Database Management System
DBMS

Higher School of Digital Culture
ITMO University
dc@itmo.ru

Contents

1 Information Systems 2

2 IS architecture 4

3 Main Functions of Data Management Systems 6

4 DBMS architecture 13

5 Introduction to Relational Databases 18

1 Information Systems

Today, we are surrounded by various information systems that make performance of basic tasks simple and unnoticeable; whereas, not so long ago these tasks required significant time. Information systems help us do shopping, register for various events, schedule doctor's appointments. In this regard, we expect quick and error-free operation of the systems.

Information system development is a complex task that requires a high level of skill. Meanwhile, different requirements are specified for the systems during the development in order to ensure their real-world application.

There are many different requirements for information systems. The main requirements include:

- reliability;
- scalability;
- development and support performance.

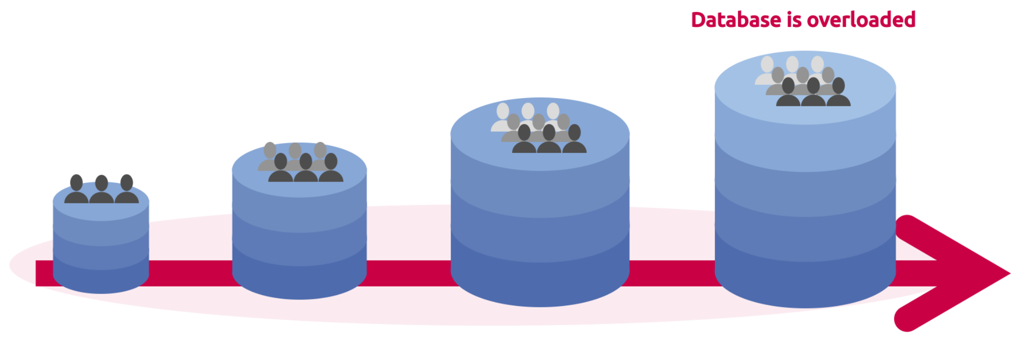
Let's take a closer look at these requirements. Reliability means that a system should continue working correctly and meeting user expectations, even when errors and disruptions occur. For example, let's assume that a student needs to enter a phone number when registering as a conference participant. The system that is used for entering data must be designed and implemented in such a way that when a field is filled with a random sequence, this doesn't lead to negative consequences. The program should recognize invalid data and alert the user. Besides, different circumstances, such as Internet disconnection, may arise. In this case, an API must be predefined. For example, the system can inform the user that it is offline and all actions are saved locally on a computer (if possible), and when connection is restored, deferred events will follow the regular procedure.

When we say "scalability", we usually mean the system's ability to handle an increased load. This can be achieved by adding the required resources.

If a system is poorly scalable or not scalable, then adding new resources will not improve the system, as increasing the load on the system will cause it to degrade or fail completely. It is noteworthy that scalability can be considered not only in terms of hardware resources on which the system is based, but also in terms of its software implementation, i. e., ways of adaptation to an increased load, algorithms used in the program or connected libraries.

An important requirement for systems is the possibility to add new functionality to the information system after it has gone live. It may be that a system is developed and successfully performing its original tasks. One day, however, the file format compatible with the original system has changed and now you need to

Scalability is the system's ability to handle an increased load



add support for a new format. In case of a well-developed system, you will need to change the code a little or even simply configure some system settings. If during system development the possibility of later changes hasn't been considered, such system would require significant costly changes at best, and at worst, the system or its part would need to be completely rewritten.

It is also important to consider the human factor. Systems are usually created by development teams. At the same time, team composition often changes. When developing a system, it is important to keep in mind that people who create the system now may not be there when the system needs to be modified. In this regard, it is essential that all developers during development and support use the same consistent and standardized technologies and tools that can be studied by new team members in a relatively short time.

What does it imply? In order to be able to implement these requirements in practice, systems are usually made up of separate small parts, which are called modules or components. Each component is responsible for a specific part of the

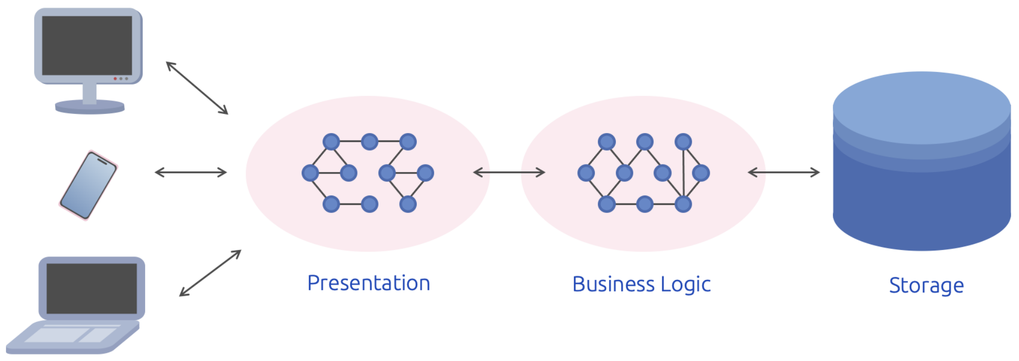
- IS is built from separate modules or components
- Components are located on different layers
- Components of different layers can interact with each other

system. For example, there may be a component responsible for sending user messages, displaying message history, generating different history data. These components, as they were different parts of an erector set, are connected to each other to create a final system with required functionality.

2 IS architecture

Among a variety of components, we can distinguish those ones that are similar in terms of their functions and used technologies. A set of such components is called an application layer of the information system.

Information system layer is a set of components with similar functions



The following layers are usually included in an information system:

- **Presentation Layer** is responsible for generating a user interface. Components that make up the presentation layer are responsible for rendering different system interfaces and system/user interaction. For example, this layer includes visualization of a user's transaction history in an online bank or a registration form of a student conference.

- **Business Logic Layer**. Functions specific to the internal processes of an information system are implemented within this layer. One example could be the implementation of a training course search algorithm in the system, allowing a student of a particular field to enroll in the necessary courses according to his curriculum. Another example could be the group-wide student ratings generation based on the students' course grades.

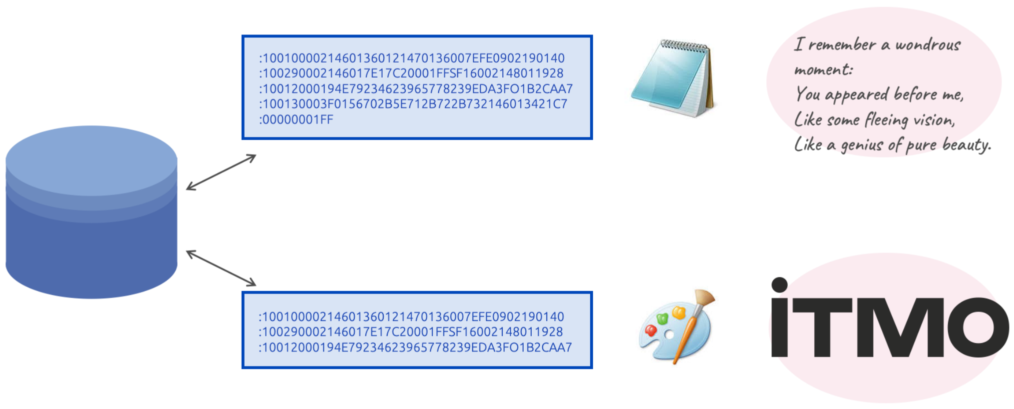
- **Storage Layer** is used to permanently store the application data. This layer may include various components responsible for organizing the interaction between the information system and storage, effective data retrieval from the durable storage or implementation of data storage.

To simplify the development and subsequent support of the system, components of the same layer can only interact with the components of adjacent layers. This means that, for example, it is not recommended to have a direct interaction between the presentation layer and the storage layer of an information system. Because such system can work, but its support and modification will be much more difficult.

As part of this lecture, we will learn how to organize the storage layer of an information system. On the one hand, we know that data is stored in files and nothing prevents us from using regular files as the basis for the storage layer. However, in most cases, special tools, such as databases and their control systems, are used for the long-term storage of data. Let's now analyze why regular files are rarely used for the storage layer organization.

When we considered file systems, we described a file as an abstraction that defined a data region and information on a physical medium that could be potentially stored for an indefinite amount of time.

Files are a linear array of bytes, the internal structure of which depends on the application that uses the file. For example, the content of text files is a sequence of characters represented by bytes. This means that if a program saves data to a



```
:10010000214601360121470136007EFE0902190140
:100290002146017E17C20001FFSF16002148011928
:10012000194E79234623965778239EDA3FO1B2CAA7
:100130003F0156702B5E712B722B732146013421C7
:00000001FF
```

```
:10010000214601360121470136007EFE0902190140
:100290002146017E17C20001FFSF16002148011928
:10012000194E79234623965778239EDA3FO1B2CAA7
```

I remember a wondrous moment:
You appeared before me,
Like some fleeing vision,
Like a genius of pure beauty.
ITMO

file in a certain way or format, another program must know exactly the structure of that file in order to access the data. At the same time, if a programmer decides to change the file format, all other programs using this file must also be changed, or they will stop working correctly.

In addition, the same data may be represented and organized differently in different programs. If there is no original schema, the data structure is determined by the programmer. It is not an easy task to put together the data used by different programs in their own formats and eliminate redundancy. This process is complicated by the fact that program data may not be static, but be changed and supplemented. In this regard, the process of consolidated acquisition and processing of data from all applications will be continuous.

In addition to all of the above, the next question then is how to implement competitive access to the same data from several different applications. The main problem is not just to read and write data, but to take into account the changes made by applications working with the same data at the same time.

Multiple implementations of such functionality, which is required for working with files, significantly increase the cost and complexity of final software. Therefore, the idea emerged to create centralized systems that provide access to data stored in a particular format.

Thus, specialized tools for storing data, called databases, were created. In essence, databases are files with descriptions of data stored in them that are managed by special software systems, called Database Management Systems (DBMS). By using such software systems, you can reduce the costs of application development, as database management systems ensure reliable storage and uniform access to the required data, which will be described in detail in the next section.

3 Main Functions of Data Management Systems

In this section, we will consider the main functions that Database Management Systems (DBMS) should have.

Ensuring the independence of data and applications was originally considered as a key element of database management systems. Those principles that are understood to be the independence of data and applications are divided into three possible groups:

- same data can be used for different applications;
- new data requirements (for example, new fields, tables, business logic of data behavior, etc.) should not affect the operation of existing applications;
- asynchronous implementation of new application versions is allowed.

To support these requirements, languages were introduced to describe the structures and business logic of data behavior. Descriptions of data structures are stored in special dictionaries provided in data management systems. Data dictionaries and languages for describing data structure and business logic are included in almost all modern DBMS in one form or another. Most systems strictly adhere to the principles of data independence. It should be noted, however, that recently there have been a number of storage systems that do not use these important principles in order to improve performance and simplicity of control. Today, we can see data storages that are strictly oriented to the specifics of data of a certain type (for example, texts, photos, etc.) and the functionality of one specific application. If data structure in such applications is changed, this usually affects the application operation and requires it to be upgraded.

Database management system controls the compliance of stored data with the described structure. In addition to the data structure, you can describe other rules that stored data must satisfy. For example, the examination score of a student must be from 1 to 5; the subject for which the score is given must be on the list of taught courses, etc. These types of rules are called integrity constraints, and they are also described using special data description languages. DBMS is responsible for checking the compliance with the given constraints. If data violates these conditions, it can neither be added, nor changed.

Typical integrity constraints of most modern DBMS are as follows:

- referential integrity constraints;
- undefined values restriction;
- duplicate values restriction;
- control of attribute values range.

However, some DBMS either have no integrity rules at all or have only partial rules. If integrity rules are not checked at the data storage layer and these rules are nevertheless important for the subject domain, it is necessary to control them at the application layer.

Consistency support is related to data states. Often a single operation in terms of business logic consists of several small data operations. For example, money transfer from one account to another, increasing of educational allowance for all students by 10%. During these operations, a situation may arise where the data for a time period will not comply with all integrity rules. A database is in a consistent state if its data entirely satisfy the declared integrity rules. The task of DBMS is to ensure the mechanisms for transforming the data from one consistent state to another.

Data consistency support is closely related to the concept of a transaction. Let's give a formal definition of this concept.

A set of operations that transform the database from one consistent state to another is commonly referred to as the transaction.

Another possible definition of a transaction is a logically indivisible sequence of operations, the result of which must be saved entirely or not saved at all.

The following example is usually given to explain the concept of a transaction. In a database containing information on bank customers, it is necessary to record money transfer from one customer's account to another. From the perspective of business (and database), this is a single indivisible transaction, commonly referred to as the transaction. However, in fact three actions stand for this operation and need to be performed in the database:

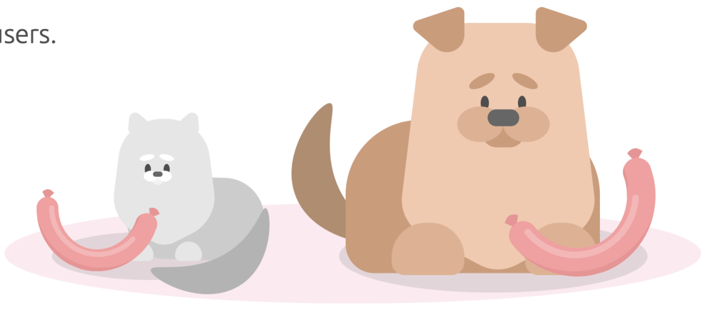
- reduce account balance of the first customer;
- increase account balance of the second customer;
- make the corresponding entry in a posting log.

Data consistency support by DBMS means that the database will be in a consistent state in any case both after successful application termination and after its failure. From the perspective of transactions, this means that all application transactions will either be completed or leave no traces in the data (i. e., DBMS will undo incomplete transactions).

This requirement is called the atomicity of transactions. Once a transaction is completed, changes to the data become permanent. This property is called the durability of transactions.

If several users are working with the data at the same time, their actions may cause the data to become incorrect. For example, two users can change the same data at the same time. In such a case, consistency support is much more difficult. To allow several users to work in DBMS in parallel, various algorithms of transaction execution were developed giving users the illusion of isolation when working with the data, but not leading the database to an inconsistent state. When DBMS support all of these properties, they are said to have ACID properties.

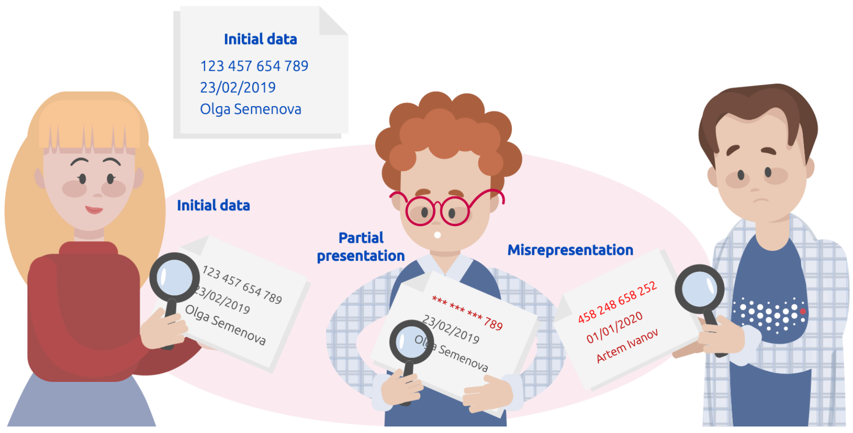
- Support of consistency.
- Atomicity of transactions.
- Durability of transactions.
- Isolation of users.



Functions of data protection against unauthorized access and access privileges are permanent tasks of almost any DBMS. Immediately after creating the database users, there is a need to control their access to different parts of the database. For example, an employee of HR Department may have the right to view employee salary data, but should not have the right to change the rates. Managers shall see the data of employees that work only in their respective departments, etc. For this purpose, various techniques have been developed in DBMS to hide real data from users. For example, with so-called representations, you can hide the actual data structure from a user and display only the data that he or she is allowed to see.

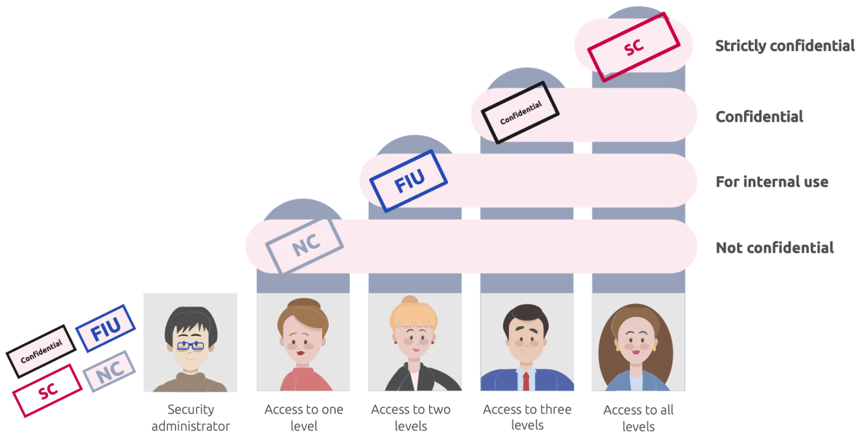
Latest versions of some industrial DBMS feature a variety of techniques for partial data representation, sometimes for deliberate and quite plausible misrepresentation of data. The figure below shows some of the techniques for presentation and misrepresentation of data.

Assignment of access privileges and roles was developed in order to explicitly



grant to each user the rights to perform certain operations in the database (for example, viewing or editing certain fragments of the database, creating new data structures, etc.).

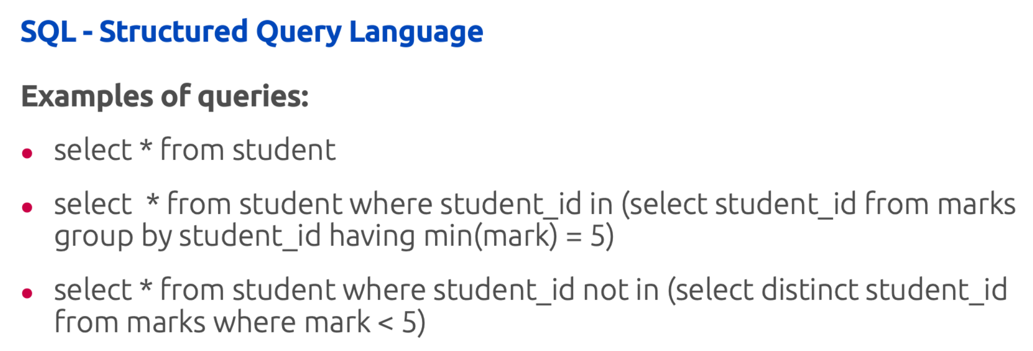
Mandatory access is based on assigning confidentiality marks to the information contained in the database and issuing to users formal permissions (admissions) to access the information of one or another confidentiality level. For example, access to materials classified as confidential.



Another promising area of data protection is isolation of database server administrators from the data itself. In most of today's DBMS, database server administrators can see and modify all data in the database. Security features in the latest versions of some DBMS allow administrators to perform all database management operations, but do not allow them to see or change the data.

A high-level and efficient query language is probably the most important feature of a DBMS.

With that in mind, high-level declarative languages for manipulating data were implemented as part of different control systems. Special attention should be paid to one of those languages – SQL (Structured Query Language). Its elegance, declarative nature and independence from DBMS specifics, as well as its support by leading database vendors – all this has made SQL the main standard for



SQL - Structured Query Language

Examples of queries:

- select * from student
- select * from student where student_id in (select student_id from marks group by student_id having min(mark) = 5)
- select * from student where student_id not in (select distinct student_id from marks where mark < 5)

data processing. When a query is written in a high-level language (SQL, in particular), you know what data to retrieve from the tables, but you don't know how. Almost any query, even the most simple, can be executed in different ways: by rearranging the elementary operations that this query consists of; by using additional structures (such as indexes) created to speed up query execution, etc. The way a query is executed is called a query plan.

How to choose the best query plan? What DBMS component is responsible for this task? Almost all DBMS that use high-level query languages feature so-called query optimizers. The main function of a query optimizer is to generate possible query execution plans and choose the best one among them. As a rule, the best plan is the one taking less time to execute. Most often it is estimated by the number of read operations, but in some cases the criteria may depend on application requirements. For example, it is required to minimize the time of fetching of the first rows of a query.

Optimizers select plans based on an explicitly or implicitly defined query cost function. Today, industrial DBMS have relatively well-developed and implemented optimization algorithms. Though fairly rare, it is required to do manual setting (using so-called optimizer hints) or complete rewriting of queries to improve their performance. Another aspect of a high-level query language is the data processing type that the query language provides for, i.e., which data units can be processed with a single language command. There are two types of processing:

- processing of separate objects;
- bulk processing.

For example, SQL was originally designed for bulk data processing. A large number of rows can be processed with a single command. Moreover, a feature of all popular SQL implementations in the traditional DBMS is the fact that such processing commands are much more efficient than processing of unique objects. It means that it is more efficient to process 10000 database objects with one command, than to process separate objects 10,000 times. However, it should be noted that systems where query languages are focused on processing of separate objects also exist and, moreover, are being actively developed and discussed.

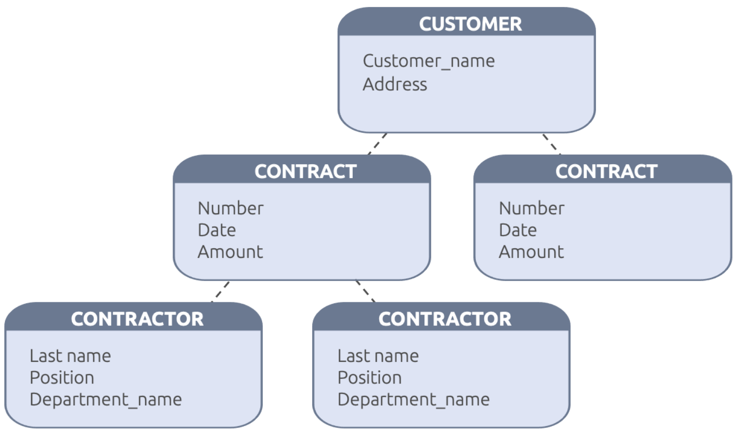
Some time ago, we could still classify DBMS based on the types of supported data models: hierarchical, network, relational, etc. Database models originated in the order listed in the Figure.

Data models

- Hierarchical
- Network
- Relational
- Etc.

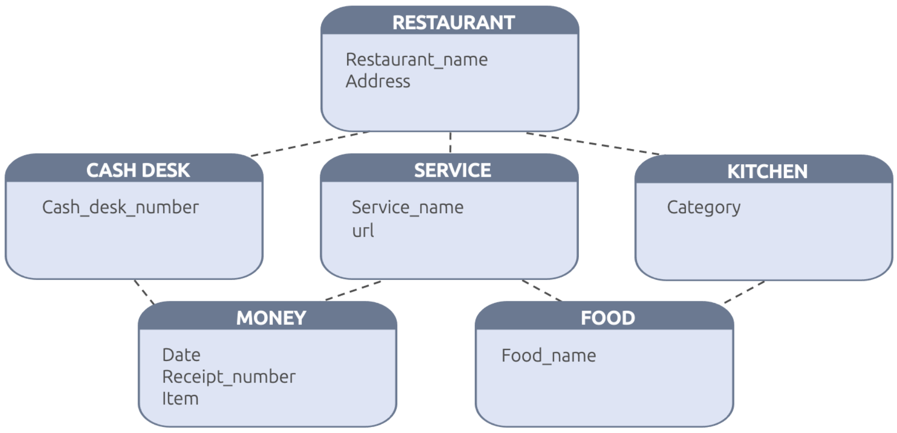
In hierarchical databases, each child record has only one parent. This creates a tree-like structure in which records are classified according to their relations with the parent record chain. The Figure shows an example of data organized in such a way.

Example of hierarchical structure



Network database models extended the functionality of hierarchical databases: records could have more than one parent. Consequently, it became possible to model quite complex relations. The Figure shows an example of data organized in such a way.

Example of network structure



Relational data models became the most widespread. They are based on the idea of presenting any data in the form of structured tables. Each column in a table has a name and type. Each row represents a separate record of an object stored in the database. DBMS that manage data of such structure are commonly referred to as relational or traditional.

Example of relational structure

Number of personal file	Last name	First name	Patronymic name	Gender	DOB	Profession
16493	Sergeyev	Petr	Mikhailovich	M	01.01.86	080104
16593	Petrova	Anna	Vladimirovna	F	15.03.85	080102
16693	Anokhin	Andrey	Borisovich	M	14.04.86	080104

Due to the diversity of existing data models, it is now common to divide models (at least at the highest level) into the following:

- structured;
- semistructured;
- schemaless.

Now let's clarify what this means. If stored data has a specific set of same-type attributes, this data is called structured. Examples of such data would be the data of bank customers, university students, goods in a store. Relational data model is ideal for working with structured data. But today there is a huge amount of data without a clear structure – photos, videos, text documents – so-called schemaless data. If you manage to extract at least some of the common attributes – creation date, document author, size and format of a photo – then such data becomes semistructured.

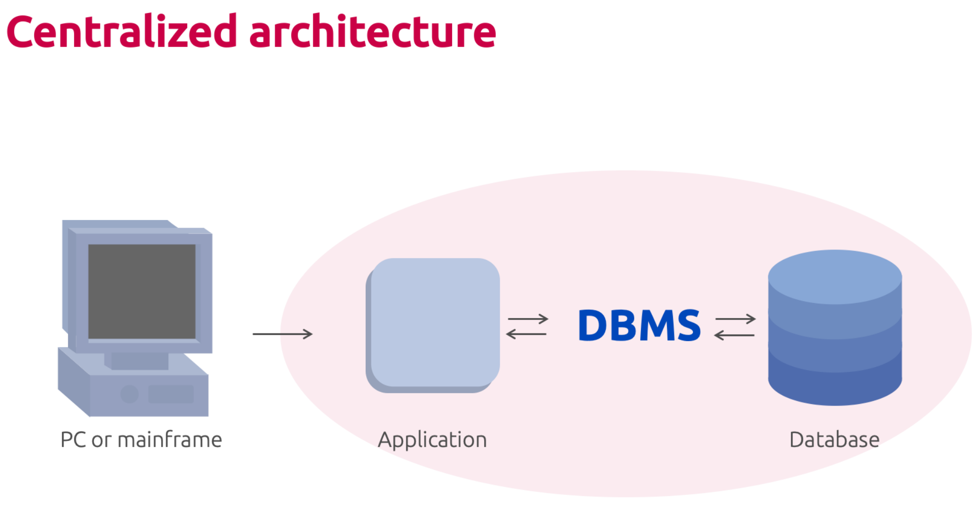
Today, most of major vendors that originally marketed their DBMS as relational ones (for example, ORACLE and DB2) allow for creating different data models, including the schemaless ones. Such databases are commonly referred to as multi-model DBMS. Our course will cover different data models, starting with the most widespread – the relational data model.

4 DBMS architecture

One of the most important characteristics of modern DBMS is the architecture that allows multiple users to work with the data. The advancement of computing technology and software have been initiating the development of architectural solutions used for implementing data access in the scope of using DBMS. Let's consider these solutions chronologically.

4.0.1 Centralized Architecture

When using a centralized architecture, the database, DBMS and application program (i.e., the application) are located on the same computer. The work is organized as follows: The work is organized as follows:

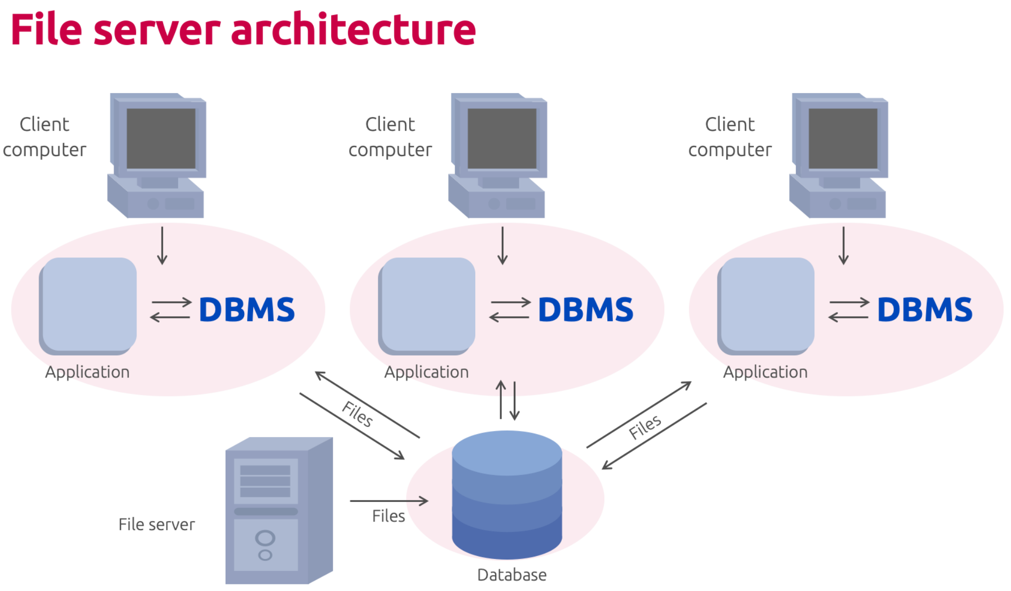


- database is located on the computer hard drive;
- DBMS and database application are installed on the same computer;
- user runs the application and initiates access to the database to retrieve/update information;

- each access to the database runs through DBMS, which contains information about the database physical design;
- DBMS ensures execution of user queries;
- DBMS returns query results to the application;
- application displays query execution results.

4.0.2 File Server Architecture

The invention of personal computers and local area networks gave rise to new architecture called file server architecture. In this architecture, one of network computers is set up as a dedicated server intended for storing database files. In



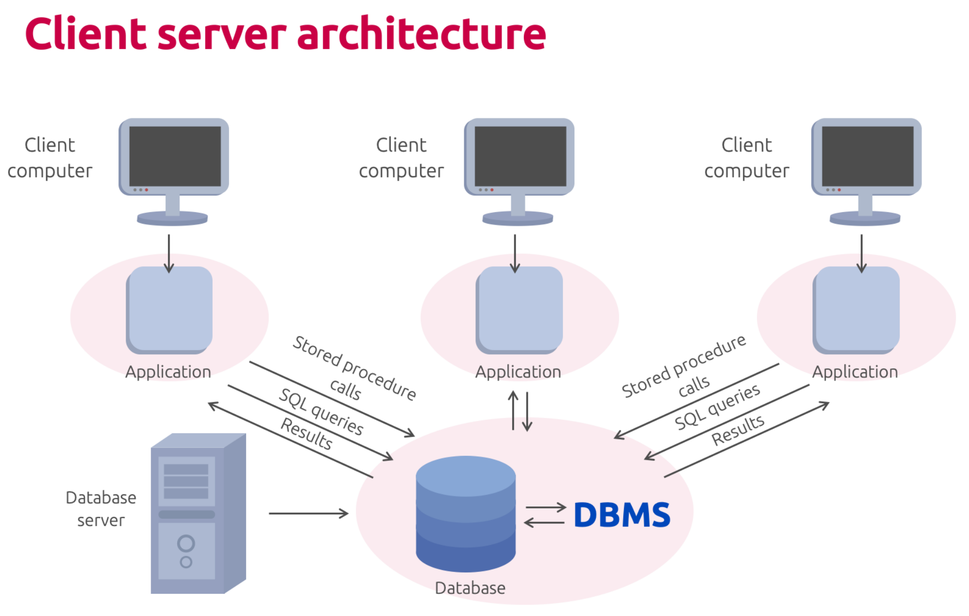
accordance with user queries, files from the file server are sent to the user's client computers, where the major part of data processing takes place. The dedicated server acts as a file repository, but it is not involved in data processing. So, the work is organized as follows:

- database is located on the hard drive of a dedicated computer (file server);
- local network connects the file server and client computers;
- DBMS and database application are installed on each client computer;
- applications initiate access to the database to retrieve/update information;
- each access to the database runs through DBMS, which contains information about the physical design of database located on the file server;

- DBMS accesses data on the file server;
- as a result of DBMS accesses, part of database is copied to the client computer and processed there;
- in the case of changes, the data is sent back to the file server to update the database;
- DBMS returns query results to the application;
- application displays query execution results.

4.0.3 Client Server Architecture

The following client-server architecture also requires networked computers, one of which performs specific control functions (it is referred to as the database server). Client-server architecture separates the functions of user application (commonly referred to as the client) and server. Client application makes a query in a special query language (usually SQL) to the server with the database. In this case, the resources of client computer are not involved in query execution. Client computer only sends a query to the database server and receives the result, which is then interpreted and displayed to the user. Since only the query result is sent to the client application, only the data needed by the client is sent over the network. As a result, network load is reduced.



So, the work is organized as follows:

- database is located on the hard drive of a dedicated computer (database server);

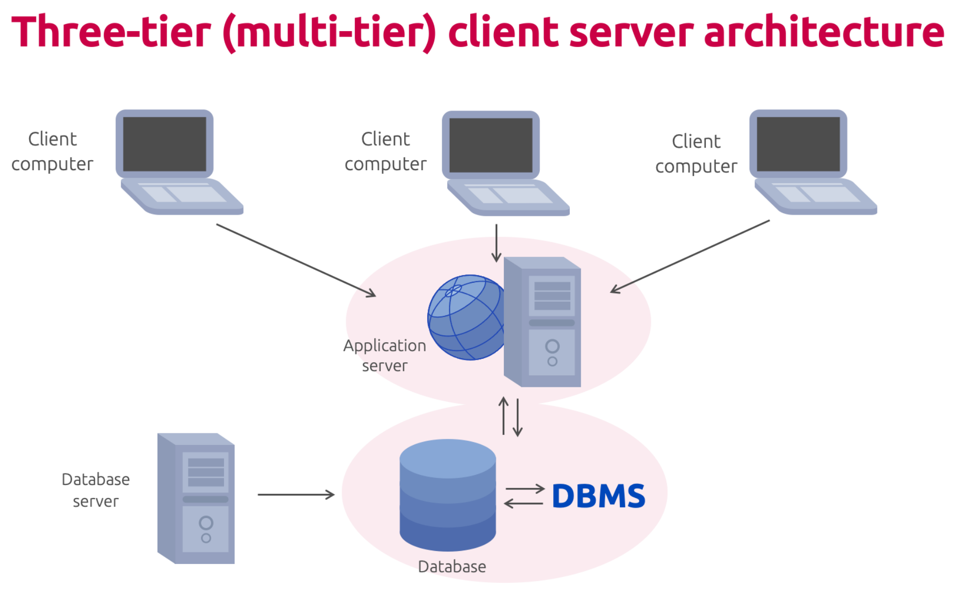
- DBMS is also located on the server;
- network connects the database server and client computers;
- database application is installed on each client computer;
- applications initiate access to the database to retrieve/update information. A query language is used (usually SQL) to communicate with the server. It means that only the query text or calls to procedures and functions stored in the database are transferred over the network from the client to the server;
- each access to the database runs through DBMS, which stores information about the database physical design;
- all queries are executed on the server;
- only the results of queries are sent to the client computer (to the application);
- applications display query execution results;
- transaction mechanism supported by DBMS does not allow for simultaneous modification of the same data by different users and backs up the database to the common state in case of software and hardware errors.

4.0.4 «Client-Server» Multi-tier Architecture

In a three-tier (or multi-tier) client-server architecture, business logic previously included in the client applications is separated into an element called the application server. This leaves the client applications only with the user interface. As a rule, the client application is a Web-browser. What advantages does a three-tier architecture provide? Now, when the business logic changes, it is no longer needed to change the client applications and update them for all users. In addition, the requirements for user devices are minimized.

As a result, the work is structured as follows:

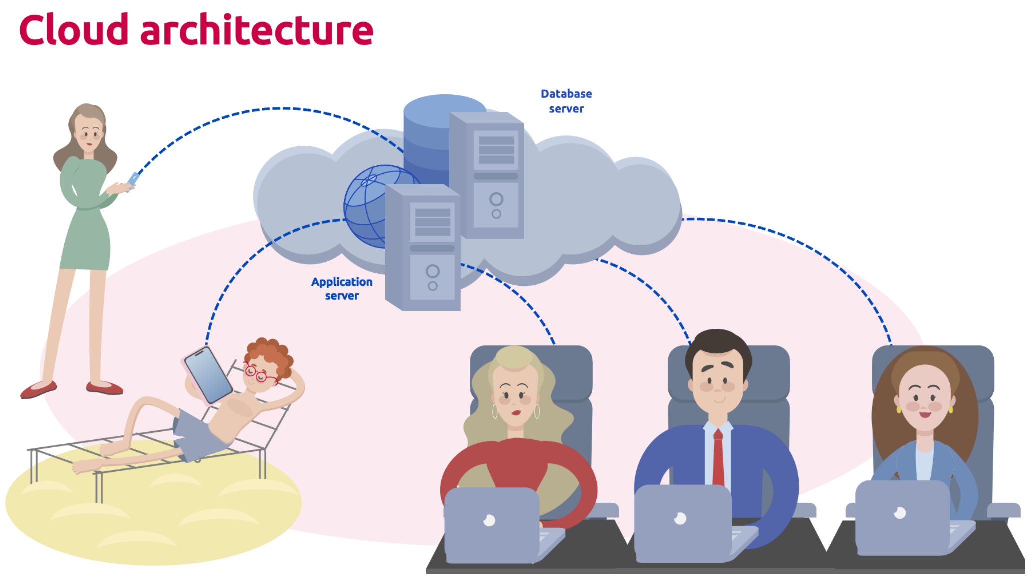
- database is located on the hard drive of a database server;
- DBMS is also located on the server;
- there is a dedicated application server that hosts the software (SW) for implementing business logic;
- there are many client computers, each with a so-called «thin client» – a client application that implements user interface (usually a Web-browser);



- network connects the database server, application server and client computers;
- users can run the application – thin client, on each of the client computers;
- thin client on the client computer initiates access to the application server SW;
- application server analyzes user's requirements and forms queries to the database. A query language is used (usually SQL) to communicate with the database. It means that only the query text is transferred over the network from the application server to the database server;
- each access to the database runs through DBMS;
- query execution results are returned to the application server;
- application server analyzes query results and returns the result to the client application;
- client application displays the result.

4.0.5 Cloud Architecture

Today, among a variety of cloud services available on the software market, there is also such useful service as cloud access to DBMS. It is particularly important for enterprises that do not want to invest in LAN infrastructure, administration, scaling and other overhead costs typical for DBMS. So, what is a cloud DBMS?



Cloud DBMS is a fully automated multi-user and unlimitedly scalable service that provides DBMS functionality via the Internet, while being managed and administered by the service provider. At the same time, cloud DBMS (DBaaS – Database as a Service) should not be confused with DBMS running in a virtual machine (i.e., when a user is given a virtual machine in the cloud, and this user independently (or with someone's help) administers DBMS, which may be installed there). Today, cloud DBMS is the most demanded architecture that makes small and large enterprises free from the routine work associated with both creating infrastructure and administering databases.

As part of our course, you will become familiar with several DBMS, and access to them will be organized using the cloud architecture.

5 Introduction to Relational Databases

As we mentioned before, the most popular model for storing structured data is the relational model. Edgar Codd proposed the relational data model in the late 1960s in his paper "A Relational Model of Data for Large Shared Data Banks". Edgar Codd later proposed the basic rules that a database management system must satisfy. Let's consider the key features specific for the relational model. According to E. Codd, all stored information can be represented as a set of tables, or so-called relations. Each table has its own unique name.

As an example, let's look at a simple table `STUDENTS` that contains data of the students of a university.

Data is recorded as rows in tables; each row corresponds to a specific data element, which in our case is a student. The table has a certain preset number of columns. Each column has its own unique name. As shown in the example, the table contains column `id` that corresponds to the student's ID number, column

STUDENTS

id	name	birth_date
1	Ivan Igorevich Maksim	22.11.1985
2	Petr Alekseevich Rebko	15.12.1992
3	Ivan Igorevich Maksimov	18.05.1995
4	Georgiy Sergeevich Petrov	11.03.1988
5	Evgeniy Ivanovich Ivanov	05.08.1984

`name` that contains a row with the student's last name, first name, and patronymic. The last column `birth_date` stores the student birth dates. Each row of the table has a fixed number of elements corresponding to the number of columns in the table. Each column stores values corresponding to a certain attribute or property of an object (in our case, a student).

Each row of the table has a fixed number of elements corresponding to the number of columns in the table. Each column stores values corresponding to a certain attribute or property of an object (in our case, a student).

Table

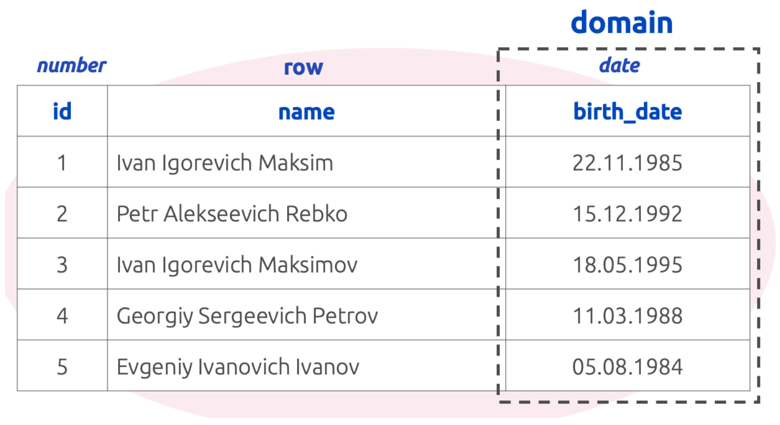
id	name	birth_date
1	Ivan Igorevich Maksim	22.11.1985
2	Petr Alekseevich Rebko	15.12.1992
3	Ivan Igorevich Maksimov	18.05.1995
1	Ivan Igorevich Maksim	22.11.1985
4	Georgiy Sergeevich Petrov	11.03.1988
4	Georgiy Sergeevich Petrov	11.03.1988
5	Evgeniy Ivanovich Ivanov	05.08.1984

Relation

id	name	birth_date
1	Ivan Igorevich Maksim	22.11.1985
2	Petr Alekseevich Rebko	15.12.1992
3	Ivan Igorevich Maksimov	18.05.1995
4	Georgiy Sergeevich Petrov	11.03.1988
5	Evgeniy Ivanovich Ivanov	05.08.1984

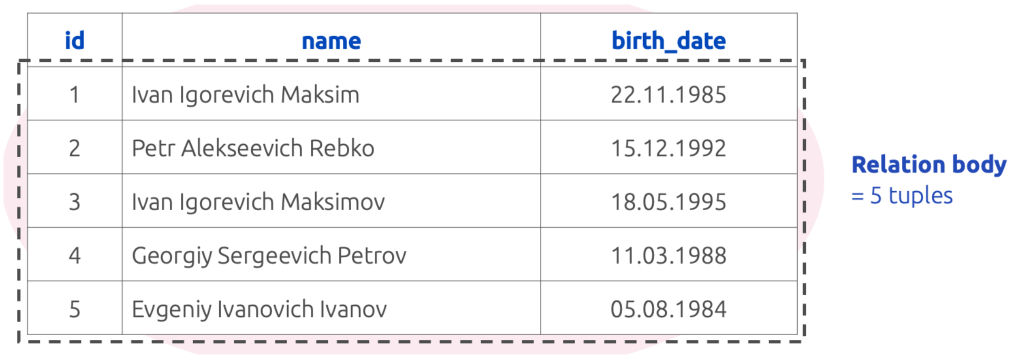
The relational model is based on the concept of relation. What is the difference between a table and a relation? A relation can be represented as a table containing many values. A relation can't have repeated rows, while the order of the rows is irrelevant. Therefore, table rows must be different by the value of at least one attribute. Relation rows are often called tuples and columns are called attributes. Let's go back to our example.

Note that every table column contains data of a certain type (dates, rows, integers). Column `id` of table `STUDENTS` stores the identification numbers represented by integer values. Column `name` stores students' first and last names as row values. The appropriate data type for the date of birth column `birth_date` is date. Thus, for each relation attribute (or table column), there is a permissible set of values. Such set of values is called **domain**.


domain
number
row
date

id	name	birth_date
1	Ivan Igorevich Maksim	22.11.1985
2	Petr Alekseevich Rebko	15.12.1992
3	Ivan Igorevich Maksimov	18.05.1995
4	Georgiy Sergeevich Petrov	11.03.1988
5	Evgeniy Ivanovich Ivanov	05.08.1984

For relations, it is also worth mentioning such concepts as **relation header** and **relation body**. The header consists of a fixed set of attributes. The body consists of a time-varying set of tuples. In our case (for relation `STUDENTS`), the header consists of attributes `id`, `name`, `birth_date`. The relation body currently consists of 5 tuples (there are 5 rows in the table). Each tuple corresponds to a certain student. The degree of relation is the number of its attributes. The degree of relation `STUDENTS` is 3, because relation `STUDENTS` contains three attributes. The cardinal number (cardinality) is the number of its tuples. For table `STUDENT`,


Relation body
= 5 tuples

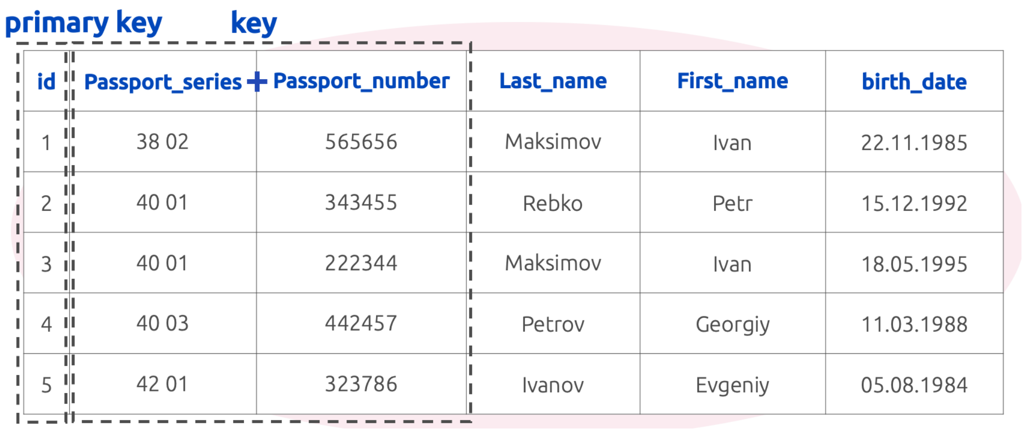
id	name	birth_date
1	Ivan Igorevich Maksim	22.11.1985
2	Petr Alekseevich Rebko	15.12.1992
3	Ivan Igorevich Maksimov	18.05.1995
4	Georgiy Sergeevich Petrov	11.03.1988
5	Evgeniy Ivanovich Ivanov	05.08.1984

the cardinal number is 5, because the relation contains 5 tuples at this time. The relation cardinal number changes over time.

So-called keys are used to identify tuples. The key is a minimum set of attributes. Using its values, it is possible to exactly determine the required relation tuple. For `STUDENTS`, the key is attribute `id`, which is the student's ID number, and these numbers cannot be repeated.

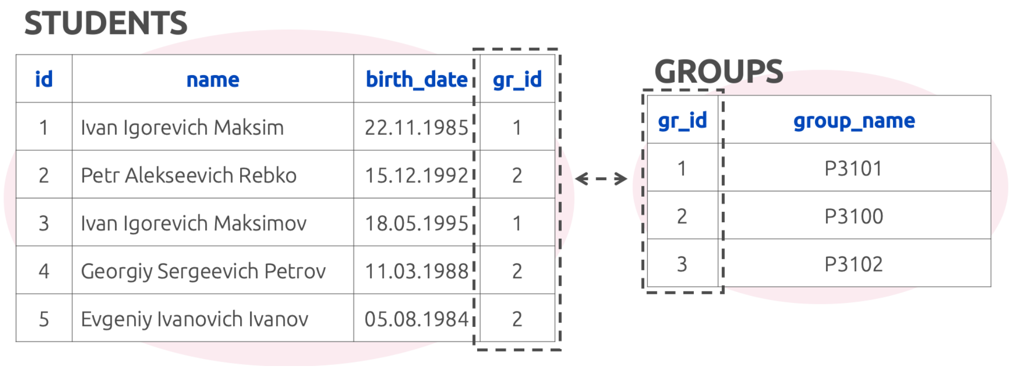
It should be noted that the key is not necessarily one attribute. It can consist of several attributes (for example, passport series and number), the values of which exactly determine the tuple in a relation.

A set of key's attributes should be minimum. In addition, the key should not include attributes that are not necessary for identification. If there are several possible options for the key in a relation, then the most useful one during search is identified. It is called the **primary key**.


primary key
key

id	Passport_series +	Passport_number	Last_name	First_name	birth_date
1	38 02	565656	Maksimov	Ivan	22.11.1985
2	40 01	343455	Rebko	Petr	15.12.1992
3	40 01	222344	Maksimov	Ivan	18.05.1995
4	40 03	442457	Petrov	Georgiy	11.03.1988
5	42 01	323786	Ivanov	Evgeniy	05.08.1984

An important part in the data storage is the organization of links between different elements. The relational data model peculiarity is that tables are also used for link representation. The keys of connected objects are used for link organization. Let's show it by the example.



STUDENTS

id	name	birth_date	gr_id
1	Ivan Igorevich Maksim	22.11.1985	1
2	Petr Alekseevich Rebko	15.12.1992	2
3	Ivan Igorevich Maksimov	18.05.1995	1
4	Georgiy Sergeevich Petrov	11.03.1988	2
5	Evgeniy Ivanovich Ivanov	05.08.1984	2

GROUPS

gr_id	group_name
1	P3101
2	P3100
3	P3102

Let's add relation `GROUPS` to represent student groups. This new relation has an attribute – `gr_id`. Each tuple has a unique value for that attribute. To connect students to their groups, you need to add a new attribute to table `STUDENTS` – let's call it `gr_id`. The values for this attribute will correspond to the values of attribute `gr_id` from relation `GROUPS`. Thus, we can make a connection between elements of different tables.

In addition to the data itself, when creating tables, you can describe the rules that the stored data must satisfy. For example, it isn't possible to add to table `STUDENTS` a value `gr_id` that does not exist in table `GROUPS`.