect the foreign keys when defining relational tables. The detailed list in which are given the names of the relational tables listing the attributes and definitions of foreign keys is called *the relational database schema*.

The advantages of the relational model: simplicity; presence of the well-developed and powerful mathematical apparatus; it is modifiable (new tables and rows can be added easily);the possibility of operations over the data without knowledge of their physical organization; very flexible and powerful; fast processing.

The disadvantages of the relational model: expensive solutions that require thorough planning; easy to create badly designed and inefficient database designs if there is noproper data analysis prior to implementation.

10 Lecture #10. The data integrity. Converting a conceptual model into a relational model

The content of the lecture: the concepts of data integrity.

The goal of the lecture:to learn specifying constraints in data models and the rules of converting a conceptual model into a relational one.

10.1 The constraints that are used for support of data integrity

Data consistency in a database is called *data integrity.* In the relational model there are several *constraints* that can be used to validate data in the database, and to give the data a meaningful structure. They maintain the integrity of the data.

The data *integrity constraints* are the rules that restrict the possible data values in the database. These constraints are derived from the rules in the mini-world that the database represents. The constraints provide a logical basis to maintain the correct data values in the database, preventing errors when updating and processing of data. These possibilities have obvious value, since the main purpose of the database is to provide accurate information for management and decision-making.

Firstly, we will remind some notions that can be considered as implicit *domain constraints*: in the relational schema every attribute is declared to have a type (integer, float, date, Boolean, string, etc.); every value in a tuple must be from the domain of its attribute (or it could be null, if allowed for that attribute).

The rows of the relational tables present in the database the elements of the specific real-world objects. The key of the relational table uniquely identifies each row. Thus, if users want to extract data of a particular row, they must know the key value of this row. That is, the element should not be written to the database as long as the values of its key attributes are fully defined. Thus, the key or any part of the key is not allowed to contain an empty value.

So, *entity integrity constraint:* no key attribute of any tuple of a relational table can have empty values. If PK has several attributes, null is not allowed in any of these attributes.

Note: Other attributes may be constrained to disallow null values, even though they are not members of the primary key.

Referential constraints are a set of validation rules applied to an entity or table such as uniqueness constraints, domain validation of columns or correspondence of foreign keys to the primary key of the related table. When relational table is created to link the tuples in one table with tuples in another table we use foreign keys. The property of a database to have all the foreign key's data correctly related to primary key's data is known as *referential integrity*.

Referential integrity constraint: the value in the foreign key column (or columns) FK of the referencing relation R1 can be either a value of an existing primary key value of a corresponding primary key PK in the referenced relation R2, or a null.

Other types of constraints:

- s*emantic integrity constraints* based on application semantics and cannot be expressed by the model:

- a constrain specification *language*; for instance, SQL allows *triggers* and *assertions* to express for some of constraints:

- the rules of the business can also determine the correct state for a database (values must comply with the business rules). Such rules are called *user-defined data integrity constraints*.

If the database does not obey all the integrity constraints, it has an *invalid state. Valid state* satisfies all the constraints in the defined set of integrity constraints.

Data integrity constraints should be enforced by DBMS or the application software. Specifying constraints is useful information to application programmers: they can write programs to prevent constraints violation. Also, knowledge of some type of constraints enables us to identify redundancy in schema sand hence specification of constraints helps in database design. Knowledge of some type of constraints can also help the DBMS in query processing.

10.2 Converting a conceptual model into a relational

The conceptual model consists of entities, relationships, attributes, composite entities. Consider the methods of conversion of each of these structures in a relational table.

Convert entities and attributes. The relational table is created for each entity of the model. Most often the names of the entities and tables are the same. But they can be different, because the names of the entities may not be subject to additional syntactic constraints besides the uniqueness of the name within the model. The names of the tables can be limited by the requirements of the particular DBMS. Most often these names are identified in a base language; they are limited in length and must not contain spaces and some special characters. Many attributes of the entity become attributes of a relational table. Renaming of attributes must occur in accordance with the same rules as renaming the object sets. For each attribute specify allowable in the DBMS: the data type and the obligation or optional of this attribute (that is, the admissibility or inadmissibility of NULL values for it).

If in the conceptual model there is a key attribute, it can be used as the key in the relational table (Primary Key). Otherwise the key attribute of the table can be created by the designer of the database. In practice, of course, designers should consult with users about the choice of key. The primary key attributes automatically receive the mandatory property (NOT NULL).

If the conceptual model consists of the specification of the entity, that is, sub-entities and generalized entities, then there are several options for submission. It is possible to create only one relation for all sub-entity of one generalizing entity. It includes all the attributes of all sub-entities. However, then for some instances the attributes will not have sense. And even if they will have undetermined values, we'll need additional rules to differentiate one sub-entity from the other. It is possible to create for each sub-entity and for the aggregated entity created own separate relationships. The disadvantage of this method of presentation is that it creates a lot of relationships, but the advantages of this method are more important, since you can only work with meaningful attributes of the sub-object.

Usually the specification of the entity will have all the attributes of the entity without which it specifies. Then we have the duplicate information. To avoid such redundancy of data all duplicate non-key attributes are deleted from the relational table specifications; the keys match the keys of the generalized entity.

The relationships are transformed by one of the ways depending on cardinality:

a) the relationship "one-to-one" is converted by placing one of the entities as attributes in the table of the second entity. Its choice is determined by the needs of the concrete application;

b) the relationship "one-to-many". In any relationship "one-to-many" table that describes the entity, the cardinality of which is equal to "many" includes a column that is a foreign key pointing to another entity.

To model an optional type of connection at the physical level, attributes that match the foreign key are set to the null ability property (NULL).At the mandatory type of the relationship attributes get the property of absence of null values (NOT NULL);

c) the relationship "many-to-many". As the relational data model are supported by only the relationships "one-to-many", and in ER-model are valid the

relationships "many-to-many", it is necessary to apply a special mechanism of converting, that will allow to reflect multiple links, non-specific for the relational model. This is done by the introduction of special additional relationship which is associated with each source connection "one-to-many". Attributes of this relationship are the primary keys of the linked relations.

So, to transform the relationship "many-to-many", the table of intersections is created. The *intersection table* is the table that represents the elements of the other two tables which are in the relation "many-to-many". The relations "many-to-many" match the multi-valued attributes and converted by creating a key from columns that match the keys of the two entities participating in the relationship. That is each of the attributes of the new table (taken from the source table) is a foreign key (FK), and together they form the primary key (PK). The intersectional table can have additional non-key attributes that are unique to it.

d) recursive relationship; when you convert a recursive relationship for the attribute that indicates the relation, creates a new semantic name.

After the transformation of all concrete structures is complete, the relational schema needs to be reviewed with a view to getting rid of redundancy. Any excess tables (i.e., tables whose information is completely contained in the other tables of the schema) should be removed from the scheme.

11 Lecture#11. Normalization of the database. Anomalies

The content of the lecture: basic concepts of the theory of database normalization.

The goal of the lecture: to learn the concept of database normalization and first normal form.

Earlier we considered a technique Entity–Relationship modeling of database. There is another one technique of modeling DB - a *normalization* technique.

Normalization is a database design technique, which is based on the analysis of relationships (*called functional dependencies*) between attributes.

The purpose of normalization is to identify a suitable set of relationships.

The characteristics of a suitable set of relationships include the following:

- the *minimal* number of attributes necessary to support the data requirements of the enterprise;
- attributes with a close logical relationship (described as *functional dependency*) are found in the same relation;
- *minimal* redundancy with each attribute represented only once with the important exception of attributes that form all or part of foreign keys, which are essential for the joining of related relations.

Normalization uses several normal forms to identify the optimal grouping for attributes to identify a set of suitable relations that supports the data requirements of the enterprise: first normal form (1NF); second normal form (2NF); third normal

form (3NF); normal form Boyce-Codd (BCNF); fourth normal form (4NF); fifth normal form (5NF).

In practical DB design there are used the first three normal forms.

The basic qualities of normal forms:

- each following normal form in a sense improves the properties of the previous one;

- when you move to the next normal form properties of the previous normal forms are preserved.

Due to negligent database design appears data redundancy (repetition), which leads not only to loss of unnecessary space; it can cause violation of integrity that is to lead to data inconsistencies in the database.

Inconsistency between data is called *anomalies*.

Relations that have redundant data may have problems called *update anomalies*, which are classified as insertion, deletion, or modification anomalies.

Consider the examples of what problems can occur due to negligent database design. For example, in the database of the University information about the teachers and departments is stored in a single relational table (see table 11.1).

This table is badly designed: in three tuples, corresponding to the teacher with id= 23, repeated the name, and office number. This data redundancy or repetition leads to loss of unnecessary space and may cause violation of data integrity (inconsistency) in the database.

For example, if we want to change the value of one of the attributes of a particular departments in the relation, for example the office number for department number d003, we must update the tuples of all teachers located at that department.

If this modification is not carried out on all the appropriate tuples of the relation, the database will become inconsistent. In this example, departments number B003 may appear to have different office number in different teacher tuples.

Table 11.1 - Information about the teachers and departments stored in one relational table

Teacher Id	**Teacher Name**	**Teacher Position**	**salary**	**Dep No**	**Office**
14	John White	professor	3000	d005	100
23	Ann Beech	tutor	2500	d003	101
23	Ann Beech	tutor	2500	d003	101
39	Mary Howe	assistant	2000	d007	105
23	Ann Beech	tutor	2500	d003	101
14	John White	professor	3000	d005	100
14	John White	professor	3000	d005	100

*Modification anomaly*is inconsistency of data due to their redundancy and a partial update.

Now suppose that a certain teacher (id=14) was fired. Then you have to delete all the records related to that teacher. In this case, information about the department is lost and there is a deletion anomaly.

Deletion anomaly is unintentional loss of data caused by deleting other data.

*Insertionanomaly*is the inability to enter data in the table caused by the lack of other data.

There are two main types of insertion anomaly, which we illustrate using the relation shown in table 11.1.

To insert the details of the new teacher into the table, we mustinclude the details of the department at which the teacher is to be located. For example,to insert the details of new teacher located at department number d007, we must enter thecorrect details of department number d007 so that the department details are consistent withvalues for department d007 in other tuples of the relation.

To insert details of a new department that currently has no members into the relation, it is necessary to enter nulls into the attributes for teacher, such asteacher No. However, as teacher No is the primary key for the relation, attempting toenter nulls for teacher No violates entity integrity, and is not allowed. Wetherefore cannot enter a tuple for a new department into the relation with a nullfor the teacher No.

Obviously, deletion, insertion and update anomalies are unwanted. The above examples demonstrate that while the relation is subject to update anomalies, we can avoid these anomalies by decomposing the original relation. When we decompose a larger relation into smaller relations any instance of the original relation can be identified from corresponding instances in the smaller relations;

Decomposing is the process of dividing tables into multiple tables to eliminate anomalies and maintain data integrity. For this purpose, normal forms or rules for structuring tables are used.

Normalization is often executed as a series of steps. Each step corresponds to a specific normal form that has known properties.

For the relational data model, it is important to recognize that it is only First Normal Form (1NF) that is critical in creating relations; all subsequent normal forms are optional. However, to avoid the update anomalies it is generally recommended that we proceed to at least Third Normal Form (3NF).

If a table contains one or more repeating groups, it is in *Unnormalized Form (UNF).*

A repeating group is an attribute, or group of attributes, within a table that occurs with multiple values for a single occurrence of the nominated key attribute(s) for that table (see table 11.2).

First Normal Form (1NF): A relation in which the intersection of each row and column contains one and only one value.

Table 11.2 - Unnormalized relation

Teacher Id	**Teacher Name**	**Teacher Position**	**Training team**
14	John White	professor	100,105

39	Mary Howe	assistant	100,101,105

We remove the repeating group by entering the appropriate data into each row (table 11.3).

Table 11.3 – The relation in First Normal Form

Teacher Id	Teacher Name	Teacher Position	Training team
14	John White	professor	100
14	John White	professor	105
39	Mary Howe	assistant	100
39	Mary Howe	assistant	101
39	Mary Howe	assistant	105

Relational table *must* satisfy the first normal form (by Codd).

The table is in the first normal form if satisfies the following requirements:

1) table has no duplicate records;
2) table must not have repeating groups of fields;
3) rows should not be ordered;
4) columns should not be ordered.

12 Lecture #12. Functional dependencies and associated normal forms

The content of the lecture: concepts of the functional dependencies and its characteristics those are useful for normalization.

The goal of the lecture: learning the other normal forms.

An important concept associated with normalization is *functional dependency*, which describes the relationship between attributes. Previously there were discussed rules of entity integrity and referential integrity constraints of the relational schema.

The functional dependence allows imposing *additional* constraints. When each value of an attribute in a tuple is associated with exactly one value of another attribute in a tuple, it means that there is *functional dependency* (FD) between these attributes.

Formally, a functional dependency is defined as follows: if A and B are attributes of relation R, B is functionally dependent on A (denoted $A \rightarrow B$), means: when two tuples have the same value of A, they also have the same value of B. The designation is read as follows: ‘A functionally determines B’.

This definition is also applicable, if A and B are sets of columns, and not just individual columns. When a functional dependency exists, the attribute or group of attributes on the left-hand side of the arrow is called the *determinant*. The determinant is the attribute (attributes) whose value determines the values of other attributes of the tuples.

Consider the attributes ‘teacherId’ and ‘teacherPosition’ of the relation in table 12.1. For a specific ‘teacherId’, for example, with the Id=14, we can determine the position of that teacher as ‘professor’.

In other words, ‘teacherId’ functionally determines ‘teacherPosition’.

However, the opposite is not true, as ‘teacherPosition’ does not functionally determine ‘teacherId’. A member of staff holds one position; however, there may be several members of staff with the same position.

Table 12.1– The relational table in 1NF

teacherId	**teacherName**	**teacherPosition**	**salary**	**depNo**	**office**
14	John White	professor	3000	d005	100
23	Ann Beech	tutor	2500	d003	101
23	Ann Beech	tutor	2500	d003	101
39	Mary Howe	assistant	2000	d007	105
23	Ann Beech	tutor	2500	d003	101
14	John White	professor	3000	d005	100
14	John White	professor	3000	d005	100

The relationship between ‘teacherId’ and ‘teacherPosition’ is one-to-one (1:1): for each staff number there is only one position.

On the other hand, the relationship between ‘teacherPosition’ and ‘teacherId’ is one-to-many (1:*): there are several staff numbers associated with a given position. In this example, ‘teacherId’ is the determinant of this functional dependency.

When identifying functional dependencies between attributes in a relation it is important to distinguish clearly between the values held by an attribute at a given point in time and the *set of all possible values* that an attribute may hold at different times. In other words, a functional dependency is a property of a relational schema and not a property of a particular instance of the schema. For example, in the relation in Fig.12.1: ‘teacherId’ functionally determines ‘teacherPosition’, but ‘teacherPosition’ does *not* functionally determine ‘teacherId’.

An additional characteristic of functional dependencies that is useful for normalization is that their determinants should have the minimal number of attributes necessary to maintain the functional dependency with the attribute(s) on the right hand-side. This requirement is called *full functional dependency*.

For A and B attributes of a relation, B is *fully functionally dependent* on A, if B is functionally dependent on A, but not dependency on any proper subset of A.

A functional dependency $A \rightarrow B$ is a fully functional dependency if removal of any attribute from A results in the dependency no longer existing. A functional dependency $A \rightarrow B$ is a partially dependency if there is some attribute that can be removed from A and yet the dependency still holds.

As example, consider the following functional dependency that exists in the relation of the table 12.1:

‘teacherId’, ‘teacherName’→’depNo’.

It is correct to say that each value of (teacherId, teacherName) is associated with a single value of’depNo’. However, it is not a full functional dependency because ’depNo’is also functionally dependent on a subset of (teacherId, teacherName)), namely ‘teacherId’. In other words, the functional dependency shown above is an example of *a partial dependency*.

In summary, the functional dependencies that we use in normalization have the following characteristics:

- there is a one-to-one relationship between the attribute(s) on the left-hand side (determinant) and those on the right-hand side of a functional dependency;
- they hold for all time;
- the determinant has the minimal number of attributes necessary to maintain the dependency with the attribute(s) on the right-hand side. In other words, there must be a full functional dependency between the attribute(s) on the left- and right-hand sides of the dependency.

There is an additional type of functional dependency called a *transitive dependency;* its existence in a relation can potentially cause the types of update anomaly. Let A, B, and C are attributes of a relation. If for these attributes there holds a condition such that: if $A \rightarrow B$ and $B \rightarrow C$, then C is transitively dependent on A via B (provided that A is not functionally dependent on B or C), the dependency will be a *transitive dependency.*

Consider the following functional dependencies within the relation shown in table 12.1:

teacherId → teacherName, teacherPosition, salary, depNo, office;
depNo → office.

The transitive dependency *depNo → office* exists on ’teacherId‘ via ‘depNo’. In other words, ‘teacherId’attribute functionally determines ‘office’ via the ‘depNo’ attribute and neither ‘depNo’ nor ‘office’ functionally determines ‘teacherId’.

On the concept of full functional dependency Second Normal Form is based. A relational table is in *Second Normal Form* (2NF)if it is in First Normal Form and every non-primary-key attribute is fully functionally dependent on the primary key.

Second Normal Form applies to relations with composite keys, that is, relations with a primary key are composed of two or more attributes. A relation with a single-attribute primary key is automatically in at least 2NF.In a relationship, which does not meet 2NF, the update anomalies can appear.

The normalization of 1NF relations to 2NF involves the removal of partial dependencies. If a partial dependency exists, we remove the partially dependent attribute(s) from the relation by placing them in a new relation along with a copy of their determinant. This process of decomposing tables consists of the following steps:

1) a new table is created, attributes of which will be the attributes of the original table, included in the contradictory FD rule. Determinants of functional dependency becomes the key of the new table;

2) attribute, located in the right side of the functional dependency, is excluded from the source table;

3) if more than one functional dependency violate 2NF, then steps 1 and 2 are repeated for each functional dependency;

4) if the same determinants are included in several functional dependencies, all of the functionally dependent attributes are placed as a nonkey attributes in a table, which will be key determinants.

We demonstrate the process of converting 1NF relations to 2NF relations in the following example. Consider the relation ‘Sales’ in the table 12.2:

Sales (vendorId, productId, vendorName, date).

For the ‘Sales’ relation we choose composite key: (vendorId, productId).

Table 12.2 – Table that do not satisfy 2NF

vendorId	**productId**	**vendorName**	**date**
23	2241	Moor	10.02
14	2241	Kurosawa	08.02
23	2518	Moor	12.02
14	2518	Kurosawa	10.02
14	1035	Kurosawa	12.02

In this table, the key consists of the attributes ‘vendorId’ and ‘productId’. ‘vendorName’ attribute is determined by the attribute ‘vendorId’, that is functionally dependent on part of key. This means that to determine the name of the vendor it isenough to know ‘vendorId’. Thus, the table does not meet 2NF.

Because of this, you may experience the following problems:

1) the name of the vendor is repeated in each row relating to his transactions;

2) if the name of the vendor is changed, you must update all rows that contain the records of his transactions;

3) this redundancy may cause a mismatch of data across the different rows containing different names for the same vendor;

4) if at any time the vendor has no transactions, then there may not be strings, in which you can store its name.

In order to solve these problems, the table must be divided into two relational tables, each of which satisfies 2НФ.

Transforming the ‘Sales’ relation into 2NF requires creation of new relations, so that the non-primary-key attributes are removed along with a copy of the part of the primary key on which they are fully functionally dependent.

This results in the creation of two new relations of the following forms:

Vendor (vendorId, vendorName),

Sales (vendorId, productId, date).

In the second table a foreign key ‘vendorId’ references a table ‘Vendor’.

These two tables are in Second Normal Form as every non-primary-key attribute is fully functionally dependent on the primary key of the relation.

Please, note that in the source table the values of the attributes ‘vendorId’ ‘productId’, ‘date’ taken together, were different, therefore in the table of ‘Sales’ will be five records (as earlier). And the table ‘Vendor’ now consists of only two rows because there are only two different sets of values for attributes ‘vendorId;, ‘vendorName’. Thus, the redundancy data and possibility of anomalies are excluded.

Although 2NF relations have less redundancy than those in 1NF, they may still suffer from update anomalies. Consider the Third Normal Form.

The relational table is in the *third normal form* (3NF) if for every functional dependency $FD: X \rightarrow Y$, X is the key (i.e. determinant is the key). Consider a relation in the table 12.3 where the attribute ‘vendorId’ is a key.

Table 12.3 - Table that does not meet 3NF

vendorId	qualification	bonus,%
14	Specialist	11
23	Supervisor	12
39	Specialist	10

The table has the following functional dependencies:

$$FD: vendorId \rightarrow qualificaton;$$

$$FD: vendorId \rightarrow bonus,\%$$

However, there is also a functional dependence

$$FD: qualification \rightarrow bonus,\%$$

The first two functional dependencies satisfy the criterion of 3NF, and a third attribute ‘qualification’ is not the key, so this criterion is violated.

The resulting problems are similar to those listed for tables that violate 2NF:

1. the fee for the type of qualification is repeated in each row. This redundant data occupies too much space;

~~2.~~ 2. if the fee for the type of qualification changes, we need to update each row. If the row is removed, it is possible to lose information on fees for this qualification. That is, the table is subject to update and deletionanomalies;

if at some time there are no workers with this qualification, it may not be the row in which you can store the bonus amount. This is insertion anomaly.

These problems are also solved by *decomposing* tables.

We create a new table *R1* by removing from the table all attributes of the right side which violate the criterion of 3NF. In our example this is ‘bonus%’.

Then create a new table *R2* that consists of attributes both from the left and from the right parts of FD that violate the 3NF criteria. These are ‘qualification’ and ‘bonus%’. The determinant ‘qualification’ is the key.

So, we have relation

R1 (<u>vendorId</u>, qualification).

The attribute ‘qualification’ is a foreign key referencing the table *R2*:

R2 (qualification, bonus%)

Instead of one table we received the two tables. If at least one of the resulting tables violates 3NF, we continue the process of decomposing as long as all tables do not meet 3NF. From the definition of 3NF it is clear that any table satisfying the 3NF also satisfy 2NF.

Because 3NF tables always satisfy 2NF, it is sufficient to use the criterion for third normal form. If every determinant in the table is the key of the table, the table satisfies the first, second and third normal forms. This simplifies the normalization process, as we need to check only one criterion. It should be noted that the definition of third normal form corresponds in some sources to normal form of Boyce-Codd.

In most cases, achieving third normal form is considered adequate for real-world database projects, however,by the theory of normalization, there are normal forms of higher orders, that are already associated not with functional dependencies between attributes of relations, but reflect the more subtle issues of the semantics of the subject domain and linked to other types of dependencies.

13 Lecture #13 Database languages

The content of the lecture: concepts of data definition and data manipulation languages.

The goal of the lecture: familiarizing with the types of language that are used by DBMS.

We consider the types of languages that are used by DBMSs. A data language consists of two parts: a *Data Definition Language* (DDL) and a *Data Manipulation Language* (DML).

All the languages of data manipulation created before the appearance of relational databases are focused on operations with the data presented in a logical file record. It is required that the user should have detailed knowledge of the organization of data storage and apply sufficient efforts to specify not only what data is needed, but also where they are placed and how to deal with them step by step. We can distinguish between two types of DML: procedural and non-procedural. The prime difference between these two data manipulation languages is that procedural languages specify *how* the output of a DML statement is to be obtained, while non-procedural DMLs describe only *what* output is to be obtained. Typically, procedural languages treat records individually, whereas non-procedural languages operate on sets of records.

Non-procedural language allows the required data to be specified in a single retrieval or update statement. With non-procedural language, the user specifies what data is required without specifying how it is to be obtained. The DBMS translates a language statement into one or more procedures that manipulate the required sets of

records. This frees the user from having to know how data structures are internally implemented and what algorithms are required to retrieve and possibly transform the data, thus providing users with a considerable degree of data independence. Non-procedural languages are also called *declarative languages*.

Procedural language is a language that allows the user to tell the system what data is needed and exactly *how* to retrieve the data. With a procedural language, the user (more often, the programmer), specifies what data is needed and how to obtain it. This means that the user must express all the data access operations that are to be used by calling appropriate procedures to obtain the information required. Typically, such a procedural language retrieves a record, processes it and, based on the results obtained by this processing, retrieves another record that would be processed similarly, and so on. This process of retrievals continues until the data requested from the retrieval has been gathered. Typically, procedural language are embedded in a high-level programming language that contains constructs to facilitate iteration and handle navigational logic. Network and hierarchical DMLs are normally procedural.

The methods of manipulation can be specific for a particular DBMS. However, there are more or less universal methods of manipulating data, supported by almost all server relational DBMS and most universal mechanisms for data access (including when using them with a desktop DBMS). Such is the method using the SQL language (Structured Query Language). Structured Query Language is non-procedural language used to manage relational DBMS, used to formulate queries to databases. This language does not contain algorithmic constructions. The term "non-procedural" means that the language can be used to specify what has to be done with the database, but you cannot instruct exactly how it should be done, it is impossible to describe the algorithm of this process.

All the algorithms processing the SQL queries are generated by DBMSs and are not dependent on the user. This non-procedural language SQL is focused on data operations, presented in the form of logically interrelated sets of tables. The peculiarity of this language is that they focus more on the end result of processing the data than on the procedure of this processing. SQL itself determines where the data resides, which indexes and even the most efficient sequence of operations should be used to obtain them: it is not necessary to specify these details in the query to the database.

*The Data Definition Language (DDL).*The database schema is specified by a set of definitions expressed by means of a special language called a Data Definition Language. The DDL is used to define a schema or to modify an existing one. It cannot be used to manipulate data. The result of the compilation of the DDL statements is a set of tables stored in special files collectively called the *system catalog.* The system catalog integrates the *metadata* that is data that describes objects in the database and makes it easier for those objects to be accessed or manipulated. The metadata contains definitions of records, data items, and other objects that are of interest to users or are required by the DBMS. The DBMS normally consults the system catalog before the actual data is accessed in the

database. The terms *data dictionary* and *data directory* are also used to describe the system catalog, although the term ‘data dictionary’ usually refers to a more general software system than a catalog for a DBMS. There are several SQL statements to manage metadata (the modification metadata) that is used to create, modify or delete databases and contained objects (tables, views, etc.).

*The Data Manipulation Language (DML).*A language that provides a set of operations to support the basic data manipulation operations on the data held in the database. Data manipulation operations usually include the following: insertion of new data into the database; modification of data stored in the database; retrieval of data contained in the database; deletion of data from the database.

The part of a DML that involves data retrieval is called a *Data Query Language* (DQL). A query language can be defined as a high-level special-purpose language used to satisfy diverse requests for the retrieval of data held in the database. The term ‘query’ is therefore reserved to denote a retrieval statement expressed in a query language.

Structured Query Language (SQL) - non-procedural language used to manage relational DBMS. This language does not contain algorithmic constructions. All the algorithms processing the SQL queries are generated by DBMS and are not dependent on the user.

14 Lecture#14. The objects of the database

The content of the lecture: the maim objects of the databasewithin the specific DBMS..

The goal of the lecture:to familiarize students with the objects of the database which are used for implementation of the designed project.

After the development of the database project and converting it to the relational model, there comes the stage of its implementation in a DBMS.

Database contains several different types of objects, they are:

- *tables*; tables are supported by all relational databases and their fields can store different kinds of data. The notion "table" generally is associated with the real table (created using ‘Create table’ statement). However SQL uses and creates a number of virtual (if available) tables: views, cursors and the unnamed worksheets in which are formed the results of the queries to receive data from the underlying tables and maybe views. These are the tables which do not exist in the database, but seem to exist from the point of view of the user.
- *views;* almost all relational databases support views. The view is the virtual table that provides the data from one or more real tables. Actually it does not contain any data but only describes their source. Often such objects are created for the storage in a database the complex queries. Actually view is a stored query.

Creating views in most modern DBMS is performed by the special visual means. They allow you to display the necessary tables, establish relationships between them, to choose the displayed fields, to input the restrictions on entry etc.

Often these objects are used for data security for example by permitting of the data view with their help without giving access directly to the tables. In addition some representations of the objects can return different data depending on for example on behalf of the user which allows him to get only the data interesting for him;

- *generating primary keys*. When creating tables it is important to determine primary keys. Very often primary keys are generated by DBMS itself. It's more convenient than their generation in the client application as at multi-user work key generation with the help of a DBMS is the only way to avoid the duplication of the keys and get their consistent values.

In different DBMS to generate the keys different objects are used. Some of these objects store the integer and the rules by which generated the following value, usually this is done using triggers. Some databases support the special field types for the primary keys. When adding records such fields are filled automatically in consecutive values (usually integers). In the case of Microsoft Access and Microsoft SQL Server the type of data in these fields are *'identity'*;

- *indexes;* in most relational DBMS the keys are realized using the objects called *indices* which can be defined as a list of numbers of records indicating the order in which to provide them.

We already know that the records in relational tables are unordered. However, any entry to a specific point in time has a definite physical location in the database file, although it may change in the editing process or the result of "internal activities" of the DBMS. A storage index requires significantly less space than storage of differently sorted versions of the tables and also reduces data retrieval time.

We already talked about the fact that the physical location of records may change during editing users, and also as a result of manipulation of the database files held by most DBMS (e.g., data compression, garbage collection, etc.).

If the corresponding changes are fulfilled in the index, the index is called *supported* and such indexes are used in most modern DBMS. The implementation of these indices leads to the fact that any change to the data in the table entails a change in the related index, and this increases the time required by the DBMS to carry out such operations. So you should create only those indexes that really necessary;

- *constraints and rules;* most modern DBMS contain this special objects. These objects contain information about restrictions on possible values of the field. For example, using this object you can set the maximum or minimum value for the given field, and then the DBMS will not allow saving in a database the records, do not satisfy this condition.

In addition to the restrictions related to installation of range data, there are also *referential* constraints. Not all DBMSs support the constraints. In this case to realize the functionality of the rules you can use other objects (such as *triggers*) or keep these rules in client applications working with this database;

- *triggers* and *stored procedures* are supported in most modern server-based DBMS; they are used to store executable code.

A stored procedure is a special type of procedure that runs the database server. Stored procedures are written in *procedural* language, which depends on the specific DBMS. They can cause each other to read and modify data in tables, and you can call them from a client application working with the database. Stored procedures are usually used to perform frequently occurring tasks. They can have arguments; return values, error codes and sometimes the sets of rows and columns.

Triggers are a special class of stored procedures that start automatically when you update data from a table. Triggers also contain executable code, but, unlike procedures cannot be called from a client application or a stored procedure.

A trigger is always associated with a specific table and running when you edit this table, there comes an event with which it is associated (e.g., inserting, deleting or updating a record). In most DBMS that supports triggers, you can define multiple triggers running when occurrence of one event, and to determine the order of execution;

- *database queries* are used for modification and data select, edit of metadata, and some other operations. The query is written in SQL. Most modern DBMS (and some application development tools) contain the tools to generate the queries.

One of the ways of data manipulation is called "queries by example" (QBE). QBE is a tool to visually link of tables and selecting the fields you want to display in the query result. Unlike the relational table in the query results the rows are ordered, and their order is determined by the initial query (and sometimes by the presence of indexes);

- *transactions* is a group of data operations that are either executed all together or all together canceled.

The completion (*Commit*) of a transaction means that all operations included in the transaction are successfully completed and the result is stored in the database.

Rollback of a transaction means that all performed operations included in the transaction are undone, and all database objects affected by these operations, returned to its original state. For the implementation of the rollback possibilities of transaction DBMS supports the recording in log files, allowing to recover the original data when a rollback.

A transaction may consist of several included transactions.

Some DBMS support two-phase commitment of transaction. It is a process that allows you to implement transactions over multiple databases related to the same DBMS. To support distributed transactions (that is transactions over databases managed by different DBMS), there are special tools, transaction monitors:

- *users and roles*. Preventing unauthorized access to data is a serious problem, which is solved in different ways. The easiest is to password protect either the whole table or certain fields.

Currently, more popular is another way to protect data: create a list of *users* with *names* and *passwords*. In this case, any object in the database belongs to a particular user, and that user provides to other users the permission to read or modify data from the data object or the modification of the object itself. This method is used in all servers, and some desktop software.

Some DBMSs, mostly server-based, support not only a list of users and roles. A *role* is a set of privileges. If a particular user receives one or more roles, with them he receives all the privileges defined for that role;

- *system catalogue*. Any relational DBMS that support a list of users and roles must store them somewhere.

In addition to these lists many DBMS maintain lists of tables, indices, triggers, procedures, etc., as well as details about who owns them. These lists are called the *system tables*, and the respective part of the database is called the *system catalog*. Note that not all DBMS support the system directories.

All information about the used data structures, logical data organization, user access rights and, finally, the physical location of data is *a metadata database*.

For the management of database metadata there is the special software of database administration (DBA), which is intended for the correct usage of common information space by many users. The database administrator is responsible for the physical realization of the database, including physical database design and implementation, security and integrity control, maintenance of the operational system, and ensuring satisfactory performance of the applications for users. The role of the DBA is more technically oriented than the role of the data administration, requiring detailed knowledge of the target DBMS and the system environment.

15 Lection #15. Database Planning, Design and Administration

The content of the lecture:

the main stages of the database system development lifecycle.

The goal of the lecture: brief description of the sequence of database development, its implementation and administration.

As we mentioned earlier a database system is a fundamental component of the information system, so the database system development lifecycle is associated with the lifecycle of the information system. The stages of the database system development lifecycle are described below. You start from the planning of database.

Database planning is the management activities that allow to realize the stages of the database system development lifecycle as efficiently and effectively as possible. An important first step in database planning is to clearly define the major aims of the database system. Database planning should also include the development of standards that manage how data will be collected, how the format should be specified, what necessary documentation will be needed, and how design and implementation should proceed. For example, specific rules may assign how data items can be named in the data dictionary, which in turn may prevent both redundancy and inconsistency (see lecture 4 about description subject domain).

Before starting to design a database system, it is essential that we first identify how it will interface with other parts of the organization’s information system. So, you must describe the major user views. It is important that we include

in our system not only the current users and application areas, but also future users and applications.

User view defines what is required of a database system from the perspective of a particular job role or enterprise application area. A database system may have one or more user views. A user view defines what the users will do with the data.

The next stage involves the collection and analysis of information about the part of the enterprise to be served by the database. There information is gathered for each major user view. This information is then analyzed to identify the requirements (or features) to be included in the new database system. These requirements are described in documents collectively referred to as *requirements specifications* for the new database system.

Requirements collection and analysis is a preliminary stage to database design. The amount of data gathered depends on the nature of the problem and the policies of the enterprise. The information collected at this stage may be poorly structured and include some informal requests, which must be converted into a more structured statement of requirements. Requirements for each user view are merged into a single set of requirements for the new database system. A data model representing all user views is created during the database design stage.

Next step is *database design*. The two main approaches to the design of a database: 'bottom-up' and 'top-down'. A bottom-up approach to database design is represented as the process of normalization (we considered its in the lecture 11). Normalization involves the identification of the required attributes and their subsequent aggregation into normalized relations based on functional dependencies between the attributes. A more appropriate strategy for the design of complex databases is to use the top-down approach. The top-down approach is illustrated using the concepts of the Entity–Relationship model, beginning with the identification of entities and relationships between the entities, which are of interest to the organization (lecture 6). These approaches are used for *data modeling*. Database design has three main phases, namely conceptual, logical, and physical design.

Selecting the DBMS and application design. We must determine a description of the criteria (based on the users' requirements specification) to be used to evaluate the DBMS products, and all necessary constraints.

The design of the user interface and the application programs that use and process the database is the process of application design. The database and application design are parallel activities of the database system development lifecycle. In most cases, it is not possible to complete the application design until the design of the database itself has taken place. On the other hand, the database exists to support the applications, and so there must be a flow of information between application design and database design. We must ensure that all the functionality stated in the users' requirements specification is present in the application design for the database system.

After choosing DBMS (assume, the relational) you convert the conceptual model into relational.

Firstly, you transform the entities and attributes for: each entity of the model there is created the relational table; the attributes of the entity become the attributes of the tables; for each attribute you specify the particular allowable in the DBMS data type; if in the conceptual model there is a key attribute, it can be used as the key in the relational table (*primary key*). Otherwise the key attribute of the table can be created by the developer of the DB (*artificial key*).The attributes that are in the primary key of the relationship automatically get the property of mandatory (NOT NULL).

Then transform the relationships:

- the relationship "one-to-one" is transformed by placing one of the entity as attributes in the table of the second object set;

- in any relationship "one-to-many" table that describes the entity, the cardinality from the side which is equal to "many" includes a column that is a foreign key pointing to another object;

- in order to convert the relation "many-to-many", there is created the table of intersections; it is a table that represents the elements of the other two tables which are in the relation "many-to-many";

- the relations "many-to-many" are converted by creating a key from columns that match the keys of the entities participating in the relationship. That is each of the attributes of the new table (taken from the original table) is a *foreign key,* and together they form the *primary key*. The intersectional table can have additional non-key attributes inherent only to her;

– recursive relationship; when you convert a recursive relationship for the attribute, denoting a relation, there creates a new meaningful name.

After the transformation of all structures is complete, the relational schema needs to be reviewed with a view to getting rid of redundancy. Every excess tables (i.e., tables whose information is completely contained in the other tables of the schema) should be removed from the scheme.

Data in tables should meet the principles of *entity integrity, referential integrity.*

Transaction design. An action, or series of actions, carried out by a single user or application program, which accesses or changes the content of the database. Transactions represent 'real world' events such as the addition of a new member of staff, the registration of a new client. These transactions have to be applied to the database to ensure that data held by the database remains current with the 'real world' situation and to support the information needs of the users.

A transaction may be composed of several operations, such as the transfer of money from one account to another. However, from the user's perspective these operations still accomplish a single task. From the DBMS's perspective, a transaction transfers the database from one consistent state to another. If the transaction cannot complete for any reason, the DBMS should ensure that the changes made by that transaction are undone.

On completion of the design stages, we are now in a position to *implement* the database and the application programs. The database implementation is achieved

using the Data Definition Language of the selected DBMS or a Graphical User Interface (GUI). The DDL statements are used to create the database structures and empty database files. Any specified user views are also implemented at this stage.

The application programs are implemented using the preferred third or fourth generation language. Parts of these application programs are the database transactions, which are implemented using the Data Manipulation Language of the target DBMS.

For managing and controlling the activities associated with the corporate data and the corporate database is used *Data Administrator* (DA) and *Database Administrator (DBA)* respectively.

Data administration is the management of the data resource, including database planning, development and maintenance of standards, policies and procedures, and conceptual and logical database design.

Database administration is the management of the physical realization of a database system, including physical database design and implementation, setting security and integrity controls, monitoring system performance, and reorganizing the database as necessary.

Operational maintenance is the process of monitoring and maintaining the system following installation.

We don't consider the problems of data security.

References

1 Toby J. Teorey. Database Modeling and Design.pdf, Lecture Notes, 3rd Edition - University of Michigan, 2009.

2 Paolo Coletti. Databases course book.pdf, Version 4.1 - Free University of BolzanoBozen, 2014.

3. Michael V. Mannino. Database Design, Application Development and Administration.pdf, Third Edition - University of Colorado at Denver, McGraw-Hill, 2012.

4. Andrew J. (Andy) Oppel. Databases: A Beginner's Guide.pdf, www.it-ebooks.info.

5. SQL Functions, Operators, Expressions, and Predicates.pdf - Teradata Corporation, 2009.

6.Ibrayeva, L.K. Designofdatabases. Teachingaid. – Almaty: AUPET, 2010.

7. Ibrayeva, L.K. Designofdatabases. Lecture notes for students of specialty "Automation and control". – Almaty: AUPET, 2010.

8. Leonard, L. et.al. Development of applications on the basis of MicrosoftSQLServer 2008 –M.: 2010

9. Date, Ch. J. Introduction to database systems. – M.: 2006.

Thecontent

Additional plan 2017, pos.11

Lida Kuandykovna Ibrayeva

DATABASE DESIGN

Lecture notes for the specialty 5B070200 - "Automation and Control"

Editor
Specialist of standardization N.K.Moldabekova

Signed for printing __.__.__.
Circulation _25 copies
Volume 3.6e.-p. sheets

Format60x84 1/16
Printing paper #1
Order .Price 1750tg.

Copying Bureau of the Noncommercial Joint-Stock Company
"Almaty University of Power Engineering and Telecommunications"
050013Almaty, 126/1, Baitursynovst.