MINISTRY OF SCIENCE AND HIGHER EDUCATION OF THE RUSSIAN FEDERATION

Federal State Autonomous Educational Institution of Higher Education

"SAINT PETERSBURG STATE UNIVERSITY

AEROSPACE INSTRUMENTATION"

DEPARTMENT OF COMPUTER TECHNOLOGIES AND SOFTWARE ENGINEERING

Modern relational databases.

Guidelines for performing laboratory work

Saint Petersburg

2024

Compiled by: N.V. Putilova, M.V. Velichko

The issues of design, development and server programming of relational and object-relational databases are considered using MySQL and PostgreSQL as examples. Many examples of writing various queries and procedures for the corresponding DBMS are given. The purpose of DBMS of other data models is also considered. The teaching aid is intended for students of all forms of study in the areas 02.03.03 and 09.03.04, studying the discipline "Database Design". In

The manual includes brief theoretical information necessary for completing laboratory work.

Content

Introduction

Most modern applications and software packages have
databases in their composition, often of different models and purposes. Often
you can find a relational database as the main storage,
key-value for caching server queries and a column-oriented database for
complex analytical work in one application. Modern
A software engineering specialist needs to understand
Features of designing and working with modern databases.
Therefore, this manual is relevant for studying by students
areas related to software engineering.
The teaching aid focuses on the issues
design and work with relational databases. The methodology is given
analysis of user stories of the subject area for construction
conceptual model. Examples of writing queries with
using various techniques. Server programming is discussed in,
represented by stored procedures, functions and triggers for DBMS
MySQL and PostgreSQL. Examples of data access control are given on
based on roles and user privileges. Tutorial
consists of 10 sections, in each of which the theoretical part is supplemented
a large number of practical examples. To gain skills and
skills, students are offered 8 laboratory works. In the first 4 sections
The design of databases is considered accordingly its stages in
in general, in the first section there is conceptual, logical and physical
design in the following. The fifth section is devoted to the use of language
data definitions in SQL. Sections six through eight show the different
aspects of data manipulation in relational databases. Ninth and
The tenth section covers the specifics of server programming and
MySQL and PostgreSQL access control. Presented in
The bibliographic list of materials will allow students to obtain more
deep knowledge and skills in the field of modern databases.

Stages of database design

The following stages of database design are distinguished[1,2]:

1. **Conceptual design.** Conceptual design **is**
   it is the process of creating a model of the information used that is independent of any physical aspects of its presentation.
2. **Logical design.** Logical design is a process
   creating a model of the information used based on the selected model data organization, but without taking into account the specific target DBMS and others physical aspects of implementation.
3. **Physical design.** Description of a specific implementation of the base data placed in external memory determines the organization files and index composition.

In this case, the concept of a data model is considered to be abstract, independent, logical definition of data structures. Thus, there is one conceptual model of the subject area and many logical data models for this conceptual, by number existing data models for DBMS, and physical ones for this there can be at least as many logical ones as there are specific DBMSs for of this data model. In this case, for each DBMS, one logical models can be set to many different storage parameters. General the relationships between models at different stages of design can be considered in Fig. 1.

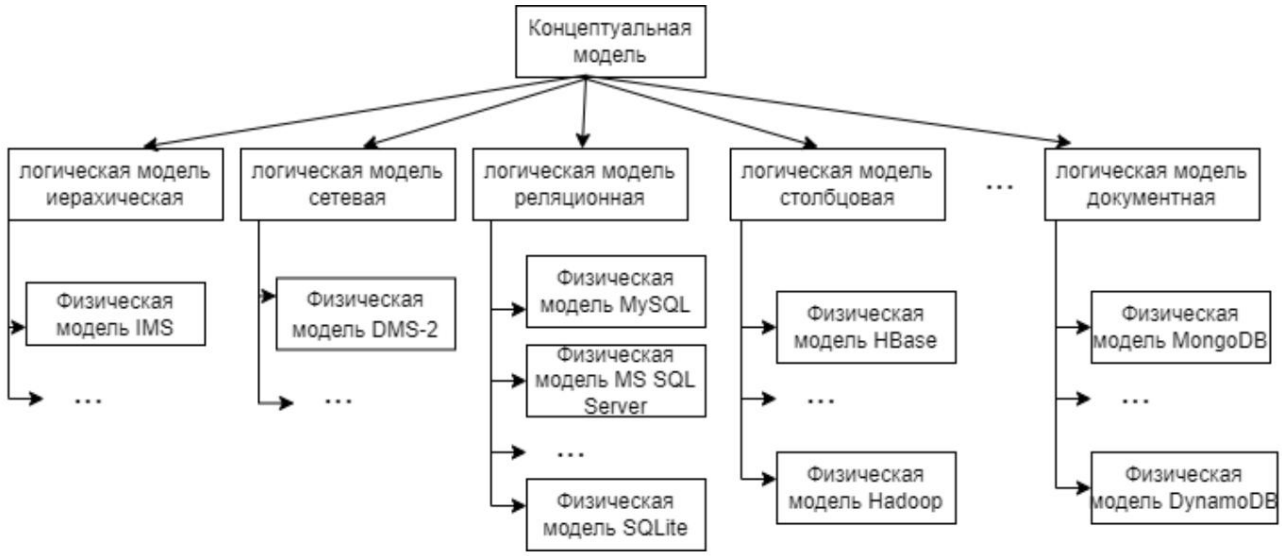

Fig. 1. Relationship of models at different stages of design

Conceptual Database Design

Entity-Relationship Model

For data modeling in the conceptual methodology paradigm the entity-relationship model (or ER model, from English) is used "Entity-Relationship"), developed by Peter Chen in 1976[3]. ER model allows to formalize the structure and relationships between information objects of the subject area under consideration. All objects of real the world have different properties: the machine is characterized by its size, weight, license plate; a person has a first name, last name, address, etc. When In developing a model, an object or class of objects is considered as some **entity,** and data elements that describe properties objects, - as **attributes** of entities. Relationships are established between entities **connections that** represent relationships between real-world objects in the model. The basic elements of the model are the concepts of attribute, entity (also called **entity types)** and relationships.

An attribute is an element of information, some characteristic of an object.

An entity is described by a set of attributes. Each attribute describes a separate property of an entity. For example, for the object "BOOK" the attributes there will be the author's surname, year of publication, place of publication, etc.

Entity type (often just entity)A group of objects with the same properties, which are considered in a specific subject area as having an independent existence.

Entity instance. A uniquely identifiable object that refers to an entity of a certain type.

Each entity type in an ER model is assigned a name. Because the name represents a class or set of objects, then to denote the type of entity a simple singular noun is used[4].

A relationship describes a connection between data (entity types)

All entities or relationships of the same type have certain common properties. Entities are divided into ordinary (strong) and weak entities. **A weak** entity is one whose existence depends on another entity, i.e. it cannot exist unless that other entity the entity does not exist. An entity that does not exist **is called ordinary (strong).** is weak. **Example: Student** (record book number, full name, date of birth) – strong entity, student's personal account (ID, login, password, creation date) - weak the entity does not exist without the student.

Property characteristics:

A simple or **compound** property. For example, the property "employee name" could be be composite if its value is composed of the values of simple properties "name", "patronymic" and "surname".

Key property or **non-key** property (Key property is a property that which may be unique only in a certain context).

For example, a student's first and last name is usually unique only in context of data about the group in which he studies.

The following can be used as a key:

- **Natural Key (NK)** – a set of attributes of the entity described by the record that uniquely identifies it (for example, a passport number for a person).
- **Surrogate Key (SK)** – an automatically generated field, no not related to the information content of the entry.

A single-valued or **multi-valued** property (i.e. this model allows repeating groups). But if a certain person has several different emails, then the login property on the site for it can be polysemantic.[5]

Missing (optional) property (i.e. property *is unknown* or *not applicable* for some instances of the entity) and **mandatory.**

A base or **derived** property. For example, the total number of students in a certain group can be calculated by summation counting individual students in a given group (date of birth is the base, age derivative).

Relationships. The name of the relationship is presented in verb form and describes a relationship between two or more entities. The entities involved in the relationship are are called its participants, and the number of participants in the connection is called its **degree** [5]. The following types of connections are distinguished by **Power** (cardinalities):"one-to-many" (1:M) (1:*),"many-to-many" (M:N) (*:*), "one-to-one" (1:1).

Description of power types is presented in Table 1.

Table 1. Description of communication power types

Power Explanation		An example
"one-to-to many" (1:M) (1:*)	One instance of entity 1 is associated with a set of instances of entity 2, but entity instance 2, associated only with one instance of entity 1	Man - mobile phone. A person can have several numbers, but the number (sim card) registered to only one
"many-co-to many" (M:N) (*:*)	One instance of entity 1 is associated with a set of instances of entity 2, and entity instance 2 is associated with many instances of entity 1	Student - interest groups. A student can go to several circles, and goes to one circle several students

"one-to- to one" (1:1)	One instance of entity 1 is associated with only one instance of entity 2 and entity instance 2 is associated only with one instance of entity 1	Student - personal account. A student has only one personal student office and personal office strictly tied to only one to a student

In the extended ER model, subtypes and supertypes of entities are also distinguished.
Where supertype is an entity that unites different subtypes, which
must be represented in the data model. A subtype is an entity that
being a member of a supertype but performing a separate role within it.

Entity-Relationship Diagram (ER Diagram)

ER is used to document conceptual models.
diagram - a special type of graph diagram representing
entities (named graph nodes) and connections between them (named arcs
graph, marked with special symbols).[6]
Despite the unified model, there are several different notations for
display diagrams. Among the main notations, the following can be distinguished:

- Peter Chen Notation

  Barker Notation
- Crow's Feet notation, which is a connection

  IE (Information Engineering) notations (J. Martin and K.

  Finkelstein (Clive Finkelstein))-
- C. Bachman notation
- ÿÿÿÿÿÿÿÿIDEF1X (Integration Definition for Information Modeling)

The two most commonly used notations are the Chen model and the bird's eye model.
paw". In the bird's paw model, the entity is represented by a rectangle with
nested attribute descriptions, and connections by lines, the ends of which
visually show power.

Basic elements of entity description in bird's foot notation
are shown in Fig. 2.

Fig.2. Display of entities and attributes in the "bird's foot" notation: a-
associative entity, b - strong entity, c - weak entity, d-
description of various attributes of an entity

An associative entity is used to reveal an M:M relationship or relationships
high degrees

Different line endings can be used to display connections.
connections denoting the plurality and obligatory nature of the connection. Total 4
combinations: (1 and only 1), optional one (0 or 1), mandatory
many (1 or many), optional many (0 or many). They are represented
in Fig. 3.

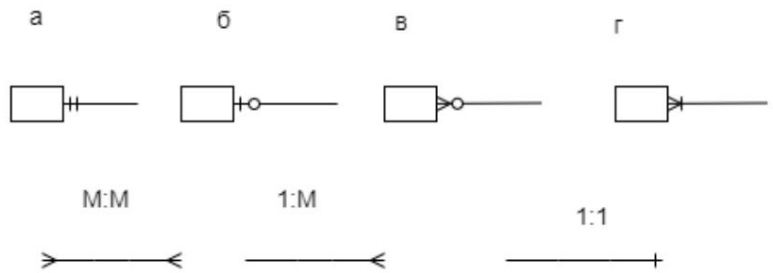

Fig. 3. Display of types of links in the "bird's foot" notation: a –
mandatory one b –1and only 1, mandatory connection, c–optional
connection many, g - mandatory connection many

In Chen's notation, the entity is also represented by a rectangle, but the attributes
located in ovals around it. Basic elements of entity description
in Chen's notation are shown in Fig. 4

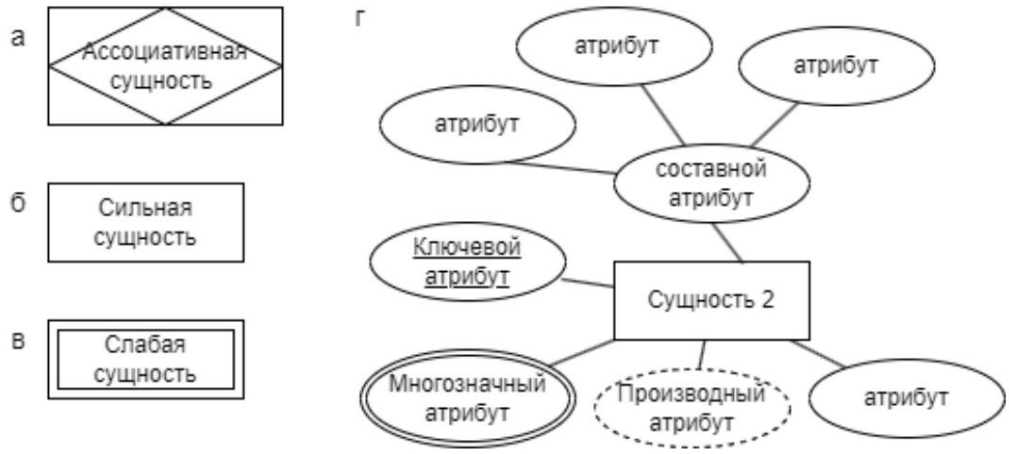

Fig. 4. Display of entities and attributes in Chen's notation a- associative essence, b - strong essence, c - weak essence, d - description of various attributes of an entity

The connections are shown as a diamond, where the power is written as a number, and the obligation is indicated by a dash, as in Fig. 5.

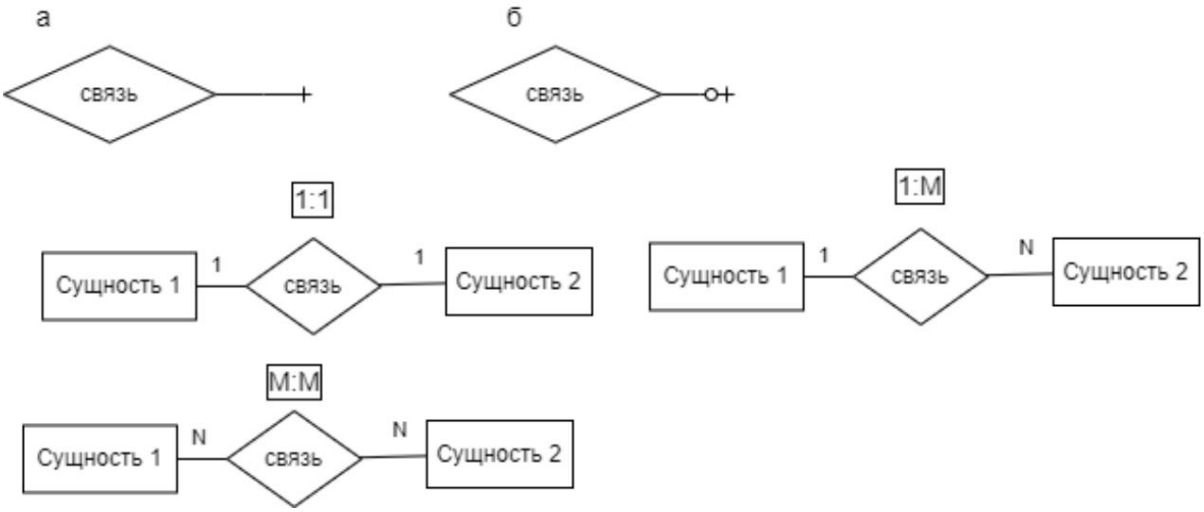

Fig. 5. Display of link types in the "bird's foot" notation: a – mandatory b – optional connection

There are quite a lot of tools for working with ER diagrams, including They include CASE tools (e.g. DB Designer, Oracle SQL Developer Data Modeler) and universal diagramming tools (Draw.io, Visio, Lucidchart)

Rules for creating an entity-relationship diagram.

• Entity names are unique within a diagram

• Attribute (property) names are unique within a diagram

• Each entity must have a key (key attributes)

Methodology of subject area analysis

Steps in domain analysis:

1. Extraction from the task description (technical assignment, user histories, requirements, etc.) of the main concepts of the subject area and their values. The concepts will then be divided into attributes and entities.
2. Determining the connections between concepts. We write out all the concepts and we connect with lines (or somehow indicate the connection) those that are logical are directly related.

All the relationships that have been defined are either relationships of an attribute with its entity or a relationship, or a connection between two or more entities. Note that a connection can exist between a concept and itself, and between two concepts, and between a large number of concepts.

3. Determining the communication power from step 2.

This is the definition of the maximum number of instances of an entity at the other ends of the connection an instance of the given one can be connected and vice versa.

4. Determining whether a concept is an attribute or an entity.

The following questions are used for this:

1) Does it exist independently? If yes – essence. If no – not. defined because it can be an attribute and a weak entity.

2)Are there any proper properties (attributes)? If yes – entity. If no – not defined. The name of something may already be a property.

3) Does the relationship involve many-to-many or one-to-many from the outside? many? If yes, the essence.

5. For each attribute, determine its affiliation.

If in step 2 the attribute was defined as being associated with a single entity, then this is an attribute of this entity. And if it is associated with 2 or more concepts (entities), then this is an attribute of the connection of these entities (if it exists) or an associated entity reflecting their connection.

6. Additional definition of attributes.

It may be necessary to add attributes to the selected entities, not specified in the description of the subject area, since many things are implied. For example, when talking about an employee, a student or in a person, the need to store the full information is rarely explicitly indicated. name. If possible, additional attributes should be provided. interview subject matter experts and customers.

7. Defining data types

At the conceptual design stage, it is sufficient to determine the date or time, text, or number (integer or floating/fixed point).

Determining the mandatory nature of relationships and attributes.

A mandatory attribute is one that will always be filled in for everyone. instances of the entity.

Example of domain analysis

Although a conceptual model can be designed on any in a convenient language in this case we will consider the final version on English. This is due to the convenience of working in the DBMS with tables in in English.

Task: Database for managing student sections (circles): sections, section leader, date of student joining section…

1. Extracting key concepts from the task description

Database for managing student sections (circles): **sections, section leader, date of student** joining **section**

Planned Queries

a. Select **the groups** whose **students** attend **the sections** in **the title** which have the word *"programmed",* but it is not the last one
b. Find **groups** without **students**
c. Find **a teacher** who leads (manages) a section on *aircraft modeling* and one more.
g. Find **a group whose students** attend all **sections** of *programming*

d. Find **the section** that holds **classes** first ***this month.***

e. **A teacher** who is **attended** by only **students from group 4231**

g. Find **the group** with the maximum number **of students**

Concepts are highlighted in bold, meanings are highlighted in italics. "This month"

is highlighted in bold italics because it is both a reference to the meaning and

indicates that there is a lesson date.

List of concepts: section, teacher-head of section, date

introduction (of a student to a section), group, student, section name, lesson,

date of the lesson. We check if this data is related.

2. Determining the connections between concepts.

Is the section name related to the section? (So related to it). Is the class date related to the class? (Yes, only to the class). Is the joining date related to the section? (Yes). Is the joining date related to the student? (Yes, only to the section and the student). Is the teacher and the section related? (Yes, he/she leads it). Is the teacher related to the student? (Not directly.) Is the teacher related to the group. (Not directly.) Is the teacher related to the class? (Yes, he/she can conduct it.) Is the section related to the student? (Yes, he/she can attend it.) Is the section related to the class? (Yes, the class is a section meeting.) Is the section related to the group? (Not directly, only through the student.) Is the class related to the group. (No, sections are not groups.) Is the class related to the student? (The presence of a connection depends on the subject area. If we want to mark the attendance of a section, then it is connected. But if we are not interested in attendance, then the connection can be neglected.) The result of displaying the connections between concepts is shown in Fig. 6.

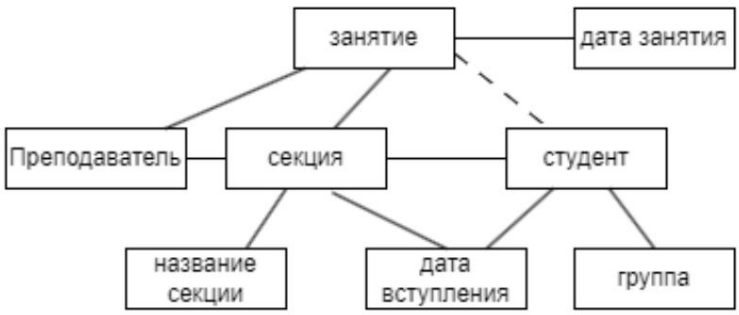

Fig. 6. Step 2 of conceptual design: connections between concepts.

Since there are no queries regarding section attendance, the connection between

We will not show the occupation and the student in the future.

3. Determining the communication power from step 2.

A section can have only one name and within one university the names
sections will be unique (belong to only 1 section) - 1:1 relationship.
A student can have many sections and many people can go to one section.
students - connection M: M. There will be many entry dates for one section, but in
A specific section can be registered by one student only once (date
entry of a specific student into a specific section) - connection 1: M. Dat
there will be many entries for one student (in different sections), but in
You can only sign up for a specific section once - connection 1: M. The section
there may be many classes, but only one specific one in one section—communication
1: M. One specific lesson has only one day (date), but on that day
there may be classes for many sections - connection 1: M. The section has one and only
one leader, but a teacher can lead many sections -
1:M connection. A teacher can teach many classes and theoretically one
the lesson can be taught by several teachers at once - M:M connection. With connection
student group is a little more complicated. If we take data with history, then in
at different times a student can study in different groups (at least bachelor's and
master's degree), and if we take current data, then only in one. In this
In this case, history can be neglected as insignificant for the subject matter
areas. There are many students in a group and a student studies in only one group—
1:M connection.

Let us display these connections in Fig. 7.

Fig. 7. Step 3 of conceptual design: connections between concepts.

4. Determining whether a concept is an attribute or an entity.

The teacher is an entity (exists independently).

A section is an entity (it exists independently).

A student is an entity (exists independently).

Occupation is an entity (participates in the M:M relationship, has a date attribute).

Date of the lesson is an attribute (does not exist independently, there are no attributes, not many participate from the outside).

The group is an entity (it exists independently, and may not have students).

Section name is an attribute (does not exist independently, no attributes, no many participate from the outside).

Date of entry is an attribute (does not exist independently, there are no attributes, no many participate from the outside).

5. For each attribute, determine its belonging

Class date is a class attribute (link only to class). Entry date is student-section relationship attribute (linked to 2 related entities student and section). Section name is a section attribute (link only to section).

6. Additional definition of attributes.

The teacher and student should clearly have a full name (at this point stage, a composite of the surname, first name and patronymic), the group number and year receipts (numbers are duplicated every 10 years).

7. Defining data types

The section title, student and teacher names are clearly textual.

We will specify them in the varchar type (strings with a limited length, which maximum length is given in brackets), date of the lesson and date of entry student in section—dates, year of admission—whole number. Group number can contain letters, so it is also varchar. For each entity also let's add surrogate keys of type INT(integer).

8. Determining the mandatory nature of relationships and attributes.

Keys are always required. Names for the student and teacher are clearly are mandatory, without them there are no people. The section without a name is also not

exists, as does a group without a number and year of admission. The occupation is clearly not is held without specifying a date. And the entry of a specific student into the section You may not know, that is, it is not mandatory.

The teacher-section connection is not mandatory for the teacher, since he may not supervise sections, but the section must have a leader. Communication - the teacher's lesson is mandatory only from the side of the lesson (someone must then conduct). The lesson is always held for some section, but the section may classes may not be scheduled yet. Also, the section may not have any students yet, and the student is not obliged to attend sections. At the same time, the student is always in some group, but the group may be without students.

Final ER diagram in bird's foot notation with entities and attributes in English is shown in Fig. 8.

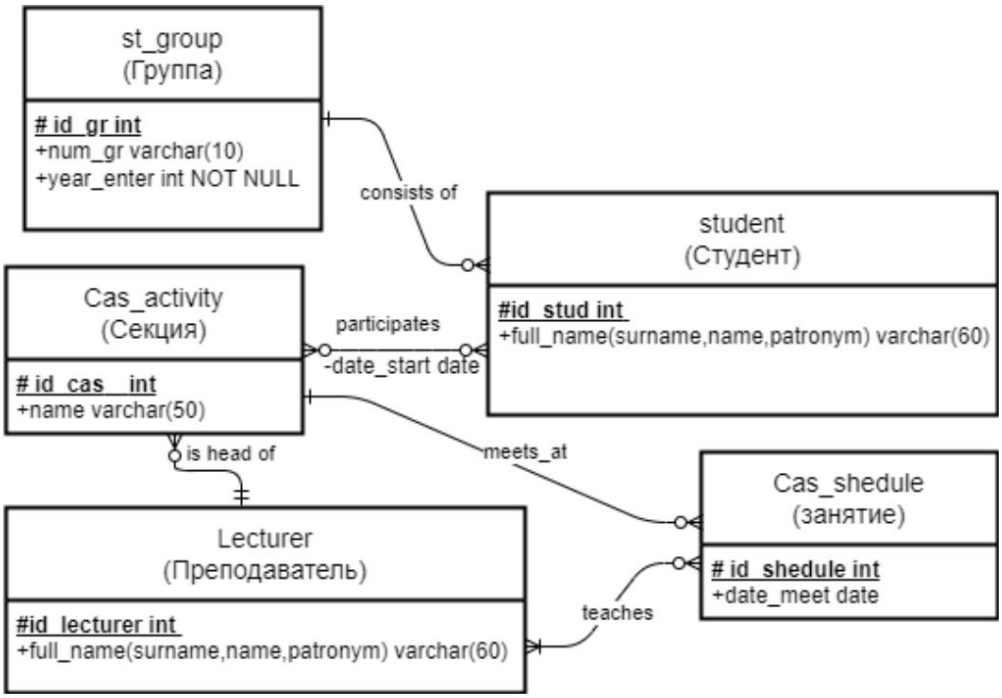

Fig.8 ER diagram of the subject area “Database for management sections"

For clarity, the names of entities are duplicated in Russian, although this is not is included in the notation.

Lab ÿ1 Development of a conceptual model of the subject area

Objective of the work: To acquire skills and abilities in constructing a conceptual model of a subject area

Task and sequence of work execution

Design a conceptual model of the subject area (ER diagram) in
in accordance with the task option. The structure of the model should ensure
storing information necessary to fulfill the requests specified in
option of the task. All entities must be named and have at least
one attribute, which must have a data type specified and
a name that is unique within the diagram. All links must be named,
They must indicate the power and obligation.

Contents of the report

- ÿ Text of the task;
- ÿ Conceptual scheme of the subject area;
- ÿ Conclusions on the construction of a conceptual model in the subject area.

Test questions and tasks

- ÿ What is the difference between an entity type and an entity instance?
- ÿ What is the difference between an entity (entity type) and a class?
- ÿ There are many students in the group, but only one of them is the head of the class. What is this a type of connection?
- ÿ How to determine if an attribute is mandatory? Is the number mandatory? passports for a person?
- ÿ What notations exist for displaying ER diagrams?
- ÿ What is the difference between a strong entity and a weak one? Give an example of a strong one and weak essence.

Logical Database Design

Logical design is primarily the design of the base in
paradigm of the selected data model. At the moment, we can distinguish

the following data models: relational, object (Object-
oriented), object-relational, key-value DBMS,
document, column family (columnar), graph.
The last 4 models are related to NoSQL DBMS. In addition to defining models
When designing a database, it is necessary to take it into account
purpose. There are 2 major areas of database purpose:
analytical and transactional, differing in structure. The first
designed for complex statistical queries for data analytics and poor
are adapted to change operations, the latter are intended for
manipulation of data in near real time. This semester
and the benefits will be considered transactional databases.

Data models and selection of applicable model for different subject areas

Despite the emergence of many different data models, relational and
object-relational databases are most suitable for
working with well-structured data or critical data. A
Non-relational models occupy niches with specific tasks.
The main purposes of the various data models are shown in Table 2.
Table 2. Purposes of various DBMS data models

Data Models Purpose	
Relational, object-relational	Working with data with a clear structure that is weakly subject to change
	Subject areas sensitive to transaction handling.
Columnar	Working with big data, including data for machine learning, Internet of Things telemetry and high traffic.
	Natural Language Processing
	Working with time series (data with timestamps that represent measurements or events)*
Count	Highly correlated data, including spam analysis and any subject areas representable as graphs

	Recommender systems
	Social media
Key-Value Data Caching	including storing session information, order baskets and so on
	Organization of locks (mutex)
	Working with queues
	User profiles, preferences
Documentary	Content Management (Blogosphere Services that include a large number of images, audio and video materials.)[7]
	Integrate disparate data to provide a unified view performance
	Mobile applications
	Real-time analytics

* —Time Series DataBase (TSDB) by model
data are most often columnar [8].

Construction of a logical relational model for a given conceptual

To obtain a logical relational model from a conceptual one
It is enough to apply a number of rules to the conceptual model.
If the entity **is strong,** then a relationship is created in the logical model, which include all simple attributes. If the entity **is weak,** then except simple attributes in the resulting relation after transformation connections with each owner entity must be defined composite primary key. **Multi-valued** attribute is converted to additional a relation representing a multi-valued attribute into which it is passed a copy of the primary key of the owner entity for use as foreign key. Transforming relationships is a little more complicated. Rules The transformations depend on the connection power.

Two-way communication of the 1:M type involves copying the primary key from the "one" side to the "many" side as a foreign key.

All attributes of the connection are also transmitted to the "many" side. An example of such The transformation is shown in Fig. 9.

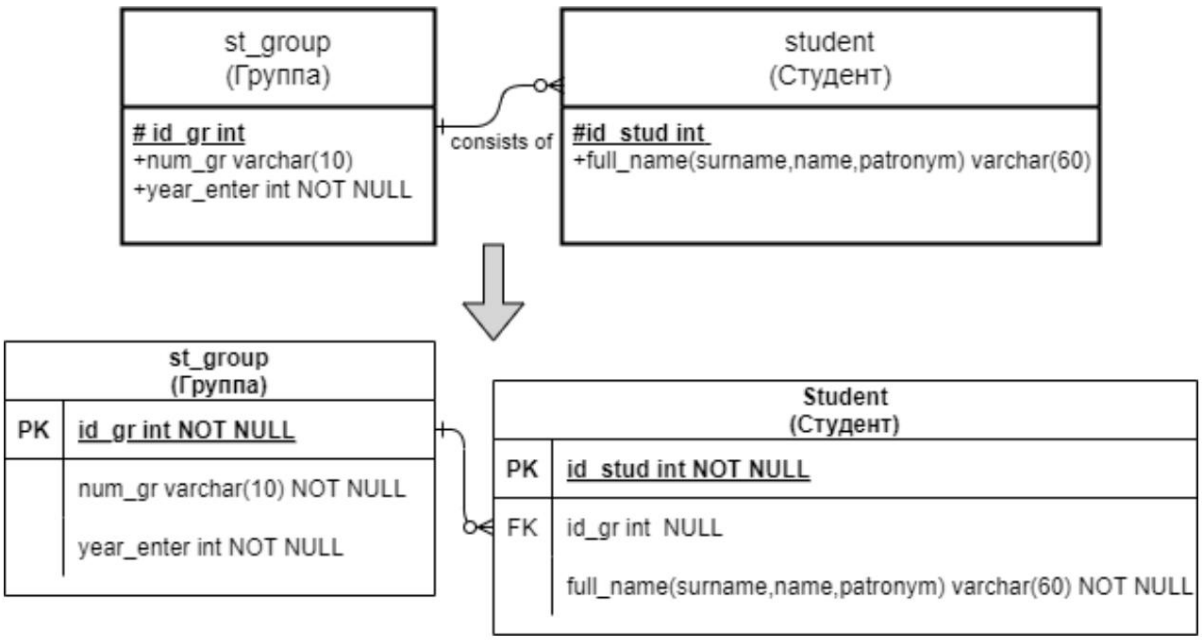

Fig. 9. Transformation of 1:M relationship from conceptual model to logical model

In the example, the primary key of the group is copied to the student as external.

Two-way M:M relationship, complex relationship (more than 2 entities) is expanded by creating an additional table representing the relationship, including all relationship attributes and receiving copies of primary keys linked tables. An example of transforming the M:M relationship is shown in Fig. 10.

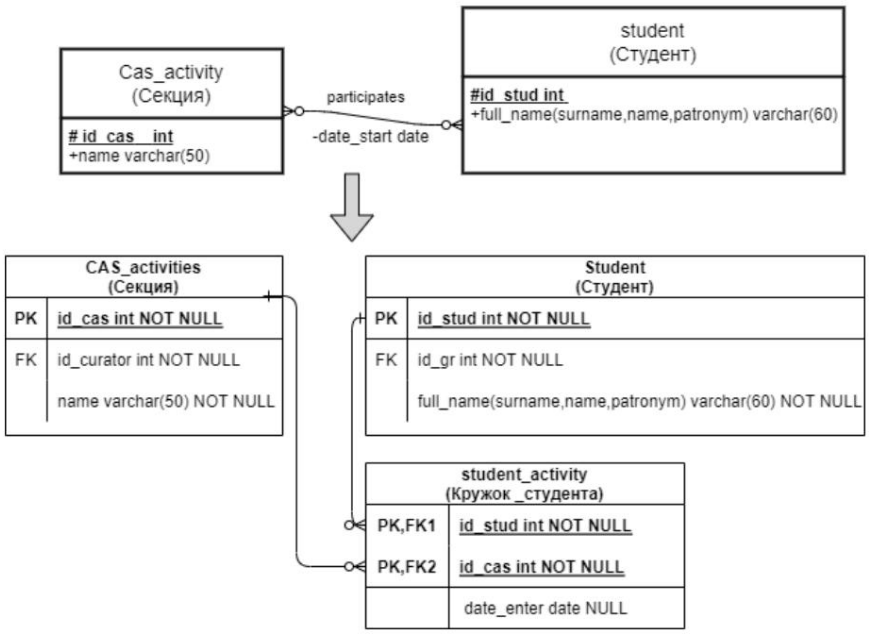

Fig.10. Transformation of 1:M relationship from conceptual model to logical model

For **a two-way 1:1 relationship,** the transformation method depends on whether the relationship is mandatory. If the relationship **is mandatory** on both sides, then the entities are **combined** into one relationship. An example is shown in Fig. 11.

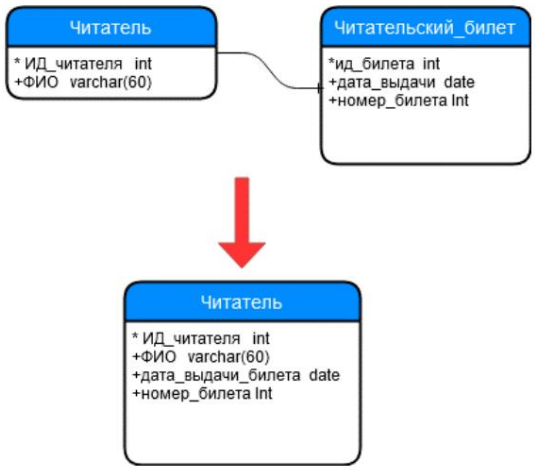

Fig. 11. Transformation of 1:1 relationship from conceptual model to logical model for obligatory parties

For **a two-way 1:1 relationship,** if the relationship is mandatory on one side, then the primary key is copied from the mandatory side (for identification) to the "optional side". An example of transformations is shown in Fig. 12.

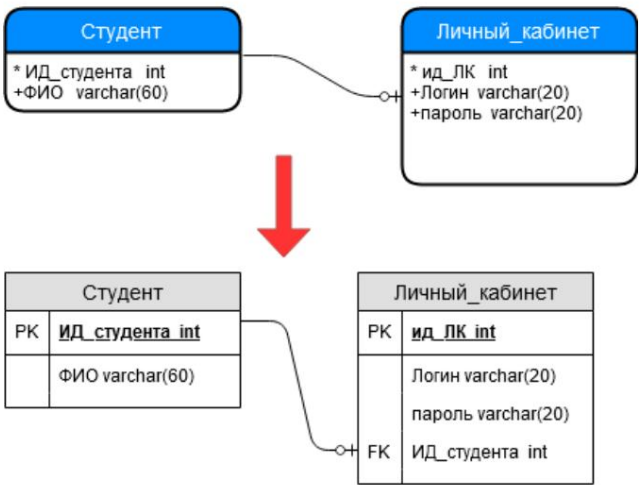

Fig. 12. Transformation of 1:1 relationship from conceptual model to logical model for mandatory one side

Database normalization

Data normalization is the process of bringing a model into a form that which allows us to obtain the structure of the database in the future, in which

storage redundancy is eliminated and anomalies are minimized
adding, deleting, changing data.

The normalization process is carried out in stages. At each stage,
some restrictions are imposed on the structure of the base. About the base with
the corresponding restrictions say that it is in one of
normal forms.

There are 8 main normal forms:

- ÿ First normal form (1NF)
- ÿ Second normal form (2NF)
- ÿ Third normal form (3NF)
- ÿ Boyce-Codd Normal Form (BCNF)
- ÿ Fourth normal form (4NF)
- ÿ Fifth normal form (5NF)
- ÿ Sixth normal form (6NF)
- ÿ Domain-Key Normal Form (DKNF)

Each normal form includes the requirements of the previous normal form.
forms and imposes its own restrictions/requirements on the structure of the database. Each
normal form on the one hand organizes the base scheme, but at the same time
time makes it a little more difficult to work with it (increases the number of tables-
relations). Therefore, as a rule, when working they are limited
using the first three normal forms.

As an example, let's consider the first 3 normal forms
non-normalized model with groups and students. It is presented in
Fig. 13.

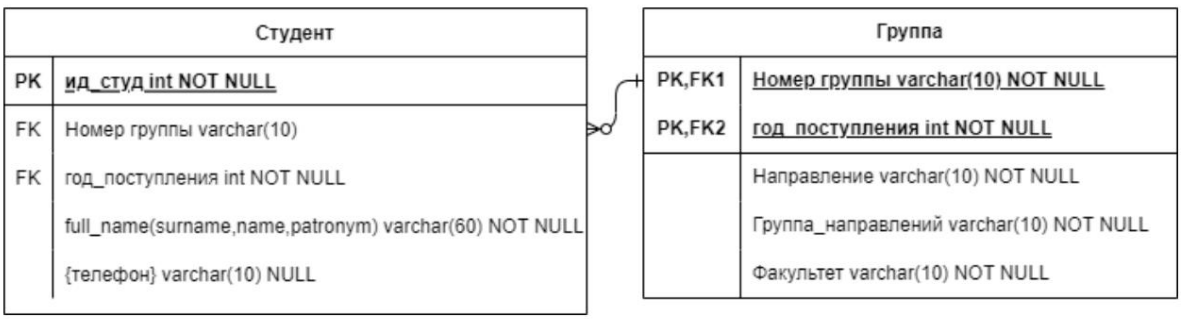

Fig. 13. Logical model of the database before conversion to the first normal form.

The group table specifies a composite primary key consisting of the group number and the year.
admission, faculty where the group studies, direction
educational program of the group in the form of a code (Direction: 09.03.04
"Software Engineering") and a large group of directions (UG) in the form
code (09 "Informatics and computer engineering"). In the direction code
The first 2 digits are the UG code.

The table (Relation) is in **first normal form (1NF)** in
in which the intersection of each row and each column contains one and
only one atomic value.

Multivalued and compound attributes violate 1NF. To convert to 1NF
it is necessary to expand composite attributes into simple ones, and multi-valued ones into
to separate into a separate entity, connected by a 1:M relationship. In the example model
composite attribute—student's name and multi-valued attribute—student's phone number.
The result of reducing the model from the example to 1 NF is shown in Fig. 14.

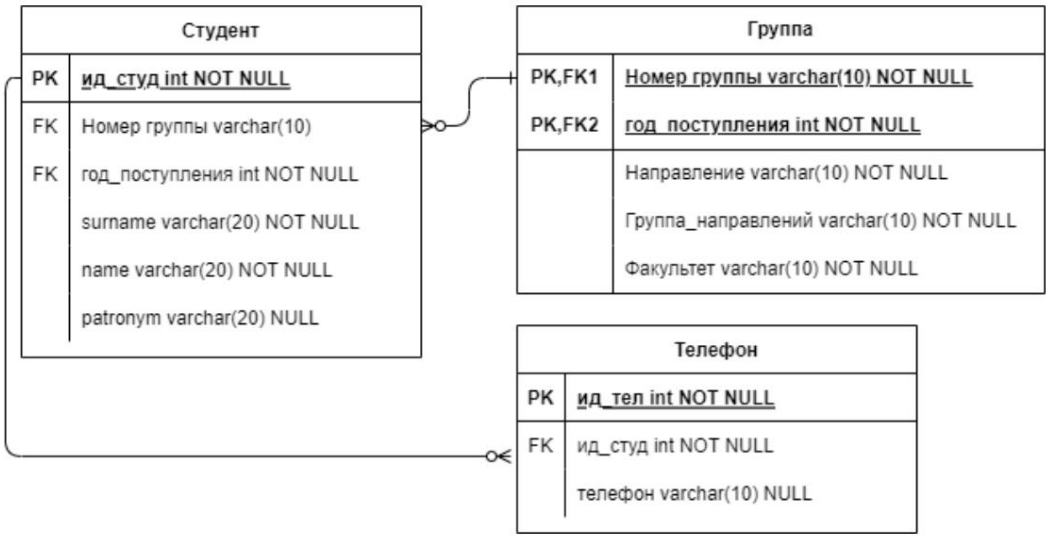

Fig. 14. Logical model of the database in the first normal form

The second and third normal forms require working with the concept
functional dependency.

Functional dependency. Field B of the table is functionally dependent on field
And the same table in that and only in that case, when at any given moment
time for each of the different values of field A there necessarily exists

only one of the different values of field B. Note that it is allowed here, that fields A and B can be composite.

Determinant. The determinant of a functional dependence is called an attribute or group of attributes located on a functional diagram dependence on the left of the arrow. For clarity, in Fig. 15, where the dependency, attribute A is the determinant of dependency.

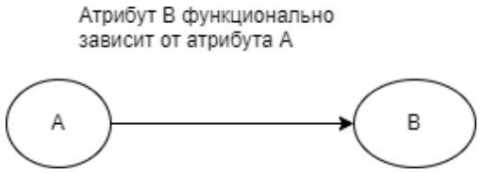

Fig. 15Functional dependency diagram

A functional dependency is called **trivial** if it remains valid under all conditions. The dependence is trivial if and only if the right-hand side of the expression defining the dependency contains a subset (but not necessarily a proper subset) of a set, which is indicated on the left-hand side (determinant) of the expression.

Full functional dependency. Field B is in full functional dependence on the composite field A if it is functional depends on A and does not depend functionally on any subset of the field A.

The table (relation) is in **second normal form** *(2NF),* if it satisfies the definition of 1NF and all its fields that are not in primary key, linked by full functional dependency with the primary key. That is, all fields must depend on the primary key completely, and not from its component. To reduce to 2NF it is necessary to take out breaking dependencies into separate tables.

In the example given, the faculty depends on the group number, but not on year of admission, which violates 2NF. The result of the reduction to 2NF is given in Fig. 16.

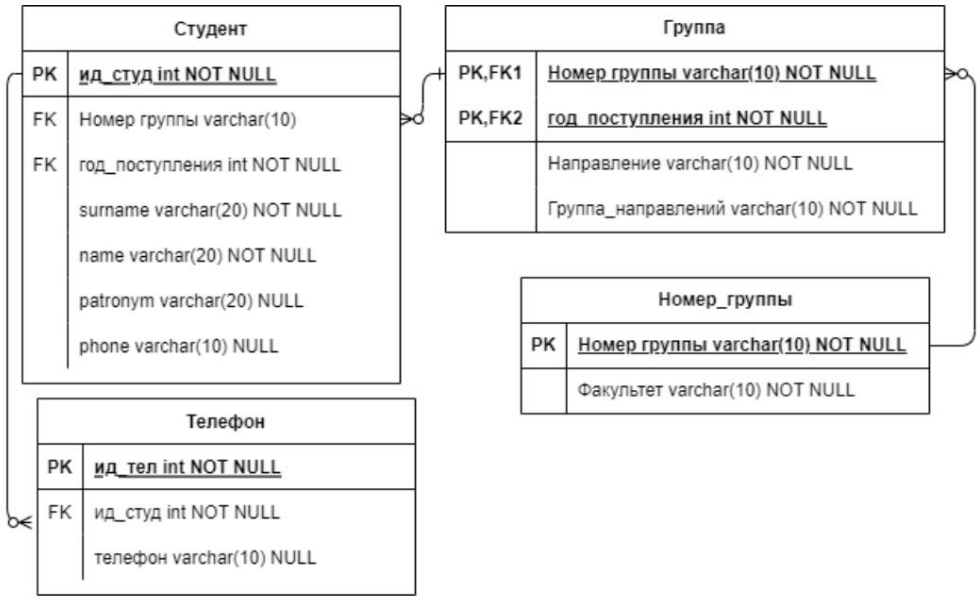

Fig. 16. Logical model of the database in 2NF conversion

A table is in ***third normal form*** *(3NF)* if it satisfies the definition of 2NF and none of its non-key fields depends functional of any other non-key field. To convert to 3NF it is necessary to move the violating dependencies into separate tables. in the given model in 2NF, the direction code uniquely determines the code UG: To reduce it to 3NF, we will put this dependence in the direction table. The result of the reduction to 3NF is shown in Fig. 17.

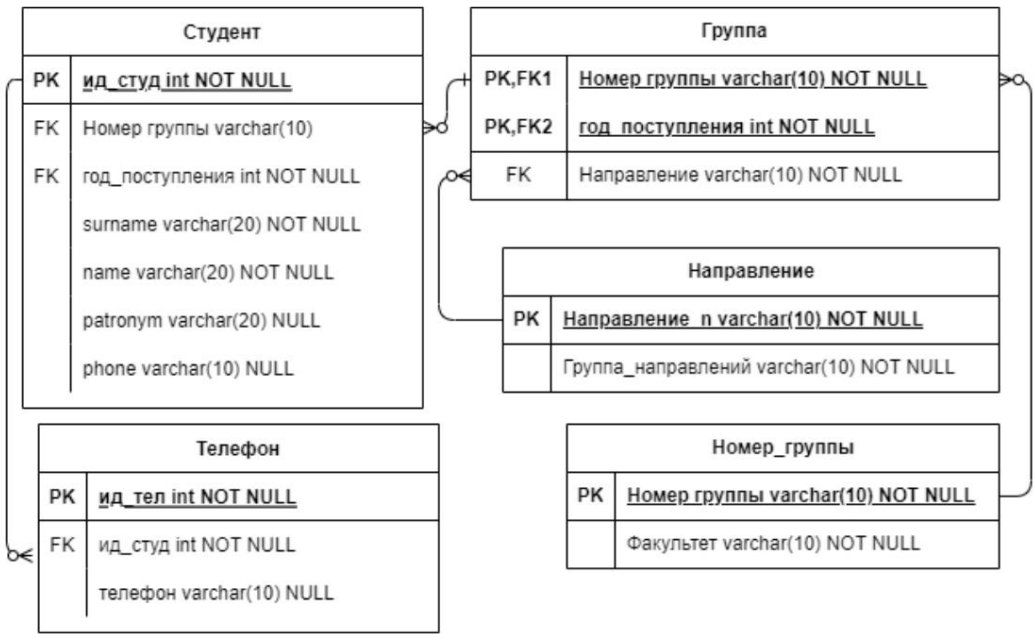

Fig. 17. Logical model of the database in 2NF conversion

There is an important aspect to working with dependencies, which is that used in the database as keys. This is due to the fact that surrogate keys in essence cannot have dependencies and when reduced to normal forms can miss the violation of 2NF, confusing it with 3NF.

Let us consider the remaining normal forms theoretically.

The relation is in **Boyce-Codd Normal Form (BCNF).** then only when each of its determinants is a potential key.

The difference between ZNF and NFBC is that the functional dependency A ÿ B is allowed in relation to ZNF if attribute B is primary key, and attribute A is not necessarily a candidate key. Whereas in relation to the NFBC this dependence is allowed *only* when attribute A is a candidate key.[5]

Multivalued dependency. Represents such a dependency between attributes of a relation (e.g. A, B, and C) such that each value of A represents the set of values for B and the set of values for C.

However, the sets of values for b and c do not depend on each other (A ÿ B, A ÿC).

A multivalued dependency can be further defined as *trivial* or *non-trivial.* For example, the multi-valued dependence A ÿ B some relation R is defined as trivial if the attribute B is a subset of attribute A or A and B = R. Conversely, a multi-valued a dependency is defined as non-trivial if neither condition is true is being carried out.

A relation is in fourth normal form (4NF) if it is in Boyce-Codd normal form and does not contain non-trivial multivalued dependencies.

Lossless join dependency is a decomposition property that ensures that there are no dummy lines when restoring the original relations using the natural join operation.

Fifth normal form (5NF), which is also called *projective*-Project- *Join Normal Form,* or PJNF (Project-Join Normal Form -

PJNF), means that the relation in this form is in 4NF and has no
connection dependencies.

The sixth normal form is considered in 2 variants: for normal data
and for temporal data. The relation is **in the sixth**
normal form (6NF) if and only if the only
the functional dependencies that are executed in it are
trivial [9]. In this case, for ordinary data, the elements are separated
relations, and in the case of temporary ones, the time of relevance of the attributes.

A domain restriction is a restriction that prescribes use for
a certain attribute of values only from a certain specified domain
(set of values). **A key constraint** is a constraint that states that
some attribute or combination of attributes represents
candidate key. The relation R is in **the domain-key**
normal form if and only if every constraint that
is performed in R, implied by domain constraints and keywords
restrictions that apply to

An example of constructing a logical relational model and normalization data

According to the given rules for the conceptual model “Database for
section control”, shown in Figure 8, we obtain a logical
model shown in Fig. 18.

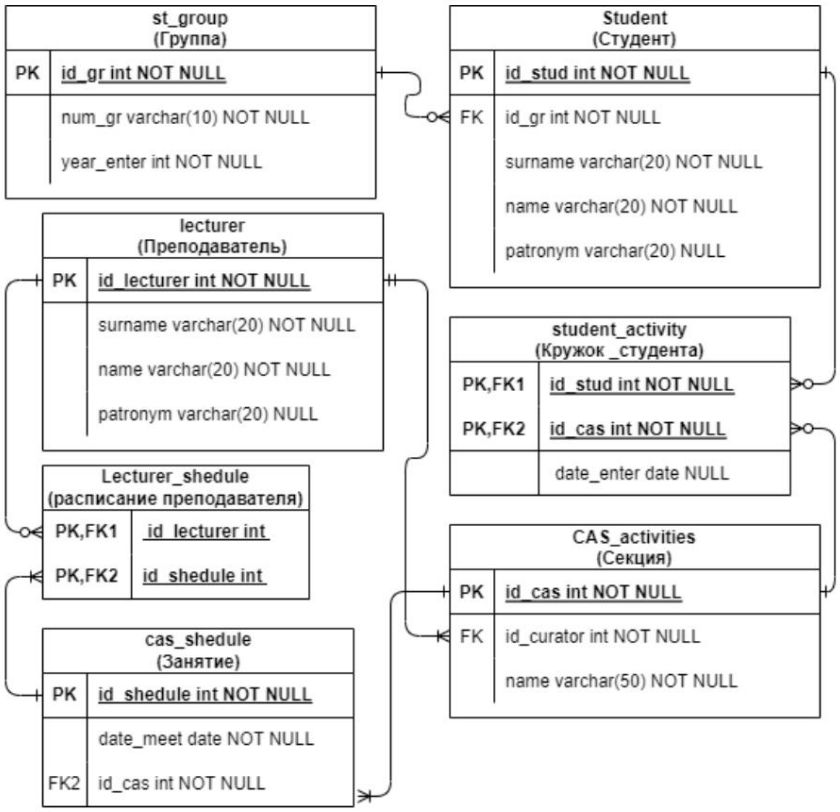

Fig. 18 Logical model "Database for section management"

The only necessary normalization action here is the disclosure of components attributes in names

Test questions and tasks

ÿ In what cases when moving from a conceptual to a logical model are new tables being added?

ÿ Why is it that when developing a database they limit themselves to bringing it to 3NF?

ÿ What are the consequences of using a non-normalized database?

ÿ How can 1NF be violated in a relational database?

ÿ Physical design of databases

The physical model concretizes the logical one, specifying the data types and storage methods for a specific DBMS. Including describes the reference integrity and can describe indexing features.

Referential integrity

An empty value (Null) indicates that the attribute value is currently unknown or unacceptable for this motorcade.

There are 2 types of integrity: entity level and link level.

Integrity at the entity level requires that all elements of the primary keys were unique and no part of the primary key was empty (null). This ensures that each entity (logical object) will have unique identification, and foreign key values can be properly refer to primary key values in a way. For example, a credit number books cannot have multiple duplicate values and cannot have empty value. That is, all students are uniquely identified by their grade book.

Referential integrity requires that a foreign key have either empty value (unless it is part of the primary key of the given table), or a value that matches the primary key value in related table. (Each non-empty foreign key value must reference an existing primary key value.) Execution link-level integrity rules make it impossible to delete a row in one table, where the primary key has a mandatory correspondence with value of a foreign key in another table or changing such a primary key. For example, a student may not be assigned a Group (yet), but It is impossible to assign a non-existent group to a student.

Referential integrity can be maintained in 2 ways: declaratively and actively. With declarative referential integrity, the methods for maintaining Integrities are specified as constraints and are explicitly declared when defining tables. It is the least flexible, but does not require additional code. When active (procedural) referential integrity, all actions on its maintenance must be implemented in triggers - special stored procedures-event handlers. For declarative reference There are several options available to maintain link integrity. integrity, given in Table 3.

Table 3. Actions in Declarative Referential Integrity

Named is the type maintain	Code on SQL	What happens when you delete What happens when you update	
Restriction restrict		Prevents deleting data from a parent table if it is linked to any data in a child table (check immediately)	Prevents changing the primary key from the parent table if any data in the child table is associated with it (check immediately)
No action constraint	Prevents data from being deleted from a parent table if it has associated data in a child table (check deferred)		Prevents changing the primary key from a parent table if any data in the child table is associated with it (check deferred)
Cascaded have	cascade	When you delete data from a parent table, the associated data in the child table will be deleted.	When a primary key in a parent table is changed, the foreign keys associated with it in the child table are changed. will change to the same value
Installation	set null	When deleting data from a parent table, the child table's foreign keys are linked to the deleted data will receive an empty value (null)	When changing the primary key of the parent table, the foreign keys of the child table associated with the changed keys will receive an empty value (null)
Installation	set default	When deleting data from a parent table, the child table's foreign keys are linked to the deleted data will receive a default value that must be set in the child table	When changing the primary key of the parent table, the foreign keys of the child table associated with the changed keys will receive the default value that must be set in the child table.

Features of DBMS and data indexing

One of the parts of designing a physical model is designing indexing. There are many types of indexes, so we define index by the way it is used, rather than focusing on structural properties.

A data structure is called an index if:

it is a redundant data structure, it is invisible to the application, it is designed to speed up the selection of data according to certain criteria. [10].

An index is unique if each indexed value has a corresponding exactly one row in a table. In particular, both PostgreSQL and MySQL automatically create a unique index to support each primary

key or unique constraint in a table. Creating other indexes

associated with the analysis of frequently used slow queries.

Lab #2 Developing a physical database model taking into account declarative referential integrity

Purpose of the work: To acquire skills in constructing logical and physical

data models.

Task and sequence of work execution

1. Create a physical model of the database located in the third

   normal form in accordance with the given option.

2. Describe the referential integrity of the database in the table and add its justification.

The table must be in the form shown for table 4 or have the format,

described below.

Table 4. Example of a description of the referential integrity of a database

Daughter l table	External so key	Parents where table	reference integrity at removal	Description reference integrity at removal	reference integrity when updating	Description reference integrity when updating
Table1	Id_t2	Table2	Cascades sia	When deleting data from Table2, all will be deleted related data from Table1	Cascades sia	When updating primary key of Table2, the foreign key will be updated key from Table1

The description may not be in the table, but must contain the same data.

3. Describe possible unique indexes in DBMS

Contents of the report

ÿ Purpose of the work

ÿ Text of the task (together with the text of the task option);

- conceptual model of the database;

- physical model of the database;

ÿ table with description of referential integrity;

ÿ description of possible unique indices;

ÿ Conclusions on physical design for this subject area.

Test questions and tasks

ÿ What actions are taken to maintain the link
integrity?

ÿ What is the difference between active and declarative referencing?
integrity?

ÿ What is physical database design?

ÿ Why can't I set the declarative referential integrity to empty for a table that exposes a many-to-many relationship?

Introduction to SQL: Data Definition Language

The composition of the SQL language

The SQL language *(Structured Query Language)* includes Data Definition Language (DDL) - Data Definition Language, Data Manipulation Language (DML) - data manipulation language, Transaction Control Language (TCL) is a transaction control language, Data Control Language (DCL) is a language data access management.

Data Definition Language

DDL (Data Definition Language) is a group of *SQL language operators,* used to define the structure of the database and its objects, such as such as tables, views, indexes and procedures. Most Common DDL statements are listed in Table 5:

Table 5 – *DDL* Operators

Create	Creates a new database object, such as a table, view, or index.
Alter	Used to modify an existing database object.

Drop	Used to delete an existing database object.
Truncate	Used to delete all rows in a table, but unlike Drop operator it preserves the table structure and indexes.
Rename	Used to rename an existing database object.

It is important to note that *DDL* statements are executed immediately and are permanent, that is, after creation, modification or deletion The change object cannot be undone.

Creating a table (create statement)

To create a table, use the *CREATE TABLE statement.* Syntax *CREATE TABLE* with basic parameters [11,12]:

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } | UNLOGGED ] TABLE [ IF NOT
EXISTS ] table_name ( { column_name
 data_type [ COMPRESSION compression_method ] sort_rule ] ] [ COLLATE
                                [ column_constraint [ ... ]
    | table_constraint
    | LIKE source_table [ copy_option ... ] [, ... ]

)
 [ INHERITS ( parent_table [, ... ] )
 [ PARTITION BY { RANGE | LIST | HASH } ( { column_name | ( expression ) } [
COLLATE sort_rule ] [ operator_class ] [, ... ] )
 [ USINGmethod ]
 [ WITH ( storage_parameter [= value] [, ... ] ) | WITHOUT OIDS ]
 [ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
 [ TABLESPACE table_space ];
```

After the *CREATE TABLE* statement , you should write the table name and after it open parentheses. In parentheses you need to list through commas fields that the table will contain. For ease of reading, each field must be on a separate line. Names may contain symbols underscores for greater clarity. The maximum length of the name and for tables, and for columns - 64 characters.

Each field needs to have its data type and restrictions specified. Below in Table 6 shows the most frequently used field types:

Table 6 – Most frequently used data types

Type	What does the column contain?	An example
VARCHAR/CHAR/TEXT1	Stores a string	'Masha', 'feather', 'flying ship'
INT	Stores an integer	1; 2; 356

DECIMAL	Stores a number with a fixed value comma	1.5 or 356.12
FLOAT/DOUBLE	Stores a floating point number comma	1.5 or 356.12
DATE	Stores the date (day, month, year) in format YYYY-MM-DD	'1999-03-03', '2017-07-25', '2023-10-27'
TIMESTAMP	Stores date and time (day, month, year, hours, minutes and seconds) in format YYYY-MM-DD HH:MM:SS	'1999-03-03 01:36:12', '2017-07-25 17:16:15', '2023-10-27 11:57:34'

Note 1:

VARCHAR (N) - a string of limited length, it can contain no more than N characters.

CHAR(N) is a fixed-length string that can hold exactly N characters. If the line is shorter, the missing characters will be replaced with spaces.

TEXT by standard refers to the CHARACTER LARGE OBJECT type (up to 2 GB of text), but not stored in the line itself and at the same time limited (and in some DBMS not possible) searching for a substring and creating indexes.

For different DBMS, the names of field types may differ. For example, INTEGER can be used instead of *INT . DATE* type depending on the DBMS can store different information - for example, only the day and month or day, month and time. It is important to check in the documentation for a specific DBMS data types used.

Example: Let's create a table to store data about groups, where *id_gr* is unique number, num_gr — group number, course — course number.

```
CREATE TABLE st_group(
  id_gr int,
  num_gr varchar(8),
coursesmallint);
```

If you try to create the table again using *CREATE TABLE,* and the table already exists in the database, the command will return an error. Therefore, before before creating it is reasonable to check whether the database already contains a table with such name. It is enough to add to *CREATE TABLE IFNOTEXISTS.*

Example:

```
GREATE TABLE IN NOT EXISTS student
```

Attributes *(ATTRIBUTES)* and constraints *(CONSTRAINTS)*

PRIMARY KEY

A special symbol is used to designate the primary key.
PRIMARY KEY() construct . This construct declares a field to be unique.
key. For example, a person's passport number may be quite good
primary key, but a person's name is a bad key, because there are people with
there are a lot of identical names. Similarly, the connection between **first and last name** is also
bad primary key. The field that is declared as a key is required
must be unique and not contain empty values. That is, in terms of
An SQL primary key is a field with the *UNIQUE* and *NOT* NULL properties.[11,12]

Example:

```
CREATE TABLE st_group(
  id_gr int,
  num_gr varchar(8),
  course smallint,
  PRIMARY KEY (id_gr));
```

NOTNULL

To prevent the creation of empty fields, the *NOTNULL operator is used.*
If there is no explicit *NOT NULL* clause and the column is not *a PRIMARY KEY,* then
the column allows storing *NULL,* that is, storing *NULL* is the behavior
default. For a primary key, this constraint can be omitted, so
as a primary key always guarantees *NOT* NULL.[11,22]

Example:

```
CREATE TABLE st_group(
  id_gr int NOT NULL,
  num_gr varchar(8),
  course smallint,
PRIMARY KEY (id_gr));
```

DEFAULT

This command allows you to specify a default value, i.e.
text or number that will be saved unless another value is specified.
Does not apply to all types: *BLOB, TEXT, GEOMETRY and JSON* are not supported.

support this restriction. The default value should be

constant, function or expression is not allowed.

For *the BOOLEAN* data type, built-in constants can be used.

FALSE and *TRUE.* Instead of *DEFAULT(FALSE) you can* specify *DEFAULT(0)* - these records

are equivalent.

Example:

```
CREATE TABLE st_group(
  id_gr int NOT NULL,
  num_gr varchar(8) DEFAULT NULL,
  course smallint DEFAULT 1
PRIMARY KEY (id_gr));
```

AUTO_INCREMENT

Every time a record is added to the table, the value of this

column will be automatically increased. This attribute is not applicable to the entire table,

to only one column, and this column must be the key.

Recommended for use with integer values. Cannot be combined with

DEFAULT. [11]

Example:

```
CREATE TABLE st_group(
  id_gr int NOT NULLAUTO_INCREMENT,
  num_gr varchar(8) DEFAULT NULL,
  course smallint DEFAULT 1
PRIMARY KEY (id_gr));
```

This attribute is applicable in *MySQL,* but is not available in *PostgreSQL.*

PostgreSQL uses the following data types for this: *smallserial*

Serial and *bigserial.* These are numeric data types whose value is formed

by auto-incrementing the value of the previous row.[12]

Example:

```
CREATE TABLE st_group(
  id_gr serial NOT NULL,
  num_gr varchar(8),
  course smallint,
PRIMARY KEY (id_gr));
```

UNIQUE

To create unique field values, use the *UNIQUE command.*

This command is used in conjunction with the *NOT NULL* command because otherwise In this case, only one *NULL value will be allowed.* [11,12]

Example:

```
CREATE TABLE student (
  id_st int NOT NULL,
  surname varchar(20) DEFAULT NULL,
  name varchar(15) DEFAULT NULL,
  patronym varchar(25) DEFAULT NULL,
  id_gr int DEFAULT NULL,
  rate int UNIQUE NOT NULL,
  PRIMARYKEY (id_st),
);
```

CHECK

The *CHECK* command allows you to add a condition that must be met. match string. Allows you to set additional checks data for a column or set of columns. If the value of the column being checked attribute is *NULL,* then the row will not be added to the table, since in The result of the condition will be an undefined result.

Example:

```
Birthday DATE NOT NULL CHECK (birthday >'1900-01-01')
```

Below is an example of using the *CHECK* command and defining date of birth restrictions and acceptable phone formats via regular expression.

```
CREATE TABLE student (
  id_st int NOT NULL,
  surname varchar(20) DEFAULT NULL,
  name varchar(15) DEFAULT NULL,
  patronym varchar(25) DEFAULT NULL,
  id_gr int DEFAULT NULL,
  rate int UNIQUE NOT NULL,
birthday DATE NOT NULL,
  PRIMARY KEY (id_st),
CONSTRAINT student_chk_birthday CHECK (birthday > '1900-01-01'));
```

To add constraints, use the CONSTRAINT statement , In this case, all names are unique, as are table names. Considering that by By default the names include the table name, it is recommended adhere to this rule.

FOREIGN KEY(foreign key)

A foreign key is a reference to a column or group of columns of another tables. This is also a constraint *(CONSTRAINT),* since you can use only values that have a foreign key match. Creates index. A table with a foreign key is called dependent.

Syntax:

```
FOREIGNKEYcolumn_name1,column_name2)
REFERENCES external_table_name(external_table_name1,external_column_name2)
```

First, *the FOREIGN KEY* expression and the set of columns are specified. tables. Then *the REFERENCES* keyword specifies the name of the external tables and the set of columns of this external table. At the end you can add *ON DELETE* and *ON UPDATE* statements , which are used to configure behavior when data in the parent table is deleted or updated. This It is not necessary to do this, as the default behavior is provided. The default behavior is to prevent records from being deleted or modified from the external tables if these records are referenced by foreign keys.

There are 3 options for *ON DELETE* and *ON UPDATE:CASCADE,.SET NULL,RESTRICT:,* described in the physical modeling section.[11,12]

Example:

```
CREATE TABLE student (
  id_st int NOT NULL,
  surname varchar(20) DEFAULT NULL,
  name varchar(15) DEFAULT NULL,
  patronym varchar(25) DEFAULT NULL,
  id_gr int DEFAULT NULL,
  rate int DEFAULT NULL,
  PRIMARY KEY (id_st),
foreign key (id_gr) references st_group(id_gr) on delete cascade on update
restrict);
```

When *CREATE TABLE,* to avoid complicating the column description, It is recommended to specify the foreign key and all its attributes after listing the columns to be created. You can add a foreign key if the table has already been created and contains data. To make changes to the table we use *ALTER TABLE.*

ALTER TABLE

The SQL statement *ALTER TABLE* is used to add, modify, deleting columns in a table or renaming a table.

The syntax for *the ALTER TABLE* statement for *PostgreSQL* is as follows:

image: [12]

```
ALTER TABLE table_name [WITHCHECK|WITH NOCHECK]
{ ADD column_name column_data_type [column_attributes] |
DROP COLUMN column_name |
ALTER COLUMN column_name column_data_type [NULL|NOTNULL] |
ADD [CONSTRAINT] definition_constraints |
DROP [CONSTRAINT] constraint_name}
```

The syntax for the *ALTER TABLE* statement for MySQL is as follows: [11]

```
ALTER TABLE table_name
{ ADD column_name column_data_type [column_attributes] |
    DROP COLUMN column_name |
    MODIFY COLUMN column_name column_data_type [column_attributes] |
    ALTER COLUMN column_name SETDEFAULTdefault_value |
    ADD[CONSTRAINT] definition_constraints |
    DROP[CONSTRAINT] constraint_name}
```

Adding column(s) to table

SQL syntax of *ALTER TABLE* statement to add a column to a table.

```
ALTER TABLE table_name ADD
column_name column_data_type [column_attributes]; Examples:
```

Adding a single column to a table. *ALTER TABLE* will add a column named name to *the student table.*

```
ALTER TABLE student
ADD name varchar(15);
```

Adding multiple columns is done in a similar manner.

```
ALTER TABLE student
ADD(name varchar(15),
        City varchar(45));
```

Alter column(s) in a table *ALTER*

TABLE syntax to alter a column in an existing table

The table for *MySQL and* PostgreSQL differ.

Examples:

Change one column.

For MySQL.

```
ALTER TABLE student
  MODIFY name varchar(15) NOT NULL
```

ÿÿÿPostgreSQL.

```
ALTER TABLE student
ALTER COLUMN name varchar(15),
ALTER COLUMN name SET NOT NULL;
```

Changing multiple columns is similar to changing

one column.

For *MySQL.*

```
ALTER TABLE student
MODIFY name varchar(15) NOT NULL,
MODIFY city varchar(55);
```

ÿÿÿ *PostgreSQL.*

```
ALTER TABLE student
ALTER COLUMN name varchar(15),
ALTER COLUMN name SET NOT NULL,
ALTER COLUMN city varchar(55);
```

Deleting column(s) in a table

ALTER TABLE syntax to drop a column in an existing table.

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

Example:

```
ALTER TABLE student DROP COLUMN name;
```

Rename a column in a table

ALTER TABLE Syntax to Rename a Column in an Existing Column

The table for *PostgreSQL and MySQL* differs.

ÿÿÿ *PostgreSQL.*

```
ALTER TABLE table_name
RENAME COLUMN old_column_name TO new_column_name For MySQL.
```

```
ALTER TABLE table_name
CHANGECOLUMNold_column_name TOnew_column_name;
```

An example

ÿÿÿPostgreSQL.

```
ALTER TABLE student
RENAME COLUMN name TO s_name;
```

ÿÿÿMySQL.

```
ALTER TABLE student
CHANGE COLUMN name TO s_name;
```

Rename table

ALTER TABLE syntax to rename a table.

```
ALTER TABLE table_name
RENAME TO new_table_name
```

Example:

```
ALTER TABLE student
RENAME TO s_student;
```

DROP TABLE

The *DROP TABLE SQL* statement allows you to remove a table from a database.

Syntax for *DROP TABLE* statement in SQL.[11,12]

```
DROP TABLE table_name;
```

Example:

```
DROP TABLE student;
```

TRUNCATE

This operator is used to quickly clear the table - it deletes everything lines from it.

TRUNCATE TABLE – deletes all rows in a table without writing to log delete individual rows. The *TRUNCATE TABLE* statement is similar to *DELETE* statement without *WHERE clause,* but *TRUNCATE TABLE* runs faster and requires fewer system resources and logs transactions.

TRUNCATE TABLE cannot be used if the table is referenced *FOREIGN KEY* constraint A table that has a foreign key that references on itself, can be truncated. [11,12]

Example:

```
TRUNCATE TABLE student
```

TRUNCATE TABLE has the following advantages: compared to the *DELETE operator:*

- A smaller transaction log is used.

Removes data, freeing up data pages used for storing table data, and only records data about it in the transaction log. freeing up pages.

- Typically fewer locks are used.

Always locks the table (including schema lock *(SCH-M))* and page, but not every line.

- The table contains zero pages, no exceptions.

The *TRUNCATE TABLE* statement deletes all rows of a table, but the structure tables and their columns, constraints, indexes, etc. are preserved. To to delete not only the table data but also its definition, you should use *DROP TABLE* operator .

If the table contains an identity column, the count of that column resets to the initial value defined for this column. If The initial value is not specified, the default value of 1 is used. To preserve the identity column, use the *DELETE statement.*

TRUNCATE *TABLE* can be rolled back.

You cannot use *TRUNCATE TABLE* on tables in the following cases:

• The table is referenced by a *FOREIGN KEY constraint.* The table, having a foreign key that references itself can be truncated.

• The table is part of an indexed view.

• The table is published using transactional replication or merge replication.

• This is a temporal table with version control.

• The table is referenced by an *EDGE constraint.*

For tables with any of these characteristics, use DELETE instruction .

Rename

RENAME TABLE renames one or more tables.

Syntax:

```
RENAME TABLE old_table_name TO new_table_name;
```

This statement is equivalent to the following *ALTER TABLE* statement:

```
ALTER TABLE old_table_name RENAME new_table_name;
```

RENAME TABLE unlike *ALTER TABLE,* you can rename multiple tables in one query:

```
RENAME TABLE old_table_name TO new_table_name,
old_table_name2 TO new_table_name2,
old_table_name3 TO new_table_name3;
```

Lab #3 Creating and modifying a database and tables

databases

Purpose of the work: To acquire the skills and abilities to create and modify tables on SQL language.

Task and sequence of work execution

In accordance with the model developed in the previous work, create a database data. Demonstrate the ability to add and remove a column using the command alter table.

Contents of the report

- ÿ Purpose of the work
- ÿ Text of the task (together with the text of the task option);
- - physical model of the database;
- ÿ table with description of referential integrity;
- ÿ Conclusions on the features of creating tables of the developed data model in selected DBMS.

Test questions and tasks

ÿ What is DDL and what operators are related to DDL?

ÿ What can be changed using the alter table command?

ÿ What is the difference between a fixed length string and a limited length string lengths?

ÿ What restrictions can be set using the create table command?

ÿ How to set constraints using the create table command?

ÿ Is it possible to create a foreign key that references something other than the primary key?

ÿ Why do you need a constraint name when setting constraints, foreign keys in particulars?

Introduction to SQL: A Data Manipulation Language

Basic data manipulation commands and examples

applications

DML (Data Manipulation Language) are *SQL* language operators (the language structured queries) that are used to manipulate data in the database. *DML* statements are used to insert, updating and deleting data in the database. Some of the most Common DML operators are presented in Table 7:

Table 7 – *DML* Operators

Select	Used to retrieve data from one or more database tables. data.
Insert	Used to insert new data into a table.
Update	Used to modify existing data in a table.
Delete	Used to delete data from a table.
Merge	Used to perform conditional *INSERT, UPDATE* or *DELETE* rows in a table.

DML statements are executed immediately and can be canceled with using the rollback operator. It is important to note that although *DML* operators used to create, modify and delete database objects, *DML* Operators are used to manipulate the data within these objects.

INSERT statement syntax

The *INSERT* statement is used to insert new data into a table.

This operator has 2 main forms:[1,2]

```
INSERT INTO table(list_of_fields) VALUES(list_of_values)
```

Inserting a new row into a table, the field values of which are formed from listed values.

```
INSERT INTO table(list_of_fields) SELECT list_of_values
FROM table_name
```

Inserting new rows into a table, the values of which are formed from row values returned by the query.

A few notes about *INSERT:*

ÿ The order in which the fields are listed does not matter, the only important thing is that

it matched the order of the values you list in brackets

after the VALUES keyword .

ÿ It is also important that values are specified for all when inserting

required fields that are marked in the table as *NOT NULL.*

ÿ It is possible not to specify the fields for which the *IDENTITY* option was specified or

the same fields that had a default value set using

DEFAULT, because either the value from

counter, or the default value.

ÿ In cases where the value of the field with the counter must be set explicitly

The IDENTITY_INSERT option is used .

UPDATE statement syntax

The SQL UPDATE statement is used to update existing records in tables.

Syntax for *UPDATE* statement when updating a table in *SQL.* Fields in [] are optional. [11,12]

```
UPDATE table
SET column_name1 = value,
column_name2 = value2,
 ...
 [WHERE condition];
```

The syntax of the *UPDATE* statement when updating a table with data from another table:

```
UPDATE table_name1
SET column_name1 = (SELECT value1
FROM table_name2
WHERE condition)
 [WHERE condition];
```

The syntax for the *UPDATE* statement when updating multiple tables is:

```
UPDATE table_name1, table_name2,
SET column_name1 = value1,
column_name2 = value2,
 ...
WHERE table_name1.column_name = table_name2.column_name
 [AND condition];
```

DELETE statement syntax

The SQL DELETE statement is used to delete one or more records from the table. Syntax of the DELETE operator:[11,12]

```
DELETEFROM table_name
```

```
[WHERE condition];
```

If you run a *DELETE* statement without conditions in the *WHERE clause,* all records from the table will be deleted.

Example:

```
DELETE FROM student WHERE name= 'Aleksandr';
```

You can specify multiple conditions in the *DELETE* statement for this using either the *AND* condition or the OR condition . The *AND* condition allows delete a record if all conditions are met. The *OR* condition deletes a record if one of the conditions is met.

```
DELETE FROM student WHERE id>50 AND name='ÿÿÿÿÿÿÿÿÿ';
```

You can delete records in one table based on values in another. table. Since you cannot list more than one table in a sentence When performing a delete, you can use the EXISTS clause *in the FROM* clause.

```
DELETE FROM student
WHERE EXISTS (SELECT * FROM st_group
    WHERE student.id_gr= st_group.id_gr AND student.name= 'ÿÿÿÿÿÿÿÿÿ');
```

Merging data (MERGE operator)

MERGE is an operator that can be used to perform conditional *INSERT, UPDATE ,* or *DELETE* operations on rows in a table.

The syntax of *the MERGE* statement for *PostgreSQL* is as follows image: [12]

```
[ WITHÿÿÿÿÿÿ_WITH [, ...] ]
MERGEINTO [ ONLY ] target_table_name [ * ]                    [ AS ] target_alias ]
USING data_source ON join_condition
proposal_when [...]
          t-
here is the data_source:
{ [ ONLY ] source_table_name [ * ] source_alias ]          ( original_query )          [ AS ]

and proposal_when:
{ WHEN MATCHED [ AND condition ] THEN { modify_on_merge | delete_on_merge | DO NOTHING }

WHEN NOT MATCHED [ AND condition ] THEN { add_on_join | DO NOTHING } and add_on_join:


INSERT [( column_name [, ...] )]
 [ OVERRIDING { SYSTEM | USER } VALUE ]
{ VALUES ( { expression | DEFAULT } [, ...] ) and change_on_merge:          | DEFAULT VALUES }

UPDATE SET { column_name = { expression | DEFAULT } ( column_name [, ...] ) =
                    ( { expression | DEFAULT } [, ...] )                                    } [,
 ...]
and delete_on_merge:
DELETE
```

How it works: insert/update or delete rows in a table *target_table_name* from *data_source* by condition.

The *MERGE* construct is somewhat similar to the *CASE conditional statement,* it also contains *WHEN* blocks , when the conditions of which are met, one or another occurs other action. Data modification is performed in the receiver table.

MERGE can have no more than two *WHEN MATCHED clauses.* If two sentences are given, the first sentence must be accompanied by an additional condition. For any row, the second The *WHEN MATCHED* clause only applies if not the first one applies.

If there are two *WHEN MATCHED clauses,* one must specify an *UPDATE action,* and the other specifies *a DELETE action.* The *WHEN NOT* clause *MATCHED* is used to insert rows from the source that do not match rows in the table being modified according to the relationship condition. The MERGE statement can have only one WHEN NOT MATCHED clause.

Example: Add records to the *customers* table with missing ones email addresses, update name if client creation date is less than, than the corresponding date in *leads,* and delete other entries:

```
MERGE INTO customers AS c
USING leads AS l
ON c.email = l.email
WHEN NOT MATCHED THEN
    INSERT (name, email, created_at)
    VALUES (l.email, l.name, DEFAULT)
WHEN MATCHED AND c.created_at < l.created_at THEN UPDATE
        SET name = l.name
WHEN MATCHED THEN
DELETE;
```

MySQL does not have *a MERGE* operator . There is a *MERGE* table (or table *MRG_MyISAM)* which is a collection of identical *MyISAM* tables that can be used as a single table.[11]

Lab #4 Filling in tables and modifying data

Objective of the work: Obtaining skills and abilities in manipulating data in a relational database

Task and sequence of work execution

1)Perform insertion of test data into the tables created during

completion of laboratory work 2.

The rows inserted into the tables must contain data that meets the requirements,

and do not satisfy the conditions of the requests given in the option

tasks. (To demonstrate this, it is necessary to create a table in the report, Where

the task for the request will be indicated, the data satisfying the conditions of the request,

data does not meet the request conditions)

2) It is necessary to provide your own examples of using the update and operators

delete and merge with a description of their purpose.

Contents of the report

ÿ Purpose of the work

ÿ text of the task (together with the text of the task variant);

- physical model of the database;

ÿ data sets contained in database tables;

ÿ test data table similar to table 8:

Table 8. Test data

Request text	data satisfying the conditions requests	data not satisfactory query conditions
a. stations in title which there are word "square"	Table 1 (station) Lenin Square	Table 1 (station) Moscow Gate

ÿ examples of using insert, update, delete and merge;

ÿ script for complete filling of the database;

ÿ Conclusions about the features of data manipulation in the selected DBMS.

Test questions and tasks

ÿ What methods of writing the data insertion operator do you know?

ÿ What is DML and what statements are related to DML?

ÿ What does the merge operator do?

ÿ When are quotation marks placed around constants when inserting data?

ÿ Provide an example of the syntax of the data update command.

ÿ Provide an example of the syntax of the data deletion command.

SQL Language. Selection Operator

SELECT statement syntax

The purpose of the SELECT statement is to retrieve and display data.

one or more database tables. The SELECT statement is most often

used by the SQL language command. The general format of the SELECT statement is

next view:

SELECT [DISTINCT | ALL] .{ *| [columnExpression [AS newName]] [FROM TableName [alias][[WHERE , ...]}
uslovievybora strok] ,..]

[GROUP BY grouping condition]
[HAVING condition for selecting groups]
[ORDERBY sorting condition]

SELECT section . Specifies which columns should be present in the

output data. The distinct keyword allows you to remove duplicate lines from

query result.

Here the *columnExpression* parameter can only include the following types:

elements:

• column names;

• **aggregation functions;**

• constants;

• expressions that include combinations of the elements listed above.

In contrast to the column list, * means to return all columns of all

query tables.

The *TableName* parameter is the name of an existing table in the database.

(or view) to which you want to access. Optional

The alias parameter is an abbreviation, a nickname, set for the name

TableName tables . The elements of the SELECT statement are processed in

the following sequence.

• *FROM.* The names of the table or tables to be used are defined,
comma separated list. Also as a data source in this
the section may contain tables linked by different types of connections,
views and other queries.

• *WHERE.* The object rows are filtered according to the specified
conditions.

• *GROUPBY.* Creates groups of rows that have the same value in
the specified column.

• *HAVING.* Filters groups of object rows according to the specified
a condition relating to the result of an aggregate function.

• *ORDERBY.* Determines the order of the execution results.
operator.

In this case, each field in the sorting list can be assigned *an ASC* (by
ascending - this is the default field sorting option) or *DESC* (by
(descending)

The order of constructs in a *SELECT statement cannot* be changed. Only
two operator constructs - *SELECT and FROM* - are mandatory,
all other constructions can be omitted.

Conditions in *WHERE* constructs

(applies to both *SELECT* and *UPDATE* or *INSERT statements)*

In the examples above, the results of executing the *SELECT* statements are
all rows of the specified table were selected. However, very often it is required that
or otherwise limit the set of rows placed in the result
query table. This is achieved by specifying the construction in the query
WHERE. It consists of the keyword *WHERE* followed by a list
search conditions that define the lines that should be selected when
query execution. There are five main types of search conditions

• Comparison. The results of calculating one expression are compared with
the results of calculating another expression. (<,>,=,<>)

• Range: Checks whether the result of evaluating the expression falls within
the specified range of values. (field_name *BETWEEN* value_1 *AND* value_2)
• Set membership. Checks whether the result belongs to a set.
evaluation of an expression to a given set of values. (field_name in
(value_1, value_2..., value_n))
• *NULL value.* Checks whether the given column contains
NULL(undefined value) (field_name *ISNULL).*
• Pattern matching: Checks whether a certain string matches
value to the specified template. (field_name *LIKE* pattern)
• %. The percent symbol represents any sequence of zero or
more characters (hence often referred to as *a wildcard).*
• _. The underscore character represents any single character. .
All other characters in the pattern represent themselves.

An example

st_group.num_grLIKE '4%'. This pattern means that the first character of the value
must be the 4 symbol, and all other symbols are not
are of interest and are not verified.
st_group.num_gr LIKE '4_ _ _ _'.This pattern means that the value should
have a length equal to exactly four characters, with the first character being
there must be a symbol 4 . (spaces between underscores
only for visualization - they are not in the template)
patronymic LIKE ' %ich'. This template defines any
a sequence of characters at least two characters long, and
the last symbols must be the symbols ich. (search for men
by patronymic)
• name LIKE '%word%'. This pattern means that we are interested in
any sequence of characters that includes the substring "word";
If the required string must also include a special character, usually
used as a wildcard character, then it should be defined with
using the ESCAPE construct, a "masking" symbol that indicates

that the symbol following it no longer has a special meaning, and

place it before the wildcard. For example, to check the values

to match the literal string 45%' you can use this

by predicate: *LIKE '15#%' ESCAPE '#'*

Example: Let's select all the students in group 4432K according to the diagram shown in

Figure 18.

```
select id_student, student.id_gr, surname, name, patronym from student, st_group where student.id_gr
=st_group.id_gr and number_gr='4432ÿ'
```

Types of connections in SQL language

Types of connections in SQL:

- ÿ Cartesian product *CROSS JOIN*
- ÿ Inner join *INNERJOIN* (often just JOIN)
- ÿ External connections:
- ÿ *LEFT JOIN*
- ÿ Right join *RIGHTJOIN*
- ÿ Full (outer) joinOUTER *JOIN*

A conjunction is a subset of a more general combination of two given entities.

tables called *the Cartesian product.* The Cartesian product of two

table is another table consisting of all possible pairs

rows that are part of both tables. The set of columns of the resulting

table represents all the columns of the first table, followed by

all columns of the second table follow. If you enter a query to two tables without

WHERE construction assignments, query execution result in SQL environment

will be the Cartesian product of these tables.

An inner join is used to relate tables by matching

column values. Outer joins are needed when the values in

one table will not always have corresponding values in another (i.e.

foreign key is empty (NULL))

Properties of compounds:

The inner join is symmetrical, i.e. no change in the order of tables occurs

nothing will change when connecting

SELECT *tableName1.*, tableName2.*,* FROM *tableName1INNER* JOIN *tableName2* Equivalent to

SELECT *tableName1.*, tableName2.*,* FROM *tableName2* INNER JOIN *tableName1*

Full external join is also symmetrical. Left and right joins

symmetrical to each other, i.e.

SELECT *tableName1.*, tableName2.*,* FROM *tableName1LEFT* JOIN *tableName2*

Equivalent

SELECT *tableName1.*, tableName2.*,* FROM *tableName2RIGHT* JOIN *tableName1*

For examples of different connections we use the tables group and student,

shown in Fig. 19

student

id_student	id_gr	surname	name	patronym
1	1	Петров	Петр	Петрович
2	2	Иванов	Иван	Иванович
3	*NULL*	Пупкин	Василий	Федорович

st_group

id_gr	number_gr
1	Z4431K
2	Z5432K
3	B5433

Fig.19. Data for examples

Select number_gr, id_student, surname,name, patronym from st_group inner join

student on student.id_gr=st_group.id_gr

The result of such a request will be as shown in Fig. 20.

st_group INNER JOIN student

number_gr	id_student	surname	name	patronym
Z4431K	1	Петров	Петр	Петрович
Z5432K	2	Иванов	Иван	Иванович

Fig.20. Result of inner join

Select number_gr, id_student, surname,name, patronym from st_group left join

student on student.id_gr=st_group.id_gr

The result of such a query is shown in Fig. 21.

st_group LEFT JOIN student

number_gr	id_student	surname	name	patronym
Z4431K	1	Петров	Петр	Петрович
Z5432K	2	Иванов	Иван	Иванович
B5433	*NULL*	*NULL*	*NULL*	*NULL*

Fig.21. Result of left connection

Select number_gr, id_student, surname,name, patronym from st_group right join

student on student.id_gr=st_group.id_gr

The result of such a query with a right join will be shown in Fig. 22.

st_group RIGHT JOIN student

number_gr	id_student	surname	name	patronym
Z4431K	1	Петров	Петр	Петрович
Z5432K	2	Иванов	Иван	Иванович
NULL	3	Пупкин	Василий	Федорович

Fig.22. Result of right connection

Select number_gr, id_student, surname,name, patronym from st_group outer join

student on student.id_gr=st_group.id_gr

The result of a query with a full outer join is shown in Figure 23.

st_group OUTER JOIN student

number_gr	id_student	surname	name	patronym
Z4431K	1	Петров	Петр	Петрович
Z5432K	2	Иванов	Иван	Иванович
NULL	3	Пупкин	Василий	Федорович
B5433	*NULL*	*NULL*	*NULL*	*NULL*

Fig.23. Result of a full outer join

The left join can be used to find groups without students according to the diagram in Fig. 18

```
Select number_gr, id_student, surname,name, patronym FROM st_group LEFT JOIN student ON
student.id_gr=st_group.id_gr

WHERE student.id_student is NULL;

or

Select number_gr, id_student, surname,name, patronym FROM student RIGHT JOIN st_group ON
student.id_gr=st_group.id_gr

WHERE student.id_student is NULL;
```

Aggregate functions in SQL select statement

The ISO standard defines the following five *aggregation functions:*

• *COUNT* — returns the number of values in the specified column;

• *SUM* — returns the sum of the values in the specified column;

• *AVG* - returns the arithmetic mean of the values in the specified column;

• *MIN* - returns the minimum value in the specified column;

• *MAX* - returns the maximum value in the specified column.

All of these functions operate on values in a single column of a table and return a single value. The functions *COUNT, MIN and MAX* are applicable

to both numeric and non-numeric fields, while the *SUM and AVG* functions can only be used for numeric fields. Except *COUNT (*) ,* when calculating the results of any functions first all empty values are excluded, after which the required operation is applied only to the remaining non-empty values of the column. *COUNT(*)* option is a special case of using the *COUNT* function - its purpose is consists of counting all the rows in a table, regardless of whether they contain empty, duplicate or any other values.

If it is necessary to exclude before applying the aggregation function duplicate values, comes before the column name in the definition functions place the keyword *DISTINCT.* If the selection list *is SELECT* contains an aggregation function, and the query text does not contain the construct GROUPBY, which provides the combination of data into groups, then none of SELECT list items cannot include any references to columns, except when that column is used as a parameter aggregation function.

Grouping results (GROUPBY construct)

All column names listed in the SELECT *list* must be present in the *GROUPBY construct,* except when the name column is used only in the aggregate function. If together with *The GROUPBY* construct uses the *WHERE construct,* then it are processed first, and only those are subject to grouping lines that satisfy the search condition.

HAVING construction

The *HAVING* design is intended for use in conjunction with *GROUPBY* construct . Basically, *HAVING* works the same as the expression *WHERE clause* in a *SELECT statement.* However, the *WHERE* clause applies to columns and expressions for a single row, while the operator *HAVING* is applied to the result of the *GROUPBY* command (aggregate functions).

Let's give an example of using aggregate functions and grouping.

Let's assume there is a table Student_Uni, shown in Fig. 24.

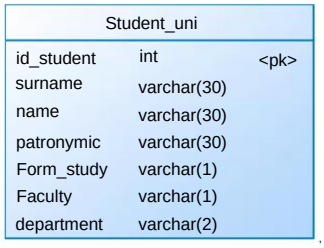

Fig.24. Attitude of university students

Where From_study is the form of study (“O”, “B”, “Z”), Faculty is the faculty, department.

If we want to count all students of a university, we need to write a query

```
SELECT count (id_student) as all_st from Student_Uni
```

then as a result we will get one number equal to the total amount students.

If we want to know how many students are studying in each form of education need to write

```
SELECT count (id_student) as all_st, Form_study from Student_Uni GROUP BY Form_study
```

In fact, by grouping by the form of training, we made the forms equal learning criteria for classifying a student into a particular group (cell), in which will then be counted. That is, they broke up the entire set students by form of study

```
SELECT count (id_student) as all_st, Faculty from Student_Uni GROUP BY Faculty
```

And if we group by faculty and by type of education

```
SELECT count (id_student) as all_st, Faculty, Form_study from Student_Uni GROUP BY Faculty, Form_study
```

Then we get a cell (group) in which both of these values are the same. Each time we add a field to the grouping, we reduce the cell, therefore, for example, a query like

```
SELECT count (id_student) as all_stfrom Student_Uni GROUP BYid_student
```

It will also give out many lines with units (the size of the cell for counting students is one student) **Therefore, pay attention if you have grouping and aggregate the function is set for one field, then this is probably an error**

Examples of implementing various queries

Queries using aliases

The need for such queries arises when the same table must be used in different capacities (the department leading the discipline of the group and department, from which the group graduates), or when it is necessary for some the attribute took on 2 values at the same time (the group that studies and Differential equations and Object-oriented programming)

The general form of an alias for a table is select … from table as **alias.** AS can sometimes be omitted.

An example of the subject area of the curriculum shown in Fig. 25.

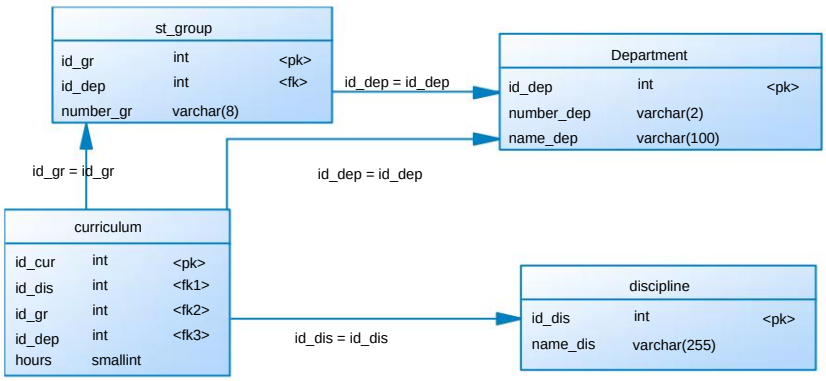

Fig. 25. Scheme of the curriculum database

Write a request: Groups that teach disciplines in departments 43 and 41.

For clarity, we will display the diagram of the request itself in Figure 26.

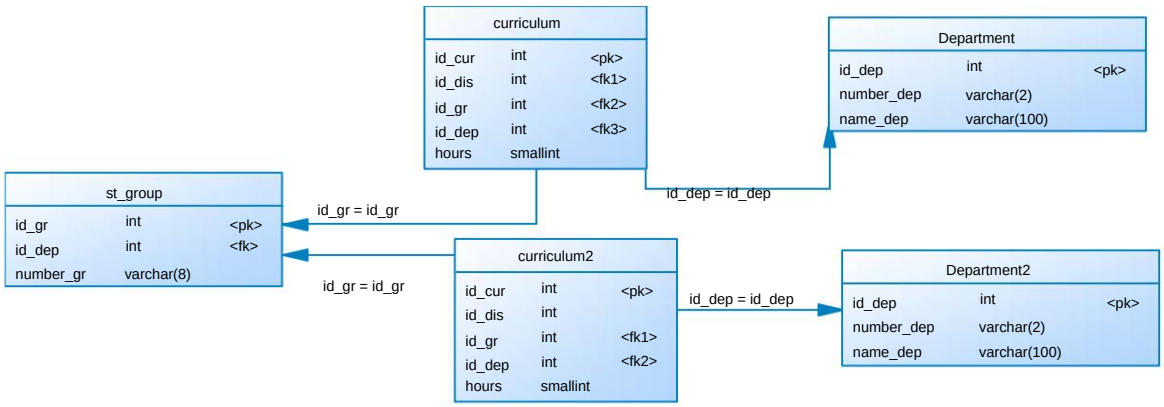

Fig.26. Query scheme with aliases

Select distinct st_group.id_gr, number_gr from st_group inner join curriculum
on curriculum.id_gr= st_group.id_gr inner join Department on
Department.id_dep=curriculum.id_dep inner join curriculum **as curriculum2** on
curriculum2.id_gr= st_group.id_gr inner join Department as **Department2** on
Department2.id_dep=curriculum2.id_dep where Department.number_dep='43' and Department2.number_dep='41'
ÿÿÿ ÿÿÿÿÿÿÿÿÿÿÿÿÿÿ ÿÿ 2 ÿÿÿÿÿÿÿÿÿÿ ÿÿÿÿÿÿ, ÿ 2 ÿÿÿÿÿÿÿÿÿ ÿÿÿÿÿÿÿÿ ÿÿÿÿÿÿ

will look like

where Department.number_dep< Department2.number_dep

Lab #5 Developing SQL queries: types of connections and templates

Purpose of the work: To acquire skills in creating queries without subqueries,

using different types of connections.

Task and sequence of work execution

1) Implement requests a) .. c) specified in the task option.

- Request for pattern search (substring search) (execute with a single

operator like)

- Query to use one table multiple times (aliases).

- Request to use external connections.

All queries must NOT contain nested queries or aggregates.

functions (use aliases and different types of connections instead).

Contents of the report

ÿ Purpose of the work

ÿ text of the task (together with the text of the task variant);

- physical model of the database;

ÿ elements of the test data table from the previous work, concerning

queries a-b:

ÿ SQL query text;

ÿ data sets returned by queries (screenshots);

ÿ conclusions about the features of implementing simple selection queries.

Test questions and tasks

ÿ What is the difference between a left connection and a right connection?

ÿ What is the difference between a left-hand connection and an internal connection? When each of them

is it applied?

ÿ What wildcards are used in pattern matching?

ÿ What is it? _ (underline) when searching by template?

ÿ What is an escape character in pattern search?

ÿ What are aliases and why are they used in queries?

SQL Language. Queries with Subqueries

Principles of constructing queries with subqueries

Here we will discuss the use of complete *SELECT statements,*
embedded in the body of another *SELECT statement. The outer* (second) statement
SELECT uses the result of *the inner* (first) operator
to determine the content of the final result of the entire operation.
Inner queries can be in *WHERE* and *HAVING* clauses.
external *SELECT* statement - in this case they are called
subqueries, or *nested queries.* In addition, internal operators
SELECT can be used in *INSERT, UPDATE ,* and *DELETE* statements.
It is also possible to find a subquery in the FROM clause. , where he should
get an alias after the brackets and you can work with it like any other
table.

There are three types of subqueries.

• *A scalar subquery* returns a value selected from the intersection
one column with one row, i.e. a single value. In principle

scalar subquery can be used anywhere where you need to specify

the only meaning.

• *A string subquery* returns values from multiple columns of a table, but

as a single row. A row subquery can be used

wherever the string value constructor is used, this is usually

predicate.

• *A table subquery* returns the values of one or more columns.

tables that span more than one row. A table subquery can

be used wherever a table can be specified, such as

operand of the predicate IN.

Example of a scalar subquery

Display a group that has the same number of students as group Z4431

```
Select st_group.* from st_group left join student on student.id_gr=st_group.id_gr

Group by student.id_gr Having
count (id_student)=
 (select count(*) from st_group gr
      left join student st on st.id_gr=gr.id_gr where number_gr=' Z4431')
```

Left join, because there may not be students in the group

Table subquery

Remove all students from the groups where Ivanov studies

```
Select student.* from student where id_gr in

 (select st_group.id_gr from from st_group inner join student on student.id_gr=st_group.id_gr
where student.surname='ÿÿÿÿÿÿ')
```

The following rules and restrictions apply to subqueries.

1. Subqueries should not use the *ORDERBY clause,* although it

may be present in the outer *SELECT statement.*

2. The SELECT list *of* a subquery must consist of the names of individual

columns or expressions composed of them, except when in

The subquery uses the EXISTS keyword .

3. By default, the column names in a subquery refer to the table name

which is specified in the *FROM* clause of the subquery. However, it is permitted

refer to columns of the table specified in the *FROM* clause of the outer

query, for which qualified column names are used (as described
below).

4. If the subquery is one of the two operands involved in the operation
comparison, then the subquery must be specified on the right side of this operation.
For example, the following query entry is incorrect,
because the subquery is placed on the left side of the comparison operation with a value
salary column .

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE (SELECT AVG(salary) FROM Staff) < salary;
```

Union, intersection, difference of queries in SQL language

• *The union* of two tables A and B is a table containing all the rows,
which are in the first table (A), in the second table (B), or in both of these
tables at the same time,

• *The intersection* of two tables is the table that contains all the rows,
present in both source tables simultaneously.

• *The difference between* two tables A and B is the table that contains all the rows,
which are present in table A but absent from table B.

Queries must have the same number of columns, and
the corresponding columns must contain data of the same type
and lengths. The three set operations provided by the ISO standard are
are called *UNION, INTERSECT* and *EXCEPT* and allow you to combine
the results of executing two or more queries into a single result
table. In each case, the format of the construction with the operation on sets
should be as follows:

Request1 operator .[ALL] Request2

Some SQL dialects do not support *INTERSECT* and *EXCEPT* operations.
(MySQL), and in others, instead of the EXCEPT keyword, the keyword is used
the word *MINUS.* For example, let's choose the names and surnames of teachers or
students with a non-empty patronymic. (logical or)

```
(SELECT surname, name FROM student
WHERE patronym IS NOT NULL)
UNION
```

```
 (SELECT surname, name FROM teacher
WHERE patronym IS NOT NULL);
```

Using ALL after UNION will result in duplicates being preserved
values.

We will select the last names, first names and patronymics of students who do not teach
(in the Master's program) (students minus teachers) (we assume that there are no full namesakes)

```
 (SELECT surname, name, patronym FROM student )
EXCEPT
 (SELECT surname, name, patronym FROM teacher);
```

Keywords *ANY* and *ALL*

The ANY and *ALL* keywords can be used with subqueries,
returning a single column of numbers. If the subquery is preceded by
the ALL keyword , the comparison condition is considered to be satisfied only if
if it is satisfied for all values in the resulting column
subquery. If the subquery text is preceded by the keyword *ANY,* then
the comparison condition will be considered fulfilled if it is satisfied
at least for some (one or more) values in the resulting
subquery column. Example Find all employees whose salary exceeds
the salary of at least one employee of the company's branch under the number 'vooz'.

```
SELECT staffNo, fName, IName, position, salary
FROM Staff WHERE salary >ANY(SELECT salaryFROM Staff
WHERE branchNo = 'B003');
```

Existential Queries in SQL

The EXISTS and *NOT* EXISTS keywords are intended for use
only in conjunction with subqueries or control structures.
The result of their processing is a logical value *TRUE* or
FALSE. For *the EXISTS* keyword, the result is *TRUE* if and only if
if the resulting table returned by the subquery
at least one row is present. If the resulting table of the subquery
empty, the result of processing *the EXISTS* keyword will be *FALSE.*
The NOTEXISTS keyword uses the reverse processing rules.
in relation to the EXISTS keyword . For example, a group where there is
students, will be received by request

```
Select st_group.* from st_group where
```

```
exists (select id_student from student where student.id_gr=st_group.id_gr)
```

Often the NOTEXISTS condition is used to implement the difference.

For example: A group without students can be found by query

```
Select st_group.* from st_group where not exists (select
id_student from student where student.id_gr=st_group.id_gr)
```

Note that the subquery has pass-through visibility.

One of the purposes of NOTEXISTS is to implement relational partitioning (so

so-called "all" queries).

Let's look at the example of a database scheme for students' participation in clubs

(sections) shown in Figure 18.

The query sounds like this: **Students who attend all clubs.**

For the simplest implementation of such a request, let's look at it from

from the point of view of relational division. In this case we must divide

students into circles, that is, a student is the dividend, and all the circles are the divisor.

such a request can be divided into 3 parts-requests:

```
ÿNOT EXISTS

 (B NOT EXISTS (C))
```

Moreover, each part has its own purpose.

Part A is the dividend, B is the divisor, and C is responsible for the connection between the dividend

and divider (student and all sections)

Queries A and C are usually similar up to aliases (end-to-end

visibility - aliases are required), but in C there is an additional connection with

A by the key of the dividend (in this case, the student) and the combination of C and B by the key

divisor (in this case, sections). Query B specifies the divisor.

Such a request will look like:

```
Select student.* from student
where not exists
 (select * from CAS_activities
where not exists
 (Select * from student student_activity as st_a
where student.id_st=st_a.id_stand st_a.id_cas= CAS_activities.id_cas))
```

If the request sounds like this, for example

Students who attend all math related clubs

The only thing that changes here is the divisor. Instead of "all circles" it will be "all

clubs related to mathematics»

Then the request will change like this

```
Select student.* from student
where not exists
 (select * from CAS_activities
where name like '%mathemat%'
and not exists
 (Select * from student student_activity as st_a
where student.id_st=st_a.id_stand st_a.id_cas= CAS_activities.id_cas))
```

Requests for difference

The difference query can be implemented in several ways:

Using except, not in, not exists, left join. Let's look at the query example.

"Groups where there is Ivanov, but no Petrov"

Here, from the groups with Ivanov, it is necessary to subtract the groups where Petrov is present.

We write queries about the subtrahend and the minuend.

Diminutive: Groups where there is Ivanov

```
select st_group.* from st_group inner join student on student.id_gr=st_group.id_gr where
student.surname='ÿÿÿÿÿÿ'
```

Subtrahend:Groups where there is Petrov

```
select st_group.* from st_group inner join student on student.id_gr=st_group.id_gr where
student.surname=' ÿÿÿÿÿÿ'
```

Implementation using except

```
select st_group.* from st_group inner join student on student.id_gr=st_group.id_gr where
student.surname='ÿÿÿÿÿÿ'

except

select st_group.* from st_group inner join student on student.id_gr=st_group.id_gr where
student.surname=' ÿÿÿÿÿÿ'
```

Implementation using not in

```
select st_group.* from st_group inner join student on student.id_gr=st_group.id_gr where
student.surname='ÿÿÿÿÿÿ'

  and st_group.id_gr not in

 (select st_group.id_gr from st_group inner join student on student.id_gr=st_group.id_gr where
student.surname=' ÿÿÿÿÿÿ')
```

Implementation using not exists

```
select st_group.* from st_group inner join student on student.id_gr=st_group.id_gr where
student.surname='ÿÿÿÿÿÿ'

And not exists(
```

```
select *from st_group as gr2 inner join student on student.id_gr=gr2.id_gr where student.surname=' ÿÿÿÿÿÿ'

And st_group.id_gr=gr2.id_gr)
```

Implementation using left join

```
select st_group.* from st_group inner join student on student.id_gr=st_group.id_gr

Left join

 (select st_group.* from st_group inner join student on student.id_gr=st_group.id_gr where
student.surname=' ÿÿÿÿÿÿ'

)q on st_group.id_gr=q.id_gr

where student.surname='ÿÿÿÿÿÿ' and q.id_gr is NULL
```

Examples of implementing different types of queries with subqueries

Group with the maximum number of students

```
Select st_group.number, count (id_st) as cnt from student join st_group on student.id_gr=st_group.id_gr group by student.id_gr
having count (id_st)
=(select max(cnt) from

    (select count (id_st) as cnt from student group by id_gr)q)
```

Through ALL

```
Select st_group.number, count (id_st) as cnt from student join st_group on student.id_gr=st_group.id_gr group by student.id_gr
having count (id_st)>
=ALL (select count (id_st) as cnt from student group by id_gr)
```

Via WITH

```
with q1 as (select st_group.id_gr, number_gr, count(id_st) as cnt from st_group join student on
student.id_gr=st_group.id_grgroup by st_group.id_gr, number_gr)select q1.id_gr, q1.number_gr, cnt from q1where cnt=(select
max(cnt) from q1);
```

Lab #6 Developing SQL queries: queries with subqueries

Objective of the work: To acquire skills in creating queries with subqueries.

Task and sequence of work execution

By analogy with the examples given in paragraph 1, implement queries g) .. f), specified in the task option. One of the requests for maximum/minimum implement with the all directive. Query for "all" (relational division) implement using 2 not exists and using an aggregate function. Queries

for the difference, implement in 3 options: Not in, except(MySQL does not support,
so just syntax), using left/right join

Contents of the report

ÿ Purpose of the work

ÿ assignment text (together with the text of the assignment option);

- physical model of the database;

ÿ elements of the test data table from work 4, concerning the requests of Ms.:

ÿ SQL query text;

ÿ data sets returned by queries (screenshots);

ÿ conclusions about the features of implementing queries with subqueries.

Test questions and tasks

ÿ There is a student table, which stores his ID, last name, first name, patronymic, Department and mode of study. How to count students in each department?

ÿ There is a table student, which stores his ID, last name, first name, patronymic, department and form of study. How to count students of each form of study in each department?

ÿ What is the difference between count(id) and count(distinct id)?When is it used each option?

ÿ What is the difference between count(id) and count(*)? When is each used? option?

ÿ How to get the value of an aggregate function from the value of another aggregate function functions?

ÿ Where in queries can subqueries be used?

Stored procedures and functions

Stored routines (procedures and functions)

Stored routines are named blocks of SQL and its
procedural extensions stored on the server in compiled form,
which can be called by name, and at the same time they can take on different
parameters. There are two types of stored routines: procedures and functions. And
procedures and functions can accept the data specified by the calling program
a set of parameters and perform a series of actions. Both types of routines can

change and return data is passed to them as a parameter. The difference between a procedure and a procedure is that a function will always be necessarily return a single value to the caller, as opposed to a procedure that may return no value at all or return one or more meanings.

The SQL language is quite rich, but it is not sufficient for all complex operations. declarative queries. That's why procedural extensions were created SQL language. In some DBMS their use must be specified explicitly (for PostgreSQL it is PL/pgSQL). For PostgreSQL stored procedures and functions can be created in SQL language when the entire procedure or function is a sequence of SQL queries, as well as in PL/pgSQL, if it is necessary to use conditions and control structures, branches or conditions. The SQL function for PostgreSQL will return the result of the last request.

Unlike individual SQL statements processed by the server individually, queries, Stored procedures and functions are compiled once and stored in a combined form in the database on the server. This allows you to get the following advantages:

- ÿ Additional requests from the client and server are minimized.
- ÿ Intermediate results that are not needed by the client application, does not need to be transmitted between the server and the client. What relevant for complex actions that cannot be performed by one person request
- ÿ It is possible to avoid unnecessary parsing of requests when calling again procedures.
- ÿ Improving data security by limiting capabilities direct access to table data, allowing users can manipulate data only through stored procedures, if the user has the appropriate rights.

For server programming in general and stored routines in particular

There are also disadvantages:

- ÿ Separation of part of the application business logic for implementation in the DBMS
- ÿ Difficulty of testing and debugging. This is due to the fact that errors in the code procedures do not manifest themselves until execution time (call) procedures.
- ÿ Poor portability between DBMS.

Stored procedures and functions can use SQL queries on sampling, data manipulation and data definition, and special control structures, as well as variables for storing values.

Stored procedures and functions are created using *CREATE PROCEDURE* commands. and *CREATE FUNCTION* and are called by *CALL* commands (EXEC for MSSQLServer) and assigning the returned value or fetching it *(SELECT)* respectively. Comparison of procedures and functions in MySQL and in PostgreSQL is shown in Table 9.

Table 9. Comparison of procedures and functions in MySQL and PostgreSQL

Differences between stored routines	MySQL	PostgreSQL
Languages for writing stored routines	SQL with built-in procedural extensions	SQL, PL/pgSQL procedural extension of the SQL language, built-in procedural elements of other languages PL/ Tcl PL/Perl and PL/Python (corresponding to languages Tcl ,Perl ,Python)
body renewal subroutines	Just delete the procedure and add it again	You can update the body using CREATE OR REPLACE
Identification of subroutines	Totality 1) the name of the subroutine, 2)database name	Totality 1) the name of the subroutine, 2) database name and schema 3) number and type of parameters [6]
Procedure parameter transfer mode	IN, OUT, INOUT	IN, OUT, INOUT,VARIADIC*
Function parameter transfer mode	IN	IN, OUT, INOUT,VARIADIC*
Return type	Any valid type	Any valid type

function values	MySQL data (single object)	PostgreSQL data, including custom types, trigger, table, set of objects of a given type

VARIADIC mode, in which an attribute can be followed by at most 1 OUT parameter, used to pass a variable number of arguments as an array, provided that all these arguments are of the same type data.

Here is a general simplified syntax for creating procedures and functions.

The syntax for creating a procedure in MySQL is given below:

```
CREATE PROCEDURE procedure_name ([procedure_parameter[,...]])
 [characteristic ...] subroutine_body
CREATE FUNCTION function_name ([function_parameter[,...]])
RETURNS type
 [characteristic ...] subroutine_body
procedure_parameter:
   [ IN | OUT | INOUT ] parameter_name type
function_parameter:
  parameter_name type
type:
  Any MySQL data type
characteristic:
  LANGUAGE SQL
  | [NOT] DETERMINISTIC
     { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
  | COMMENT 'string'
subroutine_body:
  Correct SQL expression.
```

Syntax for creating a PostreSQL procedure

```
CREATE [ OR REPLACE ] PROCEDURE
name ( [ ...] ] )    [ argmode ]            [ argname ] argtype [                    { DEFAULT | = } default_expr ] [,

    { LANGUAGE lang_name
        | TRANSFORM { FOR TYPE type_name } [, ... ]
        | [ EXTERNAL ] SECURITY INVOKER | | SET                  [ EXTERNAL ] SECURITY DEFINER
        configuration_parameter { TO value | = value | FROM CURRENT }
        | AS 'definition'
        | AS 'obj_file', 'link_symbol'
| sql_body
    } ...
```

The full syntax can be viewed on the official website documentation[11,13].

Stored procedures and functions can have the following purposes:

- ÿ To protect table data from unauthorized access;
- ÿ To manage master and reference data;
- ÿ For logical deletion of data;
- ÿ For complex analytics in transactional databases;

ÿ To cascade delete all dependent data regardless of
declarative referential integrity;
ÿ For batch execution of tasks.

Master data is data that provides context for information about
business activities presented in the form of related
activities of generally accepted abstract concepts. They include descriptions
(definitions and identifiers) of details of internal and external objects,
involved in business processes, such as customers, products, employees,
Sellers and controlled areas (code meanings)[14].

Reference data is any data used to determine
characteristics or classification of other data, or for correlation
data within the organization with external ones. [14] Mainly reference data
consist of codes and their descriptions, but can also have a more complex structure.
As part of reference data management, we may need
insert reference data into separate tables when entering data from
related to them. Similar actions may be required to insert
data into tables corresponding to weak entities.

Physical (hard delete) deletion is the deletion of a row from
databases completely. In contrast, logical or soft deletion
is the process by which a row containing deleted data is saved in
database, but in the additional column it is marked as deleted.

A comparison of the features of the removal types is given in Table 10.

Table 10 Comparison of logical and physical deletion

Differences	**Physical (hard) delete Logical (soft) delete**	
Description	The row is completely deleted from the database, the occupied memory area is freed and becomes available for further use.	The line is saved in the database, but in the service part it is marked as deleted. It is possible to store a deletion timestamp
Access data Remote data	not available	history (convenient for auditing) and dependent data are saved

Memory	Data takes up less memory The amount of memory occupied by data is constantly growing	
Requests	Queries are simpler	In queries, the conditions for relevance
Reference integrity	Ability to maintain referential integrity declaratively Storing information about the current	Referential integrity only active
Purpose	state of the subject area	Storing the history of the state of the subject area

Variables in SQL Procedural Extensions

In stored procedures, variables are used for temporary storage.
data in the body of subroutines, as well as for storing the data returned by the procedure or a function of data in the calling environment. Features of variables
SQL dialects for different DBMS may differ.
In all DBMSs, within the framework of procedural extensions, there are local variables that are scoped to the block in which they are declared.
Local variables with names without using quotation marks can
contain basic Latin letters, numbers 0–9, dollar, underscore.
special symbols that enclose the variable are possible
Unicode support. Special characters for MySQL are typewritten
backticks (gravises)``, for PostgreSQL double quotes “”, for MS
SQL Server square brackets [].
In MS SQL Server, MySQL up to version 8, the names of all local variables
start with the @ symbol.
In MySQL starting from version 8, PostgreSQL, Oracle local variables have
a name that begins with any valid character.
Additionally, MySQL has the concept of **user** variables,
whose names begin with the @ symbol and whose scope is global (all procedures, functions, triggers and beyond) within the session of a given user in this connection to the server. User variables
can be used for debugging or for substitution as actual
output parameters when calling procedures.

User variables do not need to be declared, but their value can be
request at any time, but if the variable has not yet been in this session
initialized, then its value will be null.
When writing procedures that involve inserting into child tables, you may
you will need to insert into the parent and get the value of the foreign key.
The methods for obtaining a new key value depend on the DBMS and whether
whether the key is auto-incrementing.
In general, for a non-auto-incrementing key, the key for the new record is
is calculated as the maximum value of existing keys +1. However,
It is necessary to take into account that if the table is empty, the *max* function will return null-
value, so it is necessary to use an additional function that
instead of a null value, it will substitute another value, with which we will start
counting keys in an empty table. For this purpose, you can use the function
coalesce, which takes an unlimited number of arguments and returns
the first non- *null* argument, but if there are none in the list, it returns *null.* This
function from the SQL-92 standard, so it is supported by almost all
relational DBMS. Some DBMS have additional functions
2 arguments that perform a similar task, i.e. check the first one
argument to empty and return an alternative value (the second argument),
if the first one is *NULL.* In MySQL, SQLite it is the *ifnull function,* MSSQLServer –
isnull.

Get non-autoincrement key of table st_group into variable
id_gr_new and insert data for MySQL using the following code:

```
set id_gr_new =(select ifnull(max(id_gr)+1,0) from st_group );
INSERT INTO st_group(id_gr,num_gr) VALUES (id_gr_new,gr_num);
```

For PostgreSQL the code will look like this:

```
id_gr_new := (select coalesce(max(id_gr)+1,0) from st_group );
INSERT INTO st_group(id_gr,num_gr) VALUES (id_gr_new,gr_num);
```

Obtaining an auto-increment key will vary for different DBMSs
stronger.

MySQL uses the LAST_INSERT_ID() function, while PostgreSQL allows use 2 different ways: one of which is returning the value by the operator INSERT, and the second function currval, which is fed as an argument sequence name. Here is an example of insert code for the same table st_group groups for auto-increment key.

Code for MySQL

```
INSERT INTO st_group(num_gr) VALUES (gr_num);
set id_gr_new =LAST_INSERT_ID();
```

Code for PostgreSQL

```
INSERT INTO st_group(num_gr) VALUES (gr_num);
SELECT currval('st_group_id_seq');
```

Or

```
INSERT INTO st_group(num_gr) VALUES (gr_num)RETURNING
```

Syntax for creating a procedure

Example code for inserting a student into a non-autoincrementing table with the result foreign key for MySQL DBMS

```
1. delimiter //
2. CREATE PROCEDURE ins_stud_n_group (gr_num varchar(8),name_ varchar(15),
3. surname_ varchar(20),patronym_ varchar(25),out id_stud int)
4. BEGIN
5. declare id_gr_new int;
6. declare id_st_new int;
7. if exists(select * from st_group where num_gr=gr_num)
8. then select id_gr into id_gr_new from st_group where num_gr=gr_num;
9. else begin
10. set id_gr_new =(select ifnull(max(id_gr)+1,0) from st_group );
11. INSERT INTO st_group(id_gr,num_gr) VALUES (id_gr_new,gr_num);
12. end;
13. end if;
14. set id_st_new = (select ifnull(max(id_st)+1,0) from student);
15. set id_stud = id_st_new;
16. insert into student (id_st, surname, name,patronym,id_gr)
17. VALUES (id_st_new,surname_,name_,patronym_,id_gr_new);
18. END; //
19. delimiter ;
20.call ins_stud_n_group('4131', 'Ivanov', 'Ivan', 'Ivanovich', @id_st_new);
21. select @id_st_new;
```

Line 1 changes the default separator to // characters. . and in line 19 completion of the procedure creation code returns to the separator by default.

Lines 2 and 3 show the header of the stored procedure with the input parameters: group number and last name, first name and patronymic, and day off

parameter — the identifier of the added student. Lines 5 and 6 –
declaring local variables to store primary key values
group *(id_gr_new)* and student *(id_st_new).*
Line 7 checks for the presence of a group with the given number; if so,
in line 8 the primary key of this group is written into a variable
id_gr_new. If the group is not found, then the key value is calculated in line 10
for a new record in a table with a non-auto-incrementing key, and then on line 11
a group with a calculated key is added.
In line 14 we calculate the new key value for the student, by default
considering that we add a new one in the procedure and in the next 15th line,
we write it to the output parameter. To simplify the recording, it was possible not to
use a local variable for the student key, and immediately write to
return value. In lines 16 and 17 we add a student with the found
above the values of the student and group keys. It is necessary to pay attention,
that after the end of the procedure body, the line must be
command separator specified in line 1, signaling the end of a command
creating a procedure.
Line 20 calls the student insertion function *ins_stud_n_group,* where for
return value of primary key of inserted student is used
user variable *@id_st_new.* Then the value of this variable
is displayed to the user on line 21.
In a situation where the group table has an auto-incrementing key, it is necessary
will replace lines 10 and 11 with the following lines:

```
10. INSERT INTO st_group(num_gr) VALUES (gr_num);
11. SET id_gr_new =LAST_INSERT_ID();
```

And similarly for inserting into the students table with an auto-incrementing key.
Example code for inserting a student into a non-autoincrementing table with the result
foreign key for PostgreSQL DBMS

```
1. CREATE OR REPLACE PROCEDURE
2. ins_stud_n_group ( gr_num varchar(8),name_ varchar(15), surname_ varchar(20),patronym_ varchar(25),out id_stud int )
3. AS $$
4. declare
5. id_gr_new int default null;
```

```
6. id_st_new int default null;
7. BEGIN
8. if exists (select * from st_group where num_gr=gr_num)
9. then select id_gr into id_gr_new from st_group where num_gr=gr_num;
10. else begin
11. id_gr_new := (select coalesce(max(id_gr)+1,0) from st_group );
12. INSERT INTO st_group(id_gr,num_gr) VALUES (id_gr_new,gr_num);
13. end;
14. end if;
15. id_st_new = (select coalesce(max(id_st)+1,0) from student);
16. id_stud = id_st_new;
17. insert into student (id_st, surname, name,patronym,id_gr) VALUES (id_st_new,surname_,name_,patronym_,id_gr_new);

18. END;$$
19. LANGUAGE plpgsql;
```

PostgreSQL program listing for auto-incrementing group tables and student is listed below

```
1. CREATE OR REPLACE PROCEDURE
2. ins_stud_n_group_autoinc ( gr_num varchar(8),name_ varchar(15), surname_ varchar(20),patronym_ varchar(25),out id_stud int )

3. AS $$
4. declare
5. id_gr_new int default null;
6. BEGIN
7. if exists (select * from st_group where num_gr=gr_num)
8. then select id_gr into id_gr_new from st_group where num_gr=gr_num;
9. else begin
10. INSERT INTO st_group(num_gr) VALUES (gr_num) returning id_gr into id_gr_new;

11. end;
12. end if;
13.
14. insert into student ( surname, name,patronym,id_gr)
15. VALUES (surname_,name_,patronym_,id_gr_new)returning id_st into id_stud ;
16. END;$$
17. LANGUAGE plpgsql;
18. call ins_stud_n_group_autoinc ('4131', 'Ivanov', 'Ivan', 'Ivanovich', null);
```

For cascading deletion, it will be enough to write deletion queries with subqueries, starting from the nested query itself.

Control structures

Assigning values to variables differs between MySQL and PostgreSQL The comparison is given in Table 11.

Table 11. Comparison of assignment in MySQL and PostgreSQL

Construction	MySQL	PostgreSQL
Syntax assignments	SET *variable = expr* [, *variable = expr]*	*variable* { := | = } *expression;*

An example assignments	SET *X = 10;*	*X=10;*
Syntax records via select query	SELECT ... INTO *list_of_variablesFROM* ...; *list_of_variables-* list of variables, separated by commas.	SELECT *select_expressions* INTO [STRICT] result_receiver FROM ...; For PL/pgSQL only
Example records via request	select id_gr into id_gr_new from st_group where num_gr='4131';	select id_gr into id_gr_new from st_group where num_gr=gr_num;

In PL/pgSQL, the result receiver can be a record (aggregate) variable,

variable - a table row or a list of simple variables and fields

records/lines separated by commas.

In addition, in PostgreSQL, PL/pgSQL has the ability to write a value to

variable through a data change request using the following syntax:

INSERT| DELETE | UPDATE... RETURNING *expressions* INTO [STRICT] *target;*

For example:

INSERT INTO st_group(num_gr) VALUES (gr_num) returning id_gr into id_gr_new;

The STRICT option requires the command to return exactly one row or

runtime error message: *NO_DATA_FOUND* (no rows) or

Too_MANY_ROWS (more than one row).

Conditional operator

Conditional operator MySQL and PostgreSQL have almost the same syntax.
The syntax for PostgreSQL differs from the syntax for MySQL only by writing *ELSIF* instead of *ELSEIF.* In both cases, the conditional operator has two optional parts: the else branch and additional branches of the else if selection and ends with the mandatory end if. The syntax and example of the MySQL and PostgreSQL conditional operator are given in Table 12. Table 12 Conditional operator

	MySQL	PostgreSQL
Syntax conditional	IFcondition1THENoperators1 *[ELSEIFuslovie2THENoperator*	IFcondition1THENoperators1 *[ELSIFcondition2THENoperators*

operator	y2]... *[ELSEoperators3]* ENDIF	2] ... [ELSE *statements3]* END IF

An example use conditional operator.
Procedure/function, outputting text information about positivity and argument.
	delimiter //
	create procedure test_if(numbe int)
	begin
	declare result varchar(20) default null;
	IF (numbe= 0) THEN
	set result = 'zero';
	ELSEIF numbe > 0 THEN
	set result = 'positive';
	ELSEIF numbe < 0 THEN
	set result = 'negative';
	ELSE
	set result = 'NULL';
	END IF;
	select result;
	end;//
	delimiter ;
	call test_if(1);
		create function test_if(numbe int) RETURNS varchar(20)
		LANGUAGE 'plpgsql'
		as $$
		declare result varchar(20) default null;
		begin
		IF (numbe= 0) THEN
		set result = 'zero';
		ELSIF numbe > 0 THEN
		result = 'positive';
		ELSIF numbe < 0 THEN
		result = 'negative';
		ELSE
		set result = 'NULL';
		END IF;
		return result;
		end;$$
		select test_if(1);

Cycle

Most often, loops are used for element-by-element processing of data or insertion of data, for obtaining calculations are required. A comparison of loops in MySQL and PostgreSQL is given in Table 13

Table 13. Working with cycles in DBMS

	MySQL	PostgreSQL
Cycle with	*[begin_label:]*WHILE *loop_continuation_condition* DO	[<<label>>] WHILE *loop_continuation_condition*

prerequisites m. Syntax	*loop_body_statements* END *WHILE[end_label]*	LOOP *loop_body_statements* END LOOP [label];

An example
used
and cycle.
Addition
data on
section
meetings once a
week
specified day
to specified
 	
dates,
day_of_week
_num - number
day of the week
fire1do7,
starting from
Monday
a
	create table cas_shedule (
	id_shedule int
	auto_increment not null primary key,
	
	id_cas int not null,
	date_meet date,
	foreign key (id_cas) references
	cas_activities (id_cas) on delete cascade
	on update restrict);
	
	delimiter //
	create procedure
	insert_dates_cas(id_cas_ int,day_start
	date,day_end date,day_of_week_num
	smallint)
	
	begin
	DECLARE dt DATE DEFAULT
	null;
	set
	day_of_week_num=day_of_week_
	num-1;
	set dt= DATE_ADD(day_start,
	INTERVAL
	(7+day_of_week_num-
	WEEKDAY(day_start))mod 7 day);
	
	if exists (select * from cas_activities where
	id_cas=id_cas)and(dt is not
	null)and(day_end is not null) then
	
	begin
	WHILE dt <= day_end DO
	insert into cas_shedule (id_cas,date_meet)
	values(id_cas_,dt);
	
	SET dt =
	DATE_ADD(dt,INTERVAL 1 week);
	
	END WHILE;
	end;
	end if;
	end; //
	delimiter ; call
	insert_dates(1,'2023-
	09-01','2023-10-01',1);
		create table cas_shedule (
		id_shedule serial not null primary key,
		
		id_cas int not null,
		date_meet date,
		foreign key (id_cas) references
		cas_activities (id_cas) on delete cascade
		on update restrict);
		
		create procedure
		insert_dates_cas(id_cas_ integer,day_start
		date,day_end date,day_of_week_num
		smallint)LANGUAGE 'plpgsql'
		
		as $$
		DECLARE dt DATE DEFAULT null;
		begin
		dt=day_start+
		mod(7+day_of_week_num-
		(extract(isodow from
		day_start)),7)::integer;
		if exists (select * from cas_activities where
		id_cas=id_cas)and(dt is not
		null)and(day_end is not null) then
		
		begin
		WHILE dt <= day_end LOOP
		insert into cas_shedule (id_cas,date_meet)
		values(id_cas_,dt);
		
		dt = date(DATE_ADD(dt,'1 week'::interval));
		
		END LOOP;
		end;
		end if;
		end; $
		$
		call insert_dates_cas(1,'2023-
		09-01'::date,'2023-10-
		01'::date,2::smallint);

There are also unique cycles for DBMS. PostgreSQL supports

a loop with a counter of the following syntax:

[<<label>>]
FOR *var_counter* IN [REVERSE] *start_value .. end_value* [BY *step]* LOOP
loop_body_statements
END LOOP [label];

Example of using a loop with a counter. Procedure for adding a given number of section meetings once a week on a given day day_of_week_num - day of the week number from 1 to 7, starting from Monday.

```
create or replace procedure insert_dates_cas_for(id_cas_ integer,day_start date,amount integer,day_of_week_num smallint)LANGUAGE
'plpgsql'
as $$
DECLARE dt DATE DEFAULT null;
declare i integer;
    begin
    dt=day_start+                    mod(7+day_of_week_num-(extract(isodow                    from
day_start)),7)::integer;
if exists (select * from cas_activities where id_cas=id_cas)and(dt is not null)and(amount is not null) then

begin
FOR i IN 1          .. amount BY 1
    LOOP
        insert into cas_shedule (id_cas,date_meet) values(id_cas_,dt);
    dt = date(DATE_ADD(dt,'1 week'::interval));
      END LOOP;
  end;
  end if;
end;
$$
call insert_dates_cas_for(2,'2023-09-01'::date,5,3::smallint);
```

Managing access rights

There are a number of commands to manage access rights in SQL:

CREATEUSER — create a user.

DROP USER — delete a user.

CREATE ROLE — create a role.

DROPROLE — remove a role.

GRANT —Gives rights to various objects to a role and user or assigns roles to users.

REVOKE — to revoke access rights from a role or user or to deprive them of user roles

Commands are common to most DBMSs, but have different features syntax.

PostgreSQL

CREATE USER is an alias for CREATE ROLE[13]. The only difference is is that when the command CREATE USER is written, by default the LOGIN option is assumed, meaning that the role or user can

be specified as the initial session authorization name during

client connections. whereas NOLOGIN is assumed when the command

CREATE ROLE is written, which does not allow the role to log in to the system.

An example of working with users and rights.

Listing of creating a user who can add data to the database

only through procedures, but does not have access to edit the tables themselves

for MySQL is given below

```
CREATE USER 'for_proc'@'localhost' IDENTIFIED BY '111';
GRANT select ON uni.* TO 'for_proc'@'localhost'; GRANT EXECUTE ON
PROCEDURE uni.`ins_stud_n_group` TO 'for_proc'@'localhost';
SHOW GRANTS FOR 'for_proc'@'localhost' ;
```

First, a user is created, then he is granted read permissions

all database objects, then permission to execute the procedure.

The last command displays his rights.

If implemented not through the rights of a specific user, but through the introduction

roles that are then allocated to specific users a little bit

longer.

```
1. CREATE ROLE 'deans_office';
2. GRANT select ON uni.* TO 'deans_office';
3. GRANT EXECUTE ON PROCEDURE uni.`ins_stud_n_group` TO 'deans_office';
4. CREATE USER 'deans_office_worker'@'localhost' IDENTIFIED BY '111';
5. GRANT 'deans_office' TO 'deans_office_worker'@'localhost';
6. SET DEFAULT ROLE ALL TO 'deans_office_worker'@'localhost';
7. SHOW GRANTS FOR 'deans_office_worker'@'localhost' using 'deans_office';
```

In line 1 we create a role, in lines 2 and 3

To create a role in PostgreSQL, use the CREATEROLE command.

```
CREATE ROLE name [ [ WITH ] option [ ... ]
```

```
CREATE ROLE deans_office;
  GRANT select ON ALL TABLES IN SCHEMA public TO deans_office;
GRANT EXECUTE ON PROCEDURE ins_stud_n_group TO deans_office;
GRANT EXECUTE ON PROCEDURE
ins_stud_n_group(varchar(8),varchar(15),varchar(20),varchar(25), int) TO deans_office;

CREATE USER deans_office_worker PASSWORD '111';
GRANT deans_office TO deans_office_worker;
  SET ROLE 'deans_office_worker';
SELECT * FROM information_schema.routine_privileges WHERE grantee ='deans_office';

      SELECT * from information_schema.table_privileges WHERE grantee ='deans_office';
```

However, if you try to run the procedure, there will be an access error, since the user does not have the right to insert into the table. To have the rights when running

procedures were considered by the procedure owner, it is necessary to add SECURITY DEFINER at the end after specifying the language when creating or changing a procedure.

logicaldeletion

```
alter table st_group add (is_deleted boolean default false, date_delete datetime);

alter table student add (is_deleted boolean default false, date_delete datetime);

Delimiter //
create procedure logical_delete_group(id_group int)
begin
declare time_del datetime default now();
update st_group set is_deleted=true,date_delete=time_del where id_gr=id_group
;
update student set is_deleted=true,date_delete=time_del
where id_gr=id_group
      and is_deleted=false and id_st>0;

end;//
delimiter ;

call logical_delete_group(10);
```

For PostgreSQL, despite the lack of control structures, the Plpgsql language is required due to the presence of a variable.

```
alter table st_group add is_deleted boolean default false;
alter table st_group add date_delete timestamp;
alter table student add is_deleted boolean default false;
alter table student add date_delete timestamp;
CREATE OR REPLACE PROCEDURE
logical_delete_group(id_group int)
AS $$
declare time_del timestamp default now();
BEGIN
update st_group set is_deleted=true,date_delete=time_del where id_gr=id_group
;
update student set is_deleted=true,date_delete=time_del
where id_gr=id_group
      and is_deleted=false and id_st>0;

END;$$LANGUAGE plpgsql;
```

Lab #7 Stored Procedures. Access Control

Purpose of work: To acquire skills in server programming and database access control

Task and sequence of work execution

1) Create stored procedures in the database that implement:

— insertion with replenishment of reference books (information about the student is inserted, if the specified group number is not in the database, the record is added to the table with

list of groups) (we get a link to a foreign key by the value of the data from the parent table);

— a stored function of any purpose corresponding to the subject area.

— cascade deletion: deletion of all dependent data (before deleting a record about a group, records about all students of this group and their sections are deleted);

2) Create a user who has the rights specified in the option
tasks

Contents of the report

- ÿ Purpose of the work
- ÿ The text of the task together with the task version.
- ÿ Physical model of the database.
- ÿ Assigning developed stored procedures or functions to text
- ÿ Script for creating stored procedures or functions;
- ÿ Script for creating a user, with the ability to change data only through procedures and functions
- ÿ SQL statements and screenshots of data sets illustrating the work procedures
- ÿ SQL statements and screenshots of data sets, displaying user rights and demonstrating his work with procedures.
- ÿ Conclusions on the use of procedures and access control in the database you developed

Test questions and tasks

ÿ What is a stored procedure?
ÿ What is the difference between stored procedures and functions?
ÿ What is the difference between a local variable and a user variable in MySQL?
ÿ How to return a value from a stored procedure?
ÿ How to get a new value for a bet key when inserting data into a stored procedures?

ÿ What is the difference between logical and physical deletion?
ÿ How to give a user rights to read a table?
ÿ How to assign a role to a user?

Triggers

Definition, classification and purpose of triggers.

A trigger is a stored procedure that is not called directly, but
executed when a certain event occurs (insertion, deletion,
update line).

Differences between triggers and other stored procedures:

- Called by an event, cannot be called manually
- Cannot be called from the external interface (client application)
- Do not have parameters
- Cannot be functions, i.e. return values

Composition of triggers

- **event -** an operation in the database that caused the trigger
- **Time** of trigger call, relative to the operation
- **condition** is a logical expression that must evaluate to
  value TRUE for the trigger to be executed
- **action** - the body of the trigger procedure

Purpose of triggers (most common):

- Maintaining referential integrity
- Sending warnings or messages to the user about errors
  data
- Debugging (i.e. tracking references to specified variables and/or
  control over changes in the state of these variables).
- Audit (for example, recording information about who made certain payments and when)
  other changes in certain variable relationships).
- Measuring performance (e.g. time logging)
  occurrence or tracing of specified events in the database).

- Carrying out compensatory actions (for example, cascading
  Organize the removal of the supplier tuple for removal also
  corresponding supply convoys).
- Logical deletion

Triggers are divided into triggers before, instead of and by the time of action and purpose.
after. **Before and instead** triggers are used for cascading deletions,
error handling, saving old values, debugging, data encryption
After triggers are used to log changes, conduct
compensating actions (Removal with cleaning of the directory, calculation
computable field)

Types of triggers by the method of processing commands:

- For each row processed —FOREACHROW, which
  are run for each row of the table affected by the change;
- For each command processed—FOREACHSTATEMENT, which
  are run for each of the ***insert, update, delete instructions,***
  applied to the table.

During trigger execution, there are two conditions for both MYSQL and PostgreSQL:
NEW and OLD lines, repeating the structure of the table on which the
trigger (in other SQL dialects, tables ***inserted*** and ***deleted).*** NEW contains
new versions of data (inserted by the ***insert*** statement or modified
operator ***update).*** OLD contains old versions of data (deleted
operator ***delete*** or subject to change by the operator ***update).*** Link to

The specified tables are produced in the same way as for the main database tables.

Implementation of triggers for MySQL and PostgreSQL.

Support for triggers in MySQL began with version 5.0.2. MySQL uses the ANSI SQL:2003 standard for procedures and other functions.

MYSQL Triggers

Syntax for creating a trigger

```
CREATE TRIGGER trigger_name trigger_time trigger_event
ON tbl_name FOR EACH ROW trigger_stmt
```

trigger_name — trigger name

trigger_time — Time of trigger operation. BEFORE — before the event. AFTER — after the event.

trigger_event — Event:

insert — the event is raised by the insert, data load, replace operators

update — the event is raised by the update operator

delete — the event is raised by the delete, replace operators.

DROP TABLE and TRUNCATE statements do not cause trigger execution.

tbl_name — table name

trigger_stmt is an expression that is executed when the trigger is activated.

Example of a trigger implementing referential integrity (cascade update)

Let's create a simple trigger that will delete all of its students when a group is deleted:

```
Create trigger my_trigger
before delete on st_group FOR EACH ROW Begin delete from
student where id_gr =OLD.id_gr; End//
```

Example of a trigger for calculated fields

Another example of a post-update trigger with a computed column for

tables student and st_group taking into account the presence of the stud_count column in the group.

```
Create trigger my_trigger after update on student FOR EACH ROW Begin

Update st_group Set stud_count=stud_count-1 where id_gr =OLD.id_gr;
Update st_group Set stud_count=stud_count+1where id_gr =NEW.id_gr; End//
```

Example of a data backup trigger

Let's create another table that will store backup copies of rows.

from the st_group table, and then backup triggers on update and delete

```
CREATE TABLE backup_group (
 `id` INT( 11 ) UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
```

```
 `row_id` INT( 11 ) UNSIGNED NOT NULL,
 `num_gr` varchar(10) NOT NULL
) ENGINE = MYISAM
DELIMITER |
CREATE TRIGGER `update_gr` before update ON `st_group`
FOR EACH ROW BEGIN
    INSERT INTO backup_group Set row_id = OLD.id, num_gr = OLD. num_gr t;
END;|
CREATE TRIGGER `delete_test` before delete ON `test`
FOR EACH ROW BEGIN
    INSERT INTO backup_group Set row_id = OLD.id, num_gr = OLD. num_gr;
END |
DELIMITER ;
```

Example of a trigger with an error signal

```
create table trigger_test(id int not null);
delimiter //
create trigger trg_trigger_test_ins before insert on trigger_test for each row begin

declare msg varchar(255);
if new.id < 0 thenset msg = concat('MyTriggerError: Trying to insert a negative value in trigger_test: ', cast(new.id as char));

signal sqlstate '45000' set message_text = msg;
end if; end//
delimiter ;
```

Syntax DROP TRIGGER

```
DROP TRIGGER [schema_name.]trigger_name
```

The statement deletes a trigger. The database name is optional.

Syntax for creating a trigger in PostgreSQL

Triggers in PostgreSQL can be both row-level (FOREACHROW) and row-level (FOREACHROW).

operator level(FOREACHSTATEMENT).

The body of a trigger in PostgreSQL is a call to a stored function that

returns type "trigger":

```
CREATE OR REPLACE FUNCTION function_name() RETURNS trigger
```

The general syntax elements for creating a trigger are shown in Table 14.

Table 14. Trigger Syntax

Trigger Elements	**PostgreSQL**

General syntax	CREATE [OR REPLACE] [CONSTRAINT] TRIGGER *trigger_name* *trigger_time trigger_event* ON *table_name* [FROM *referenced_table_name]* [NOT DEFERRABLE | [DEFERRABLE] [INITIALLY IMMEDIATE | INITIALLY DEFERRED]] [REFERENCING { { OLD | NEW } TABLE [AS] *transition_relation_name }* [...]] [FOR [EACH] { ROW | STATEMENT }] [WHEN (condition)] EXECUTE *trigger_body*
trigger time trigger_time { BEFOR E| AFTER | INSTEAD OF }	
trigger event *trigger_event*	{INSERT |DELETE|TRUNCATE|UPDATE [OF *column_name* [, ...]] } *event* [OR ...]
referenced_table_name	The name of another table referenced by the constraint. This is used for foreign key constraints and is not recommended for general use. It is only allowed for constraint triggers.
trigger body	{ FUNCTION | PROCEDURE } *function_name* (arguments)

Example of PostgreSQL Counting Implementation

```
CREATE OR REPLACE FUNCTION calc_stud_gr_after() RETURNS trigger
AS $$
BEGIN
update st_group set stud_count = stud_count + 1 where st_group.id_gr = new.id_gr; update st_group set stud_count = stud_count - 1 where
st_group.id_gr =
old.id_gr; RETURN NEW; END;$$ LANGUAGE plpgsql;



CREATE TRIGGER calc_stud_gr_after AFTER UPDATE OF id_gr
  ON student
FOR EACH ROW EXECUTE PROCEDURE calc_stud_gr_after()
```

Example of PostgreSQL check implementation

```
CREATE OR REPLACE FUNCTION insert_existing_gr() RETURNS trigger
AS $$
BEGIN
 IF EXISTS (SELECT * FROM st_group WHERE st_group.num_gr = NEW.num_gr)
THEN RAISE EXCEPTION 'Cannot add group % because a group with the given number already exists', NEW.num_gr;

END IF;
RETURN NEW;
END;$$
LANGUAGE plpgsql;

CREATE TRIGGER insert_existing_gr1 BEFORE INSERT ON st_group
FOR EACH ROW EXECUTE PROCEDURE insert_existing_gr()
```

Syntax DROP TRIGGER

DROP TRIGGER [IF EXISTS] *name* ON *table_name* [CASCADE | RESTRICT]
Drops an existing trigger. If CASCADE is specified, drops dependent triggers.

objects from it

Lab #8 Triggers. Ensuring Active Database Data Integrity

Objective of the work: To acquire skills and abilities in designing and creating database triggers, including their use to maintain active referential integrity.

Task and sequence of work execution

Implement triggers for all events (insert, delete,

update) before and after.(6 triggers) Some of which will provide

referential integrity, the rest may have a different purpose from others

proposed, but not less than 2 different ones.

- Computing/maintaining up-to-date computables

(derived) attributes

- logging (recording) of changes;

- checking the correctness of the actions performed.).

Calculated fields can be added if needed.

Contents of the report

ÿ Purpose of the work

ÿ The text of the task together with the task option. ÿ
The physical model of the
database. ÿ The assignment of the developed triggers by text.

ÿ Script for creating triggers; ÿ SQL
operators and screenshots of data sets illustrating the work
triggers.

ÿ Conclusions about the use of triggers in the database you developed.

Test questions and tasks

ÿ What is a trigger?

ÿ How are triggers different from other stored procedures?

ÿ How to access added and deleted data inside a trigger?

ÿ What is the difference between before and after triggers in terms of execution and purpose?

ÿ How to return a value from a trigger?

ÿ How to pass parameters to a trigger?

Literature

1. Osipov D. L. Database design technologies. – M.: DMK Press, 2019. –498 p.: ill.
2. Connolly T. Databases. Design, implementation and maintenance. Theory and Practice: trans. from English / T. Connolly, K. Begg. - M. and others: Williams, 2017. - 1439 p
3. Shustova, L. I. Databases: textbook / L. I. Shustova, O. V. Tarakanov. — Moscow: INFRA-M, 2023. — 304 p. + Add. material [Electronic resource]. — (Higher education: Bachelor's degree). —URL: https://znanium.com/catalog/product/1986697 (accessed: 18.10.2023).
4. Valacich Joseph S. Modern Systems Analysis and Design 9th edition/ Valacich Joseph, George Joey F-— New York: Pearson, 2020. — 529 ÿ.
5. Date, K. Introduction to Database Systems. Eighth edition. – Moscow: Dialectics, 2019— 1328 p.
6. Volk V.K. Databases. Design, programming, management and administration: textbook /V.K. Volk. - St. Petersburg: Lan, 2020.
7. Sadalaj P.D. NoSQL: A New Methodology for Developing Non-Relational Databases data / Pramodkumar J. Sadalaj, Martin Fowler; [translated from English and edited by D. A. Klyushina]. - Moscow: Williams, 2013
8. Time Series Databases and Their Importance https://expeed.com/time-series-databases-and-their-importance/(ÿÿÿÿÿÿÿÿÿÿÿÿ: 19.10.2023)
9. Date, C.J.: Database Design and Relational Theory: Normal Forms and All That Jazz,. APRESS, Healsburg (2019)- 450 ÿ.
10. Dombrovskaya G. Optimization of queries in PostgreSQL / Dombrovskaya G., Novikov B., Beilikova A. , per. s English D. A. Belikova. - M.: DMK Press, 2022. – 278 p.: ill.

11. MySQL 8.1 Reference Manual URL:
https://dev.mysql.com/doc/refman/8.1/en/ (accessed 22.10.2023)
12. Documentation in Russian for PostgreSQL 15.4 URL:
https://postgrespro.ru/docs/ (date of access 10/22/2023)
13. Official documentation of PostgreSQL version 16
https://www.postgresql.org/docs/16/index.html(date of access 20.10.2023)
14. DAMA-DMBOK : Data Management Body of Knowledge. Second Edition /
DamaInternational [per. s English G. Agafonova]. — Moscow: Olymp-Business,
2020. — 828 p

Appendix 1 Distribution of points

Semester 5

	No. Name of laboratory	Cart thing points	Limit number weeks surrender
1.	Development of a conceptual model of the subject area	5	3
2.	Developing a Physical Database Model Taking into Account Declarative Referential Integrity	3	5
3.	Creating and modifying a database and database tables data	3	7
4.	Filling tables and modifying data	4	9
5.	Developing SQL Queries: Join Types and Templates	5	11
6.	Developing SQL Queries: Queries with Subqueries	7	13
7. Stored Procedures. Access Control		6	15
8.	Triggers: Ensuring Active Database Data Integrity	7	17
	Total	40	

Appendix 2 Variants of laboratory tasks 1-8

1. Graph drawing program: vertex name, upper left coordinates
points of the displayed vertex, and its dimensions, author of the graph, edge types
graph (directed and undirected), edge weight.
a. Vertices whose text contains the word "exit" but does not end with it
b. Users-authors who have graphs
c. Graphs that have a vertex with a loop (A loop in a graph is an edge incident
the same peak.)
g. Graphs whose width exceeds 200 pixels. Graph width in pixels (from
maximum sum of horizontal coordinate with width subtract minimum
left coordinate)
d. authors of graphs with the largest number of edges
e. Vertices that have outgoing edges leading to all other vertices
her count
f. Authors whose graphs do not contain vertices with the word "beginning" in the vertex name
2. Gardening: plots, owners taking into account joint ownership, lines/number
plot, area, cost of construction, type of buildings, electric meters
owner of the site
a. plot numbers of owners with patronymics ending in "ich", and
starting with the letter "B"
b. plots on which more than 1 type of building is registered
c. Type(s) of buildings that are absent from the sites
g. Owner(s) of the plot of maximum area
d. Owners of plots with more than average number of building types
e. Owners who have electric meters on all their plots
f. Areas where there are no meters but toilets

3. HOA: apartment, owner, number of rooms, area, living area, floor,
utility bills
a. apartments of owners whose patronymics end with "na" and begin with "A"
b. owners who have three-room apartments and studio apartments
v. floors on which there are no apartments
g. owners of apartments with a minimum area
d. the type of payment that the fewest people paid for
i.e. floors in apartments, for which all types of payments have been paid in total

f. owners who do not have two-room apartments, but have paid the removal fees
TKO in June last year

4. Park: trees, species, planting date, pruning date, location, alleys

a. alleys where poplars are found, which have in their name a clarification other than
words poplar

b. trees standing at the crossroads of the alleys: Triple Linden Alley and Theatre Alley

in. species not planted in the park

g. an alley where the trees that were pruned last grow

d. the species of trees of which there are the fewest in the park with ID 1

i.e. a type of tree found in all the alleys of a given park

f. an alley where linden trees grow, but no trees were planted in 2021

5. exam schedule: exam dates, discipline, teacher, group,
audience, one exam can be conducted by several teachers

a. audiences where exams are taken on subjects beginning with the word
"mathematics", but that's not the only part of the name

b. the discipline(s) for which exams are held in and in room 23-10 and in
23-16

v. teachers who do not take exams

g. Teachers administering the most recent exams

d. a group that has fewer exams on average

i.e. the auditorium where exams are held for all groups of the 4th faculty

f. date on which exams are held only for group 4131

6. outpatient clinic: appointment, patient, procedure/appointment, doctor, cost

a. doctors who performed any procedures with equipment associated with a laser (laser in
anywhere in the name)

b. a doctor who performed both an appointment and a procedure for one patient in different offices

v. procedures that have never been done to anyone

g. procedures with the lowest cost among all procedures performed by the doctor
Ivanov Ivan Ivanovich

d. an office in which more than the average number of appointments were held

i.e. a patient who has undergone all procedures whose names begin with "P"

f. the doctor who saw Ivanov Petr, but did not perform procedures with the devices
magnetic therapy.

7. personal account: discipline, work, type of work, student, teacher, date of submission
points

a. tasks that start with the word "development", but these are not the only words in them

b. a discipline that includes laboratory and other types of assignments

c. a task for which no work is attached

g. student who submitted his OOP coursework later than everyone else

d. a discipline for which more than average number of papers were submitted this year

i.e. a teacher who is responsible for all types of classes in group 4332 in the discipline
"Object-oriented programming"

f. a group whose students did not attach reports on laboratory work, but
attached for coursework

8. theatre costume department: role, performance, costume name, costume detail, size,
model author, development date

a. authors who developed costumes that had the word "princess" in the title, but did not
starting with it

b. suits that include both a coat and trousers

in a play that hasn't had any roles added yet

g. performances for which the oldest costumes were developed

d. an author who has designed a suit with a less than average number of parts

i.e. a role in which the costume contains all types of details

f. an author who did not design the costumes for Valkyrie, but designed the costumes in
current year

9. guarded parking: parking address, car, owner, place, car registration number,
date and time of arrival, date and time of departure

a. all parking lots located on streets with the word "Morskaya" in their name,
but the name doesn't start with it

b. a car owner who has several cars of different brands

in. an area where there is no parking

g. parking lot that opens later than everyone else

d. street with less than average number of parking spaces

i.e. a car that was parked in all the parking lots of the Central District

f. owner who parked on Nevsky Prospect but did not park in
Admiralty district

10. The database on the topic of “getting ready for a trip” should have a structure that allows implement the following queries:

a. Find which trips for the current year had the word "excursion" in the title, but not end with it

b. Find a category without items

c. Find out what type of trips both the lighter and the knife were taken on

g. Find the user who made the most recent trip last year

d. What categories of things are taken on trips of all types

e. What categories of things are not taken on a conference type trip, but are taken on a excursions

g. Find the type(s) of trips with the number of trips less than the average.

11. Database of TV series.

a. A series that has the word "home" in its title, but it's not the last one

b. A character played by different actors in different seasons

c. A series in which not a single character is announced.

g. Season and series of this season, with more than average number of episodes

d. genre of the oldest series

i.e. the season in which all the characters of the series meet

f. Actor who played in historical series but did not act in 2021

12. vacancies: vacancy title, employer organization, employer address, Salary range, Education requirements, Duties, Work schedule, requirements mandatory, desirable, date of posting of vacancy

a. vacancies that have "data" in the title (from the words data/data), but not ending with it

b. a schedule that is not present in any vacancy

in. employers who posted vacancies both in Moscow and St. Petersburg

g. vacancy with maximum salary at the lower limit of the salary range

d. vacancies with a lower than average number of mandatory requirements

i.e. a requirement present in all vacancies for a translator (any vacancies with in words translation)

f. vacancies that exist in St. Petersburg, but that Gazprom does not have

13. Family Budget Calculator: Income Category (Sales, Salary), Category
expenses (food, utility bills, health...), income and expense items, date of expense/income.
Category is a more general concept than an article. For example, the category is food, and the articles in it meat, fish, delicious with tea, and the specific consumption of "kurabye cookies for tea 11.09"

a. incomes of all categories name, which (categories) contain the word "gift", but not end with it

b. a month in which there were expense items for transport and health for the same person user

in. category without income

category, for which the income this year was the highest for a one-time payment size

d. category for which there were no expenses in January, but there were in May

i.e. the category of expenses for which all family members spent

f. month in which the number of items spent was less than the average

14. Contact book: people, tags (friends, colleagues, family), birthdays, phone, email, notes to a person or contact, holidays that people are celebrating

a. relatives whose phone number ends in 44

b. People (contacts) without email

v. people who celebrate both February 23 and Christmas

g. the oldest people among those celebrating February 23

d. a month when friends have holidays, but colleagues don't have birthdays

i.e. labels that include more than average number of people

f. month in which there are birthdays of people in the aggregate, related to all tags

15. universities for applicants: city, university, faculties, areas, specializations, Unified State Exam which need to be submitted, start/end date of the admission campaign.
(Direction -09.03.04 "Software Engineering", Focus - its specification "Development of software and information systems", it is the focus that is assigned to the department and, accordingly, the faculty)

a. directions in which the word "system" is present, but it is not the first

b. A department that does not accept any direction

in. the direction for which you need to pass mathematics and computer science

g. faculty accepting more than average number of applications

d. a city in which there are all large groups of directions and specialties (UGSN)
(the first 2 digits of the specialty number, i.e. for 09.03.04 UGSN=09, and for 02.03.03-02)

i.e. the university with the last name in alphabetical order

f. a field for which you don't have to take the Unified State Exam in mathematics, but you do have to
foreign language

16. school excursions: type (entertainment/educational), subjects to which
educational excursion, cost per person, list of participants,
teacher responsible for conducting

a. excursions, the title of which contains the word museum, but it is not the first

b. excursions related to history and geography

in. places where there are no excursions

g. students who did not go on excursions in May 2024, but went to the history museum
religions

d. teacher responsible for less than average number of excursions

i.e. teachers responsible for most daytime excursions

f. places all the students went to

17. dog kennel: dog, owner, breed, parents, medals from shows, date
birth

a. breeds whose names contain part of the word "terrier" but do not begin with it

b. dogs without medals

v. dog(s) whose puppies received the same medals

Mr. owner who has dogs of all breeds starting with the letter "p"

d. owner who has shepherds but does not have any dogs born in 2021

e. a dog that has more than the average number of puppies

f. owners of the oldest but still living parent dogs

18. Delivery service: delivery address, contact person, parcel cost, range
desired delivery time. actual delivery time, parcel weight, mark about
delivery, sender company

a. all parcels sent to the area, the name of which contains the beginning "mosk", but this is not
the only letters in it

b. Street where there are no delivery addresses

c. contact person who received parcels on Stroiteley Street and Lenin Avenue

g. the company that sent more than the average number of parcels

d. parcel with weight less than average

e. contact person who received parcels from all companies starting with the letter B

f. a contact person who never received parcels in the past year, but did receive
parcels in February of this year

19. tourist guide: city, landmark, address, type
attractions (monument, architectural complex, natural complex),
date of creation

a. landmarks that contain the word "house" but don't have a name for it
ends

b. a city without streets

v. street, which has both monuments and fountains

a city in which there are no architectural complexes, but there are museums on Lenin Street

d. a city with fewer museums than average

e. the street with the newest attractions

f. a type of attraction that is found in all cities where there is
attractions

20. metro: line, station, closing time, opening time, exit address,
travel time between stations, station opening date

a. stations whose names contain the word "prospect" but do not end with it

b. station without exits

v. a station that has exits onto 2 different streets

g. line with travel time below average

d. line on which all stations are collected, starting with the letter "n"

i.e. the station with the earliest opening

f. line, which does not have transfer stations to line 1, but has exits to the street
Lenin

21. Medical center of the organization: employee of the organization, fluorography data (result,
date), vaccination data, medical visits

a. all employees whose patronymic ends with "na" and begins with "I"

b. a type of disease for which no vaccinations have been given

in. employees of the organization who were vaccinated against measles last year and were not vaccinated against measles this year.
diphtheria

g. employee who last had a fluorography scan in the current year

d. department, whose employee has received all types of vaccinations

i.e. an employee with a lower than average number of visits to the medical center

f. an employee who had a fluorography this year but did not have one last year

22. Inventory: responsible person, equipment, categories of property
(computer equipment, office supplies, furniture, lighting, special equipment
etc.), premises, delivery date, write-off date

a. any premises where specialized tables are located (equipment containing
the words "table" , but it doesn't start with it.

b. premises in which no property is currently listed

in. a room in which there is equipment of categories and computer equipment and
another type

g. category of property written off last year

d. mat. responsible person(s) responsible for equipment of all categories

i.e. the material responsible person(s) responsible for the amount of property is greater
average

f. mat responsible person responsible for furniture, but not equipment from the premises
131

23. Corporate events calendar: time date, event, event status for
participant (mandatory, recommended, neutral), positions and departments
participants, venues

a. All events of the current year, in the title of which there is the word "seminar", but on
his name doesn't end there

b. A room where no events take place

c. A room where 2 different events took place on the same day in May of this year

g. Participants of the very first event in May 2023

d. The event with the largest number of participants who did not attend but were invited to it
above average. (Participants may be invited to the event and at the same time
may come/may not come).

e. An event in which the presence of all employees of the second department is mandatory

g. An employee who did not attend events in July last year but did attend
activities in room 11-11.

24. Distribution of duties: duty, title, frequency of execution, category
responsibilities (household, paying bills, attending events, social)

a. People who perform duties that have the word "washing" in their title, but do not
starting with it

b. Category of duties without duties

c. A person who has duties, both daily and with a different frequency of execution.

Mr. Ivanov Ivan Petrovich's duty, which is performed last by date
(day)

d. The frequency with which duties are performed by everyone who performs them at all
has responsibilities

e. A category in which the responsibilities are greater than average.

g. A person who performs duties in the category "repair" but does not perform
monthly duties

25. Bug tracking: projects, bugs, testers, developers (testers do not
are being corrected)

a. Bugs that have the word "memory" in their name, but it is not the first one

b. Bugs found in one project by one tester on different days

c. Developer without tasks

g. Critical bug that was added last

d. Tester, with less than average number of added bugs

e. A developer who has no bugs fixed late in the past year

f. A tester who has worked on all projects whose names begin with the word
"system"

26. Home medicine cabinet. Group of drugs, expiration date.

a. Find a medicine that contains the text "norm" in the name, but not at the end
b. Find a medicine that has analogues of the active substance in both cream and gel form.

c. Find a group of drugs that does not contain any drugs
g. Find the medicine with the closest expiration date, but not expired
suitability.
d. Find an active ingredient with a higher than average amount of medicine.
e. Find groups of drugs for which there are drugs in all forms.
g. Find a group of drugs that does not have tablets but does have drugs in other forms
expired.

27. gym: memberships, sports, trainers, membership type, equipment for the sport
sports

a. sports for which there are season tickets with the word "morning" in the type name, but this is not the first word of the title

b. a trainer who teaches judo and wushu

v. a sport for which there is no equipment

g. subscription (offered) for rumba with a minimum validity period in the subscription

d. a coach who teaches fewer than average sports

i.e. equipment that is needed in fitness, but not needed in boxing

f. a combined subscription for all types of ballroom dancing.

28. Create a fanfiction database (fanfiction is fantasies expressed in additions, revisions and continuations of cult works.) for storage the following information: original work (fandom), author of the fanfic (works based on), the name of the fanfic, the date it was added and the last editions, title and text of fanfic chapters.

a. list of fanfics, with a name starting with the letter b, added by the author with id=1 in the past year;

b. fan fictions written based on more than one work (crossovers), like "Alien vs. Predator");

v. fanfic, which does not have a single chapter yet

g. the oldest fandom(s);

d. the fandom about which the largest number of fanfics have been written;

i.e. authors who write only Star Wars-related works;

f. authors writing for all fandoms;

29. Create a database of idioms and catchphrases to store the following information: individual words idioms, idiom, definition, usage example, date adding an idiom (if any), the user who added the idiom to the dictionary

a) a list of idioms that contain the word "labor" but do not begin with it added in the current year;

b) pairs of idioms with the same words;

c) a word in the database that does not appear in idioms

d) the user who added the most idioms;

d) idioms with the largest number of examples of use;

e) users who added idioms with the word "time" but did not add idioms in 2023;

g) the year when idioms were added by all users;