

Министерство науки и высшего образования Российской Федерации

Национальный исследовательский университет «МИЭТ»

---

**С.А. Балабаев, Д.В. Киселев**

## **Базы данных**

Лабораторный практикум

Утверждено редакционно-издательским советом университета

Москва 2024

УДК 004.65(076.5)

ББК 32.973-018.2я73

И20

Рецензент: Горбунов Владимир Леонидович, д.т.н., профессор

**Балабаев С.А., Киселев Д.В.**

И20 Базы данных: практикум. М.: МИЭТ, 2024. 172 с.: ил.

**ISBN**

В учебном пособии рассмотрены и систематизированы основы баз данных, языка SQL, процедурного расширения PL/pgSQL. В качестве СУБД была предложена PostgreSQL, являющаяся одной из самых популярных на текущий момент на современном рынке. В завершающей части рассматривается способ взаимодействия баз данных с разработанным ПО на языке программирования С.

Учебное пособие предназначено для студентов бакалавриата направления «Информатика и вычислительная техника», «Программная инженерия», а также может быть полезна для студентов других специальностей.

**ISBN**

**© МИЭТ, 2024**

## **Оглавление:**

Введение.....	3 стр.
Лабораторная работа 1.....	6 стр.
Лабораторная работа 2.....	29 стр.
Лабораторная работа 3.....	60 стр.
Лабораторная работа 4.....	84 стр.
Лабораторная работа 5.....	108 стр.
Лабораторная работа 6.....	116 стр.
Лабораторная работа 7.....	142 стр.
Лабораторная работа 8.....	156 стр.

## **Введение**

Настоящий курс лабораторных работ был разработан для студентов, обучающихся по направлению 09.03.01 «Информатика и вычислительная техника». Но он также может использоваться студентами других специальностей, изучающих теорию и практику баз данных.

В лабораторных работах используется объектно-реляционная СУБД PostgreSQL. Эта СУБД достаточно известна и широко распространена. Кроме того, эта СУБД является свободной, что позволит обучающим и обучающимся избежать проблем лицензирования.

PostgreSQL включает в себя широкий спектр функций СУБД, что позволяет в результате работы с ней как получить начальные знания в области баз данных, так и глубоко изучить многочисленные возможности и стать профессионалом в этой области.

Курс начинается со знакомства с СУБД и выполнения элементарных запросов. Затем в нем рассматриваются более сложные вопросы: использование подзапросов, проектирование структуры БД, использование транзакций и индексов. Лабораторные работы позволят изучить основы программирования баз данных на примере процедур, функций и триггеров.

В конце курса обучающийся может попробовать силы в разработке программных приложений, взаимодействующих с БД.

В лабораторных работах используется небольшая демонстрационная БД, описывающая учебный процесс в Университете, что не потребует от студентов изучения незнакомой им предметной области, но позволит подробно изучить детали функционирования БД и СУБД.

Лабораторные работы содержат в себе практические задания двух типов – практикоориентированные и исследовательские. Первый тип задач предназначен для получения навыка работы с запросами на языке SQL. Второй тип предполагает изучение тонкостей СУБД PostgreSQL и приобретение опыта

написания оптимальных запросов. Практические задания разделены на несколько уровней сложности. Практикум составлен таким образом, что его можно использовать как под руководством преподавателя, так и самостоятельно.

Авторы пособия хотели бы выразить благодарность следующим людям за советы и ценные замечания: Аверкину Глебу, Богатырёву Кириллу, Внуковой Милене, Галустову Дмитрию, Дергачеву Вадиму, Елисееву Эдуарду, Макаровой Оксане, Петрушишину Александру, Рахимкулову Захару, Сорокину Александру, Фатыхову Алику, Шорохову Дмитрию, Яронскому Серафиму и многим другим неравнодушным к нашей работе студентам.

Отдельная благодарность Балабаевой Наталии Владимировне за неоценимый вклад в разработку пособия!

С уважением, авторы пособия.

# Лабораторная работа №1

## «Знакомство с PostgreSQL»

### 1. Теоретическая часть

#### 1.1. Основные понятия теории реляционных баз данных

Каждый день мы сталкиваемся с огромным количеством информации. Это могут быть формулы по математике, оценки за контрольные, даты исторических событий, цены на выпечку в буфете. Чтобы легче ориентироваться во всем этом разнообразии данных, каждый из нас стремится их упорядочить, разложить на простые составляющие. Для этого создаются различные схемы, графики и таблицы. Например, гораздо проще анализировать рост цен на булочки с корицей, построив график зависимости их цены от текущей даты. Или запоминать даты по истории, выписав их в столбик, в виде таблицы.

Такой подход делает работу с информацией более наглядной. Кроме того, структурированные данные, хранящиеся, например, в таблице *Microsoft Excel* проще обрабатывать, чем данные, записанные через запятую в блокноте. Однако при большом объеме информации использование обычных таблиц (*Excel*) становится неэффективным. В этом случае для хранения структурированной информации используют базы данных, а для ее обработки – одну из существующих СУБД.

**База данных (БД)** – совокупность данных, организованных по определённым правилам, предусматривающим общие принципы описания, хранения и манипулирования данными, независимая от прикладных программ. Эти данные относятся к определённой предметной области и организованы таким образом, что могут быть использованы для решения многих задач многими пользователями.

**Система управления базами данных (СУБД)** – совокупность программных и лингвистических средств общего или специального

назначения, обеспечивающих управление созданием и использованием баз данных.

В данном курсе будет рассматриваться объектно-реляционная СУБД **PostgreSQL**. PostgreSQL является одной из наиболее популярных СУБД в настоящее время. Одной из основных её особенностей является открытый исходный код. [1]

Базы данных строятся на основе некоторой модели данных.

**Модель данных** – абстракция, описывающая структуру (организацию) данных и методы их обработки.

Существуют различные виды моделей данных. Наиболее известными являются иерархическая, сетевая и реляционные модели. Наибольшую распространенность в настоящее время получила реляционная модель данных. Реляционная модель данных основана на понятии отношения (*relation*). В теории баз данных оно соответствует таблице, состоящей из столбцов (**атрибутов**) и строк (**кортежей**).

Приведем пример.

Таблица 1 — Пример списка студентов

Номер в списке	Номер студенческого билета	Фамилия	Имя	Отчество	Группа	Дата рождения	Номер паспорта
1	825644	Семенов	Павел	Романович	ИВТ-31	23.10.04	4356 678958
2	845227	Семенов	Иван	Иванович	ИВТ-32	21.07.05	4456 458326
3	861247	Орлова	Екатерина	Леонидовна	ИВТ-34	02.10.03	4244 523417

В нашей таблице, являющейся физической реализацией отношения, каждая из трех строк с информацией о студенте – кортеж, а столбцы «Номер», «Группа», «Фамилия», «Номер зачетной книжки», «Дата рождения», «Номер паспорта» – атрибуты.

Среди атрибутов или совокупности атрибутов можно выделить такие, которые обладают свойством уникальности и неизбыточности. Они

называются **потенциальными ключами**. Из них можно выделить один, который будет **первичным** ключом, остальные – **альтернативными**. Первичный ключ должен однозначно идентифицировать каждую запись в таблице. Если среди атрибутов потенциальных ключей нет, или их неудобно использовать, то возможно создать дополнительный атрибут (столбец). Дополнительный атрибут должен быть также уникален и избыточен для каждой записи. Содержательного значения он не несет. Такой дополнительный атрибут называется **суррогатным** ключом.

В примере «Номер студенческого билета», «Номер паспорта», «Фамилия+Дата рождения» – потенциальные ключи. Для однозначной идентификации достаточно одного, выберем «Номер зачетной книжки», тогда он будет первичным ключом, а «Номер паспорта», «Фамилия+Дата рождения» альтернативными. В случае, если нам неудобно использовать потенциальные ключи, можно ввести дополнительный атрибут (например, «Номер в списке»), он будет суррогатным ключом.

В реляционной базе данных таблицы могут быть связаны между собой. Они будут соотноситься как главные и подчиненные. Одной записи главной таблицы может соответствовать множество записей из подчиненной. Связь происходит посредством первичного и внешнего ключей. **Внешний ключ** – поле подчиненной таблицы, соответствующее первичному ключу главной таблицы.

Рассмотрим простой пример. Предположим, база данных состоит из двух таблиц – *Группа*, содержащая информацию об учебной группе вуза, и *Студент* – содержащая информацию о студенте, обучающемся в вузе. В качестве первичного ключа в таблице *Группа* выберем «Номер группы» (он будет уникальным во всем вузе). Каждый студент вуза обязательно состоит в одной конкретной группе. В одной группе может быть много студентов. Таким образом, возможно связать между собой эти две таблицы. Таблица *Группа*



будет главной, а *Студент* – подчиненной. Для связи в таблице *Студент* возьмём поле «Номер группы» и назначим его внешним ключом. Данное поля обязательно должно содержать одно из значений «Номера группы» из таблицы *Группа* – студент не может быть прикреплен к несуществующей в вузе группе.



Рисунок 1 — Связь Группа – Студент

Более подробно о реляционной теории можно прочитать в источнике [2].

## 1.2. Общее устройство PostgreSQL

СУБД PostgreSQL представляет собой клиент-серверную архитектуру. Рабочий сеанс PostgreSQL включает следующие взаимодействующие процессы:

- Главный серверный процесс, управляющий файлами баз данных, принимающий подключения клиентских приложений и выполняющий различные запросы клиентов к базам данных.
- Клиентское приложение пользователя, которому требуется выполнять различные операции над базами данных. Это может быть текстовая утилита, графическое приложение, веб-сервер, использующий базу данных для отображения веб-страниц.

Как и в других типичных клиент-серверных приложениях, клиент и сервер могут располагаться на разных компьютерах. В этом случае они взаимодействуют по сети TCP/IP. Важно не забывать это и понимать, что файлы, доступные на клиентском компьютере, могут быть недоступны (или доступны только под другим именем) на компьютере-сервере. [2]

Взаимодействие между клиентским приложением и сервером происходит посредством запросов. [3]

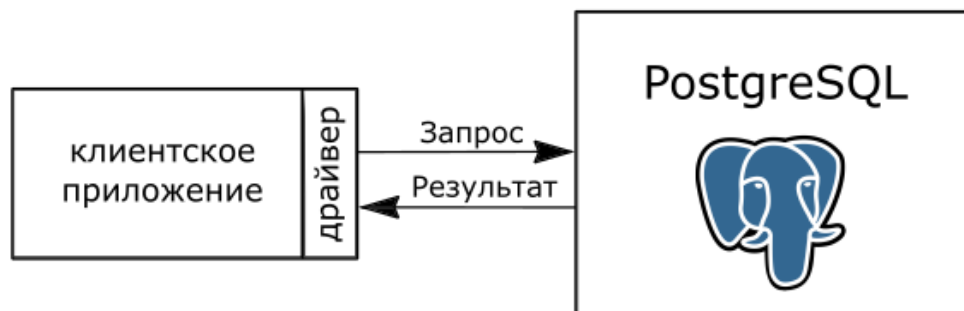


Рисунок 2 — Взаимодействие клиента и сервера

Работу с PostgreSQL возможно осуществить из терминала или с помощью графического приложения.

### 1.3. Язык программирования SQL

Работа с базами данных будет осуществляться с помощью языка программирования SQL. В отличие от знакомых вам императивных языков программирования C, C++, Python, Pascal и т.п. SQL является декларативным языком. [4]

**Декларативное программирование** (от *declare* – описание) — парадигма программирования, в которой задаётся спецификация решения задачи, то есть описывается ожидаемый результат, а не способ его получения.

Сравним между собой два подхода. Предположим, что нам необходимо найти в некоторой базе данных, содержащей информацию о студентах вуза, всех молодых людей по имени Александр. Напишем на псевдо-языке программирования решение данной задачи.

Таблица 2 — Подходы к программированию

Императивный подход	Декларативный подход
Для всех строчек таблицы Студент Если (имя студента = Александр)	<b>Выбери</b> всю информацию <b>Из</b> таблицы Студент <b>Где</b> имя студента = Александр

То вывести информацию о нем на экран	
--------------------------------------	--

Как можно увидеть из таблицы, в первом случае мы задаем последовательность действий, которые приведут к желаемому результату. Во втором случае – описываем результат того, что хотим получить.

Приведем более простой пример. Предположим, мы хотим приготовить на обед салат овощей. Императивный подход к решению задачи выглядит следующим образом:

*Купить огурцы, помидоры, лук, редис, оливковое масло;*

*Порезать огурцы, помидоры, лук, редис;*

*Полить оливковым маслом.*

При декларативном подходе описание будет звучать так: *хочу на обед салат из свежих овощей, заправленный оливковым маслом.*

Язык SQL включает в себя операторы, инструкции, вычисляемые функции.

Операторы SQL делятся на:

- операторы определения данных (Data Definition Language, DDL)
- операторы манипуляции данными (Data Manipulation Language, DML)
- операторы определения доступа к данным (Data Control Language, DCL)
- операторы управления транзакциями (Transaction Control Language, TCL)

Более подробно данные операторы будут рассмотрены в дальнейшем.

#### **1.4. Аутентификация в PostgreSQL.**

При работе с базами данных важно разделять права доступа между различными пользователями. В первую очередь это необходимо в целях безопасности. Например, рядовой сотрудник не должен иметь возможности удалить или испортить базу данных. С подобными разграничениями мы сталкивались при работе с электронной системой ОРИОКС. При подключении

в качестве студента имеется возможность лишь просматривать оценки, но при авторизации в качестве преподавателя – их выставять и редактировать.

PostgreSQL использует концепцию ролей (*roles*) для управления разрешениями на доступ к базе данных. Роль можно рассматривать как пользователя базы данных или как группу пользователей. Для каждой роли существуют свои права, от возможности любого изменения (администратор), до простого просмотра (простой пользователь). Роли назначаются администратором базы данных.

В теории баз данных существует разделение между понятиями пользователь (*user*) и роль (*role*). Пользователь – физическое лицо, которому могут быть выделены особые привилегии – роли. Однако, в последних версиях *PostgreSQL* данные определения имеют одинаковый смысл.

При подключении к серверу базы данных клиентское приложение указывает имя пользователя PostgreSQL, так же, как и при обычном входе пользователя на компьютер с ОС Unix. При работе в среде SQL по имени пользователя определяется, какие у него есть права доступа к объектам базы данных. Для соотнесения имени пользователя и его роли в PostgreSQL служит аутентификация.

Существуют понятия, которые необходимо различать:

**Аутентификация** — это процесс подтверждения права на доступ с помощью ввода пароля, пин-кода, использования биометрических данных и других способов.

**Идентификация** используется для определения, существует ли конкретный пользователь в системе. Проводится, например, по номеру телефона или логину.

**Авторизация** — процесс предоставления пользователю или группе пользователей определенных разрешений, прав доступа и привилегий в компьютерной системе.

PostgreSQL предлагает несколько различных методов аутентификации клиентов. Метод аутентификации конкретного клиентского соединения может основываться на адресе компьютера клиента, имени базы данных, имени пользователя. После установки Postgres настроена на использование аутентификации *ident*, что значит, что выполняется привязка ролей *postgres* с соответствующей системной учетной записи Unix/Linux.

### **1.5. Подключение к учетной записи.**

В ходе установки была создана учетная запись пользователя *postgres*, которая связана с используемой по умолчанию ролью *postgres*. Чтобы использовать *postgres*, вы должны войти в эту учетную запись. Сделаем это с помощью утилиты *sudo* операционной системы Linux.

Для этого в командной строке выполните следующую команду:

```
sudo -i -u postgres
```

Система предложит ввести пароль для текущего пользователя. После этого произойдет переключение на учетную запись *postgres*.

### **1.6. Создание резервной копии базы данных**

Одной из задач администратора баз данных является периодическое создание резервной копии базы данных. Существует несколько способов решить поставленную задачу. Рассмотрим простейший из них. Для создания бэкапа будем использовать встроенную утилиту *pg\_dump*.

Для создания резервной копии в командной строке из-под пользователя *postgres* выполните следующий скрипт:

```
pg_dump название_БД > название_выходного_файла
```

Например,

```
pg_dump postgres > template_dump.sql
```

Для восстановления резервной копии воспользуйтесь утилитой *psql*

```
psql название_БД < название_выходного_файла
```

Например,

```
psql postgres < template_dump.sql
```

Аналогичное действие возможно выполнить с помощью утилиты *pg\_restore*. Для этого файл резервной копии сохраняется утилитой *pg\_dump* в определённом формате с помощью ключей *-Fc* и далее восстанавливается.

```
pg_dump -Fc название_БД > название_выходного_файла
```

```
pg_restore -d название_БД название_входного_файла
```

## 1.7. Работа с командной строкой PostgreSQL

Подключение к командной строке PostgreSQL осуществляется следующей командой:

```
psql
```

После ввода данной команды будет установлено соединение с сервером PostgreSQL. По умолчанию, произойдет подключение к базе данных, название которой совпадает с именем роли – **postgres**. Таким образом, в системе определено **три разные** сущности с одинаковым именем *postgres* – имена учетной записи *Linux*, роли в *PostgreSQL* и базы данных.

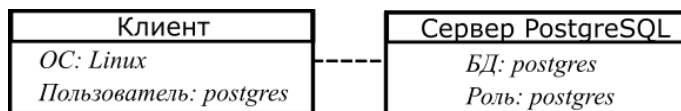


Рисунок 3 — Подключение к серверу PostgreSQL

Для работы с командной строкой *PostgreSQL* необходимо использовать специализированные команды. Некоторые из них приведены в таблице ниже.

Таблица 3 — Список команд *psql*

Команда	Описание работы
<code>\connect db_name</code>	подключение к базе данных <i>db_name</i>
<code>\dt</code>	вывести все таблицы
<code>\dt+</code>	вывести все таблицы с описанием
<code>\l</code>	вывести список баз данных
<code>\l+</code>	вывести список баз данных с описанием
<code>\dS</code>	вывести системные таблицы
<code>\dv</code>	вывести представления
<code>\dn</code>	вывести все схемы
<code>\du</code>	вывести всех пользователей
<code>\d имя_таблицы</code>	вывести информацию о таблице
<code>\o</code>	пересылка результатов запроса в файл
<code>\di</code>	вывести все индексы
<code>\help</code>	вывести справочник SQL
<code>\i</code>	запуск команды из внешнего файла, например <code>\i /my/directory/my.sql</code>
<code>\?</code>	вывести справочник <i>psql</i>
<code>\q</code>	выход из терминала <i>psql</i>

Если выводимые на экран данные будут превышать допустимые размеры терминала, то произойдёт открытие текстового редактора, в котором будет выведена информация. Для выхода из него нажмите клавишу *q*.

Запросы на языке SQL возможно выполнять из командной строки. **Каждая команда SQL должна заканчиваться символом «;»**. Однако, более удобным и привычным для обучения способом является использование клиентских

приложений. Ниже будет рассмотрена работа с клиентским приложением *pgAdmin*.

## 1.8. Работа с pgAdmin

После запуска программы *pgAdmin* вам будет предложено ввести пароль, указанный при установке программы. Если программа была установлена администратором – спросите пароль у преподавателя.

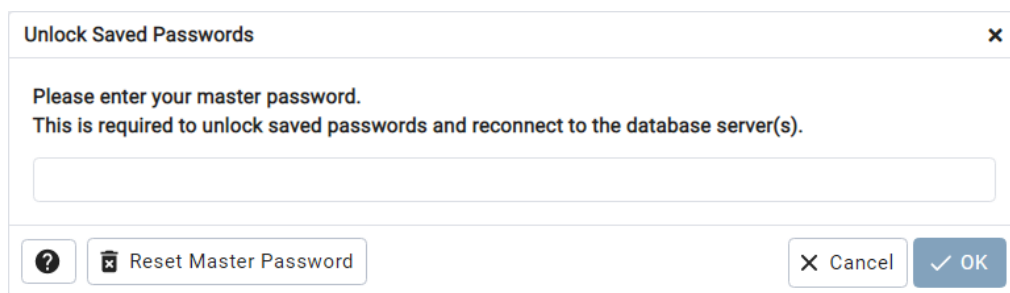


Рисунок 4 — Диалоговое окно с предложением ввода пароля

Основное окно программы выглядит следующим образом:

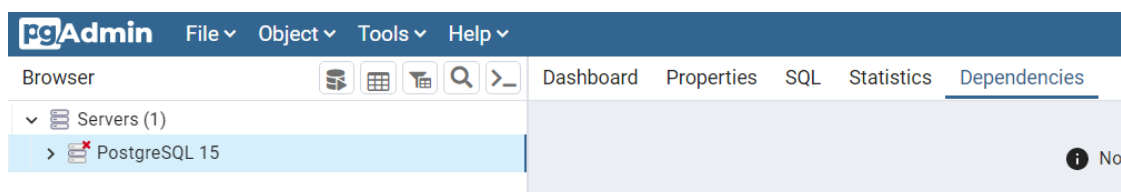


Рисунок 5 — Основное окно pgAdmin

Для подключения к серверу дважды щелкните на название сервера в окошке слева

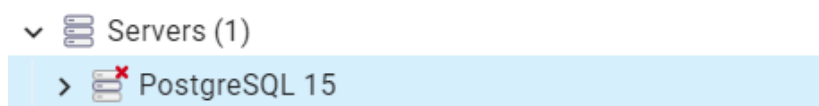


Рисунок 6 — Список серверов pgAdmin

При удачном подсоединении появляются три новые вкладки



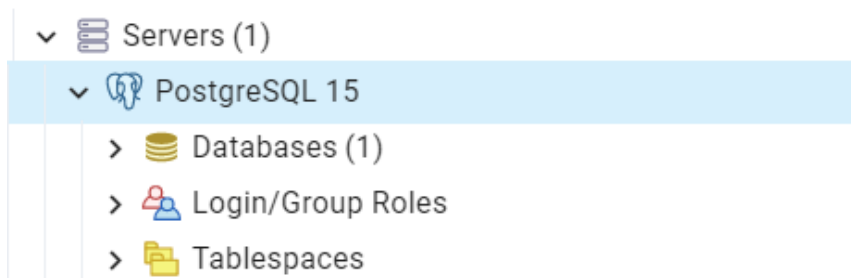


Рисунок 7 — Содержимое PostgreSQL 15

Первая вкладка – Database содержит всю информацию о хранимых базах данных. На текущий момент база данных всего одна – **postgres**

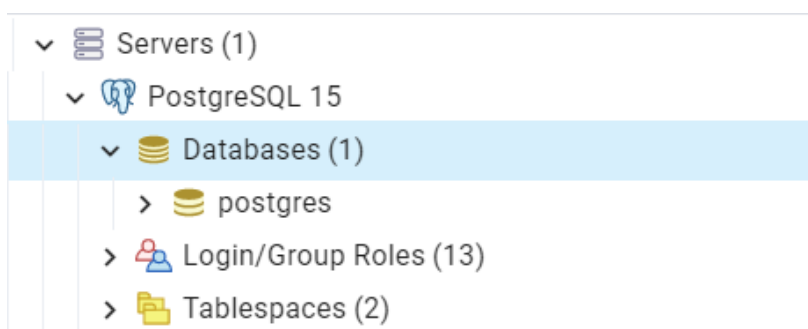


Рисунок 8 — Содержимое вкладки Database

Вторая вкладка – Login/Group Roles. В ней содержатся все созданные роли и группы, в которые данные роли могут входить. Это предназначено для разделения прав пользователей базы данных, например, между администратором и программистом. По умолчанию создана одна роль – *postgres*.

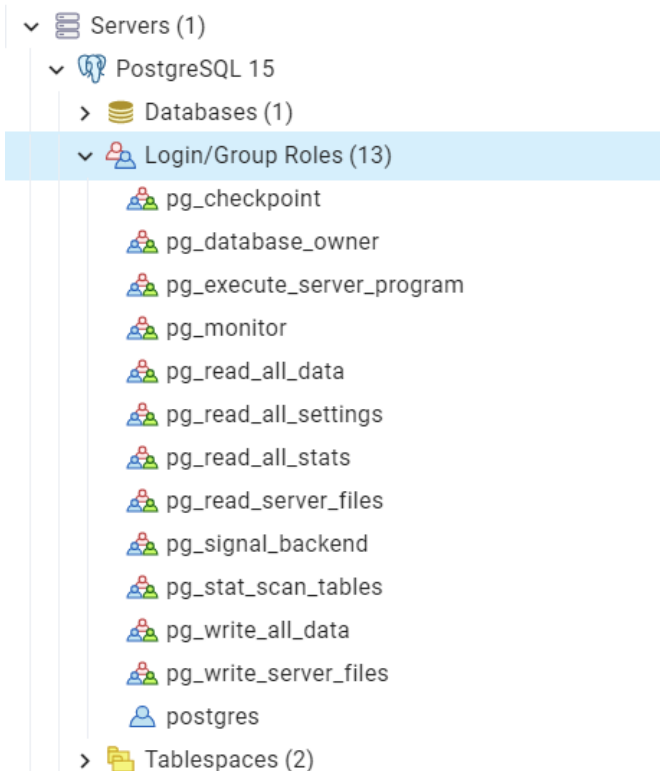


Рисунок 9 — Содержимое вкладки Login/Group Roles

Третья вкладка — Tablespaces. В ней располагаются *табличные пространства*, которые определяют физическое расположение данных. Например, табличные пространства возможно использовать, чтобы расположить архивные данные на медленных носителях, а данные, с которыми идет постоянная работа, на быстрых. При инициализации создается два табличных пространства – *pg\_default*, для хранения данных по умолчанию и *pg\_global* для хранения общих объектов.

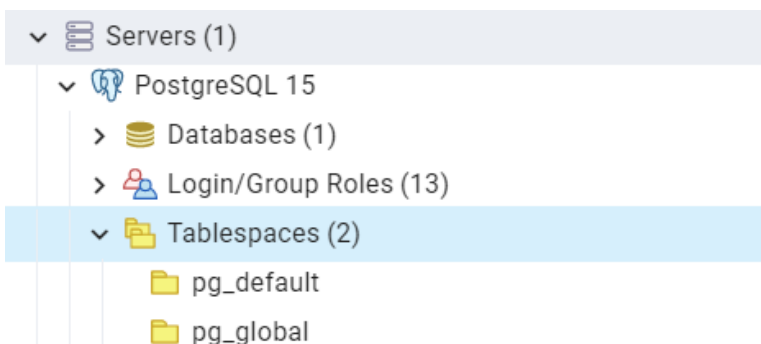



Рисунок 10 — Содержимое вкладки Tablespaces

Для того, чтобы создать запрос на языке SQL в программе *pgAdmin* необходимо воспользоваться утилитой Query tool. Для этого перейдите во вкладку Databases – postgres и нажмите на символ .

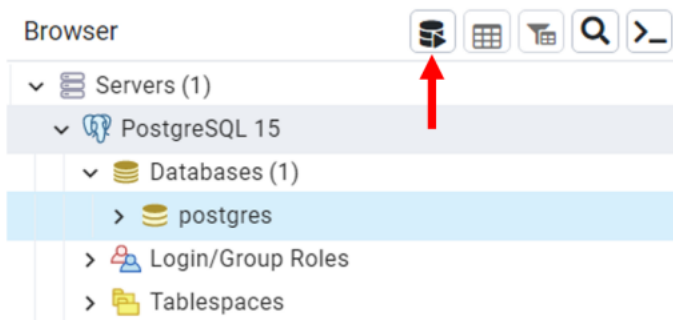


Рисунок 11 — Кнопка создания запроса

Перед вами откроется командное окно, в которое возможно вводить запросы и запускать их на выполнение. Утилита доступна только если курсор установлен на имени БД (в нашем случае **postgres**).

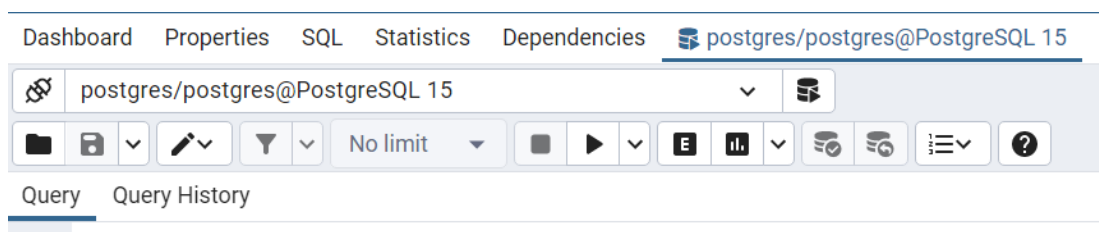


Рисунок 12 — Командное окно pgAdmin

Обратим внимание на строку с подключением. Она записана в формате «база данных/роль@сервер». Для данного примера база данных называется **postgres**, пользователь – **postgres**, сервер – PostgreSQL 15

### 1.9. Создание новой роли.

В СУБД PostgreSQL разграничение доступа реализуется с помощью понятий роли и привилегий.

Каждому пользователю в СУБД назначается роль, обладающая определенными привилегиями. Данный процесс называется авторизацией.

Например, определенной роли возможно выделить привилегию только на чтение данных из таблиц.


Для создания роли используется оператор **CREATE ROLE**

```
CREATE ROLE имя [ [ WITH ] параметр [ ... ] ]
```

Здесь параметр:

```
SUPERUSER | NOSUPERUSER  
| CREATEDB | NOCREATEDB  
| CREATEROLE | NOCREATEROLE  
| INHERIT | NOINHERIT  
| LOGIN | NOLOGIN  
| REPLICATION | NOREPLICATION  
| BYPASSRLS | NOBYPASSRLS  
| CONNECTION LIMIT предел_подключений  
| [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'пароль'  
| VALID UNTIL 'дата_время'  
| IN ROLE имя_роли [, ...]  
| IN GROUP имя_роли [, ...]  
| ROLE имя_роли [, ...]  
| ADMIN имя_роли [, ...]  
| USER имя_роли [, ...]  
| SYSID uid
```

Подробно о каждом из параметров возможно прочитать в приложении к документации PostgreSQL [2]

Создадим роль, название которой будет содержать ваши инициалы и наделим её правами администратора. Для этого скопируем следующий скрипт в рабочую область и запустим его с помощью символа . Имя пользователя и пароль должны быть выбраны вами.

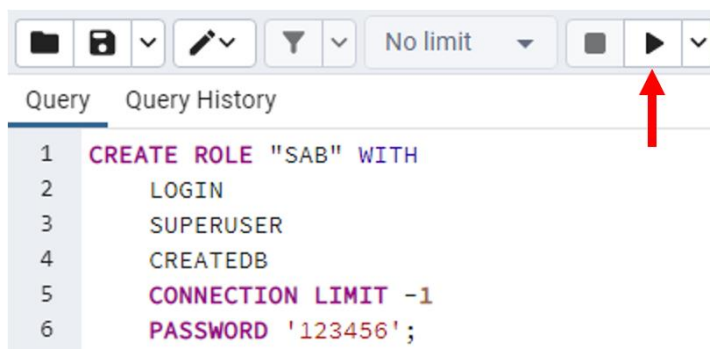


Рисунок 13 — Пример создания роли SAB

```
CREATE ROLE "SAB" WITH
    LOGIN
    SUPERUSER
    CREATEDB
    CONNECTION LIMIT -1
    PASSWORD '123456';
```

Данный скрипт создает роль SAB, наделяет её привилегиями на вход (параметр LOGIN), создание базы данных (CREATEDB) и делает её суперпользователем (SUPERUSER). Созданный пользователь может подключаться неограниченное число раз (CONNECTION LIMIT -1) и имеет пароль для входа 123456.

Обратите внимание, что после успешного выполнения запроса в поле Messages появилась информация об этом.

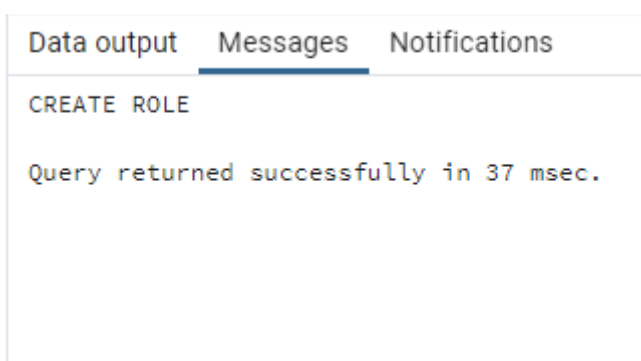


Рисунок 14 — Сообщение об успешном выполнении запроса

### 1.9.1. Работа с базами данных

Перейдем непосредственно к работе с базой данных. Описание предметной области учебной базы данных расположено в приложении.

### 1.9.2. Создание базы данных

Для создания БД служит команда CREATE DATABASE. Её синтаксис представлен ниже.

**CREATE DATABASE** *имя*

```
[ [ WITH ] [ OWNER [=] имя_пользователя ]  
  [ TEMPLATE [=] шаблон ]  
  [ ENCODING [=] кодировка ]  
  [ LC_COLLATE [=] категория_сортировки ]  
  [ LC_STYPE [=] категория_типов_символов ]  
  [ TABLESPACE [=] табл_пространство ]  
  [ ALLOW_CONNECTIONS [=] разр_подключения ]  
  [ CONNECTION LIMIT [=] предел_подключений ]  
  [ IS_TEMPLATE [=] это_шаблон ] ]
```

Создадим учебную базу данных с информацией о студентах вуза. Для этого выполним запрос:

**CREATE DATABASE** *students*;

Напомним, этот запрос возможно выполнять как из командной строки PostgreSQL, так и с помощью программы *pgAdmin*.

Обновим информацию о базах данных на сервере. Для этого щелкнем по строке Databases в левой колонке правой кнопкой мыши и выберем пункт *Refresh*.

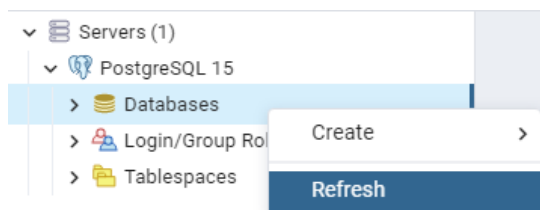


Рисунок 15 — Контекстное меню Databases

Раскроем выпадающий список и убедимся, что появилась новая база данных.

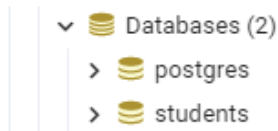


Рисунок 16 — Обновленный список Databases

Нажмем на созданную базу данных и войдем в Query tool. Обратите внимание, что теперь произведено подключение к базе данных *students*.

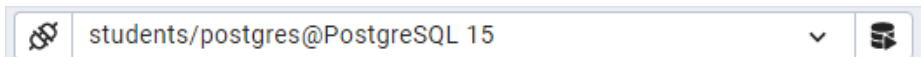


Рисунок 17 — Строка для подключения к БД

### 1.9.3. Смена роли пользователя

Для подключения к базе данных от имени другой роли откройте новое окно Query tool, нажмите на строку с соединением и в выпавшем окне выберите строку <New Connection>

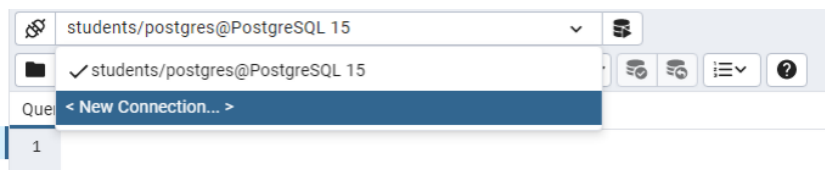


Рисунок 18 — Кнопка для создания нового соединения

Создадим новое соединение

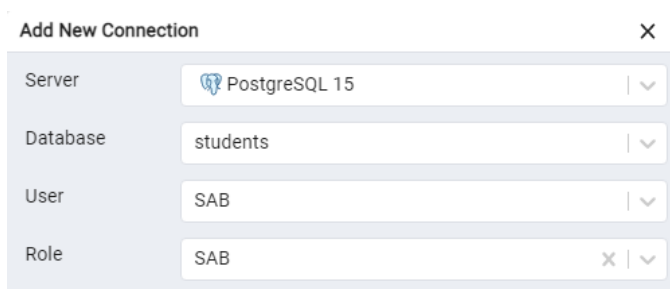
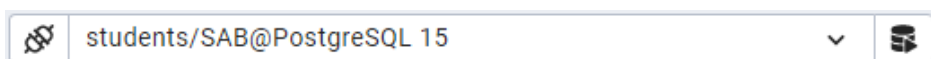


Рисунок 19 — Окно создания нового соединения

После успешного подключения обратим внимание на то, что имя пользователя изменилось.



Обратите внимание, что от имени выбранного пользователя будут выполняться только команды, запущенные в открытом окне. В других окнах по умолчанию будет использоваться роль, под которой было осуществлено подключение к серверу баз данных. В нашем случае это роль *postgres*.

#### 1.9.4. Работа с таблицами в базе данных

Рассмотрим простейшие действия с таблицами. В выпадающем списке найдем созданные таблицы по пути: «Servers – PostgreSQL 15 – students – Schemas – public – Tables».

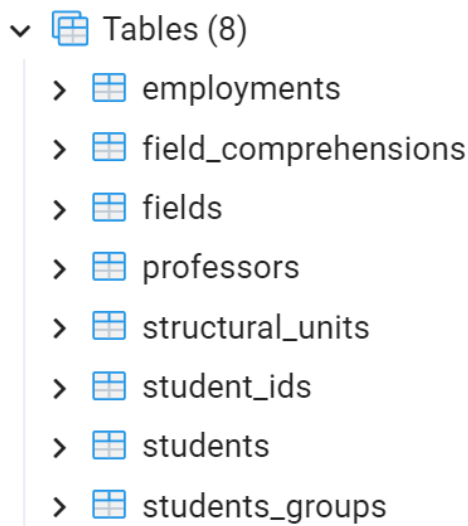


Рисунок 21 — Содержимое БД students

Нажав правой кнопкой мыши на название таблицы, мы можем выбрать некоторые действия. Рассмотрим некоторые из них.

- Count Rows возвращает число строк в таблице.
- View/Edit Data позволяет вывести на экран содержимое таблицы.
  - All rows – выводит все строки таблицы
  - Filtered rows – выводит строки, удовлетворяющие определенному условию. Например, для поиска всех студентов, чье имя – Иван – введите следующее условие: `first_name='Иван'`

Дважды щелкнув по любой ячейке таблицы, возможно изменить её значение.



После внесенных изменений необходимо зафиксировать их, нажав на символ



или клавишу F6.

### 1.9.5. Работа с ERD моделью базы данных

Для графического представления схемы базы данных используется диаграмма «Сущность-связь» – ER модель. Чтобы её открыть в программе pgAdmin необходимо в левом столбце правой кнопкой мыши нажать на название базы данных и выбрать пункт ERD for Database.



Рисунок 22 — Графическое представление схемы БД

## 2. Практическая часть

Вариант к практической части выбирается по формуле:  $V = (N \% 10) + 1$ , где N – номер в списке группы, % – остаток от деления.

## **2.1. Задание 1.**

Создание базы данных

2.1.1. Создайте учебную базу данных Students. Для этого необходимо войти в учетную запись postgres и подключиться к программе psql.

2.1.2. Выйдите из программы psql и заполните базу данных, используя файл резервной копии.

2.1.3. Используя программу pgAdmin, ознакомьтесь со схемой данных, содержимым таблиц БД. Определите число строк в каждой из таблиц.

2.1.4. Определите, какие таблицы в базе данных Students являются главными, а какие для них подчиненными.

## **2.2. Задание 2.**

Администрирование СУБД

2.2.1. Подключитесь к созданной базе данных Students из-под командной строки. Определите, какой размер на диске занимает таблица student?

2.2.2. Создайте новую роль «Ваши инициалы junior». Выделите ей привилегии на вход и установите пароль «654321». Подключитесь от её имени к базе данных students и попробуйте удалить её с помощью запроса:

```
DROP DATABASE students;
```

Удалось ли вам это сделать?

## **2.3. Задание 3.**

Редактирование содержимого базы данных

2.3.1. Выполните в соответствии с вариантом задание на изменение содержимого базы данных. Используйте графическое приложение pgAdmin. Номер варианта соответствует номеру задания из списка ниже.

1. Студентка группы ИВТ-32 Барашкова Настасья Филлиповна пересдала экзамен по Основы теории информации и кодирования на 4. Исправьте её оценку.
2. В связи с ошибкой при заполнении документов, студенту Безухову Пьеру Кирилловичу из группы ИТД-12 указали неверно дату рождения. Исправьте её на 12 августа 2004 года.
3. Студентка группы ИТД-32 Ольга Ильинская вышла замуж за своего одногруппника, родившегося 6 февраля, и сменила свою фамилию на его. Выполните соответствующее изменение в базе данных.
4. Из-за конфликтов с одногруппниками, Роман Хлудов из группы ИВТ-41 решил перевестись в группу ИВТ-42. Выполните данное изменение.
5. Преподаватель, чье имя совпадает с именем беллетриста из пьесы Чехова повысили ставку на 20% и перевели в институт МПСУ. Исправьте значение в базе данных.
6. Добавьте кафедре маркетинга сокращенное название – МИУП.
7. Измените ЗЕТ дисциплины, которую читает Михаил Астров на 3.
8. Почта студента, родившегося в 1999 году, изменилась на oldeststudent@miet.ru. Произведите данное изменение.
9. После провала на второй пересдаче по философии студент Андрей Алехин был отчислен. Удалите его из базы данных
10. У студентки Анны Незвановой был обнаружен пропавший после её рождения отец. Добавьте ей отчество Александровна.

2.3.2. После внесенных изменений, создайте новую резервную копию базы данных Students.

### **3. Контрольные вопросы**

1. Что такое база данных?

2. Что такое СУБД?
3. Чем потенциальный ключ отличается от первичного ключа? Когда используются внешний и суррогатный ключи.
4. Опишите работу метода аутентификации *ident*. В чем различие между аутентификацией и авторизацией?
5. Для чего предназначено создание ролей в СУБД?

#### 4. Список использованной литературы

- [1] «Исходный код СУБД postgres,» [В Интернете]. Available: <https://github.com/postgres/postgres>. [Дата обращения: 30 01 2023].
- [2] Документация к PostgreSQL 15.1, 2022.
- [3] Е. Рогов, PostgreSQL изнутри, 1-е ред., Москва: ДМК Пресс, 2023, 662.
- [4] Б. А. Новиков, Е. А. Горшкова и Н. Г. Графеева, Основы технологии баз данных, 2-е ред., Москва: ДМК пресс, 2020, стр. 582.
- [5] Е. П. Моргунов, PostgreSQL. Основы языка SQL, 1-е ред., Санкт-Петербург: БХВ-Петербург, 2018, стр. 336.

## Лабораторная работа №2 «Оператор SELECT языка SQL»

### 1. Теоретическая часть

Основой работы с базами данных является умение правильно составить запрос на фильтрацию данных. Для данной задачи используется оператор SELECT. Ниже представлен сокращенный синтаксис данного оператора. Более подробно можно прочитать в документации [1].

```
SELECT [ ALL | DISTINCT [ ON ( выражение [, ...] ) ] ]  
      [ * | выражение [ [ AS ] имя_результата ] [, ...] ]  
      [ FROM элемент_FROM [, ...] ]  
      [ WHERE условие ]  
      [ GROUP BY элемент_группирования [, ...] ]  
      [ HAVING условие ]  
      [ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] выборка ]  
      [ ORDER BY выражение [ ASC | DESC | USING оператор ] ]  
      [ LIMIT { число | ALL } ]  
      [ OFFSET начало [ ROW | ROWS ] ]
```

Оператор SELECT получает строки из базы данных и возвращает их в виде таблицы результатов запроса. Результат выполнения запроса может оказаться пустым множеством. Ключевые слова указываются строго в порядке, приведенном выше.

Рассмотрим выполнение SQL запроса на фильтрацию данных по шагам.

#### 1.1. Фильтрация строк

Первыми и обязательными элементами запроса являются ключевые слова SELECT и FROM. После ключевого слова SELECT указывается список столбцов, значения которых необходимо вывести. Возвращаемые столбцы могут быть получены непосредственно из базы данных или вычислены во время выполнения запроса. Для краткого обозначения того, что необходимо вывести все столбцы таблицы используют символ \*.

После ключевого слова FROM указывается имя таблицы, из которой будет производиться выборка данных. Данная таблица может быть реальной таблицей из базы данных, или виртуальной, полученной в результате подзапроса.

Рассмотрим пример. *(В результате данного запроса должно быть выведено 163 строчки. Для экономии места, в данном лабораторном практикуме будет оставлено не более 8 строк от результата любого запроса)*

**SELECT \***

**FROM students;**

student_id	last_name	first_name	patronymic	student..number	birthday	email
847516	Блюм	Андрей	Антонович	ИВТ-11	2005-09-01	BljumAndrej@miet.ru
813156	Верховенский	Петр	Степанович	ИВТ-11	2005-12-15	VerhovenskiyPetr@miet.ru
844688	Верховенский	Степан	Трофимович	ИВТ-11	2004-11-18	VerhovenskiyStepan@miet.ru
862231	Виргинская	Арина	Прохоровна	ИВТ-11	2004-07-02	VirginskajaArina@miet.ru
816390	Гаганов	Артемий	Павлович	ИВТ-11	2004-09-23	GaganovArtemij@miet.ru
830114	Гаганов	Павел	Павлович	ИВТ-11	2004-06-20	GaganovPavel@miet.ru
827840	Дроздов	Маврикий	Николаевич	ИВТ-11	2005-07-15	DrozdovMavrikiy@miet.ru
834759	Дроздова	Прасковья	Ивановна	ИВТ-11	2005-03-17	DrozdovaPraskovja@miet.ru

Данный запрос возвращает список всех студентов университета. Для того, чтобы вывести только ФИО студентов, необходимо вместо знака \* указать названия требуемых полей.

**SELECT last\_name, first\_name, patronymic**

**FROM students;**

last_name	first_name	patronymic
Блюм	Андрей	Антонович
Верховенский	Петр	Степанович
Верховенский	Степан	Трофимович
Виргинская	Арина	Прохоровна
Гаганов	Артемий	Павлович
Гаганов	Павел	Павлович
Дроздов	Маврикий	Николаевич
Дроздова	Прасковья	Ивановна

Следующий запрос

**SELECT first\_name**

```
FROM students;
```

Выведет все имена студентов, с учетом повторений одинаковых имен. Чтобы их исключить возможно использовать ключевое слово DISTINCT

```
SELECT DISTINCT first_name  
FROM students;
```

Для вывода ограниченного числа строк из базы используется ключевое слово LIMIT. Например, запрос

```
SELECT last_name, first_name, patronymic  
FROM students  
LIMIT 1;
```

Выведет только одну строку из общего числа.

## 1.2. Условия отбора строк

Для наложения условий поиска используется оператор WHERE. В качестве операторов сравнения используются операторы: =, <>, >, >=, <, <=.

Например, чтобы найти всех студентов с именем «Сергей», возможно выполнить следующий запрос:

```
SELECT last_name, first_name, patronymic  
FROM students  
WHERE first_name = 'Сергей';
```

last_name	first_name	patronymic
Липутин	Сергей	Васильевич
Сокольский	Сергей	Петрович
Голубков	Сергей	Павлович
Тальберг	Сергей	Иванович
Дорофеев	Сергей	Евгеньевич
Каренин	Сергей	Александрович
Кознышев	Сергей	Иванович
Паратов	Сергей	Сергеич

Обратите внимание, что сравниваемая строка заключается в одинарные кавычки. Об их использовании подробнее будет рассказано ниже.

Аналогично можно найти всех студентов, у которых номер студенческого билета больше числа 850000.

```
SELECT student_id, last_name, first_name, patronymic
FROM students
WHERE student_id > 850000;
```

last_name	first_name	patronymic
Виргинская	Арина	Прохоровна
Лебядкина	Марья	Тимофеевна
Липутин	Сергей	Васильевич
Ставрогин	Николай	Всеволодович
Ставрогина	Варвара	Петровна
Тушина	Лизавета	Николаевна

Для наложения нескольких условий возможно воспользоваться логическими операторами – AND, OR, NOT.

Например, следующий запрос вернет ФИО студентов, чей день рождения попадает в диапазон с 25 июня 2002 до 25 июня 2003 года включительно.

```
SELECT last_name, first_name, patronymic, birthday
FROM students
WHERE birthday > '25/06/2002' AND birthday < '25/06/2003';
```

last_name	first_name	patronymic	birthday
Прозоров	Андрей	Сергеевич	2003-03-23
Прозорова	Мария	Сергеевна	2003-01-14
Солёный	Василий	Васильевич	2003-01-11
Пралинский	Иван	Ильич	2002-07-17
Зубиков	Аким	Петрович	2003-02-18
Валковский	Петр	Александрович	2003-02-14
Сизобрюхов	Степан	Терентьевич	2002-12-11
Заметов	Александр	Григорьевич	2003-04-04

Аналогичный запрос можно составить, используя ключевое слово BETWEEN.

```
SELECT last_name, first_name, patronymic, birthday
FROM students
```



```
WHERE birthday BETWEEN '25/06/2002' AND '25/06/2003'
```

### 1.3. Проверка на членство во множестве

С помощью ключевого слова IN возможно отобрать только те кортежи, заданный атрибут которых находится в указанном списке.

Например, следующий запрос выводит список студентов, кто родился 20 июня 2005 года или 28 июля 2002 года.

```
SELECT last_name, first_name, birthday
FROM students
WHERE birthday IN ('20/06/2005', '28/07/2002');
```

last_name	first_name	birthday
Карамазова	Аделаида	2005-06-20
Суриков	Иван	2002-07-28
Степанов	Федор	2002-07-28
Пежёнова	Анфиса	2002-07-28

(4 rows)

### 1.4. Поиск с использованием шаблона

Для наложения более сложных условий поиска возможно воспользоваться оператором поиска шаблонов LIKE и регулярными выражениями.

*строка* LIKE *шаблон* [ESCAPE *спеcсимвол*]  
*строка* NOT LIKE *шаблон* [ESCAPE *спеcсимвол*]

Оператор LIKE сравнивает анализируемую строку с заданным шаблоном и в случае совпадения отбирает эту строку. Для построения шаблона используются следующие спецсимволы:

% – любая последовательность символов

\_ – строго один символ

Также возможно использовать регулярные выражения POSIX.

^ – начало строки

\$ – окончание строки

- \* – повторение предыдущего символа любое число раз
- \ – проверка наличия указанного после \ символа
- | – выбор одного из двух вариантов
- ~ – поиск с учетом регистра
- [...] – указание класса символов

При проверке по шаблону `LIKE` всегда рассматривается вся строка. Поэтому, если нужно найти последовательность символов где-то в середине строки, шаблон должен начинаться и заканчиваться знаками процента.

Например, данный запрос позволяет найти всех студентов, чье имя начинается на букву 'А'.

```
SELECT last_name, first_name, patronymic
FROM students
WHERE first_name LIKE 'A%';
```

last_name	first_name	patronymic
Блюм	Андрей	Антонович
Виргинская	Арина	Прохоровна
Гаганов	Артеми	Павлович
Кириллов	Алексей	Нилыч
Лембке	Андрей	Антонович
Телятников	Алексей	null
Вельчанинов	Алексей	Иванович
Лобов	Александр	null
Погорельцев	Александр	Павлович

Данный запрос вернет всех студентов, чье имя состоит из 5 символов и заканчивается на букву "я".

```
SELECT last_name, first_name, patronymic
FROM students
WHERE first_name LIKE '____я';
```

last_name	first_name	patronymic
Лебядкина	Марья	Тимофеевна

Улитина	Софья	Матвеевна
Шатова	Дарья	Павловна
Шатова	Мария	null
Карамазова	Софья	Ивановна
Прозорова	Мария	Сергеевна
Мармеладова	Софья	Семеновна
Епанчина	Аглая	Ивановна

Рассмотрим пример с регулярными выражениями

```
SELECT last_name, first_name, patronymic
FROM students
WHERE last_name ~ '^[ЛМН]';
```

last_name	first_name	patronymic
-----+	-----+	-----
Лебядкин	Игнат	Тимофеевич
Лембеке	Андрей	Антонович
Липутин	Сергей	Васильевич
Макаров	Михаил	Макарович
Миусов	Петр	Александрович
Нелюдов	Николай	Парфенович
Мейер	Карл	Федорович
Незванова	Анна	null

Данный пример вернет всех студентов, чьи фамилии начинаются на буквы Л, М, Н. Здесь символом ^ обозначается, что следующий символ будет первым в строке, а в квадратных скобках указывается список допустимых символов. Аналогично можно записать через диапазон – [Л-Н]. Символ ~ указывает, что сравнение идет с учетом регистра.

Дополним выражение таким образом, чтобы отбираемые строки оканчивались на буквосочетание `ин`.

```
SELECT last_name, first_name, patronymic
FROM students
WHERE last_name ~ '^[ЛМН].*ин$';
```

last_name	first_name	patronymic
-----+	-----+	-----
Лебядк <b>ин</b>	Игнат	Тимофеевич

Липутин	Сергей	Васильевич
Лужин	Петр	Петрович
Мышкин	Лев	Николаевич
Нащокин	Ипполит	Александрович
Маркин	Даниэль	Ильич
Левин	Константин	Дмитриевич

Символом \$ указывается, что предыдущие символы будут стоять в конце строки. Т.к. между ними и начальными символами могут находиться еще любое число символов, то обозначим их с помощью ‘.\*’. В данном случае точка обозначает «любой символ», а звездочка продублирует его от 0 до любого числа раз.

Если убрать символ окончания строки \$, то будет производиться поиск строк, начинающихся на буквы Л, М, Н и содержащие в себе «ин».

```
SELECT last_name, first_name, patronymic
FROM students
WHERE last_name ~ '^[ЛМН].*ин';
```

last_name	first_name	patronymic
-----+-----+-----		
Лебядкин	Игнат	Тимофеевич
Лебядкина	Марья	Тимофеевна
Липутин	Сергей	Васильевич
Мизинчиков	Иван	Иванович
Лужин	Петр	Петрович
Мышкин	Лев	Николаевич
Нащокин	Ипполит	Александрович
Маркин	Даниэль	Ильич
Левин	Константин	Дмитриевич

Более подробно про регулярные выражения можно прочитать в источнике [2].

## 1.5. Вычисляемые столбцы

На языке SQL возможно добавлять к итоговой выборке отдельные столбцы, значения которых будут вычисляться в процессе фильтрации. Для этого, к

отбираемым столбцам после ключевого слова **SELECT** добавляется выражение, которое будет вычисляться для каждой строки.

Например, рассчитаем возраст всех студентов вуза и выведем его вместе с ФИО. Для этого добавим еще один столбец и укажем в нем функцию *age*, позволяющую вычислить разницу между двумя датами. В качестве точки отсчета выберем текущую дату, значение которой можно получить с помощью функции **CURRENT\_DATE**.

```
SELECT last_name, first_name, patronymic, age(CURRENT_DATE, birthday)  
FROM students;
```

last_name	first_name	patronymic	age
Блюм	Андрей	Антонович	18 years 5 mons 22 days
Верховенский	Петр	Степанович	18 years 2 mons 8 days
Верховенский	Степан	Трофимович	19 years 3 mons 5 days
Виргинская	Арина	Прохоровна	19 years 7 mons 21 days
Гаганов	Артемий	Павлович	19 years 5 mons
Гаганов	Павел	Павлович	19 years 8 mons 3 days
Дроздов	Маврикий	Николаевич	18 years 7 mons 8 days
Дроздова	Прасковья	Ивановна	18 years 11 mons 6 days

Для того, чтобы переименовать столбец *age*, воспользуемся ключевым словом **AS**. После него укажем новое название столбца – «Возраст».

```
SELECT last_name, first_name, patronymic, age(CURRENT_DATE, birthday) AS  
"Возраст"  
FROM students;
```

last_name	first_name	patronymic	Возраст
Блюм	Андрей	Антонович	18 years 5 mons 22 days
Верховенский	Петр	Степанович	18 years 2 mons 8 days
Верховенский	Степан	Трофимович	19 years 3 mons 5 days
Виргинская	Арина	Прохоровна	19 years 7 mons 21 days
Гаганов	Артемий	Павлович	19 years 5 mons
Гаганов	Павел	Павлович	19 years 8 mons 3 days
Дроздов	Маврикий	Николаевич	18 years 7 mons 8 days
Дроздова	Прасковья	Ивановна	18 years 11 mons 6 days

## 1.6. Агрегирование и группировка

Для проведения статистических вычислений в SQL существуют агрегатные функции. Данные функции принимают на вход множество значений и возвращают одно.

Основные агрегатные функции:

1. AVG: находит среднее значение
2. COUNT: находит количество строк в запросе
3. SUM: находит сумму значений
4. MIN: находит наименьшее значение
5. MAX: находит наибольшее значение

Например, в данном примере рассчитывается средний оклад всех преподавателей вуза. Т.к. атрибут «Оклад» (*salary*) имеет тип *money*, то для применения агрегатной функции необходимо привести его к числовому значению. Для этого используется конструкция `::numeric`. Более подробно о приведении типов указано в п. 0.

```
SELECT AVG(salary::numeric)::numeric(10,2) AS "Average salary"  
FROM professors;
```

```
Average salary  
-----  
          78214.29  
(1 row)
```

В данном запросе рассчитывается общее число ставок, выделенное на весь вуз

```
SELECT SUM(wage_rate) AS "Общее число ставок"  
FROM employments;
```

```
Общее число ставок  
-----  
          33.60  
(1 row)
```

Однако, если потребуется рассчитать общее число ставок для каждого из направлений, то такой подход не сработает. Перед тем, как применять агрегатную функцию необходимо сгруппировать кортежи по определенному признаку – в данном примере, по номеру направления. Затем необходимо применить функцию SUM к каждой из получившихся групп.

Для группировки строк в SQL служит оператор GROUP BY. Данный оператор распределяет строки, образованные в результате запроса по группам, в которых значения некоторого столбца, по которому происходит группировка, являются одинаковыми. Группировку можно производить как по одному столбцу, так и по нескольким.

Дополним запрос из предыдущего примера. В нем происходит подсчет числа сотрудников в каждом структурном подразделении. Например в структурном подразделении 9 работают два сотрудника.

```
SELECT structural_unit_id AS "Structural unit number", count(*) AS "Number of employees"
FROM employments
GROUP BY structural_unit_id;
```

Structural unit number	Number of employees
11	1
8	3
19	1
4	10
14	4
3	8
17	1
20	1

Рассмотрим еще один пример. В данном случае сумма будет рассчитана для каждого структурного подразделения. В подразделении 9 работают три преподавателя – поэтому их ставки суммировались и результатом стало значение 1.00.

```
SELECT structural_unit_id AS "Structural unit number", SUM(wage_rate) AS "Wage
rate sum"
FROM Employments
GROUP BY structural_unit_id;
```

Structural unit number	Wage rate sum
11	0.35
8	1.75
19	0.35
4	4.25
14	1.40
3	3.65
17	0.35
20	0.35

Для фильтрации строк перед группировкой использовалось ключевое слово WHERE. В случае, если нужно отфильтровать строки после неё – используется ключевое слово HAVING.

Добавим в приведенный выше пример условие, чтобы число сотрудников выводилось только для подразделений, в которых более 2х преподавателей.

```
SELECT structural_unit_id AS "Structural unit number", count(*) AS "Number of
employees"
FROM employments
GROUP BY structural_unit_id
HAVING count(*) > 2;
```

Structural unit number	Number of employees
8	3
4	10
14	4
3	8
1	19
5	5
2	6
6	3



## 1.7. Сортировка

Для сортировки результата запроса необходимо использовать ключевое слово ORDER BY. После него указывается атрибуты, по которым производится сортировка. Далее указывается порядок с помощью слов DESC (в порядке убывания) и ASC (в порядке возрастания). По умолчанию строки сортируются по возрастанию, поэтому ASC можно опускать.

```
SELECT structural_unit_id AS "Номер структурного подразделения", count(*) AS  
"Число сотрудников"  
FROM employments  
GROUP BY structural_unit_id  
ORDER BY structural_unit_id;
```

Номер структурного подразделения | Число сотрудников

-----+-----	
1	19
2	6
3	8
4	10
5	5
6	3
7	1
8	3

```
SELECT structural_unit_id AS "Номер структурного подразделения", count(*) AS  
"Число сотрудников"  
FROM employments  
GROUP BY structural_unit_id  
ORDER BY structural_unit_id DESC;
```

Номер структурного подразделения | Число сотрудников

-----+-----	
20	1
19	1
18	1
17	1

16	1
15	1
14	4
13	1

## 1.8. Оконные функции

При составлении запросов с использованием агрегатных функций применялась группировка. Все строки с одинаковыми значениями, указанными после GROUP BY объединялись в одну и над каждой из данных групп совершалось определенное действие. Таким образом, число строк в результирующей таблице уменьшалось. Однако в некоторых случаях бывает полезным провести вычисления над группой и добавить вычисленное значение в качестве дополнительного столбца для каждой строки таблицы. Например, необходимо вывести студентов всех групп и пронумеровать в алфавитном порядке внутри каждой группы. Для этого можно воспользоваться оконными функциями.

В общем виде оконные функции выглядят следующим образом:

```
SELECT
Название функции (столбец для вычислений)
OVER (
PARTITION BY столбец для группировки
ORDER BY столбец для сортировки
)
```

В качестве функции может выступать одна из агрегатных функций (SUM, COUNT, AVG, MIN, MAX) или специальные функции, предназначенные для оконных вычислений.

Приведем некоторые из них:

**ROW\_NUMBER** – данная функция возвращает номер строки и используется для нумерации;

**FIRST\_VALUE** или **LAST\_VALUE** — с помощью функции можно получить первое и последнее значение в окне. В качестве параметра принимает столбец, значение которого необходимо вернуть;

**CUME\_DIST** — вычисляет интегральное распределение (относительное положение) значений в окне;

После ключевых слов **PARTITION BY** необходимо указать поле, по которому будет производиться объединение в группы. Далее возможно отсортировать значения внутри каждой из групп. [3]

В итоге запрос для вычисления порядкового номера студента будет выглядеть следующим образом:

```
SELECT row_number() OVER (  
                                partition by students_group_number ORDER BY last_name),  
       last_name, first_name, patronymic, students_group_number  
FROM students;
```

row_number	last_name	first_name	patronymic	students_group_number
1	Беляков	Иван	Константинович	ИБ-11
2	Быков	Лев	Михайлович	ИБ-11
3	Виноградов	Иван	Макарович	ИБ-11
4	Волков	Кирилл	Денисович	ИБ-11
5	Григорьев	Егор	Матвеевич	ИБ-11
6	Губанов	Артём	Андреевич	ИБ-11
7	Денисов	Леонид	Маркович	ИБ-11
8	Еремин	Лука	Львович	ИБ-11

## 1.9. Несколько важных замечаний

### 1.9.1. Экранирование кавычек

При использовании текстовых строк в запросах их необходимо обрамлять одинарными кавычками. Как, например, в следующем запросе:

```
SELECT last_name, first_name  
FROM students
```

```
WHERE first_name = 'Анна'
```

Однако, как поступить, в случае наличия в самой строке символа кавычки. Например, в институт поступил иностранный студент с фамилией O'Brien.

```
SELECT last_name, first_name
FROM students
WHERE last_name = 'O'Brien'
```

Данный запрос выполнен не будет, т.к. СУБД распознает как строку только символ 'O'. Чтобы одинарная кавычка воспринималась как часть строки, а не указание на её окончание, в PostgreSQL предусмотрена возможность дублирования данной кавычки.

```
SELECT last_name, first_name
FROM students
WHERE last_name = 'O' 'Brien'
```

Способ, указанный выше, сработает.

### 1.9.2. Типы данных в PostgreSQL

PostgreSQL предоставляет пользователям большой выбор встроенных типов данных. В таблице Таблица приведены основные из них. Более подробно можно прочитать в Главе 8 документации PostgreSQL. [1]

Таблица 1 — Типы данных PostgreSQL

Имя	Описание
bigint	знаковое целое из 8 байт
boolean	логическое значение (true/false)
date	календарная дата (год, месяц, день)
integer	знаковое четырёхбайтное целое
json	текстовые данные JSON
money	денежная сумма
numeric [ (p, s) ]	вещественное число заданной точности
real	число одинарной точности с плавающей точкой (4 байта)

Продолжение Таблицы 1

smallint	знаковое двухбайтное целое
serial	четырёхбайтное целое с автоувеличением
text	символьная строка переменной длины
time [ (p) ] [ without time zone ]	время суток (без часового пояса)
uuid	универсальный уникальный идентификатор
varchar(n)	строка ограниченной переменной длины

К основным целым числовым типам можно отнести типы smallint(2 байта), integer (4 байта), bigint (8 байт). В большинстве случаев рекомендуется использовать тип integer.

Для хранения дробных чисел произвольной точности возможно использовать тип numeric.

**NUMERIC(точность, масштаб)**

где точность – число значащих цифр (по обе стороны запятой), масштаб – число цифр после запятой. По умолчанию, тип *numeric* может хранить числа, содержащие до 1000 значащих цифр.

Для хранения чисел с плавающей точкой используются типы real и double. Обратим внимание, что при арифметике чисел с плавающей точкой возможны ошибки округления и неточности при хранении данных, поэтому при необходимости точности вычислений рекомендуется использовать тип *numeric*. Данная проблема может стать критичной при хранении денежных сумм. Поэтому в случае хранения денежных единиц рекомендуется использовать тип *numeric* или *money*.

Для хранения строковых типов ограниченного размера рекомендуется использовать тип *varchar(N)*, где N – максимальный размер строки. При необходимости хранения длинного текста возможно применять тип *text*.

Отдельно остановимся на типе, хранящим значения даты и времени. В PostgreSQL существует множество подобных типов. Наиболее популярными

являются типы *date* и *time*. Все даты в PostgreSQL считаются по Григорианскому календарю, даже для времени до его введения.

Данные о дате могут храниться разными способами. В соответствии со стандартом ISO 8601, рекомендуется хранить дату в формате «год-месяц-день». Например, 1998-08-26. Также возможно использовать следующую запись: месяц/день/год в режиме MDY или день/месяц/год в режиме DMY. Для проверки, в каком режиме работает PostgreSQL возможно выполнить команду:

```
SELECT current_setting('datestyle');
```

```
current_setting
-----
ISO, DMY
(1 row)
```

Для переключения между режимами возможно использовать команду:

```
SET datestyle TO "ISO, DMY";
```

Для работы со значениями типа дата и время существует набор функций, встроенных в PostgreSQL. Приведем некоторые из них.

Таблица 2 — Набор нескольких функций PostgreSQL

Функция	Описание	Пример
<code>extract(<i>field</i> from timestamp)</code>	Извлечение полей из даты	<pre>SELECT extract(month from '2024-02-23'::date)</pre>
<code>age(timestamp, [timestamp])</code>	Вычисление временного интервала между датами.	<pre>SELECT age('2024-02-22'::date, '2024-01-22'::date)</pre>

	При указании одного аргумента, вычисляется интервал между ним и текущей датой.	
--	--	--

В качестве первичного ключа очень часто используют последовательность из чисел от 1 и выше. Для хранения подобной последовательности используется тип *serial*. При создании переменной данного типа создается автоинкрементирующийся счетчик, увеличивающийся с добавлением новой записи. При вызове команды INSERT поле с данным типом возможно отметить значением DEFAULT или просто оставить пустым.

В некоторых случаях необходимо, чтобы первичным ключом служил уникальный идентификатор, не повторяющийся ни в одной другой базе данных. Это может быть полезно, при объединении двух таблиц из различных баз данных. Таким идентификатором может служить GUID – глобальный уникальный идентификатор. Для его хранения в PostgreSQL используется тип *uuid*. UUID записывается в виде последовательности шестнадцатеричных цифр, разделённых знаками минуса на несколько групп: группа из 8 цифр, три группы из 4 цифр, группа из 12 цифр, что в сумме составляет 32 цифры и представляет 128 бит.

Пример *uuid*: 47a8229b-9e8a-0473-fd20-21c889da75bf

PostgreSQL позволяет хранить данные в одной ячейке в виде массива. Для его объявления необходимо после ключевого слова, обозначающего тип элемента добавить квадратные скобки. Например, *int[]*. Размерность массива указывается в виде целого числа внутри скобок. Для объявления одномерных массивов возможно использовать ключевое слово `ARRAY`.

```
SELECT ARRAY[1,2,3];
```

```
array
-----
{1,2,3}
(1 row)
```

Обращение к элементам массива происходит с помощью оператора [].

### 1.9.3. Преобразование типов

Преобразование типов в PostgreSQL возможно выполнить несколькими способами. Самый простой из них – напрямую указать тип данных через символ “::”

```
SELECT 1234::int;
```

```
int4
-----
1234
(1 row)
```

```
SELECT 1234::text;
```

```
text
-----
1234
(1 row)
```

Аналогичного результата можно добиться, используя оператор CAST

```
SELECT CAST(1234 AS int);
```

```
int4
-----
1234
(1 row)
```

```
SELECT CAST(1234 AS text);
```

```
text
-----
1234
(1 row)
```



### 1.9.4. Оператор CASE

В SQL возможно создавать конструкции, подобные SWITCH – CASE из языка программирования С. Общий синтаксис выглядит следующим образом:

```
CASE WHEN условие THEN результат
      [WHEN ...]
      [ELSE результат]
END
```

Если условие оказывается верным, то значением выражения CASE становится результат, идущий после ключевого слова THEN. Если не выполняется ни одно из указанных условий, то результатом выражения будет значение после ветви ELSE или NULL при её отсутствии.

Рассмотрим пример, в котором каждой оценке сопоставляется словесная запись.

```
SELECT mark, CASE
              WHEN mark=5 THEN 'Отлично'
              WHEN mark=4 THEN 'Хорошо'
              WHEN mark=3 THEN 'Удовлетворительно'
              WHEN mark=2 THEN 'Неудовлетворительно'
            END
FROM field_comprehensions
```

```
mark |      case
-----+-----
  3 | Удовлетворительно
  2 | Неудовлетворительно
  5 | Отлично
  4 | Хорошо
```

### 1.9.5. Некоторые полезные функции

Для выполнения практических заданий лабораторных работ вам могут потребоваться некоторые функции. Приведем их в виде таблицы.

Таблица 3 — Набор функций PostgreSQL

Функция	Описание	Пример
<code>overlay(string placing string from int [for int])</code>	Заменяет подстроку	<pre>SELECT overlay('Беляков' placing 'улан' from 2 for 4);</pre> <pre>overlay ----- Буланов (1 row)</pre>
<code>char_length(string)</code>	Число символов в строке	<pre>SELECT char_length('Базы данных');</pre> <pre>char_length ----- 11 (1 row)</pre>
<code>substring(string [from int] [for int])</code>	Извлекает подстроку	<pre>SELECT substring('Базы данных' from 1 for 4);</pre> <pre>substring ----- Базы (1 row)</pre>
<code>position(substring in string)</code>	Положение указанной подстроки	<pre>SELECT position('данных' in 'Базы данных');</pre> <pre>position ----- 6 (1 row)</pre>
<code>left(str text, n int)</code>	Возвращает первые n символов строке.	<pre>SELECT left('Базы данных', 4);</pre> <pre>left ----- Базы (1 row)</pre>
<code>right(str text, n int)</code>	Возвращает последние n символов строке.	<pre>SELECT right('Базы данных', 6);</pre> <pre>right ----- данных (1 row)</pre>

Продолжение таблицы 3

<code>round(x numeric)</code>	Округление ближайшего числа	ДО  <code>SELECT round(9.5) ;</code>  round ----- 10 (1 row)
<code>mod(y int, x int)</code>	Остаток от деления y/x	<code>SELECT mod(9,4) ;</code>  mod ----- 1  (1 row)
<code>random()</code>	Генерация случайного числа в диапазоне $0.0 \leq x < 1.0$	<code>SELECT random() ;</code>  Random ----- 0.9942907746544045 (1 row)
<code>ascii(string)</code>	Возвращает ASCII-код первого символа аргумента.	<code>SELECT ascii('x') ;</code>  ascii ----- 120 (1 row)
<code>chr(int)</code>	Возвращает СИМВОЛ данным кодом.	<code>SELECT chr(120) ;</code>  chr ----- x (1 row)

## 2. Практическая часть

Вариант к практической части выбирается по формуле:  $V = (N \% 10) + 1$ , где N – номер в списке группы, % – остаток от деления.

### 2.1. Задание 1

### *Исследование типов данных*

Предположим, что в магазине новый конструктор стоит 999 рублей и 99 копеек. Студент С. решил приобрести для дальнейшей перепродажи 100000 таких товаров. Для расчета общей суммы, которую необходимо заплатить был создан следующий скрипт на языке PL/pgSQL. Более подробно о нём будет рассказано в одной из следующих лабораторных работ. Обратите внимание, что значение суммы имеет тип *real*.

```
DO
$$
DECLARE
    summ real :=0.0;
BEGIN
FOR i IN 1..100000 LOOP
    summ := summ + 999.99;
END LOOP;
RAISE NOTICE 'Summ = %;', summ;
--RAISE NOTICE 'Diff = %;', 99999000.00 - summ;
END
$$ language plpgsql;
```

Запустите скрипт. Раскомментируйте строку с вычислением разницы и определите, сколько денег переплатил студент С? Объясните полученный результат. Измените тип суммы на *numeric* и *money*. Какой результат был получен в обоих случаях?

## **2.2. Задание 2**

### *Написание запросов на языке SQL*

Напишите SQL запросы к учебной базе данных в соответствии с вариантом. Запросы брать из сборник запросов к учебной базе данных, расположенного ниже

Таблица 4 — Список вариантов

№ варианта	№ запросов
1	1, 11, 21, 31, 41, 51, 61, 71
2	2, 12, 22, 32, 42, 52, 62, 72
3	3, 13, 23, 33, 43, 53, 63, 73
4	4, 14, 24, 34, 44, 54, 64, 74
5	5, 15, 25, 35, 45, 55, 65, 75
6	6, 16, 26, 36, 46, 56, 66, 76
7	7, 17, 27, 37, 47, 57, 67, 77
8	8, 18, 28, 38, 48, 58, 68, 78
9	9, 19, 29, 39, 49, 59, 69, 79
10	10, 20, 30, 40, 50, 60, 70, 80

### 2.3. Задание 3

Самостоятельно разработайте **7 осмысленных** запросов к базе данных, используя приведенные в данной лабораторной работе материалы.

#### Сборник запросов к учебной базе данных

Условия *WHERE*, *ORDER BY*

1. Вывести всех студентов группы отсортировав по возрасту
2. Вывести возраст студентов группы, отсортировав по номеру студенческого билета
3. Вывести самого младшего студента группы ИВТ-41
4. Вывести самого старшего студента группы ИВТ-42
5. Выведите студентов, у которых дата рождения совпадает с вашей (месяц и день)
6. Вывести почту всех Кириллов, отсортировав их по дате рождения
7. Выведите студентов, которые моложе 20 лет
8. Вывести номер студенческого билета всех Евгениев, отсортировав их по возрасту

9. Вывести почту всех студентов группы ИТД-21, отсортировав их по фамилии обучающихся
10. Вывести фамилии всех студентов, чей студенческий лежит в интервале от 820000 до 830000
11. Вывести список всех предметов, отсортировав по уникальному Id
12. Вывести должности всех преподавателей, чей оклад выше 100000, отсортировать по размеру оклада
13. Вывести всех студентов, родившихся зимой и весной
14. Вывести всех студентов, родившихся летом и осенью
15. Вывести всех студентов, родившихся под летними знаками зодиака
16. Вывести всех студентов, родившихся под осенними знаками зодиака
17. Выведите все группы, обучающиеся очно
18. Выведите все группы, обучающиеся заочно
19. Вывести номера студенческих билетов студентов, имеющих двойки.
20. Вывести фамилию и академические звание преподавателя с максимальным стажем

#### *Группировка GROUP BY*

21. Подсчитать количество двоечников, троечников, хорошистов и отличников среди студентов.
22. Выведите количество студентов обучающихся в каждой группе и отсортируйте их по количеству
23. Выведите количество студентов, которые обучаются на третьем курсе.
24. Выведите коды дисциплин, по которым зачёт получило больше 100 студентов
25. Вывести средний оклад преподавателей по должностям в порядке возрастания, включить в расчет только тех у кого оклад больше 40000
26. Посчитать число преподавателей, имеющих разные ученые степени, отсортировать их по количеству, оставив только те строчки, где число больше 2
27. Найти среднюю оценку каждого студента по всем дисциплинам, отсортировать записи по оценке и дисциплине, оставив только тех, у кого средняя оценка больше 4

28. Вывести неуникальные фамилии и их количество для групп ИБ, отсортировать по алфавиту. Рядом вывести количество таких фамилий, оставив только тех, которых больше 2
29. Вывести список студентов (номера студенческих билетов) и число их оценок больших, чем 4, отсортировать по числу оценок в порядке убывания. Оставить только тех, у кого число таких оценок больше 5.
30. Подсчитать количество долгов у студентов, вывести номер студенческого билета и количество долгов по всем дисциплинам, отсортировав их по номеру студенческого. В вывод включить только тех, у кого долгов больше 10.
31. Подсчитать количество должников по каждой дисциплине и вывести на экран код дисциплины и количество должников, превышающих 50 человек.
32. Вывести число учащихся по каждой дисциплине, отсортировать по количеству. Оставить только те числа, где число учащихся больше 100
33. Вывести среднюю оценку учащихся по каждой дисциплине, отсортировать по ней же. Оставить только средние оценки больше 3.5
34. Вывести количество отличных оценок у каждого студента, отсортировать по количеству. Оставить только тех, у кого пятерок больше 10
35. Вывести стаж преподавателя, среднюю зарплату для каждого стажа и количество преподавателей, имеющих этот стаж. При подсчете учитывать только преподавателей, имеющих стаж больше 2 лет и вывести записи, у которых число преподавателей, имеющих данный стаж больше 1
36. Вывести дату рождения (только число) студентов и подсчитать количество студентов, родившихся в один день, отсортировать по возрастанию числа. Оставить только те дни, в которые родилось меньше 15 студентов.
37. Подсчитать количество различных отчеств у девушек студенток, вывести только те записи, где количество отчеств  $> 1$  и отсортировать их в порядке возрастания количества
38. Вывести коды структурных подразделений и количество работающих в них преподавателей, оставив только те в которых больше 1 сотрудника. Отсортировать по количеству.
39. Подсчитать количество каждой оценки у каждого студента, отсортировать по номеру студенческого и оценке. Вывести номер студенческого билета, оценку и ее количество, оставив билеты из диапазона 820000–840000.

40. Вывести номера студенческих билетов, среднюю оценку студента по всем дисциплинам и количество оценок. В подсчет включить только студентов с четными номерами студенческих., заполненными полями оценок и вывести записи, у которых средняя оценка лежит в диапазоне 3,2 – 3,6. Среднюю оценку округлить до сотых.

#### *Регулярные выражения*

41. Вывести весь 3-й курс ИБ, отсортировать по возрасту
42. Вывести весь 2-й курс ИВТ, отсортировать по фамилии
43. Вывести весь 4-й курс ИТД, отсортировать по фамилии, имени и отчеству
44. Вывести студентов с почтой, начинающейся на латинскую букву А, отсортировать по имени студентов
45. Вывести студентов, у которых ВНУТРИ имени почты содержится латинская буква А, отсортировать по фамилиям в обратном порядке
46. Вывести всех студентов, название почты которых начинается с букв из диапазона А-Г, отсортировать по студенческому билету
47. Вывести всех студентов ИБ с именем Андрей
48. Вывести всех студентов ИВТ чьи фамилии начинаются на буквы из диапазона А-Г, отсортировать по фамилии
49. Вывести всех студентов ИБ с отчеством, заканчивающимся на «нович» или «ловна»
50. Вывести всех студентов, у которых номер студенческого начинается на 8 и заканчивается на 1, отсортировать по нему же
51. Вывести всех студентов, родившихся зимой, используя регулярные выражения
52. Вывести всех студентов ИТД с фамилией, заканчивающейся на -ин/ов/ев, отсортировать по фамилии
53. Вывести всех студентов вечерних отделений
54. Выведите всех студентов, обучающихся на 2 курсе, исключая номер группы
55. Вывести всех студентов третьего курса, исключая группы ИБ в порядке убывания номеров студенческих билетов
56. Выведите количество по группам студентов из 3 курса, чьи фамилии не заканчиваются на 'а'. Вывести группу и количество.



57. Была потеряна контрольная работа студента с инициалами В.Ф. Необходимо определить фамилию, группу, в которой обучается студент и вернуть работу.
58. Выведите количество групп в каждом подразделении с очной формой обучения, исключая 4 подразделение.
59. Посчитать количество групп 4-го курса
60. Посчитать всех Михайловичей, отсортировать по фамилиям в алфавитном порядке. Оставить только тех, у кого в имени есть буква "А" или "а"

*Общее*

61. Выведите фамилии и количество их повторений, которые начинаются на ту же букву, что и ваша фамилия, а две последние буквы фамилии с вашей не совпадают.
62. Вывести всех студентов-тёзок, родившихся в 2004-2005 годах
63. Подсчитать количество групп в каждом структурном подразделении, столбцы назвать "код подразделения" и "количество" соответственно. Исключить из подсчёта вторые группы (оканчивающиеся на 2), отсортировать по коду подразделения.
64. Вывести общую сумму оклада по должностям, преподавателей чей стаж выше трех лет, округлив ее до сотых. Сумма превышающая 500т не выводится.
65. Подсчитать и вывести на экран количество студенческих билетов для каждой даты выдачи. Дайте столбцам русские наименования. Исключите из подсчета билеты заканчивающиеся на '3'.
66. Вывести фамилии и группы всех студентов второго и четвертого курса, исключая вечерников. Дать столбцам русские названия. Отсортировать по фамилиям в обратном порядке.
67. Выведите количество студентов девушек, которые обучаются на третьем курсе.
68. Подсчитать количество студентов обучающихся в каждой группе. В подсчет включить только студентов третьего и четвертого курсов. Вывести только те группы, у которых количество студентов > 20, дать столбцам русские названия. Отсортировать по количеству.
69. SQL запрос показывает сколько студентов получили определенную оценку за все технические дисциплины, не считая те случаи, когда оценку всего получили меньше 10 человек.
70. Найти все дисциплины, которые ведут преподаватели с кодами: 810006, 840010, 805004, 840004. Вывести код подразделения, код преподавателя,

название дисциплины. Отсортировать по коду подразделения и коду преподавателя. Всем столбцам дать русские имена.

71. Подсчитать количество студентов, родившихся раньше '01/01/2004' в каждой группе, вывести названия и количество студентов тех групп, в которых таких студентов больше 20. Отсортировать по количеству. Всем столбцам дать русские имена.

72. Подсчитать количество студентов в каждой группе, при подсчете учитывать только первые группы, включая вечерников. Вывести группы, в которых количество студентов  $>20$ , а также число студентов по группам. Отсортировать по количеству. Всем столбцам дать русские имена.

73. Подсчитать количество студентов в каждой группе имя которых состоит из пяти символов, вывести только те группы, у которых количество таких студентов больше одного. Отсортировать по количеству. Всем столбцам дать русские имена.

74. Вывести всех студентов всех курсов второй группы ИТД, фамилии которых начинаются на букву из диапазона А-Н, убрать повторяющиеся фамилии, отсортировать по фамилии. Всем столбцам дать русские имена.

75. Вывести средний оклад преподавателей по всем должностям, кроме доцента. В подсчёте учитывать только тех у кого оклад  $> 15000$ . Отсортировать по окладу. Всем столбцам дать русские имена. Средний оклад округлить до сотых.

76. Подсчитать количество дисциплин, которые ведет каждый преподаватель, в подсчете учитывать только тех, чей код лежит в диапазоне 810000-850000. Вывести коды преподавателей, которые читают больше одной дисциплины. Отсортировать по количеству. Всем столбцам дать русские имена.

77. Вывести пять самых популярных отчеств у студентов и их количество, Значение null в подсчёт не включать.

78. Подсчитать количество однофамильцев в каждой группе. Вывести фамилию, номер группы и количество таких фамилий. Уникальные фамилии не выводить. Отсортировать по фамилии и группе. Всем столбцам дать русские имена

79. Найти самого младшего студента. у которого нет отчества, и фамилия состоит из шести символов. Вывести всю возможную информацию о нем, в конце добавить столбец с возрастом.

80. Найти студентов, родившихся в високосном году и вывести только тех, у кого на текущий момент прошло меньше месяца со дня рождения. В вывод включить всю информацию о таких студентах и добавить столбец с возрастом. Отсортировать записи по возрасту студента.

### 3. Контрольные вопросы

1. Как задать условие фильтрации на языке SQL?
2. Сколькими способами возможно произвести поиск с помощью шаблона?
3. Для чего предназначены оконные функции?
4. Чем тип *real* отличается от типа *numeric*?
5. Для чего предназначены *uuid*?

### 4. Список использованной литературы

- [1] Документация к PostgreSQL 15.1, 2022.
- [2] «Regular-expressions,» [В Интернете]. Available: <https://www.regular-expressions.info/tutorial.html>. [Дата обращения: 23 06 2023].
- [3] Р. Романчук, «Учимся применять оконные функции,» [В Интернете]. Available: <http://thisisdata.ru/blog/uchimsya-primenyat-okonnyye-funktsii/>. [Дата обращения: 29 06 2023].
- [4] «Исходный код СУБД postgres,» [В Интернете]. Available: <https://github.com/postgres/postgres>. [Дата обращения: 30 01 2023].
- [5] Е. Рогов, PostgreSQL изнутри, 1-е ред., Москва: ДМК Пресс, 2023, стр. 662.
- [6] Б. А. Новиков, Е. А. Горшкова и Н. Г. Графеева, Основы технологии баз данных, 2-е ред., Москва: ДМК пресс, 2020, стр. 582.
- [7] Е. П. Моргунов, PostgreSQL. Основы языка SQL, 1-е ред., Санкт-Петербург: БХВ-Петербург, 2018, стр. 336.

## Лабораторная работа №3

### «Использование объединяющих и вложенных запросов языка SQL»

#### 1. Теоретическая часть

В предыдущей лабораторной работе были рассмотрены примеры запросов на выборку данных из одной таблицы. Однако, существует большое число задач, когда требуется проанализировать информацию из нескольких таблиц. Для этого существуют операции соединения.

##### 1.1. Соединение таблиц

###### 1.1.1. Неявное соединение таблиц

Самым простым способом является неявное соединение таблиц, когда таблицы объединяются перекрестно. Другими словами, каждой строке одной таблицы будет совмещаться с каждой строкой второй таблицы. В данном случае мы получаем прямое (декартово) произведение двух таблиц.

На языке SQL для неявного соединения необходимо указать требуемые таблицы через запятую после оператора FROM.

Рассмотрим для примера таблицы *Structural\_units*, *Employments*, *Professors* из учебной базы данных. Для упрощения сократим число строк в таблицах, оставив следующие значения.

Таблица *structural\_units*

structural_unit_id	abbreviated_title	phone_number
1	МПСУ	25-13
2	СПИНТех	23-45
3	ИЦОД	66-36

Таблица *professors*

professor_id	last_name
810001	Широков
820001	Воронов
840001	Быкова

Таблица *employments*

structural_unit_id	professor_id	wage_rate
1	810001	0,25
2	820001	0,35

Рисунок 1 — Таблицы БД

Обратите внимание, что к третьему структурному подразделению не прикреплен ни один преподаватель и преподаватель по фамилии Быкова не прикреплена ни к одному подразделению.

Предположим, что нам необходимо вывести код всех преподавателей и название подразделения, в котором они трудоустроены. Для этого необходимо объединить таблицы *Structural\_units* и *Employment*.

Выполним следующий запрос:

```
SELECT abbreviated_title, professor_id
FROM structural_units, employments
```

abbreviated_title	professor_id
МПСУ	810001
СПИНТех	810001
ИЦОД	810001
МПСУ	820001
СПИНТех	820001
ИЦОД	820001

(6 rows)

В результате запроса будут выведены значения из двух таблиц, где каждому значению из первой будут соответствовать все значения из второй таблицы. Таким образом, будет выведено (число строк в первой таблице) \* (число строк во второй таблице). В данном примере будет выведено 6 строк.

В итоговой таблице будут присутствовать все преподаватели и все подразделения независимо от реального трудоустройства. Очевидно, что в этом нет никакого смысла.

Для того, чтобы выведенная информация связывала фамилию преподавателя и только то структурное подразделение, на котором он трудоустроен необходимо добавить условие отбора. Для вывода фамилии преподавателя вместо его кода используем таблицу *Professors*.

Выберем только те строки, где значения номера подразделения в таблицах *Structural\_units* и *Employments*, а также значения номера преподавателя в таблицах *Employments* и *Professors* совпадают.

Итоговый запрос будет выглядеть следующим образом:

```

SELECT structural_units.abbreviated_title, professors.last_name
FROM   structural_units, employments, professors
WHERE  structural_units.structural_unit_id = employments.structural_unit_id
AND    employments.professor_id = professors.professor_id

```

```

abbreviated_title | last_name
-----+-----
МПСУ              | Широков
СПИНТех           | Воронов
(2 rows)

```

Этот результат несет реальную смысловую нагрузку. Как можно заметить из результата запроса, преподаватель Быкова и подразделение ИЦОД в итоговую выборку не попали, т. к. для них указанное условие не было выполнено.

Обратите внимание, что для того, чтобы обращаться к атрибуту конкретной таблицы, необходимо указывать название таблицы, отделив его от имени атрибута точкой (название\_таблицы.имя\_атрибута).

Для сокращения возможно использовать более короткую запись. Для этого после названия таблицы в выражении FROM нужно указать для неё псевдоним. Пример ниже аналогичен запросу, рассмотренному выше, однако выглядит более компактным.

```

SELECT su.abbreviated_title, p.last_name
FROM   structural_units su, employments e, professors p
WHERE  su.structural_unit_id = e.structural_unit_id
AND    e.professor_id = p.professor_id

```

### 1.1.2. Соединение с помощью JOIN

Кроме неявного соединения таблиц в языке SQL существует альтернативная форма записи операций соединения таблиц с помощью ключевого слова JOIN. Объединение происходит по столбцу, который есть в

каждой из таблиц. В результате запроса формируется соединенная таблица. Соединённая таблица — это таблица, полученная из двух других таблиц ( $T_1$ ,  $T_2$ ) в соответствии с условием соединения. Общий синтаксис описания соединённой таблицы:

**T1 тип\_соединения T2 [ условие\_соединения ]**

По типу соединения операторы JOIN подразделяются на внутренние и внешние — INNER JOIN и OUTER JOIN, а также CROSS JOIN.

**Внутреннее соединение – INNER JOIN** используется для отбора строк из двух таблиц, в которых совпадают значения поля, по которому происходит объединение.

Формат:

```
SELECT столбцы FROM таблица1
[INNER] JOIN таблица2    ON условие1
[[INNER] JOIN таблица3 ON условие2]
```

Составим запрос, аналогичный примеру выше, в котором необходимо было вывести фамилии трудоустроенных преподавателей.

```
SELECT su.abbreviated_title, p.last_name
FROM structural_units su
INNER JOIN employments e ON su.structural_unit_id = e.structural_unit_id
INNER JOIN professors p ON e.professor_id = p.professor_id
```

```
abbreviated_title | last_name
-----+-----
МПСУ              | Широков
СПИНТех           | Воронов
(2 rows)
```

Результат выполнения запроса совпал с примером выше.

**Внешнее соединение – OUTER JOIN** позволяет включить в вывод все строки из одной или обеих таблиц. Его можно разделить на правое (**RIGHT**), левое (**LEFT**) и полное (**FULL**).

Формат:

**SELECT столбцы FROM таблица1**

**{ LEFT | RIGHT | FULL } [OUTER] JOIN таблица2 ON условие1**

**[ { LEFT | RIGHT | FULL } [OUTER] JOIN таблица3 ON условие2 ]...**

Операция левого внешнего соединения (**LEFT**) возвращает все строки левой (первой) таблицы, включая те, для которых не нашлось парного значения в правой (второй) таблице. Вместо ненайденных значений атрибутов правой таблицы будет указано неопределенное значение **NULL**. Аналогично, операция правого внешнего соединения (**RIGHT**) возвращает все строки правой (второй) таблицы, включая те, для которых не нашлось пары в левой. Обратите внимание, операции правого и левого соединения не коммутативны: **A LEFT JOIN B**  $\neq$  **B LEFT JOIN A**. Вывод **A LEFT JOIN B** совпадает с **B RIGHT JOIN A**.

Полное внешнее соединение (**FULL**) включает в себя все пересекающиеся строки и все непарные строки из обеих таблиц.

Рассмотрим примеры:

Выберем из таблиц *Structural\_units* и *Employments* значения номеров телефонов всех структурных подразделений и ставку трудоустроенных профессоров.

```
SELECT abbreviated_title, phone_number, professor_id, wage_rate  
FROM structural_units su  
LEFT JOIN employments e ON su.structural_unit_id = e.structural_unit_id;
```

abbreviated_title	phone_number	professor_id	wage_rate
МПСУ	25-13	810001	0.25
СПИНТех	23-45	820001	0.35
ИЦОД	66-36	null	null

(3 rows)



В данном примере была взята вся информация из первой (левой) таблицы и к ней присоединены значения, удовлетворяющие указанному условию. В противном случае проставлено значение *null*.

Рассмотрим второй пример. Выберем фамилии всех преподавателей и ставки тех из них, кто трудоустроен.

```
SELECT wage_rate, last_name
FROM employments e
RIGHT JOIN professors p ON e.professor_id = p.professor_id
```

```
wage_rate | last_name
-----+-----
      0.25 | Широков
      0.35 | Воронов
      null | Быкова
(3 rows)
```

В текущем примере к таблице трудоустройства была присоединена таблица, содержащая преподавателей с использованием RIGHT JOIN. Таким образом была получена вся вторая (правая) таблица и удовлетворяющие условию значения из левой. Аналогично в пустые ячейки проставлено значение *null*.

## Декартово произведение – CROSS JOIN

Формат:

```
SELECT столбцы FROM таблица1
CROSS JOIN таблица2
```

Аналогично неявному соединению, возможно произвести декартово произведение таблиц с помощью оператора JOIN. Для этого существует ключевое слово CROSS.

```
SELECT abbreviated_title, professor_id
FROM structural_units
CROSS JOIN employments
```

```
abbreviated_title | professor_id
-----+-----
МПСУ              |      810001
СПИНТех           |      810001
```

ИЦОД		810001
МПСУ		820001
СПИНТех		820001
ИЦОД		820001

(6 rows)

Ключевые слова OUTER и INNER являются устаревшим и команды LEFT OUTER JOIN, RIGHT OUTER JOIN, INNER JOIN возможно сокращать до LEFT JOIN, RIGHT JOIN и просто JOIN.

Если столбцы, по которым производится соединение, имеют одинаковое название, то вместо записи условия с помощью ключевого слова ON возможно использовать ключевое слово USING.

Формат:

```
SELECT столбцы FROM таблица1
{ [INNER] | { LEFT | RIGHT | FULL } [OUTER] } JOIN таблица2
USING ( список столбцов соединения )
```

В отличие от соединения с использованием ключевого слова ON, USING удаляет в итоговой таблице повторяющийся столбец, по которому происходит соединение. Например запрос, указанный ниже, аналогичен рассмотренному примеру на внутреннее соединение.

```
SELECT abbreviated_title, last_name
FROM structural_units
INNER JOIN employments USING(structural_unit_id)
INNER JOIN professors USING(professor_id)
```

### 1.1.3. Объединение, разность, пересечение таблиц

Кроме соединения таблиц, когда в результате операции атрибуты (столбцы) одной таблицы будут добавлены к атрибутам другой существуют операции *объединения*. В данном случае число атрибутов не изменяется, но в итоговой таблице будут содержаться значения из нескольких таблиц. При объединении таблиц необходимо соблюдать условие, при котором тип данных

каждого столбца первой таблицы должен совпадать с типом данных соответствующего столбца во второй таблице. Также должно совпадать количество выбранных столбцов из всех таблиц. Имена столбцов в объединяемых таблицах не обязательно должны быть одинаковыми.

В языке SQL для объединения таблиц используется оператор UNION.

Формат:

```
SELECT_выражение1
UNION [ALL] SELECT_выражение2
[UNION [ALL] SELECT_выражениеN]
```

Исходные таблицы могут содержать идентичные строки, тогда при объединении по умолчанию повторяющиеся строки удаляются. Если необходимо чтобы общая таблица содержала все строки, включая повторяющиеся используют параметр ALL.

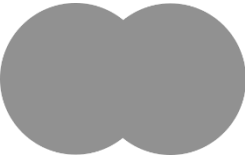
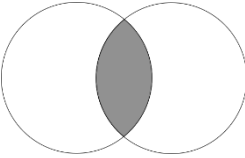
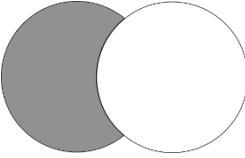
Кроме операции объединения существуют операции *пересечения* (INTERSECT) и *разности* (EXCEPT). Операция пересечения оставляет только общие строки из двух таблиц, а операция разности удаляет из первой таблицы значения, содержащиеся во второй. Данные три операции в языке SQL тесно связаны с логическими операциями булевой алгебры.

Таблица 1 — Операции с таблицами

<i>Операция</i>	<i>SQL операция</i>	<i>Логическая операция</i>
Объединение	UNION	A <b>OR</b> B
Пересечение	INTERSECT	A <b>AND</b> B
Разность	EXCEPT	A <b>AND NOT</b> B

Рассмотрим еще один пример. Вычислим

Таблица 2 — Пример SQL запросов и операций с таблицами

SQL запрос	Возвращаемые значения	Комментарий
<b>SELECT professor_id FROM professors</b> <b>UNION</b> <b>SELECT professor_id FROM employments;</b> 	<pre> professor_id -----       820001       840001       810001 (3 rows) </pre>	Объединение всех кодов преподавателя без повторяющихся значений.
<b>SELECT professor_id FROM professors</b> <b>INTERSECT</b> <b>SELECT professor_id FROM employments;</b> 	<pre> professor_id -----       810001       820001 (2 rows) </pre>	Вывод только трудоустроенных преподавателей
<b>SELECT professor_id FROM professors</b> <b>EXCEPT</b> <b>SELECT professor_id FROM employments</b> 	<pre> professor_id -----       840001 (1 row) </pre>	Вывод нетрудоустроенных преподавателей.

## 1.2. Подзапросы

Результатом выполнения запроса является набор кортежей, оформленный в виде таблицы. Данную таблицу возможно вывести на экран для просмотра или использовать в других запросах. Запрос, используемый внутри другого запроса, называют вложенным запросом или подзапросом.

Существуют два типа подзапросов:

**Некоррелированный подзапрос** – оператор SELECT вложенный в другой запрос SQL, не связанный с внешним запросом (он может быть выполнен отдельно от него).

**Коррелированный подзапрос** – оператор SELECT вложенный в другой запрос SQL, и ссылающийся на один или несколько столбцов внешнего запроса.

Рассмотрим типы запросов на примерах.

Для создания подзапросов иногда бывают полезными ключевые слова ALL, ANY, EXISTS, IN.

Таблица 3 — Ключевые слова

Ключевое слово	Применение
условие <b>ALL</b> (подзапрос)	Если все значения подзапроса удовлетворяют условию, то оператор возвращает <i>true</i>
условие <b>ANY</b> (подзапрос)	Если хотя бы одно значение подзапроса удовлетворяет условию, то оператор возвращает <i>true</i>
<b>EXISTS</b> (подзапрос)	Если подзапрос содержит хотя бы одну строку, то оператор возвращает <i>true</i>
Выражение <b>[NOT] IN</b> (подзапрос)	Если подзапрос [не] содержит строку, полученную в результате вычисления выражения, то он возвращает <i>true</i>

### 1.2.1. Некоррелированный запрос.

Выведем всех преподавателей, чей оклад больше среднего. Для этого создадим подзапрос, вычисляющий среднюю зарплату всех преподавателей вуза. Далее, используя скалярный результат этого подзапроса найдем все большие значения.

```
SELECT last_name, first_name, salary::numeric
FROM professors
WHERE salary::numeric >
(
```

```

SELECT AVG(salary::numeric)
FROM professors
)

last_name | first_name | salary
-----+-----+-----
Широков   | Василий   | 98000.00
Семенов   | Андрей    | 210000.00
Филатов   | Илья      | 85000.00
Воронов   | Николай   | 98000.00
Серебряков | Александр | 150000.00
Воронов   | Артём     | 98000.00
Михайлова | Анастасия | 210000.00

```

Выведем всех студентов, которые имеют хотя бы одну оценку 2 за один из экзаменов. Для этого создадим подзапрос, выбирающих всех двоечников и используя их номера студенческого билета найдем их фамилию и имя.

```

SELECT surname, name
FROM student
WHERE student.student_id IN(
    SELECT field_comprehension.student_id
    FROM field_comprehension
    WHERE field_comprehension.mark = 2
)

```

### 1.2.1. Коррелированный запросы

Создадим аналогичный запрос с поиском всех студентов, имеющих хотя бы одну оценку 2, но с помощью коррелированного запроса. Обратите внимание, что во вложенном запросе происходит обращение к таблице *Students*, не указанной после ключевого слова FROM данного подзапроса. Если мы попытаемся запустить отдельно от основного данный подзапрос, то в результате будет получена ошибка.

```

SELECT last_name, first_name
FROM students

```

```

WHERE 2 = ANY
(
    SELECT mark
    FROM field_comprehensions
    WHERE field_comprehensions.student_id = students.student_id
)

```

Создадим запрос, выводящий средний балл каждого из студентов. Для этого создадим подзапрос, вычисляющий средний балл и ссылающийся на атрибут **student\_id** из таблицы *Students*, используемой во внешнем запросе.

```

SELECT last_name, first_name, (
    SELECT CAST(AVG(mark) AS NUMERIC(2,1))
    FROM Field_comprehensions
    WHERE Field_comprehensions.student_id = Students.student_id
) AS "Средняя оценка"
FROM Students
ORDER BY "Средняя оценка" DESC;

```

last_name	first_name	Средняя оценка
Савельев	Максим	4.3
Гаганов	Павел	4.2
Лембке	Юлия	4.1
Фадеева	Ксения	4.1
Лобов	Александр	4.1
Иртеньев	Николай	4.1
Верховцев	Иван	4.1
Ракитин	Михаил	4.0
Ильин	Владимир	4.0

### 1.3. Общие табличные выражения

Для упрощения сложного запроса возможно использовать конструкцию CTE (Common table expression), позволяющую разбить его на несколько частей. Эта конструкция определяет временные таблицы, которые существуют только для одного запроса. CTE похожи на вложенные запросы, но более оптимизированы. Вложенный запрос в отличие от CTE повторяется для каждой

строки, которую нашел основной запрос, что повышает ресурсоемкость и замедляет работу кода.

Формат:

```
WITH <название выражения> AS (  
SELECT | INSERT | UPDATE |DELETE  
) , AS ...  
SELECT ...
```

Например, следующий запрос вычисляет всех студентов, чьи оценки выше средней по всему вузу.

```
WITH avg_mark AS (  
    SELECT cast(avg(mark) AS numeric(2,1)) AS a_mark FROM field_comprehensions  
)  
SELECT s.student_id,  
       last_name,  
       first_name,  
       cast(avg(mark) AS numeric(2,1)) AS "avg. mark"  
FROM students s  
INNER JOIN field_comprehensions fc ON s.student_id = fc.student_id  
GROUP BY s.student_id  
HAVING avg(mark) > (SELECT a_mark FROM avg_mark)  
ORDER BY "avg. mark" DESC;
```

student_id	last_name	first_name	avg. mark
828170	Савельев	Максим	4.3
830114	Гаганов	Павел	4.2
892391	Фадеева	Ксения	4.1
834421	Иртеньев	Николай	4.1
816993	Лембке	Юлия	4.1
825852	Лобов	Александр	4.1
833207	Верховцев	Иван	4.1
895508	Раkitин	Михаил	4.0
894671	Бессонов	Алексей	4.0

## 1.4. Представления



При работе с базами данных очень часто приходится выполнять одинаковые сложные и объемные запросы. Каждый раз составлять длинный запрос оказывается весьма трудоемко. Для упрощения работы возможно сформировать из такого запроса представление, к которому далее возможно обращаться, как будто это обычная таблица какое угодно число раз. Представление состоит из строк и столбцов, которые могут формироваться из одной или нескольких таблиц. Сокращенный формат команды:

```
CREATE VIEW имя AS запрос
```

Например, создадим представление на основе запроса, выводящего трудоемкость дисциплины в формате «Количество часов/ЗЕТ»

```
CREATE VIEW labor_intensity AS
SELECT field_name AS "Field name", (36*zet::numeric)::varchar || '/' || zet as
"Labor intensity"
FROM field
ORDER BY "Labor intensity"
```

Далее к созданному представлению возможно обратиться, как к таблице.

```
SELECT * FROM labor_intensity
```

Field name	Labor intensity
Риторика	108/3
Основы управления проектами	108/3
Основы рыночной экономики	108/3
Безопасность жизнедеятельности	108/3
Экология	108/3
Иностранный язык	108/3
История	108/3
Правоведение	108/3

## 2. Практическая часть

### 2.1. Задание 1.

```
with count_degree AS (
    SELECT DISTINCT degree from professors
) SELECT count(degree) FROM count_degree
```

Рассмотрите следующий запрос. Какой результат был получен после его выполнения. Измените запрос, поставив вместе значения degree в агрегатную функцию символ «\*». Сравните результаты выполнения запросов и объясните его.

## 2.2. Задание 2.

Напишите SQL запросы к учебной базе данных в соответствии с вариантом. Вариант к практической части выбирается по формуле:  $V = (N \% 10) + 1$ , где N – номер в списке группы, % – остаток от деления. Обратите внимание, что группировка заданий по разделам означает не столько подсказку, сколько **обязательное** использование данных параметров. Некоторые запросы возможно решить без использования указанных операторов, однако задача лабораторной работы познакомить вас с разнообразием возможностей PostgreSQL, поэтому их использование необходимо.

Таблица 4 — Список вариантов

№ варианта	№ запросов
1	1, 11, 21, 31, 41, 51, 61, 71, 81
2	2, 12, 22, 32, 42, 52, 62, 72, 82
3	3, 13, 23, 33, 43, 53, 63, 73, 83
4	4, 14, 24, 34, 44, 54, 64, 74, 84
5	5, 15, 25, 35, 45, 55, 65, 75, 85
6	6, 16, 26, 36, 46, 56, 66, 76, 86
7	7, 17, 27, 37, 47, 57, 67, 77, 87
8	8, 18, 28, 38, 48, 58, 68, 78, 88

9	9, 19, 29, 39, 49, 59, 69, 79, 89
10	10, 20, 30, 40, 50, 60, 70, 80, 90

### 2.3. Задание 3.

Самостоятельно разработайте 3 **осмысленных** запроса к базе данных, используя приведенные в данной лабораторной работе материалы. Вариант выбирается в соответствии с номером по списку. Из созданных запросов создайте представления.

Сборник запросов к лабораторной работе

#### INNER JOIN

1. Вывести ФИО, должности, оклад, номер трудового договора всех преподавателей
2. Вывести ФИО, номера студенческих билетов, оценки по освоению дисциплин и названия дисциплин для всех студентов
3. Вывести наименование группы и наименование структурного подразделения, к которому она относится (для всех групп)
4. Вывести ФИО преподавателей, названия преподаваемых дисциплин и название групп, у которых они ведутся. Отсортировать по номеру группы.
5. Вывести ФИО преподавателя, названия преподаваемых дисциплин и все оценки ими выставленные. Исключить из вывода незаполненные поля оценок. Отсортировать по фамилии, имени и дисциплине.
6. Вывести номер трудового договора и название структурного подразделения, к которому он относится. Отсортировать по номеру трудового договора.
7. Вывести название структурного подразделения, номер трудового договора и ФИО преподавателя, который трудоустроен в этом структурном подразделении. Отсортировать по названию структурного подразделения и фамилии и имени преподавателя.
8. Вывести ФИО студента и освоенные им дисциплины с оценками. Отсортировать по фамилии и имени.
9. Вывести ФИО преподавателей, названия преподаваемых ими дисциплин. Отсортировать по фамилии и имени.

10. Вывести фамилию и имя преподавателей, у которых отсутствует отчество, а также названия преподаваемых ими дисциплин. Отсортировать по фамилии и имени.
11. Вывести названия групп, сокращенной название подразделения и ФИО руководителей структурного подразделения.
12. Вывести ФИО всех студентов, освоивших дисциплину Базы Данных на 5. Отсортировать по фамилии и имени.
13. Вывести все дисциплины, которые ведутся у группы ИТД-31
14. Вывести полные названия структурных подразделений и названия дисциплин, преподаваемых в них. Отсортировать по названию структурного подразделения.
15. Вывести ФИО и ставку всех трудоустроенных преподавателей. Отсортировать по ставке.
16. Вывести ФИО каждого студента и срок действия его студенческого билета.
17. Вывести ФИО всех студентов и форму их обучения. Включить в список только заочников.
18. Вывести ФИО преподавателей, их ставку и стаж. Включить в список преподавателей со стажем больше 5 лет.
19. Вывести ФИО всех студентов с оценками, освоивших дисциплину «Философия» меньше, чем на 4. Отсортировать по фамилии и имени.
20. Вывести ФИО, номер группы студентов имеющих оценку 5 по дисциплине 'Фотографика'.
21. Подсчитать количество отличников в каждой группе по каждой дисциплине, включающей в себя слово «Физика».
22. Вывести названия групп, в которых есть хотя бы 1 студент, освоивший дисциплину «Физика» на 5 (название дисциплины должно содержать слово «Физика»). Отсортировать по номеру группы.
23. Вывести руководителя структурного подразделения, номера групп входящие в его подразделение и количество студентов в каждой группе. Отсортировать по фамилии руководителя.
24. Вывести ФИО всех преподавателей и их зарплату, если зарплата больше средней по России (зарплата = оклад \* ставка)
25. Найти среднюю оценку студентов по каждой дисциплине. Вывести полное название дисциплины и среднюю оценку. Всем столбцам дать русское название. Отсортировать по оценке.
26. Подсчитать количество каждой оценки («2», «3», «4», «5»), которые выставили преподаватели. Вывести фамилию, имя, отчество преподавателя,

наименование дисциплины, оценку и ее количество. Исключить из вывода незаполненные поля оценок. Отсортировать по фамилии, имени и дисциплине.

27. Вывести названия первых групп (включая вечерников) и названия их структурных подразделений.

28. Найти среднюю оценку каждого студента, родившегося в 2004 году по всем дисциплинам, отсортировать, по средней оценке, в порядке убывания. Вывести фамилию, имя, номер студенческого и среднюю оценку.

29. Подсчитать количество всех оценок у студентов чьи номера студенческих билетов лежат в интервале 820000–850000. Вывести фамилию, имя, номер студенческого, оценку и ее количество. Исключить из подсчета незаполненные поля оценок. Отсортировать по номеру студенческого билета.

30. Вывести сокращенное название подразделения, которым руководит Соколова, фамилию и должность преподавателей, которые входят в это подразделение, а также дисциплины, которые ведут преподаватели этого подразделения.

31. Вывести ФИО студентов и преподавателей, у которых совпадают фамилии.

32. Вывести список студентов из групп ИВТ. Которые сдали на отлично более 10 предметов.

33. Создать пароль студентам по следующему правилу: первые пять символов почты + день рождения + месяц рождения + две цифры номера группы. Вывести фамилию, имя, дату рождения, почту, форму обучения, дату выдачи студенческого билета. Оставить только студентов очной формы обучения с датой выдачи студенческого билета > 01.01.2022.

34. Вывести список всех студентов, обучающихся заочно, отсортировать по фамилиям, в алфавитном порядке оставить только тех, кому больше 20 лет

35. Вывести всех студентов группы ИБ, отсортировать по числу 5 по дисциплине «Базы Данных». Оставить только тех, у кого 5к больше 5

36. Вывести фамилию, имя, номер студенческого и среднюю оценку студента по всем дисциплинам исключая 'Философию'. Отсортировать, по средней оценке, в порядке убывания.

37. Вывести название предмета, фамилию и имя преподавателя, который ведет этот предмет и среднюю оценку у всех студентов за этот предмет. Оставить только те предметы, которые читаются в первом семестре.

38. Вывести фамилию, имя преподавателя, количество дисциплин, которые он преподает и его оклад. Оставить только тех, у кого количество дисциплин > 5. Отсортировать по фамилии.

39. Выведите полные названия структурных подразделений, название групп в него входящих и количество студентов в каждой группе. Оставьте только группы содержащие в своем названии буквы “В” и “Б” и оканчивающиеся цифрой “1”. Отсортируйте по номеру группы.

40. Вывести полное название структурного подразделения, его код, средний, минимальный и максимальный оклад преподавателей, которые работают в этом подразделении. Дать столбцам соответствующие русские названия. Отсортировать по коду подразделения.

### **LEFT JOIN, RIGHT JOIN, FULL JOIN**

41. Вывести руководителей всех структурных подразделений и название дисциплины, относящееся к этим структурным подразделениям (если они есть, иначе – null). Отсортировать по дисциплине в обратном порядке.

42. Вывести фамилии преподавателей и фамилии их однофамильцев среди студентов (если есть, иначе – null). Использовать *LEFT JOIN* или *RIGHT JOIN*.

43. Вывести названия структурных подразделений и группы, которые им принадлежат (если они есть, иначе – null)

44. Вывести ФИО преподавателя и название дисциплины, которую он ведёт (если нет – null)

45. Вывести список преподавателей и наименование структурного подразделения, в котором они трудоустроены (если не трудоустроены – null)

46. Вывести ФИО всех студентов и форму их обучения

47. Вывести все пары фамилий преподавателей, у которых одинаковые имена

48. Вывести ФИО всех студентов и предмет, по которым у них долг (если долга нет – null), отсортировать по ФИО.

49. Вывести всех руководителей структурных подразделений, их номера телефонов и ФИ и электронные почты всех студентов

50. Вывести все пары фамилий студентов родившихся в один день

### **UNION/EXCEPT/INTERSECT**

51. Вывести фамилии всех студентов и преподавателей в (дубли оставить). Отсортировать по фамилии. Добавить столбец, в котором обозначить преподаватель это или студент.

52. Вывести название всех групп и всех структурных подразделений. Отсортировать по названию.
53. Вывести ФИО всех преподавателей и руководителей структурных подразделений в одной колонке. Добавить колонку, где указать – “руководитель” для соответствующей записи.
54. Вывести в одном столбце: максимальный оклад среди преподавателей, средний оклад среди преподавателей, минимальный оклад среди преподавателей. В дополнительном столбце сделать соответствующие пояснения.
55. Вывести однофамильцев среди студентов и преподавателей (фамилию). Использовать *UNION/EXCEPT/INTERSECT*.
56. Вывести названия всех групп, в последней строчке вывести: ‘Итого студентов’ и количество всех студентов в институте.
57. Выведите фамилии всех преподавателей и их зарплату. В первой строчке вывести: ‘Средняя зарплата’ (можно начать с латинской буквы “С” ) и значение средней зарплаты у преподавателей.
58. Вывести фамилии преподавателей, у которых нет однофамильцев среди студентов.
59. Вывести ФИО, оклад и должность всех преподавателей, кроме преподавателей МПСУ. Сортировать по убыванию оклада. Использовать EXCEPT.
60. Вывести ФИО студентов и сокращенное название подразделения, к которому относится их группа. В список включить всех студентов, кроме тех, у кого группа принадлежит МПСУ. Использовать EXCEPT.
61. Вывести название всех групп, количество студентов в них, код всех структурных подразделений, к которым привязаны группы и количество групп в них.
62. Выведите в два столбца название семестра (например, “Семестр 1”) и количество читаемых в этом семестре дисциплин. Внизу в эти же два столбца добавьте фамилии преподавателей, которые читают дисциплины в 8 семестре и количество этих дисциплин.
63. Вывести название всех групп и число студентов в них, в последней строчке вывести: ‘Итого студентов’ и количество всех студентов в институте.

64. Вывести всех преподавателей и студентов, у которых совпадают первые буквы имени и фамилии. Добавить столбец, в котором обозначить преподаватель это или студент.
65. Вывести ФИО всех студентов вторых групп (включая вечерников), их номера групп и преподавателей-ассистентов, читающих дисциплины в 5 семестре. У преподавателей в столбец номера группы записать 'преподаватель'.
66. Вывести имена студентов из первой и второй групп, количество этих имен, столбец, содержащий значение 'первая' или 'вторая' в зависимости от номера группы.
67. Вывести ФИО всех мужчин в институте (среди преподавателей и студентов). Учесть очень большое, в отличие от женщин, разнообразие окончаний фамилий и частое отсутствие отчеств. Добавить столбец, в котором обозначить преподаватель это или студент.
68. Вывести ФИО всех женщин в институте (среди преподавателей и студентов). Пол определить по окончанию фамилии или отчества. Добавить столбец, в котором обозначить преподаватель это или студент.
69. Найдите ФИО всех девушек среди студентов (пол определить по окончанию фамилии или отчества). Используя найденную информацию выведите всех юношей среди студентов.
70. Вывести фамилии всех студентов, чьи фамилии начинаются на букву “Ф” и преподавателей-ассистентов, чьи фамилии начинаются на буквы “К” и “И”. Добавить столбец, в котором обозначить преподаватель это или студент.

### ***Вложенные запросы***

71. Вывести фамилию, имя студентов и их оценку по дисциплине ‘ Базы Данных’, если она больше средней оценки всех студентов за эту дисциплину.
72. Вывести номера трудовых договоров и ставки тех преподавателей, у которых ее значение выше средней. В конце таблицы добавить строку – ‘Средняя ставка’ и вписать ее значение.
73. Вывести фамилию, имя и оценку студентов, у которых оценка хотя бы по одной из освоенных дисциплин выше, чем средняя по ВУЗу.
74. Вывести фамилию, имя и оценку студентов группы ИБ-21, у которых хотя бы одна оценка выше средней по группе.



75. Вывести всех отличников (студентов, у которых за все дисциплины оценка 5)
76. Вывести средний возраст студентов каждого структурного подразделения
77. Вывести фамилию, имя, группу и возраст студента, у которого возраст меньше среднего по всем студентам.
78. Вывести максимальное количество студентов в группе по всему ВУЗу. Использовать вложенный запрос.
79. Вывести фамилия, имя и оклад всех преподавателей, чей оклад меньше среднего. Отсортируйте по окладу в порядке убывания. Первой строчкой выведите средний оклад.
80. Вывести фамилию и имя преподавателя с максимальным стажем работы.
81. Вывести фамилию, имя, группу и возраст тех студентов, у которых возраст больше среднего по группе. Первой строчкой в каждой группе при выводе должна быть запись – ‘Средний возраст’ (для сортировки можно использовать первую букву латинскую ‘С’) + номер группы + средний возраст по группе. Далее должны быть выведены студенты этой группы, чей возраст больше среднего.
82. Найдите название дисциплины, по которой получено наибольшее количество пятерок. Выведите фамилию, имя, отчество студентов, получивших пять по этой дисциплине.
83. Вывести фамилию, имя преподавателей и максимальную трудоемкость среди преподаваемых ими дисциплин. В выборку включить только тех преподавателей, у которых максимальная трудоемкость меньше средней. Первой строкой вывести среднюю трудоемкость.
84. Вывести фамилию, имя, номер группы студентов, которые учатся в группе с максимальным количеством человек.
85. Вывести преподавателя, у которого наибольшее количество выставленных двоек за определенную дисциплину. Вывести его фамилию, имя, дисциплину и количество двоек.
86. Вывести фамилию, имя и группу студентов, у которых троек больше, чем четверок.
87. Выведите все дисциплины семестра, в котором читаются наибольшее количество дисциплин.
88. Вывести фамилию, имя стаж и зарплату преподавателей всех преподавателей, у которых стаж работы больше среднего, а зарплата ниже средней. Отдельной строкой вывести средний стаж и зарплату всех преподавателей.

89. Вывести фамилию, имя всех студентов, чей средний балл выше среднего балла в ИТД-21

90. Выведите фамилию, имя и группу студентов, в которой наибольший средний возраст

### ***Дополнительные запросы***

91. Вывести номера студенческих билетов студентов, у которых нет ни одной двойки.

92. Вывести номера всех преподавателей, которые не являются преподавателями дисциплины 'Базы Данных' (использовать EXCEPT).

93. Выведите всех преподавателей, которые преподают в первом семестре, но не преподают во втором. Используйте INTERSECT.

94. Вывести номера нетрудоустроенных преподавателей (без трудового договора)

95. Вывести ФИ и номера студенческих билетов студентов, а также дисциплины по которым у них нет двоек

96. Вывести в одном столбце: - самое популярное имя среди студентов; - фамилии всех студентов с этим именем.

97. Вывести в одном столбце: - email-адреса всех студентов; - число уникальных доменных адресов (gmail.com, yandex.ru)

98. Выведите фамилию, имя и группу студентов, у которых троек больше, чем четверок. Отдельными столбцами выведите количество этих оценок.

99. Вывести ФИО и дату рождения самого молодого студента

100. Выведите 10 лучших студентов 3-го курса института МПСУ (имеющих самую высокую среднюю оценку), окончивших учебу без троек и двоек. Отсортируйте ее по убыванию среднего балла. Таблица должна содержать ФИО, номер студенческого билета и средний балл.

### **3. Контрольные вопросы**

1. Существует ли отличие между использованием ключевых слов INNER JOIN или перечисления таблиц через запятую в предложении FROM?
2. Для чего предназначено соединение LEFT JOIN?
3. В чем отличие коррелированного от некоррелированного подзапроса?
4. Назовите отличие соединения таблиц от их объединения.
5. В каких случаях используется ключевое слово INNER и OUTER?

6. В чем разница использования конструкции CTE и вложенного запроса?
7. В чем отличие представления от обычного запроса?

#### **4.Список использованной литературы**

- [1] Документация к PostgreSQL 15.1, 2022.
- [2] Исходный код СУБД postgres, [В Интернете]. Available: <https://github.com/postgres/postgres>. [Дата обращения: 30 01 2023].
- [3] Е. Рогов, PostgreSQL изнутри, 1-е ред., Москва: ДМК Пресс, 2023, стр. 662 .
- [4] Б. А. Новиков, Е. А. Горшкова и Н. Г. Графеева, Основы технологии баз данных, 2-е ред., Москва: ДМК пресс, 2020, стр. 582.
- [5] Е. П. Моргунов, PostgreSQL. Основы языка SQL, 1-е ред., Санкт-Петербург: БХВ-Петербург, 2018, стр. 336.

## **Лабораторная работа №4**

### **«Создание логической и физической модели базы данных»**

#### **1. Теоретическая часть**

##### **1.1. Проектирование базы данных**

Проектирование баз данных в общем виде является сложной и трудоемкой задачей. В общем случае её можно сформулировать, как выбор подходящей логической структуры для заданного массива данных, который необходимо поместить в базу данных [1]. Другими словами, необходимо выделить требуемые отношения (таблицы) и содержащиеся в них атрибуты (столбцы). По окончании проектирования созданную базу данных необходимо преобразовать в форму, которая может быть воспринята конкретной СУБД.

Проектирование базы данных можно разделить на несколько этапов:

- Концептуальное (семантическое, инфологическое) – на данном этапе происходит анализ и определение понятий предметной области
- Логическое – создание схемы данных на основе заданной модели без учета специфики СУБД
- Физическое – создание базы данных с учетом специфики СУБД

Данная лабораторная работа посвящена созданию логической и физической моделей базы данных. Логическая модель данных будет изучаться на примере создания ER-модели или модели «сущность-связь». Физическая модель данных будет представлена в виде запросов для СУБД PostgreSQL.

##### **1.2. Логическое моделирование. ER-модель**

ER-модель является одним из самых распространенных средств для представления структуры баз данных.

В ней структура данных отображается графически в виде диаграммы, которая состоит из элементов трех типов:

- Сущности
- Атрибуты
- Связи

**Сущность** – абстрактный объект определенного типа. Например, в базе данных **Students\_mark** можно выделить сущности *Студенты*, *Преподаватели*, *Структурные подразделения* и др.

**Атрибут** – свойство множества сущностей. Например, сущности *Студенты*, могут быть поставлены в соответствии атрибуты *фамилия*, *имя*, *номер студенческого билета* и др. У каждой сущности возможно выделить подмножество атрибутов или суммы атрибутов, обладающее свойствами уникальности и неизбыточности – потенциальный ключ. Напомним, что из потенциальных ключей можно выделить один, который будет первичным ключом (*PK*, от *Primary Key*), остальные – альтернативными. Первичный ключ должен однозначно идентифицировать каждую запись в таблице. На каждый атрибут можно наложить ограничения, чтобы не допускать добавление в базу данных некорректных значений.

**Связи** – соединения между двумя и большим числом сущностей. Например, между сущностями *Студенты* и *Группы* возможно провести связь – «студенты, обучающиеся в группе». Наиболее распространенными являются бинарные связи, соединяющие два множества сущностей.

Существуют несколько типов связи – 1:1, 1: N, N:N. Рассмотрим их подробнее.

Связь «один к одному» (1:1) – в случае данной связи каждой записи из одной таблицы будет соответствовать одна уникальная запись в другой. Например, у каждого студента есть студенческий билет, поэтому связь между сущностями *Студенты* и *Студенческие билеты* будет один к одному. Связь между таблицами происходит за счет совпадения значений первичных ключей в обеих таблицах.

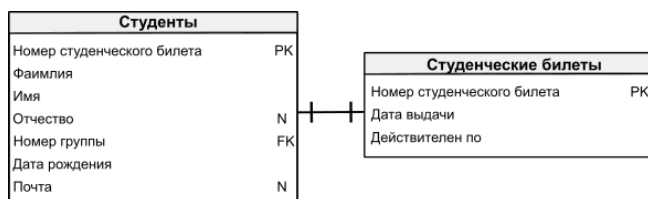


Рисунок 1 — Связь 1:1

Связь «один ко многим» (1:N) – самый распространенный тип связи. В данном случае, одной записи главной таблицы можно сопоставить несколько записей подчинённой таблицы. Например, в одной группе может обучаться множество студентов. Таким образом, связь между атрибутами сущностей *Группы* и *Студенты* будет один ко многим.

Связь между таблицами происходит за счет внешних ключей (*FK*, от *Foreign Key*) Напомним, внешний ключ – поле подчиненной таблицы, соответствующее первичному ключу главной таблицы. Например, атрибут *номер группы* в таблице *Студент* является внешним ключом, так как он связан с первичным ключом *номер группы* таблицы *Студенческие группы*. Данный атрибут отмечен красной стрелочкой.



Рисунок 2 — Связь 1:N

Связь «Многие ко многим» (N: N) предполагает возможность связи одного или нескольких элементов из одной таблицы с одним или несколькими элементами из другой таблицы. В случае рассматриваемого примера один преподаватель может работать в нескольких структурных подразделениях, а в одном структурном подразделении могут содержаться несколько преподавателей. Данный тип связи нереализуем в рамках физической модели, поэтому представляется в виде создания дополнительной таблицы,

содержащей ключевые поля соединяемых отношений. Например, *Структурные подразделения* и *Преподаватели* соединены посредством таблицы *Трудоустройства*.

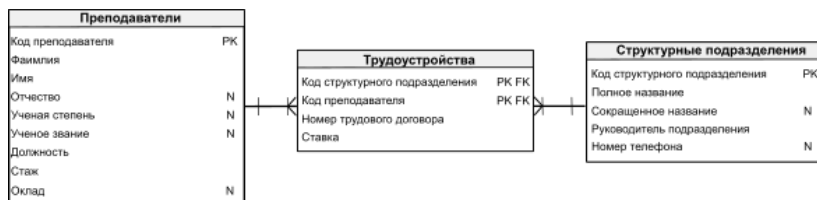


Рисунок 3 — Связь N:N

Существует несколько вариантов записи (нотации) ER-моделей в форме диаграмм. В данном лабораторном практикуме будет рассмотрена ER-диаграмма в записи Гордона Эвереста.

Согласно данной нотации, сущность изображается в виде прямоугольника, содержащее её имя. Имя сущности должно быть уникальным в рамках одной модели. Атрибуты сущности записываются внутри данного прямоугольника. Связь изображается линией, которая связывает две сущности, участвующей в отношении. Каждая связь должна именоваться глаголом или глагольной фразой, записываемой над стрелкой. Например, «состоит из», «включает». В зависимости от типа связи возможно различное обозначение конца линии.



Рисунок 4 — Тип линии

В случае связи один к одному, на конце и в начале линии указывается вертикальная черта, в случае один ко многим в начале линии также указывается вертикальная черта, а в конце – изображение вилки или вороньей лапки.

### 1.3. Пример ER-диаграммы

Рассмотрим пример разработки ER-диаграммы на основе базы данных о студентах вуза. В ней необходимо хранить информацию о студенте, группе его обучения, структурном подразделении, в котором состоит эта группа, об оценках за дисциплины и о преподавателях, проставивших эти оценки.

Проанализировав всю требуемую информацию выдели очевидные независимые сущности: *Студенты*, *Дисциплины*, *Студенческие группы*, *Преподаватели*, *Структурные подразделения*. Рассмотрим сущности по отдельности.

Сущность ***Студенты*** должна содержать всю относящуюся к нему информацию. Выделим следующие атрибуты: *фамилия, имя, отчество, дата рождения, номер студенческого билета, дата выдачи билета, дата окончания действия билета, номер группы и адрес электронной почты*. Номер студенческого билета однозначно определяет студента, то есть обладает свойством уникальности. Поэтому, его можно выбрать в качестве первичного ключа. На практике, часто необходимо на определенные атрибуты накладывать ограничения (*constrain*), позволяющие задавать точный формат записи, диапазон, в котором может лежать значение и др. О данных ограничениях можно задуматься на этапе разработки ER-диаграммы. Добавим ограничение на формат электронной почты. Она будет представлять из себя строку вида [TEXT]@[TEXT].[DOMAIN], где TEXT может содержать любые английские буквы и числа, а DOMAIN только английские буквы.

Обратим внимание, что поля с информацией о студенческом билете (*дату выдачи билета, дату окончания действия*) относятся к билету, а не к студенту и логично будет разделить данную сущность на две – одна будет описывать студента (человека), вторая – студенческий билет (документ). Исходя из этого выделим еще одну сущность – ***Студенческие билеты*** и соединим её с сущностью *Студенты* связью 1:1 (каждому студенту будет соответствовать



свой студенческий билет). В качестве атрибутов сущности *Студенческие билеты* выделим *дату выдачи билета* и *дату окончания действия*.

Сущность ***Структурные подразделения*** будет содержать в себе атрибуты *полное и сокращенное название, фамилия руководителя подразделения, телефон*. Для удобства уникальной идентификации введем дополнительный атрибут – *код структурного подразделения*. Он будет являться суррогатным ключом для данной сущности. Напомним, суррогатный ключ – дополнительный атрибут сущности, не имеющий смысловой нагрузки, но являющийся уникальным и неизбыточным, что дает право использовать его в качестве первичного ключа.

Сущность ***Студенческие группы*** будет содержать в себе следующие атрибуты: *номер группы* (первичный ключ), *форма обучения* и *код структурного подразделения*, в котором состоит эта группа. На *номер группы* будет наложено ограничение – [ТЕКСТ]-[НОМЕР], где ТЕКСТ – любое буквосочетание на кириллице, а НОМЕР – любые цифры. Значение атрибута *форма обучения* может быть только очной, заочной или очно-заочной. Для обозначения очно-заочной формы к номеру группы добавляется символ «В».

Сущность ***Преподаватели*** будет содержать в себе его *фамилию, имя, отчество, ученую степень, ученое звание, должность, стаж, оклад*. *Ученая степень* должна иметь формат [к/д].[ТЕКСТ].н, где ТЕКСТ – название отрасли наук, а к./д. н. – кандидат/доктор наук. В качестве ключа выбран суррогатный ключ – *код преподавателя*.

Сущность ***Дисциплины*** будет включать в себя атрибуты: *код дисциплины* (суррогатный ключ), *название дисциплины, код структурного подразделения, код преподавателя, зачетные единицы трудоемкости (ЗЕТ), семестр*.

Создав сущности и их атрибуты, для построения ER-диаграммы осталось определить связи между сущностями.

Сущности *Студенты* и *Студенческие билеты* можно связать по одноимённому атрибуту – *номер студенческого билета* (связь «один к одному»). *Студенческие группы* и *Студенты* связаны по атрибуту *номер группы* связью «один ко многим». Причем атрибут *номер группы* сущности *Студенты* будет внешним ключом. Сущности *Структурные подразделения* и *Дисциплины* связаны по атрибуту *код структурного подразделения* связью «один ко многим». Атрибут *код структурного подразделения* сущности *Дисциплины* является внешним ключом. Сущности *Преподаватели* и *Дисциплины* также связаны «один ко многим» по атрибуту *код преподавателя* и одноименный атрибут у сущности *Дисциплины* является внешним ключом.

Исходя из того, что один студент может иметь оценки по разным дисциплинам и одной дисциплине могут обучаться несколько студентов, связь между сущностями *Студенты* и *Дисциплины* будет «многие ко многим». Для реализации данной связи добавим еще одну сущность – ***Результаты освоения дисциплины***, Она будет содержать в себе атрибуты *номер студенческого билета* и *код дисциплины*. Данные атрибуты будут являться внешними ключами для связи с сущностями *Студенты* и *Дисциплины*. Также *Результаты освоения дисциплины* будет содержать в себе атрибут с оценкой за дисциплину. Допустимые значения атрибута *Оценка* – от 2 до 5 включительно.

Аналогично, связь между сущностями *Преподаватели* и *Структурные подразделения* будет «многие ко многим». Поэтому добавим сущность ***Трудоустройства***, содержащую внешние ключи из связанных таблиц – код структурного подразделения, код преподавателя, а также номер трудового договора и ставку.

Итоговая ER-диаграмма приведена на Рисунок .



```

structural_unit_id SERIAL,
unit_type VARCHAR(64),
full_title TEXT,
abbreviated_title VARCHAR(20),
head_of_the_unit VARCHAR(40)
);

```

#### 1.4.2. Изменение таблиц

Для изменения таблиц базы данных используется команда ALTER TABLE.

```

ALTER TABLE [ IF EXISTS ] [ ONLY ] имя [ * ]
    действие [, ... ]

```

В качестве действия могут быть добавление, изменение и удаление конкретного столбца, изменение типов данных и др. Синтаксис некоторых возможных действий представлен ниже:

```

ALTER TABLE [ IF EXISTS ] [ ONLY ] имя [ * ]
    ADD [ COLUMN ] [ IF NOT EXISTS ] имя_столбца тип_данных [ COLLATE
правило_сортировки ] [ ограничение_столбца [ ... ] ]
    DROP [ COLUMN ] [ IF EXISTS ] имя_столбца [ RESTRICT | CASCADE ]
    ALTER [ COLUMN ] имя_столбца [ SET DATA ] TYPE тип_данных [ COLLATE
правило_сортировки ] [ USING выражение ]
    ALTER [ COLUMN ] имя_столбца SET DEFAULT выражение
    ALTER [ COLUMN ] имя_столбца DROP DEFAULT

```

Значение всех параметров команды можно посмотреть в литературе [2], В данной лабораторной работе будут использоваться параметры, приведенные в примерах ниже. Примеры:

Добавление в таблицу *Students* столбца *address*:

```

ALTER TABLE student
ADD COLUMN address varchar(30);

```

Изменение столбца адрес – увеличение его объема.

```

ALTER TABLE student

```

```
ALTER COLUMN address TYPE varchar(80);
```

Удаление столбца с адресом

```
ALTER TABLE student  
DROP COLUMN address;
```

### 1.4.3. Удаление таблиц

Удаление таблицы происходит с помощью команды DROP TABLE.

```
DROP TABLE [ IF EXISTS ] имя [, ...] [ CASCADE | RESTRICT ]
```

Параметры:

IF EXISTS – не считать ошибкой, если таблица не существует, выдается только замечание.

CASCADE – автоматически удаляются объекты, зависящие от данной таблицы (например, представления).

RESTRICT – отказ в удалении таблицы, если от неё зависят какие-либо объекты (по умолчанию).

Пример удаления таблицы *Students*

```
DROP TABLE students;
```

### 1.4.4. Ограничения

Отнесение данных к определенному типу (например, INTEGER, DATE) накладывает на них общие ограничения. Однако, часто требуются, чтобы данные находились в необходимом диапазоне или имели строго определенный вид. Например, значение атрибута *оценка* должно быть целым и лежать в диапазоне от 2 до 5. Для указания таких ограничений существуют специальные ключевые слова.

Таблица 1 — Ключевые слова для установки ограничений

Ключевое слово	Описание
PRIMARY KEY	Первичный ключ

FOREIGN KEY...REFERENCES	Внешний ключ
NOT NULL	Значение не может принимать значение NULL
NULL	Значение может принимать значение NULL
CHECK (условие)	Проверка условия
UNIQUE	Уникальность атрибута
DEFAULT	Установка значения по умолчанию

Самый простой способ установки ограничений – использование ключевого слова CHECK. В его определении возможно указать, что значение данного столбца будет удовлетворять логическому выражению (проверке истинности).

Например, укажем, что оценка может быть от 2 до 5:

```
mark INTEGER NOT NULL CHECK (mark >=2 AND mark <=5)
```

Ограничения NOT NULL/ NULL указывает на то, что столбцу нельзя /можно присваивать значение NULL.

Ограничение уникальности UNIQUE, гарантирует, что данные в столбце должны быть уникальными для всех строк таблицы.

Например:

```
patronymic VARCHAR(30) NULL
birthday DATE NOT NULL
email VARCHAR(30) UNIQUE
```

Если NOT NULL и UNIQUE используют вместе, то это ограничение эквивалентно определению первичного ключа. Однако, ограничений уникальности и NOT NULL может быть сколько угодно, а первичный ключ может быть только один.

Для указания того, что атрибут является первичным ключом, используются ключевые слова PRIMARY KEY. Например,

```
structural_unit_id SERIAL PRIMARY KEY
```

Ограничение FOREIGN KEY, подразумевает, что данный столбец является внешним ключом, то есть его значения должны быть связаны со значениями аналогичного столбца в другой таблицы, являющейся главной. После слова FOREIGN KEY в скобках указывают названия столбцов подчиненной таблицы, которые являются внешними ключами. После слова REFERENCES указывают название главной таблицы, а в скобках ее атрибут, по которому эти таблицы связаны. В случае если внешний ключ один, слово FOREIGN KEY можно опустить.

Например, для указания ссылки на таблицу со структурными подразделениями из таблицы студенческих групп можно выполнить следующим образом:

```
structural_unit_id      INTEGER      NOT      NULL      REFERENCES  
Structural_units(structural_unit_id)
```

```
Аналогично возможно выполнить ограничение с помощью FOREIGN KEY  
FOREIGN KEY(structural_unit_id)  
REFERENCES Structural_units(structural_unit_id)
```

Параметр DEFAULT <выражение> задает значение для столбца, которое будет присваиваться ему по умолчанию. Тип выражения должен соответствовать типу столбца. Это выражение будет присвоено столбцу во всех операциях добавления данных, где его значение не задается напрямую. В случае, если параметр DEFAULT не используется, значением по умолчанию будет NULL.

Каждому ограничению возможно присвоить отдельное имя. Для этого перед ним указывается ключевое слово **CONSTRAINT** и **название ограничения после него.**

```
CONSTRAINT email_cheak  
CHECK (email ~* '^[A-Za-z0-9._+%~]+@[A-Za-z0-9.-]+[.][A-Za-z]+$')
```

Все ограничения возможно выносить за пределы строки с атрибутом.

Рассмотрим пример создания таблицы «Студент»

```
CREATE TABLE Student(  
    student_id INTEGER NOT NULL,  
    surname VARCHAR(30) NOT NULL,  
    name VARCHAR(30) NOT NULL,  
    patronymic VARCHAR(30) NULL,  
    students_group_number VARCHAR(7) NOT NULL,  
    birthday DATE NOT NULL,  
    email VARCHAR(30) UNIQUE,  
    PRIMARY KEY(student_id),  
    CONSTRAINT Students_group_key  
        FOREIGN KEY(students_group_number)  
            REFERENCES Students_group(students_group_number)  
            ON DELETE CASCADE,  
    CONSTRAINT email_cheak  
        CHECK (email ~* '^[A-Za-z0-9._+%~]+@[A-Za-z0-9.-]+[.][A-Za-z]+$')  
);
```

В данном случае все ограничения на ключи и CHECK были вынесены за пределы строк с атрибутами. Для указания внешнего ключа использовалось ключевое слово FOREIGN KEY. Для обеспечения каскадного удаления всех связанных с записью строк по ключам используются ключевые слова ON DELETE CASCADE.

В завершении работы приведем итоговую схему данных с учетом всех заданных типов.



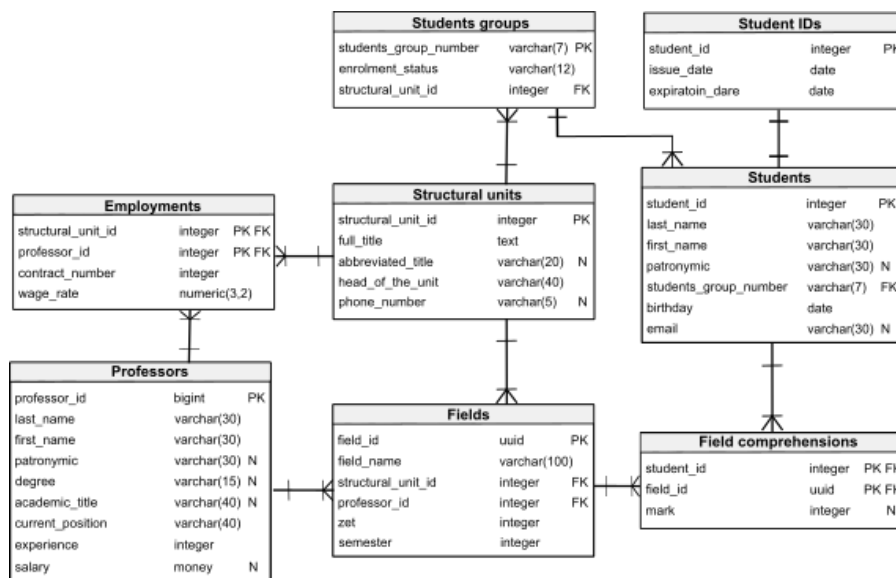


Рисунок 6 — Схема данных

На практике использование кириллических символов при разработке физической модели не используется, поэтому учебная база данных полностью написана латинскими буквами.

### 1.5. Работа с программой pgModeler

Приложение pgModeler – open-source проект, предназначенный для моделирования баз данных. Программа позволяет создавать графические модели базы данных и экспортировать их в файлы различных форматов – от SQL-скриптов для создания физической модели, до изображения имеющих тип png.

Для запуска программы необходимо в командной строке набрать название программы – *pgmodeler*. Окно запустившейся программы выглядит следующим образом:

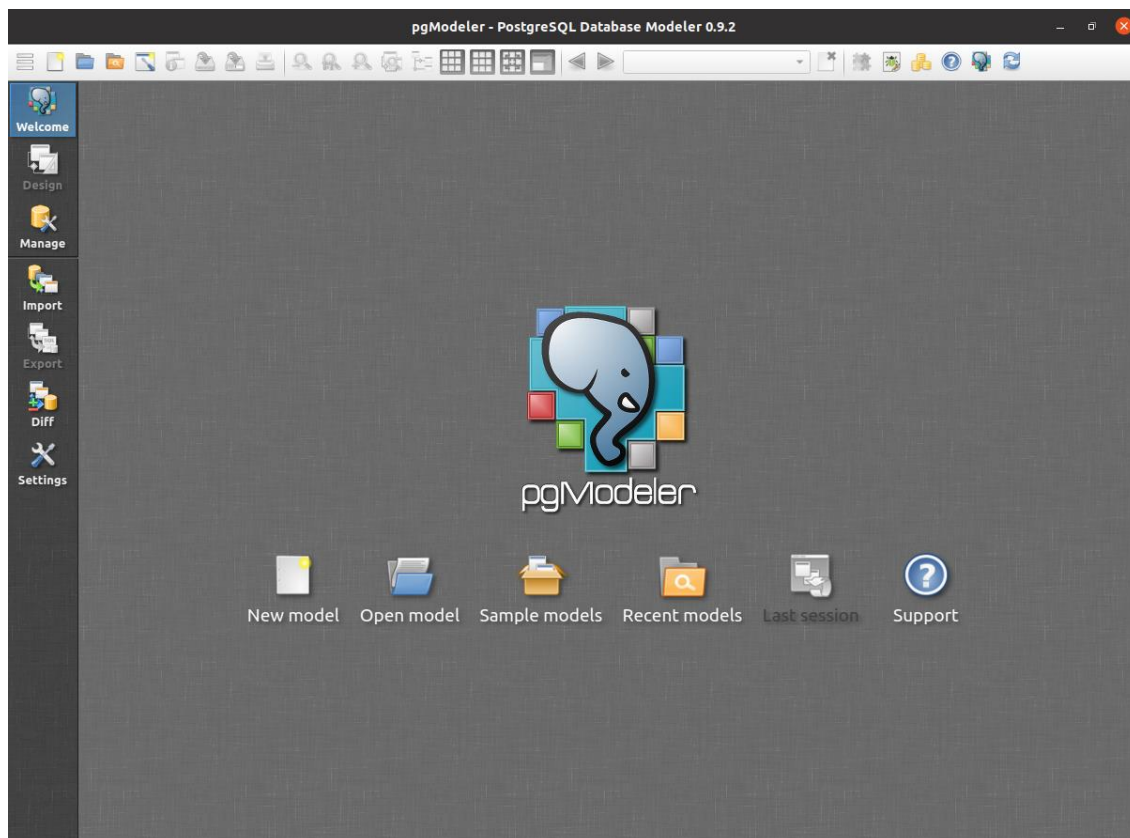



Рисунок 7 — Основное окно pgModuler



Для создания нового проекта необходимо нажать на кнопку «New model» или на рабочей панели сверху выбрать пункт с символом . Аналогичного действия возможно добиться, нажав комбинацию клавиш Ctrl+N.



После создания новой модели откроется рабочее окно, на котором будет производиться разработка модели. Рассмотрим кратко основные функции *pgmodeler*.

#### 1. Основные элементы управления.




Элементы управления располагаются в левой части экрана.

Таблица 2 — Список основных элементов управления

 Welcome	Окно приветствие пользователей. Содержит базовый функционал для создания и открытия проекта.
 Design	Окно разработки даталогической модели.

	Окно для подключения к существующей базе данных.
	Окно для импорта модели из существующей базы данных в программу <i>pgmodeler</i>

Продолжение Таблицы 2

	Окно для экспорта модели на сервер баз данных, в формат SQL файла, графический формат или в html.
	Окно для сравнения нескольких моделей и определения отличий между ними
	Окно настроек программы

### 1.5.1. Подключение к существующей базе данных.

Для подключения к существующей базе данных в поле Manage необходимо в выпадающем слева сверху окне найти и отредактировать существующее стандартное подключение или добавить новое.

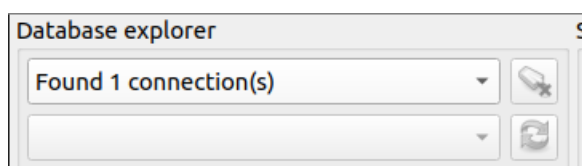


Рисунок 8 — Окно подключения к БД

При выборе «Edit connections», выберите кнопку «Edit selected connection».



Рисунок 9 — Иконка «Edit connections»

В открывшейся вкладке необходимо указать значения, соответствующие вашему соединению. Их вы можете посмотреть в настройках подключения в программе *pgAdmin*.

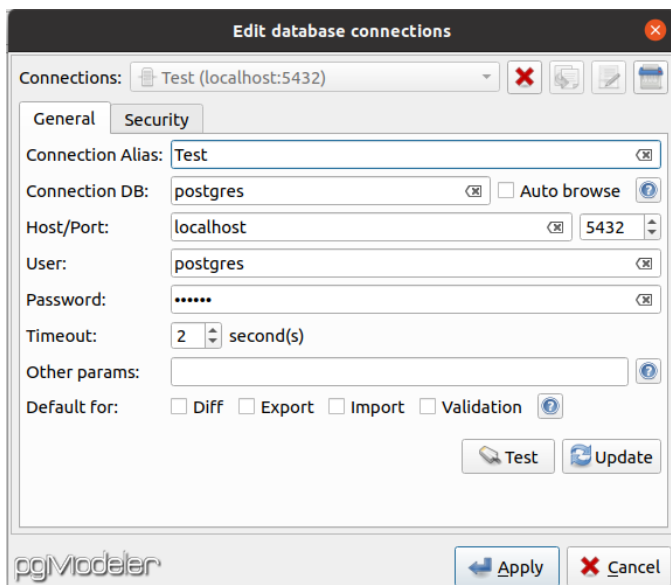


Рисунок 10 — Окно редактирования параметров соединения

После настройки подключения, проведите его тестирование, нажав на кнопку Test. В случае удачного соединения будет выведено окно с сообщением Success.

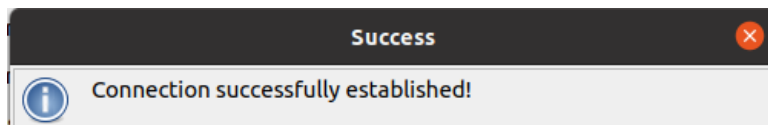


Рисунок 11 — Диалоговое окно успешного подключения

### 1.5.2. Импорт базы данных.

При успешном подключении к серверу возможно импортировать схему базы данных в ваш проект. Для этого в окне import нужно выбрать соединение с нашим сервером и в правом выпадающем окне выбрать необходимую базу данных.

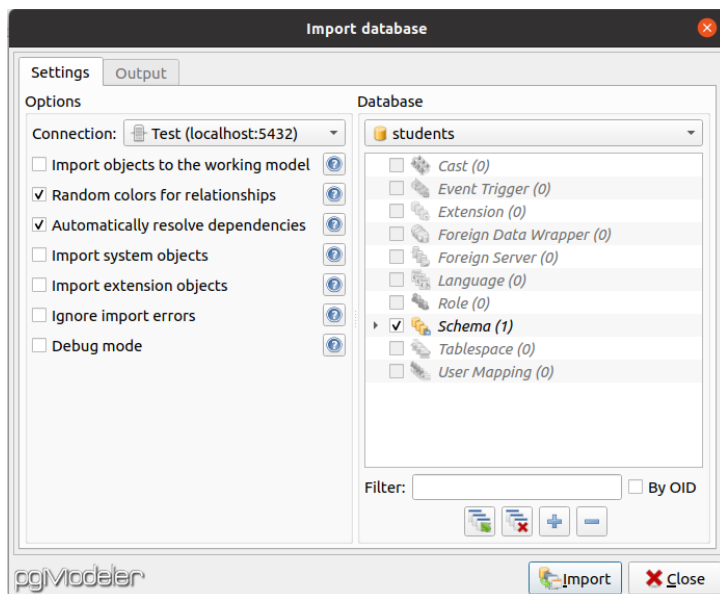


Рисунок 12 — Окно импорта БД

После импорта в ваш проект будет добавлена ERD модель базы данных.

### 1.5.3. Экспорт базы данных.

Для экспорта проекта в окне *Export* необходимо выбрать желаемый выходной формат файла – sql, png или html. Далее укажите путь, по которому файл должен быть сохранен и нажмите кнопку *Export* в нижнем углу открывшегося экрана.

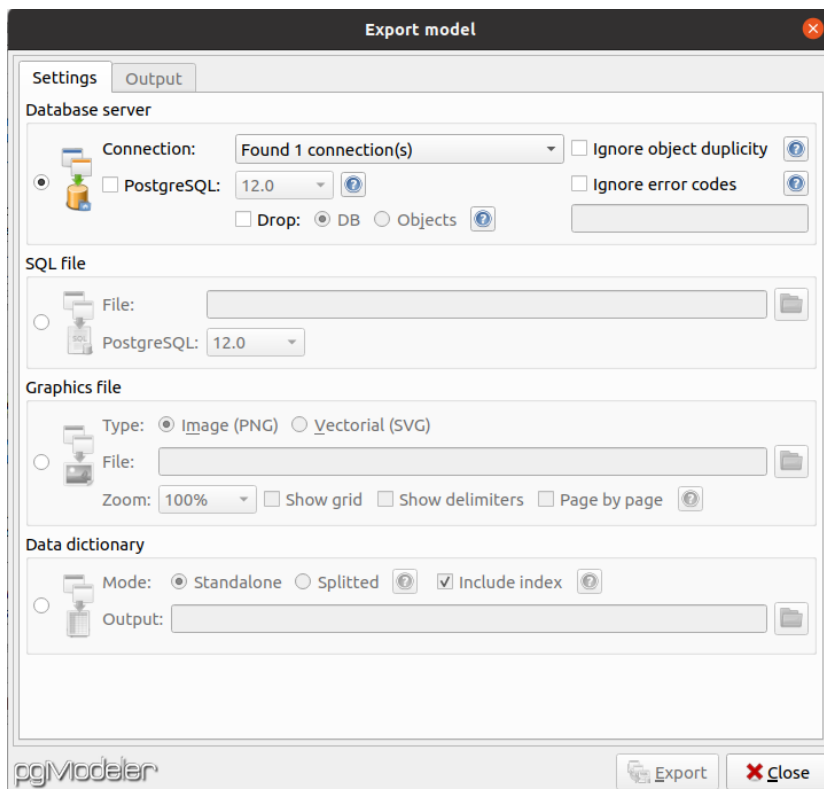


Рисунок 13 — Окно экспорта БД

#### 1.5.4. Разработка модели базы данных.

В качестве примера рассмотрим добавление в импортированную базу данных сущности «Парковочное место», связанное с преподавателем. Для добавления новой таблицы щелкните правой кнопкой мыши по рабочему пространству окна *design* и в выпадающем окне выберите вкладку New -> Table

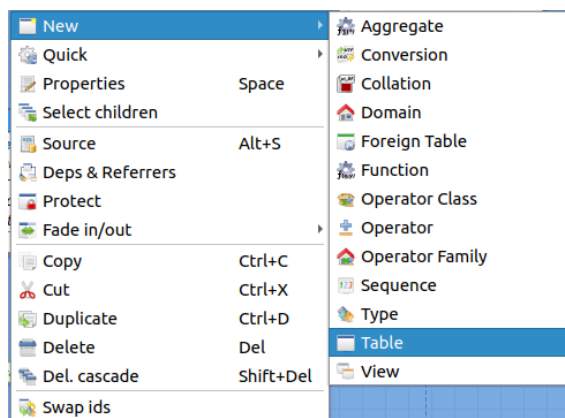


Рисунок 14 — Создание новой таблицы из контекстного меню

Заполним открывшееся окно данными. В поле Name внесем название таблицы – car\_places.

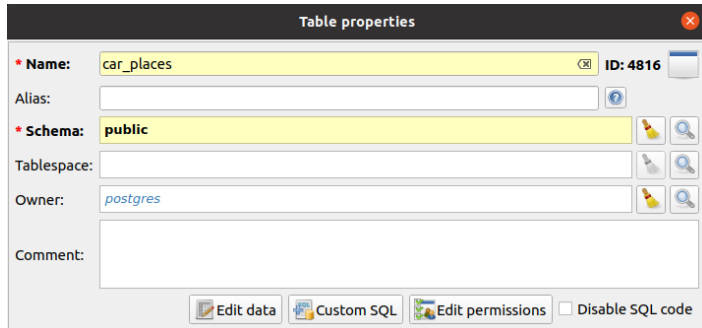


Рисунок 15 — Окно свойств таблицы

В нижнем поле, во вкладке Columns добавим атрибуты созданной сущности. Для этого нажмем на кнопку «Add Item».

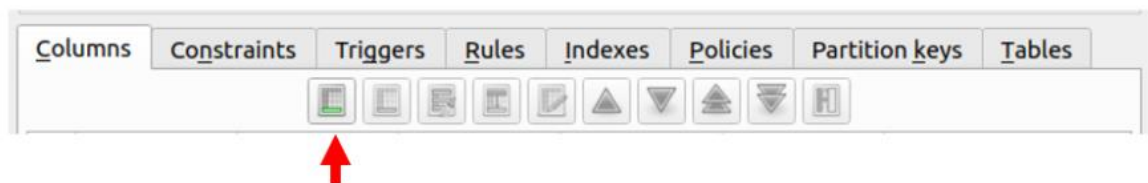


Рисунок 16 — Кнопка Add Item

В открывшемся окне укажем название атрибута – place\_id. Тип – smallint. Аналогично создадим столбец owner целочисленного типа integer, car\_number, состоящий из 8 символов и логическое поле valid, по умолчанию хранящее в себе значение TRUE.

**Column properties**

\* Name:  ID: 4385

Alias:

Collation:

Comment:

☐ Disable SQL code

Data Type

Type:  L:  P:  []:

Format:

Default Value:

☒ Expression:

☐ Sequence:

☐ Identity:  ☐ NOT NULL

Рисунок 17 — Окно свойств столбца

В качестве первичного ключа выберем значение `place_id`. Для этого установим галочку в левой колонке РК. Нажав кнопку *Apply*, создадим таблицу.

Добавим ограничения на столбцы для созданной таблицы. Предположим, что владельцем машины может быть только кто-то из профессоров. Для этого создадим внешний ключ и соединим его с таблицей *Professors*. Для этого дважды щелкнем по созданной таблице и перейдем во вкладку *Constraints*. По кнопке «Add Item» войдем в окно добавления ограничений. Далее выставим настройки следующим образом: Name – `owner_fk`, Constraint Type – `FOREIGN KEY`, Match – `MATCH SIMPLE`. Далее в поле *Columns* выберем столбец `owner` и добавим его с помощью кнопки «Add Item». Аналогично в поле *Reference columns* выберем таблицу *Professors* и поле `professor_id`.

Подтвердите настройки, нажав кнопку *Apply*.



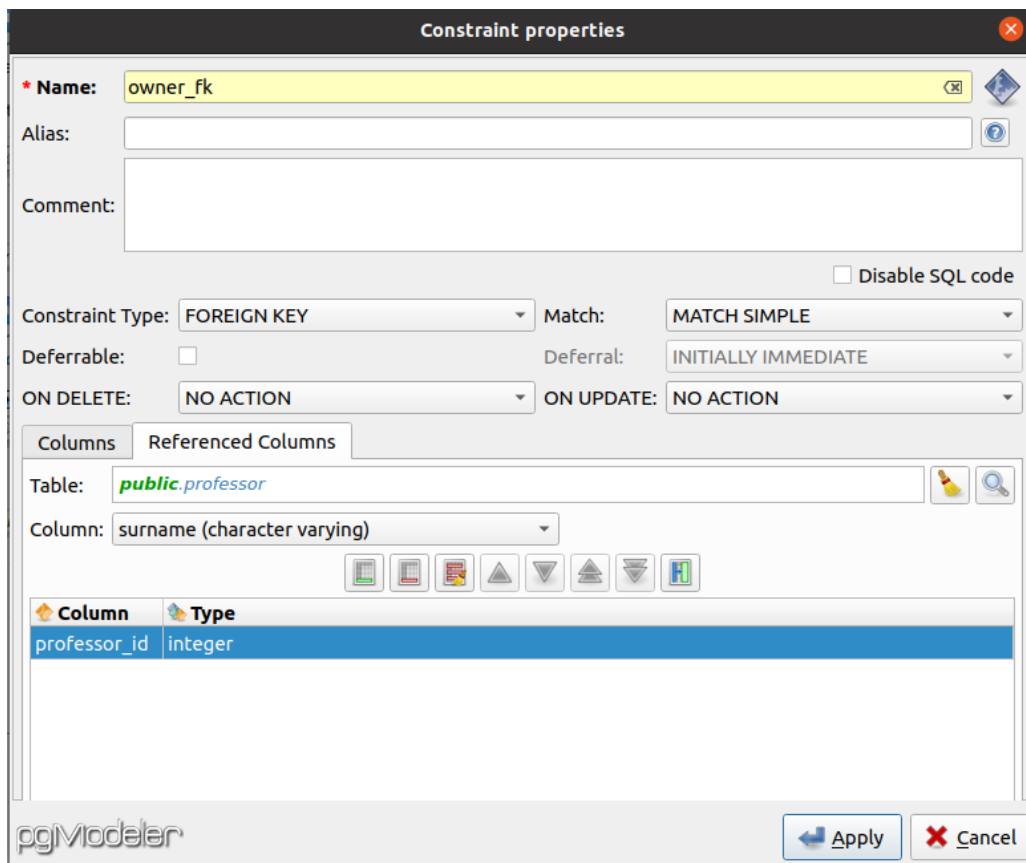


Рисунок 18 — Окно ограничений для столбцов

Для атрибута `car_number` добавим проверку на корректность введенного номера автомобиля. Создадим ограничение `car_min_valid` и выберем Constraint type CHECK. В поле ниже укажем значение регулярного выражения для российских автомобильных номеров - `car_number ~* '^[АВЕКМНОРСТУХ]\d{3}(?!000)[АВЕКМНОРСТУХ]{2}\d{2,3}$'`.

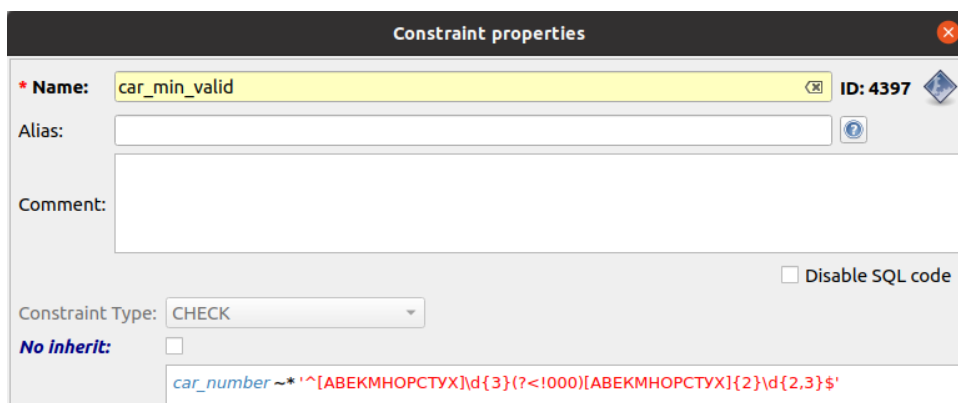


Рисунок 19 — Проверка на корректность введенного номер для атрибута car\_number

Подтвердим создание ограничения и вернемся в основное рабочее окно. Созданная таблица получила связь с таблицей преподавателей.

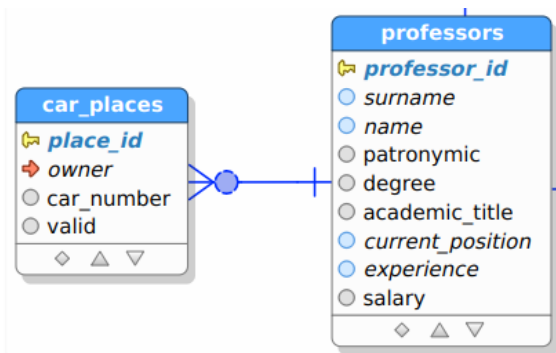


Рисунок 20 — Связь таблиц car\_places и professors

Перед тем, как экспортировать измененную модель обратно на сервер, необходимо убедиться в том, что после внесенных изменений не были нарушены ограничения, связанные с базой данных. Например, не создана таблица, имеющая аналогичное название на сервере. Для этого в нижней вкладке необходимо выбрать пункт Validation. Установим значение SQL Validation, выберем адрес сервера и нажмем кнопку Validate.

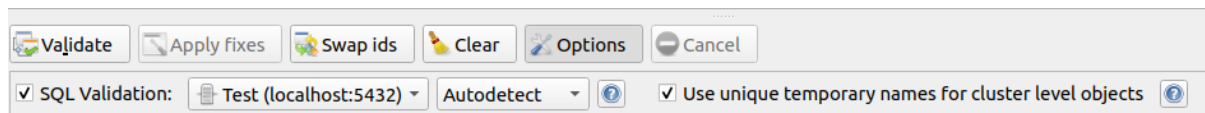


Рисунок 21 — Установка SQL Validation

Если валидация пройдет успешно, то появится сообщение «Database model successfully validated». Далее, экспортируем модель обратно на сервер. Для этого перейдем во вкладку Export, выберем сервер базы данных и установим пункт Ignore object duplicity.

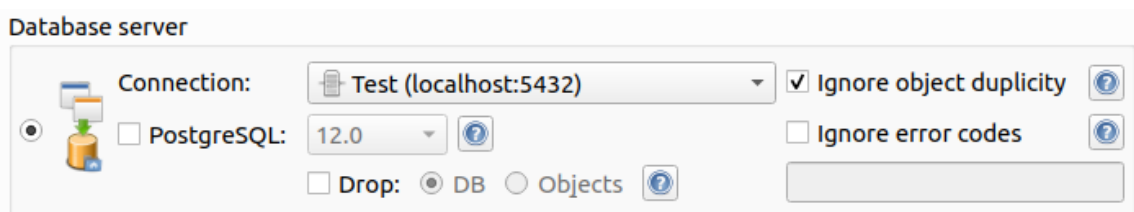


Рисунок 22 — Установка Ignore object duplicity

Нажмем на кнопку Export и обновим базу данных на сервере. Вернувшись в pgAdmin, и выполнив запрос, убедимся в том, что таблица была создана.

```
INSERT INTO car_places VALUES (1,81001,'A777TM77')
```

```
SELECT * FROM car_places
```

```
place_id | owner | car_number | valid
-----+-----+-----+-----
1 | 81001 | A777TM77 | t
```

## 2. Практическая часть

Вариант выбирается в соответствии с формулой:  $N = (N_{\text{в списке}} \bmod 10) + 1$ .

### 2.1. Задание 1.

2.1.1. В учебной базе данных одним из допущений является возможность прикрепить только одного преподавателя к дисциплине. Исправьте его.

### 2.2. Задание 2.

Отредактируйте базу данных в соответствии с вашим вариантом.

1. Добавьте в таблицу students\_id поле, содержащее цвет студенческого билета. Цвет может быть синим, белым или зеленым.
2. Добавьте в таблицу students\_id поле, содержащее статус студенческого билета: «заблокирован», «активен».
3. Добавьте в таблицу students поле «Пол», способное принимать значения «м» и «ж»
4. Добавьте в таблицу students поле «Паспорт», способное хранить значения в формате XX-XXXXXX, где X – цифра.
5. Добавьте в таблицу students поле «СНИЛС», способное хранить значения в формате XXX-XXX-XXX X, где X – цифра.
6. Добавьте в таблицу students поле «ИНН», способное хранить значения в формате XXNNAAAAAABB, где все указанные символы – цифры. XX – код субъекта РФ, может быть выбран из любых 5 субъектов, на ваш выбор.
7. Добавьте в таблицу professors поле, содержащее его контактный телефон. Сделайте ограничение, позволяющее хранить номер телефона в формате: +7(XXX)XXX-XX-XX

8. Добавьте в таблицу `students` поле, содержащее его контактный телефон. Сделайте ограничение, позволяющее хранить номер телефона в формате: 8(XXX)XXX-XX-XX

9. Добавьте в таблицу `structural_units` поле, содержащее номер аудитории подразделения. Сделайте ограничение, позволяющее хранить номер аудитории в формате: ABXX, где А может принимать значения 1,3,4; В – от 1-3. Значение XX может лежать от 00 до 39

10. Добавьте в таблицу `students_groups` поле, содержащее данные о кураторе группы. В его качестве должен выступать один из преподавателей вуза.

### **2.3. Задание 3.**

2.3.1. В соответствии с вариантом доработайте логическую модель базы данных. При доработке БД должно быть добавлено не менее трех новых таблиц. Постройте схему новой базы данных в редакторе `pgmodeler` или `Erwin`. Экспортируйте её в созданную базу данных.

1. Добавить возможность прикрепления студентов на места практики
2. Добавить возможность занятости студентов в спортивных секциях
3. Добавить студенческие объединения
4. Добавить военную кафедру
5. Добавить общежитие
6. Добавить медпункт и возможность выхода студента на больничный
7. Добавить возможность получения студентами льгот и стипендии
8. Добавить курсы повышения квалификации для преподавателей
9. Добавить портфолио студентам
10. Добавить студенческий офис

### **3. Контрольные вопросы**

1. Для чего необходимо проектирование базы данных?
2. Что такое ER-модель?
3. Перечислите известные вам типы связи?
4. Каким образом возможно физически реализовать связь «многие ко многим»?
5. Какие команды для установки ограничения вам известны?

### **4. Список использованной литературы**

[1] К. Дейт, Введение в системы баз данных, Москва: Диалектика, 2019, стр. 1316.

[2] Документация к PostgreSQL 15.1, 2022.

## Лабораторная работа №5

### «Операторы модификации базы данных»

#### 1. Теоретическая часть

В предыдущей лабораторной работе рассматривались вопросы, связанные с проектированием баз данных. После выполнения оператора CREATE TABLE создается таблица, не содержащая никаких данных. На следующем этапе ее необходимо заполнить. В данной лабораторной работе обсуждаются темы, связанные с изменением данных в таблицах – вставкой, обновлением и удалением записей.

##### 1.1. Добавление данных в таблицу

Для добавления данных в таблицу существует оператор SQL INSERT. Его сокращенный синтаксис представлен ниже.

```
INSERT INTO имя_таблицы [ ( имя_столбца1 [, ...] ) ]  
    { DEFAULT VALUES | VALUES ( { выражение1 | DEFAULT }[, ...] ) [, ...] | запрос  
    }
```

Данный оператор добавляет строки в таблицу. С помощью INSERT возможно добавить одну или несколько строк, указанных напрямую, либо ноль и более строк, возвращенных с помощью дополнительного запроса.

Параметры:

*имя\_таблицы* – таблица, в которую добавляются данные

*имя\_столбца1,...* – список столбцов в таблице, которые заполняются данными из списка выражений записанного после VALUES. Имя столбцов можно не указывать, тогда таблица начнет заполняться с крайне левого столбца и далее по порядку, пока не закончится список выражений. В случае если указаны только некоторые столбцы, остальные будут заполнены выражениями по умолчанию или NULL.

DEFAULT VALUES – все столбцы таблицы получат значение по умолчанию,

VALUES ( *выражение1*,...) – значение выражения которое будет присвоено заданным столбцам ( *имя\_столбца1*,...). Тип столбца должен соответствовать типу записываемого в него выражения.

VALUES DEFAULT – столбцу будет присвоено выражение по умолчанию. Аналогичное действие происходит при отсутствии имени столбца в списке ( *имя\_столбца1*,...), однако для наглядности часто используют именно этот формат.

*Запрос* – запрос (оператор SELECT), который формирует строки для добавления в таблицу.

Например, добавим в таблицу *Students\_groups* восемь записей.

```
INSERT INTO Students_group (students_group_number, enrolment_status,
structural_unit_number) VALUES
('ИВТ-41', 'Очная', 1),
('ИВТ-42', 'Очная', 1),
('ИВТ-43', 'Очная', 1),
('ИВТ-21В', 'Заочная', 1),
('ИБ-21', 'Очная', 3),
('ИТД-31', 'Очная', 4),
('ИТД-32', 'Очная', 4),
('ИТД-33', 'Очная', 4);
```

В качестве строк для добавления в таблицу могут быть использованы значения, сформированные в результате запроса к другой таблице. Для иллюстрации приведем следующий пример. Создадим ещё одну таблицу, которая будет содержать информацию о должниках по предметам (студентам, имеющим хотя бы одну оценку 2).

Для этого создадим еще таблицу *Debtor\_students*:

```
CREATE TABLE debtor_students
(
    id SERIAL PRIMARY KEY,
```

```

    surname VARCHAR(30) NOT NULL,
    name VARCHAR(30) NOT NULL,
    patronymic VARCHAR(30) NULL,
    group_id VARCHAR(7) NOT NULL,
    debt_number INTEGER NOT NULL
)

```

Данная таблица будет хранить информацию о ФИО и группе студента, а также о числе его долгов.

Для заполнения таблицы составим SQL запрос.

```

INSERT INTO debtor_students (surname, name, patronymic, students_group_number,
debt_number)
(
    SELECT surname, name, patronymic, students_group_number, COUNT(*) AS "Number
of debts"
    FROM student
    INNER JOIN field_comprehension ON field_comprehension.student_id =
student.student_id
    WHERE field_comprehension.mark = 2
    GROUP BY surname, name, patronymic, students_group_number
);

```

Данный запрос выбирает всех студентов, которые имеют хотя бы одну оценку 2, и записывает данные значения в созданную таблицу с должниками.

## 1.2. Изменение значений

Для изменения существующих значений в базе данных существует оператор UPDATE.

```

UPDATE имя_таблицы
SET { имя_столбца = { выражение | DEFAULT } |
    ( имя_столбца [, ...] ) = ( { выражение | DEFAULT } [, ...] ) |
    ( имя_столбца [, ...] ) = ( вложенный_SELECT )
    } [, ...]
[ FROM список_FROM ]
[ WHERE условие ]

```



После оператора UPDATE указывается целевая таблица, которая должна быть модифицирована. В предложении SET указывается, какие столбцы в выбранных строках таблицы должны быть обновлены, и для них задаются новые значения. Остальные столбцы сохраняют свои предыдущие значения. Возможно изменение строк таблицы на значения, из другой таблицы. Для этого в параметре FROM указывается её название, с помощью ключевого слова WHERE, возможно указать условие, при котором произойдет выборка.

Например, составим запрос, убирающий один долг у всех студентов групп ИВТ.

```
UPDATE debtor_students  
SET debt_number = debt_number - 1  
WHERE students_group_number LIKE 'ИВТ%';
```

### 1.3. Удаление значений

Для удаления значений из базы данных существует оператор DELETE.

```
DELETE FROM имя_таблицы [ * ] [ [ AS ] псевдоним ]  
[ WHERE условие ]  
[ RETURNING * ]
```

Оператор DELETE удаляет из указанной таблицы строки, удовлетворяющие условию WHERE. Если предложение WHERE отсутствует, она удаляет из таблицы все строки.

Составим запрос на удаление из списка должников всех студентов, не имеющих задолженностей (число долгов = 0).

```
DELETE FROM debtor_students  
WHERE debt_number = 0;
```

В случае отсутствия оператора WHERE из таблицы будут удалены все значения. Аналогичного результата возможно добиться, используя оператор

TRUNCATE. Этот оператор работает гораздо быстрее безусловной команды DELETE и полезна для больших таблиц.

```
TRUNCATE debtor_students;
```

Для вывода на экран списка удаленных строк возможно использовать параметр RETURNING \*;

## **2. Практическая часть**

Вариант выбирается в соответствии с формулой:  $N = (N_{\text{в списке}} \bmod 10) + 1$ .

### **2.1. Задание 1.**

Напишите запрос в соответствии с вашим вариантом

1. Переведите всех учащихся 3-го курса на 4-й, изменив номер группы
2. Предположим, что одна из групп была расформирована. Составьте запрос таким образом, чтобы оставшиеся студенты были распределены по другим группам потока.
3. Уволился преподаватель, имеющий наибольший оклад. Распределите его оклад поровну между преподавателями, у которых такая же должность. Уволившемуся преподавателю поставьте оклад 0. Если преподавателей, имеющих наибольший оклад несколько выберите человека с минимальным кодом преподавателя.
4. Составьте запрос, увеличивающий значение стажа преподавателя на 1 год, для тех, чья ставка меньше 0.25.
5. Увеличьте зарплату вдвое у тех преподавателей, которые преподают больше 6 дисциплин.
6. Поставьте “!” перед фамилией студента, у которого долгов более 12
7. Появилась ставка доцента. Переведите старшего преподавателя имеющего наибольший стаж на должность доцента.
8. Поставьте “!” перед фамилией преподавателя если средний бал по всем дисциплинам, которые он преподаёт меньше среднего балла по всему структурному подразделению к которому он прикреплен.
9. Пересчитайте ставки преподавателей в соответствии с средним значением (округлите до целого) ЗЕТ дисциплин, которые он ведет, из расчета, что полная ставка соответствует 6 ЗЕТ. Значение ставки округлите до сотых.
10. Рассчитайте реальные оклады преподавателей, с учетом, что полная ставка ассистента 25000 у.е., старшего преподавателя 50000 у.е., доцента 75000 у.е., профессора 100000 у.е.

## 2.2. Задание 2.

Напишите запрос в соответствии с вашим вариантом

1. Заполните поле, содержащее цвет студенческого билета следующим образом: студенты с четными номерами (кроме оканчивающихся на 0) получают синий студенческий, нечетными – белый, оканчивающиеся на 0 – зеленый.
2. Заблокируйте пропуск всем студентам, имеющим больше 4 долгов.
3. Автоматически заполните поле «Пол». Подсказка: обратите внимание на окончание отчества и фамилий студентов. Учтите отсутствие отчеств у ряда студентов.
4. Автоматически заполните поле «Паспорт» в формате, заданном в предыдущей лабораторной работе (XX-XXXXXX, где X – цифра). Первые две цифры паспорта – дата рождения студента, последние восемь – номер студенческого билета. В случае если дата состоит из одной цифры, добавьте после нее 0.
5. Автоматически заполните поле «СНИЛС» в формате, заданном в предыдущей лабораторной работе (XXX-XXX-XXX X, где X – цифра). В качестве первых трех цифр возьмите число и месяц рождения студента (если они составят строку меньше трех символов, то добавить 0, если больше – обрезать). Следующие шесть взять из номера студенческого билета, разбив его по три цифры через тире. Последнюю цифру взять сгенерировать случайно.
6. Автоматически заполните поле «ИНН» в формате, заданном в предыдущей лабораторной работе (XXXXAAAAAABV, где все указанные символы – цифры). XXXX – за код налогового органа возьмите год рождения студента, следующие шесть цифр (AAAAAA) – номер студенческого билета. Две последние контрольные цифры вычислите по правилу:

Вычислить 1-ю контрольную цифру:

- Вычислить сумму произведений цифр ИНН (с 1-й по 10-ю) на следующие коэффициенты — 7, 2, 4, 10, 3, 5, 9, 4, 6, 8 (т.е.  $7 * \text{ИНН}[1] + 2 * \text{ИНН}[2] + \dots$ ).
- Вычислить младший разряд остатка от деления полученной суммы на 11.

Вычислить 2-ю контрольную цифру:

- Вычислить сумму произведений цифр ИНН (с 1-й по 11-ю) на следующие коэффициенты — 3, 7, 2, 4, 10, 3, 5, 9, 4, 6, 8 (т.е.  $3 * \text{ИНН}[1] + 7 * \text{ИНН}[2] + \dots$ ).
- Вычислить младший разряд остатка от деления полученной суммы на 11.

7. Автоматически заполните поле телефон преподавателя в формате, заданном в предыдущей лабораторной работе (+7(XXX)XXX–XX–XX). Укажите код оператора 903, в случае четного порядкового номера преподавателя в таблице professors, иначе – 967. Следующие три цифры номера – первые три цифры кода преподавателя. Предпоследнюю пару цифр сгенерируйте рандомно, вместо последней используйте порядковый номер преподавателя в таблице professors, отсортированной по фамилии. Если порядковый номер <10, добавьте вначале 0.
8. Автоматически заполните поле телефон студента в формате, заданном в предыдущей лабораторной работе (8(XXX)XXX–XX–XX). Укажите код оператора 903, в случае четного студенческого билета, иначе – 967. Следующие три цифры номера сгенерируйте рандомно. Предпоследняя пара цифр – номер группы, последняя – количество человек в группе, в которой учится студент.
9. Автоматически заполните поле, содержащее номер аудитории подразделения в формате, установленном в предыдущей лабораторной работе (AAXX, где AA могут принимать значения 29, 25, 14, 33, 87). Выбор первых двух цифр аудитории организуйте рандомно из списка допустимых значений. Последние две цифры – количество букв полном названии подразделения.
10. Назначьте случайным образом куратора каждой группе. Для этого в соответствующее поле таблицы students\_group добавленное в предыдущей лабораторной работе запишите код преподавателя, выбранного случайным образом из таблицы professors.

### **2.3. Задание 3.**

В зависимости от варианта, добавьте значения в исправленную вами базу данных в прошлой лабораторной работе. В каждую из таблиц необходимо добавить не менее 10 осмысленных значений.

### **3. Контрольные вопросы**

1. Какие ключевые слова возможно использовать внутри оператора INSERT?
2. Каким образом возможно обновить сразу несколько значений
3. Возможно ли использовать операции объединения в операторе INSERT? Если да, то приведите пример.
4. Возможно ли изменять порядок атрибутов при вводе данных в таблицу?
5. В чем отличие между операторами DELETE и TRUNCATE?

#### **4. Список использованной литературы**

- [1] Е. П. Моргунов, PostgreSQL. Основы языка SQL, 1-е ред., Санкт-Петербург: БХВ-Петербург, 2018, стр. 336.
- [2] «PL/pgSQL — процедурный язык SQL,» [В Интернете]. Available: <https://postgrespro.ru/docs/postgresql/15/plpgsql>. [Дата обращения: 09 03 2023].
- [3] «Исходный код СУБД postgres,» [В Интернете]. Available: <https://github.com/postgres/postgres>. [Дата обращения: 30 01 2023].
- [4] Документация к PostgreSQL 15.1, 2022.
- [5] Е. Рогов, PostgreSQL изнутри, 1-е ред., Москва: ДМК Пресс, 2023, стр. 662
- [6] Б. А. Новиков, Е. А. Горшкова и Н. Г. Графеева, Основы технологии баз данных, 2-е ред., Москва: ДМК пресс, 2020, стр. 582.

## Лабораторная работа №6 «Язык программирования PL/pgSQL.

### Процедуры, функции, триггеры»

#### 1. Теоретическая часть

##### 1.1. Язык программирования PL/pgSQL

##### 1.1.1. Введение в PL/pgSQL.

До текущего раздела мы писали запросы к базе данных на языке SQL – декларативном языке программирования. Однако, в некоторых случаях, при составлении запросов со сложной логикой, было бы удобнее писать запрос в стандартном для нас императивном стиле, когда все команды выполняются последовательно. В качестве решения предлагается использовать процедурный язык для СУБД PostgreSQL - PL/pgSQL. Он является наследником языка PL/SQL, предназначенного для работы с СУБД Oracle, который в свою очередь наследует языки Ada и Pascal. Таким образом, структура программы на PL/pgSQL может напомнить структуру программы на языке Pascal.

В общем виде, структуру программы на языке PL/pgSQL можно представить следующим образом:

**DECLARE**

*Объявления переменных*

**BEGIN**

*операторы*

**END;**

Первым блоком в программе является объявление переменных, которые будут использоваться в программе. Переменные могут иметь любой тип данных, поддерживаемый PostgreSQL.

Общий синтаксис объявления переменной:

**имя [ CONSTANT ] тип [ NOT NULL ] [ { DEFAULT | := | = } выражение ]**

Параметр `DEFAULT` задает начальное значение переменной. По умолчанию, переменным присваивается значение `NULL`. В случае необходимости, переменную возможно проинициализировать значением, используя оператор «:=» для присваивания. Параметр `CONSTANT` указывает, что переменную нельзя изменять, а `NOT NULL` нельзя присваивать ей значение `NULL`.

Далее следует код программы в виде последовательно записанных операторов. Код должен быть заключен в специальные скобки из операторов `BEGIN` и `END`.

Для запуска скрипта на PL/pgSQL необходимо перед ним указать ключевое слово `do` и заключить скрипт в одинарные кавычки для того, чтобы он распознавался, как строка. Однако в большинстве случаев такой подход неудобен – все кавычки внутри скрипта придется экранировать и исчезнет встроенная подсветка синтаксиса. Поэтому для удобства, вместо кавычек используется символ `$$`. (В общем виде строковые константы можно представить в виде `$tag$<string>$tag$`, где `tag` – необязательное поле. Между подобными скобками может содержаться любое число кавычек, специальных символов, которые не нужно экранировать)

Приведем простейший скрипт на языке PL/pgSQL.

```
do
$$
BEGIN
    raise notice 'Hello, MIET';
END
$$;

NOTICE:  Hello, MIET
```

Команда *raise notice* служит для вывода служебных сообщений на экран.

Усложним скрипт, сохранив значение строки в переменной *our\_text*.

```
do
```

```

$$
DECLARE
    our_text TEXT := 'MIET';
BEGIN
    raise notice 'Hello, %', our_text;
END
$$;

```

Для вывода значения переменной с помощью *raise notice*, необходимо указать символом % место, на которое будет подставлено её значение.

```
NOTICE: Hello, MIET
```

Внутри скрипта возможно использовать стандартные команды языка SQL. Обращаться к значениям, полученным в результате подобного запроса возможно несколькими способами.

Чтобы сохранить результат выполнения запроса *select* в переменной, возможно использовать ключевое слово *into*.

Общий синтаксис запроса:

```

SELECT []
INTO variable
FROM Table

```

Приведем пример, выводящий общее количество студентов в институте:

```

do
$$
DECLARE
    num_of_students INTEGER;
BEGIN
    SELECT count(*)
    INTO num_of_students
    FROM students;
    raise notice 'Количество студентов: %', num_of_students;
END
$$;

```



Обратите внимание, что подобным образом возможно сохранить только одно значение.

Если мы хотим сохранить в переменной значение целой строки, то возможно использовать тип *record*. Данный тип сохраняет внутри себя всю полученную строку, обращение к полям которой возможно с помощью точки.

В следующем примере происходит поиск фамилии и имени студента с указанным номером студенческого билета.

```
do
$$
DECLARE
    stud record;
BEGIN
    SELECT last_name, first_name
    INTO stud
    FROM students
    WHERE student_id = 866017;
    raise notice 'Имя: %, Фамилия: %', stud.first_name, stud.last_name;
END
$$;
```

Модернизируем скрипт, сохранив значение номера студенческого билета в качестве константы. Напомним, что константа – значение, которое инициализируется в начале выполнения кода программы и не может быть изменено внутри него. Для этого, после имени переменной укажем ключевое слово *constant*.

```
do
$$
DECLARE
    stud record;
    studentid constant INTEGER := 866017;
BEGIN
    SELECT last_name, first_name
```

```

    INTO stud
    FROM students
    WHERE student_id = studentid;
    raise notice 'Имя: %, Фамилия: %', stud.first_name, stud.last_name;
END
$$;

```

Представим ситуацию, что пользователь неверно ввел значение студенческого билета и студент не был найден в таблице.

Тогда указанный выше скрипт вернет значения NULL.

```
NOTICE:  Имя: <NULL>, Фамилия: <NULL>
```

Чтобы обойти подобную ошибку, возможно использовать оператор ветвления.

### 1.1.2. Оператор ветвления

В общем виде оператор ветвления можно представить в следующем виде:

```

if condition then
    statements;
else
    alternative-statements;
END if;

```

Как и во многих других языках программирования, ветвь *else* является необязательной.

Для определения результата выполнения запроса воспользуемся переменной *found*. Она принимает значение *true*, в случае успешно выполненного запроса и *false*, если было возвращено пустое множество.

```

do
$$
DECLARE
    stud record;
    studentid constant INTEGER := 8660107;

```

```

BEGIN
    SELECT last_name, first_name
    INTO stud
    FROM students
    WHERE student_id = studentid;
    if not found then
        raise notice 'The student % could not be found', studentid;
    else
        raise notice 'Имя: %, Фамилия: %', stud.first_name, stud.last_name;
    end if;
END
$$;

```

Тогда при вводе некорректного значения будет выведено сообщение о том, что указанный студент не был найден.

```
NOTICE: The student 8660107 could not be found
```

### 1.1.3. Операторы циклов

В языке PL/pgSQL существует несколько способов задать цикл. Рассмотрим некоторые из них.

Один из самых простых способов задать цикл с условием – цикл с использованием ключевого слова **WHILE**. В общем виде цикл можно представить следующим образом:

```
WHILE логическое-выражение LOOP
```

```
    операторы
```

```
END LOOP
```

**WHILE** выполняет *операторы* до тех пор, пока истинно *логическое-выражение*. Проверка истинности осуществляется перед каждым входом в цикл.

Приведем простой пример вывода последовательных чисел на экран

```

do $$
declare
    iterator integer := 0;
begin

```

```

while iterator < 5 loop
    raise notice 'I = %', iterator;
    iterator := iterator + 1;
end loop;
end$$;

NOTICE: I = 0
NOTICE: I = 1
NOTICE: I = 2
NOTICE: I = 3
NOTICE: I = 4

```

Аналогичную задачу возможно решить с помощью цикла с заданным числом итераций **FOR**. В общем виде цикл представляет собой следующую структуру:

```

FOR имя IN [REVERSE] выражение .. выражение [BY выражение] LOOP
    операторы
END LOOP;

```

Тогда решение можно записать в виде следующего скрипта:

```

do $$
begin
    for iterator in 0..4 loop
        raise notice 'I = %', iterator;
    end loop;
end $$;

```

Для прохода по циклу в обратном порядке возможно использовать ключевое слово **REVERSE** и указания диапазона от большего числа к меньшему.

```

do $$
begin
    for iterator in REVERSE 4..0 loop
        raise notice 'I = %', iterator;
    end loop;
end $$;

```

```
NOTICE: I = 4
NOTICE: I = 3
NOTICE: I = 2
NOTICE: I = 1
NOTICE: I = 0
```

По умолчанию, за каждый шаг цикла значение итератора инкрементируется на 1. Чтобы изменить данный шаг, возможно использовать ключевое слово BY.

```
do $$
begin
  for iterator in 0..4 by 2 loop
    raise notice 'I = %', iterator;
  end loop;
end; $$
```

```
NOTICE: I = 0
NOTICE: I = 2
NOTICE: I = 4
```

В предыдущих примерах мы могли получать только одно значение из таблицы. Используя цикл, возможно проходиться по всем записям таблицы, выводить и обрабатывать их последовательно. Он называется цикл по результатам запроса.

Для его задания используется следующая конструкция

```
FOR цель IN запрос LOOP
  операторы
END LOOP;
```

Переменной *цель* последовательно присваиваются строки из результата *запроса* и для каждой выполняется цикл.

В следующем примере выведем всех студентов с именем Александр.

```
do
$$
DECLARE
  stud record;
  student_name constant TEXT := 'Александр';
```

```

BEGIN
    FOR stud IN SELECT last_name, first_name
                FROM students
                WHERE first_name = student_name
    loop
        raise notice 'Имя: %, Фамилия: %', stud.first_name, stud.last_name;
    end loop;
END
$$;

```

```

NOTICE:  Имя: Александр, Фамилия: Лобов
NOTICE:  Имя: Александр, Фамилия: Погорельцев
NOTICE:  Имя: Александр, Фамилия: Вершинин
NOTICE:  Имя: Александр, Фамилия: Заметов

```

Сделаем небольшое замечание. При выполнении запроса `SELECT` внутри скрипта результирующее значение загружается с сервера, где хранится база данных и передается на клиент, где сохраняется внутри переменной *stud* типа *record*. Представьте себе, что результат запроса вернет огромное число строк. Передача подобного объема данных займет множество ресурсов. Также, сохранять их всех внутри данной переменной не существует возможности. Поэтому, создатели языка предусмотрели определенную переменную, называемую **курсором**. С его помощью возможно получать данные от запроса порциями, не переполняя при этом память. В языке PL/pgSQL такая переменная создается автоматически при использовании цикла «*FOR цель IN запрос*». Однако, во многих других процедурных надстройках SQL такой возможности нет, поэтому необходимо создавать курсор вручную. Более подробно об этом возможно прочитать в документации [1].

Для удобства сведем наиболее популярные управляющие структуры в таблицу.

Таблица 1 — Наиболее популярные управляющие структуры

Присваивание	<b>переменная := выражение;</b>
Ветвление	<b>IF условие THEN</b> оператор; <b>ELSE</b> оператор; <b>END IF</b>
Бесконечный цикл	<b>LOOP</b> оператор; ... <b>END LOOP;</b>
Цикл while	<b>WHILE логическое-выражение LOOP</b> операторы <b>END LOOP</b>
Целочисленный цикл	<b>FOR имя IN [REVERSE] выражение .. выражение [BY выражение] LOOP</b> операторы <b>END LOOP;</b>
Цикл по результатам запроса	<b>FOR цель IN запрос LOOP</b> операторы <b>END LOOP;</b>

## 1.2. Хранимые подпрограммы

При выполнении запросов к базе данных происходит пересылка данных по сети от приложения клиента к серверу баз данных и обратно. В некоторых случаях, данная операция является затратной по времени выполнения. Для того, чтобы выполнять запросы более эффективно, возможно создать отдельную подпрограмму, которая будет выполняться в рамках процессов сервера баз данных. В СУБД PostgreSQL существуют два типа подобных подпрограмм – процедуры и функции. Их код может быть написан как на языке SQL, так и на других языках программирования – C, PL/pgSQL, Python, Tcl,

Perl, R, Java, JavaScript и др. В данном лабораторном практикуме будут рассмотрены подпрограммы, написанные на языке SQL.

Функции и процедуры имеют несколько особенностей, отличающих их друг от друга. Приведем их сравнение в виде таблицы.

Таблица 2 — Сравнение функции и процедуры

Функция	Процедура
<b>Имеет</b> возвращаемый тип и возвращает значение	<b>Не имеет</b> возвращаемого типа
Использование запросов на добавление, изменение и удаление строк <b>невозможно</b> . Разрешены только SELECT-запросы	Использование запросов на добавление, изменение и удаление строк <b>возможно</b>
Использовать транзакции <b>запрещено</b>	<b>Возможно</b> использовать операции управлением транзакциями

Рассмотрим их более подробно.

### 1.2.1. Процедуры

Процедуры – подпрограммы, которые не могут возвращать значения. В отличие от обычных запросов процедуры представляют собой объекты, уже готовые к выполнению и не требующие интерпретации. Чаще всего процедуры используются для модификации данных в таблице – добавление, изменение или удаление.

Для создания процедуры существует команда CREATE PROCEDURE. Её сокращенный синтаксис приведен ниже.

CREATE [ OR REPLACE ] PROCEDURE

```
имя ( [ [ режим_аргумента ] [ имя_аргумента ] тип_аргумента [ { DEFAULT
| = } выражение_по_умолчанию ] [, ...] ] )
{ LANGUAGE имя_языка
```



```

AS $$
...
$$;
}

```

После имени процедуры в скобках указывают аргументы. Режим\_аргумента: IN (входной), OUT (выходной), INOUT (входной и выходной) или VARIADIC (переменный). По умолчанию подразумевается IN. Далее следует имя, тип аргумента и, если параметр не задан явно, выражение, используемое для вычисления значения по умолчанию. В теле процедуры указывается язык, на котором она написана. Например, это может быть SQL, С. Команда CREATE OR REPLACE PROCEDURE создает новую или заменяет уже существующую процедуру.

Рассмотрим несколько примеров. Создадим процедуру, обновляющую значение ЗЕТ у определенной дисциплины.

```

CREATE PROCEDURE update_zet(param_field_id uuid, new_zet integer)
LANGUAGE SQL
AS
$$
    UPDATE fields SET zet = new_zet
    WHERE field_id = param_field_id
$$;

```

Для того, чтобы вызвать процедуру необходимо использовать ключевое слово CALL.

```

CALL update_zet('f81d63d6-ccd0-4cf0-a13a-340c44b852af', 5)

```

Для изменения процедуры возможно использовать ключевое слово REPLACE вместо CREATE, для удаления – DROP.

```

DROP PROCEDURE update_zet;

```

Аналогично создадим процедуру для добавления студента в базу данных.

```

CREATE PROCEDURE add_Student(Student_ID bigint,

```

datebegin	date,
dateend	date,
F	VARCHAR(30),
I	VARCHAR(30),
O	VARCHAR(30),
groupe	VARCHAR(7),
birthday	date,
email	VARCHAR(30))

LANGUAGE SQL

AS

\$\$

INSERT INTO students VALUES (Student\_ID, F, I, O, groupe, birthday, email);

INSERT INTO student\_ids VALUES (Student\_ID, datebegin, dateend);

\$\$;

### 1.2.2. Функции

Функции – подпрограммы, которые могут возвращать значения. Сокращенный синтаксис функции представлен ниже.

CREATE [ OR REPLACE ] FUNCTION

*имя* ( [ [ *режим\_аргумента* ] [ *имя\_аргумента* ] *режим\_аргумента* [ { DEFAULT | = } *выражение\_по\_умолчанию* ] [, ...] ] )

[ RETURNS *тип\_результата*

| RETURNS TABLE ( *имя\_столбца* *тип\_столбца* [, ...] ) ]

{ LANGUAGE *имя\_языка*

}

Параметры команды схожи с рассмотренными ранее параметрами CREATE PROCEDURE. Команда CREATE OR REPLACE FUNCTION применяется для замены уже существующей функции, однако надо учитывать, что невозможно изменить имя или аргументы функции. При попытке произвести такие изменения возникнет новая, независимая функция. Удобство же использования команды замены состоит в том, что при изменении функции сохраняются все ссылающиеся на нее объекты, тогда как при удалении функции нужно будет

удалить вместе с ней и все представления, триггеры, а при задании новой функции воссоздать их заново.

Параметр RETURNS тип\_результата задает возвращаемый тип данных. Если функция не возвращает значение, то используется слово void. Если в качестве режима\_аргумента используют OUT или INOUT, параметр RETURNS можно не указывать. Если же он присутствует, то должен согласовываться с режимом\_аргумента выходных аргументов.

В зависимости от количества возвращаемых значений функции могут быть скалярными и составными.

**Скалярные функции** – функции, возвращающие одно значение базового типа, например integer.

Приведем пример скалярной функции, возвращающей значение даты окончания действия студенческого билета по его номеру.

```
CREATE FUNCTION find_date(Student_ID bigint) RETURNS date
LANGUAGE SQL
AS $$
    SELECT expiration_date
    FROM student_ids
    WHERE student_id = Student_ID
$$;
```

Вызов скалярной функции происходит с помощью запроса SELECT:

```
SELECT find_date(841576)

 find_date
-----
2027-08-31
```

Функция с аргументами **составных типов** может возвращать набор некоторых значений. Далее к этой функции возможно обращаться, как к таблице и выбирать из нее данные. Для того, чтобы она возвращало множество значений, необходимо определить составной тип. Составной тип очень похож

на структуру в языке С. Он состоит из списка имён полей и соответствующих им типов данных. В общем виде, синтаксис можно представить следующим образом:

```
CREATE TYPE название AS (  
    переменная      тип,  
    переменная      тип,  
    ...  
);
```

Приведем пример следующей задачи. Необходимо написать функцию, которая выводит информацию о студентах, чей студенческий билет закончил действовать до указанной даты. Для этого создадим составной тип Student\_ID\_date.

```
CREATE TYPE Student_ID_date AS(  
    last_name varchar(30),  
    first_name varchar(30),  
    id_date date  
);
```

Далее создадим составную функцию find\_id\_date. Её основное отличие в синтаксисе от скалярной заключается в том, что функция возвращает тип Student\_ID\_date с использованием ключевого слова SETOF.

```
CREATE OR REPLACE FUNCTION find_id_date(id_date date) RETURNS SETOF  
Student_ID_date  
LANGUAGE SQL  
AS $$  
    SELECT last_name, first_name, expiration_date  
    FROM student_ids  
    INNER JOIN students ON students.student_id = student_ids.student_id  
    WHERE expiration_date < id_date  
$$;
```

Вызов функции будет совпадать с обращением к таблице:

```
SELECT * FROM find_id_date('10/09/2024');
```

last_name	first_name	id_date
Корзухина	Серафима	2024-08-31
Голубков	Сергей	2024-08-31
Чарнота	Григорий	2024-08-31
Хлудов	Роман	2024-08-31

### 1.3. Триггеры

Для упрощения работы с базой данных было бы полезно совершать некоторые постоянные автоматические действия при наступлении определенного события. Например, вести записи обо всех изменениях в таблице с оценками студентов. Или автоматически рассчитывать поле, содержащее количество должников при получении студентом двойки. Для подобной работы используются триггеры.

В общем случае синтаксис команды создания триггера выглядит следующим образом:

```
CREATE [ CONSTRAINT ] TRIGGER имя { BEFORE | AFTER | INSTEAD OF } { событие [ OR ... ] }  
ON имя_таблицы  
[ FOR [ EACH ] { ROW | STATEMENT } ]  
[ WHEN ( условие ) ]  
EXECUTE PROCEDURE имя_функции ( аргументы )
```

Триггер – подпрограмма, автоматически выполняющаяся при наступлении заданного события. Созданный триггер будет связан с таблицей (*имя\_таблицы*) и будет выполнять заданную функцию (*имя\_функции* (*аргументы*)) при наступлении определенного события. Допустимые *события*:

INSERT  
UPDATE [ OF *имя\_столбца* [, ... ] ]  
DELETE  
TRUNCATE

Триггер может выполняться до указанного события (*BEFORE*), после него (*AFTER*) или вместо (*INSTEAD OF*). Обратите внимание, что *INSTEAD OF* работает только в представлениях. В качестве события может быть операция добавления (*INSERT*), обновления (*UPDATE*) или удаления значений (*DELETE*). [1]

Существуют два типа триггеров – построчные и операторные. Построчные триггеры (параметр *FOR EACH ROW*) вызываются один раз для каждой строки, с которой произошло изменение. Операторный триггер (*FOR EACH STATEMENT*) срабатывает только один раз, при запуске оператора, независимо от количества строк (даже если таких строк нет).

С помощью параметра *CONSTRAINT* команда создаёт *триггер ограничения*. Это обычный триггер, время срабатывания которого можно изменить командой *SET CONSTRAINTS*. [2]

Логическое выражение *WHEN ( условие )* определяет при каких условиях выполняется функция триггера. В триггерах *FOR EACH ROW* условие *WHEN* может ссылаться на значения столбца в старой и/или новой строке, в виде *OLD.имя\_столбца* и *NEW.имя\_столбца*, соответственно. Триггеры *INSTEAD OF* не поддерживают условия *WHEN*.

Создание триггера происходит в два этапа – создание триггерной функции и прикрепление её к определенной таблице. Триггерная функция может быть написана только на одном из процедурных языков программирования, поддерживаемых PostgreSQL, например PL/pgSQL.

### 1.3.1. Создание триггерной функции

В общем виде триггерную функцию можно представить в следующем виде:

```
CREATE FUNCTION trigger_function()  
RETURNS TRIGGER  
LANGUAGE PLPGSQL
```

```

AS
$$
BEGIN
    -- trigger logic
END;
$$

```

Триггерная функция получает данные об изменениях в базе данных через специальную структуру *TriggerData*, содержащую набор переменных. Наиболее часто используемые переменные – NEW и OLD, содержащие строки до и после выполнения события, вызвавшего триггер.

Например, создадим триггерную функцию, сохраняющую информацию об изменениях студентами фамилий. Перед этим создадим таблицу, которая будет хранить общую информацию об изменениях.

```

CREATE TABLE sys_log (
    id INT GENERATED ALWAYS AS IDENTITY,
    info TEXT NOT NULL
);

```

С помощью параметра GENERATED ALWAYS AS IDENTITY задается столбец идентификации, последовательность целых чисел, автоматически увеличивающаяся на единицу при добавлении записи.

Создадим функцию:

```

CREATE OR REPLACE FUNCTION log_last_name_changes()
RETURNS TRIGGER
LANGUAGE PLPGSQL
AS
$$
BEGIN
    IF NEW.last_name <> OLD.last_name THEN
        INSERT INTO sys_log(info)
            VALUES(concat(OLD.last_name, ' сменил фамилию на ',NEW.last_name));
    END IF;

```

```
RETURN NEW;  
END;  
$$
```

### 1.3.2. Прикрепления триггерной функции к таблице.

Для прикрепления триггерной функции к таблице создадим непосредственно сам триггер.

```
CREATE OR REPLACE TRIGGER last_name_changes  
BEFORE UPDATE  
ON students  
FOR EACH ROW  
EXECUTE PROCEDURE log_last_name_changes();
```

Теперь, при изменении фамилии любого студента информация об этом будет сохранена в таблице *sys\_log*.

## 2. Практическая часть

### 2.1. Задание 1.

Напишите скрипт на языке *PL/pgSQL*, вычисляющий среднюю оценку студента. Аналогичный запрос напишите на языке *SQL*. Сравните время выполнения работы в обоих случаях. Для расчета времени выполнения скрипта, запустите его в терминале *psql*, перед этим запустив таймер с помощью команды *\timing*. Для того, чтобы отключить таймер после окончания работы, выполните команду *\timing off*.

### 2.2. Задание 2.

Напишите *SQL* запросы к учебной базе данных в соответствии с вариантом. Вариант к практической части выбирается по формуле:  $V = (N \% 10) + 1$ , где *N* – номер в списке группы, % - остаток от деления.



№ варианта	№ запросов
1	1, 11, 21, 31, 41, 51, 61
2	2, 12, 22, 32, 42, 52, 62
3	3, 13, 23, 33, 43, 53, 63
4	4, 14, 24, 34, 44, 54, 64
5	5, 15, 25, 35, 45, 55, 65
6	6, 16, 26, 36, 46, 56, 66
7	7, 17, 27, 37, 47, 57, 67
8	8, 18, 28, 38, 48, 58, 68
9	9, 19, 29, 39, 49, 59, 69
10	10, 20, 30, 40, 50, 60, 70

## Сборник запросов к учебной базе данных

### *Скрипты на языке PL/pgSQL*

1. Напишите скрипт, выводящий количество всех оценок 5, 4, 3, 2
2. Напишите скрипт, в результате работы которого будет выведено ФИО преподавателя, его ставка и реальная занятость – сумма ЗЕТ всех читаемых им дисциплин
3. Напишите скрипт, формирующий таблицу со средним баллом по каждой дисциплине у преподавателей Института МПСУ
4. Напишите скрипт, который возвращает фамилию, имя студентов со счастливым студенческим билетом (сумма первых трех цифр номера билета совпадает с суммой последних трех)
5. Напишите скрипт, генерирующий случайным образом вариант контрольной работы для каждого студента определенной группы. Первая цифра варианта – сумма цифр номера группы, остальные две выбираются случайно.
6. Напишите скрипт, выводящий ФИО студента и его пол.
7. Напишите скрипт, который считает частоту появления каждого имени студента
8. Напишите скрипт, определяющий, сколько дней рождений приходится на каждый из месяцев
9. Напишите скрипт определяющий знак зодиака каждого студента. Выведите Фамилию, имя, дату рождения и знак зодиака.
10. Напишите скрипт, вычисляющий количество студентов, у которых совпадает имя и отчество (Например, Сергей Сергеевич).
11. Напишите скрипт, шифрующий с помощью шифра Цезаря все названия дисциплин.
12. Напишите скрипт, генерирующий случайное местоположение каждого студента на карте (его координаты в градусах).
13. Напишите скрипт, который случайным образом разбивает студентов на пары для вальса (мальчик/девочка). Мальчикам, которым не хватило пары в поле партнерши поставить значение «Teddy bear :(»
14. Напишите скрипт, отбирающий случайным образом 10 студентов из списка студентов 3-го курса, имеющих больше 10 двоек для участия в субботнике. Выведите фамилию имя и количество двоек у студента.
15. Найти двух студентов, имеющих максимальное совпадение имени, фамилии и отчества.
16. Напишите скрипт, генерирующий случайное местоположение каждого студента на карте (его координаты в градусах). Определите материк,

соответствующий этим координатам. Считайте, попаданием на материк, если координаты лежат в пределах:

Европа – между 1° с.ш. и 77° с.ш. и между 9° з.д. и 67° в.д.

Африка – между 37° с.ш. и 34° ю.ш. и между 13° з.д. и 51° в.д.

Австралия – между 10° ю.ш. и 39° ю.ш. и между 113° в.д. и 153° в.д.

Северная Америка – между 7° с.ш. и 71° с.ш. и между 55° з.д. и 168° з.д.

Южная Америка – между 12° с.ш. и 53° ю.ш. и между 34° з.д. и 81° з.д.

Антарктида – ниже 63° ю.ш.

17. Напишите скрипт, выводящий фамилию, имя, дату рождения и знак зодиака студентов совместимых с вами по знаку зодиака. Совместимость найти в любом гороскопе.

18. Напишите скрипт имитирующий шахматный турнир. Выберите случайным образом 10 студентов. Каждый играет с каждым по одному разу. Результат игры между двумя участниками выбирается случайно из победы (победивший получает 2 очка, проигравший 0) и ничьи (каждому добавляется по одному баллу). Вывести результат каждой игры (фамилии участников и результат игры) и итоговую таблицу всех участников с суммой полученных баллов в порядке убывания очков.

19. Напишите скрипт имитирующий шахматный турнир. Выберите случайным образом 10 студентов. Каждый играет с каждым по одному разу. Результат игры между двумя участниками выбирается случайно из победы (победивший получает 2 очка, проигравший 0) и ничьи (каждому добавляется по одному баллу). После проведения первого тура отсеиваются 2 участника набравшие наименьшее количество очков. Проводится 2 тур между оставшимися 8 студентами. И так продолжать до тех пор, пока не останутся два победителя. Вывести результаты каждой игры каждого тура (фамилии участников и результат игры) и также для каждого тура итоговые таблицы всех участников с суммой полученных баллов в порядке убывания очков.

20. Написать скрипт, имитирующий пересдачу экзамена студентами. Для одного (любого) преподавателя выбрать студентов, имеющих долги дисциплинам, которые он ведет. “Пересдача” будет состоять в сравнении суммы цифр id преподавателя и студента (если у студента сумма больше – пересдал, иначе нет). Суммирование проводить до тех пор, пока не останется одна цифра (898899 -> 51 -> 6). В начале списка вывести фамилию, имя, id преподавателя. Далее вывести id студентов, информацию о пересдаче и суммы цифр id студента и преподавателя.

*Процедуры на языке SQL*

21. Создайте процедуру перемещения студента из одной группы в другую
22. Создайте процедуру изменения курса в определенной группе. Входные параметры – курс, номер группы.
23. Создайте процедуру увеличения зарплаты преподавателя в зависимости от стажа. Стаж до 10 лет – увеличиваем на **a**, от 11 до 20 – на **2a**, более на **3a** рублей. Где **a** входной параметр процедуры
24. Создайте процедуру, изменяющую преподавателя у данной дисциплины. Входные параметры – id нового преподавателя, название дисциплины).
25. Создайте процедуру изменяющую оценку у студента по определенной дисциплине. Входные параметры – id студента, название дисциплины, новая оценка.
26. Создайте процедуру увеличивающую на один бал оценку студентов именинников по определенной дисциплине. Входной параметр - название дисциплины
27. Создайте процедуру изменения почты у студента.
28. Создайте процедуру продления студенческих билетов у определенной группы на 1 год. Номер группы вводится в качестве параметра.
29. Создайте процедуру продления студенческих билетов у определенной группы на 1 год. Входной параметр - номер группы.
30. Создайте процедуру изменения ставки у преподавателя. Учтите ограничения –  $0 < \text{ставка} \leq 2$ , содержит не более двух цифр после запятой и последняя цифра либо 0, либо 5. . В случае если входной параметр не удовлетворяет диапазону – ставку не менять, последняя цифра не 0, и не 5 – округлить по правилам. Входной параметр – id преподавателя, ставка.

### *Функции на языке SQL*

31. Создайте функцию, которая выводит количество всех студентов по определенной группе
32. Создайте функцию, которая выводит среднюю оценку по всем предметам у студента (по id студента)
33. Создайте функцию, которая определяет разницу в возрасте между двумя студентами
34. Создайте функцию, рассчитывающую среднюю зарплату преподавателей в определенном структурном подразделении.
35. Создайте функцию, рассчитывающую дисперсию оценки у конкретного студента
36. Создайте функцию, рассчитывающую моду оценки у конкретного студента

37. Создайте функцию, возвращающую фамилию случайного студента
38. Создайте функцию, которая будет рассчитывать зарплату преподавателя, по id преподавателя
39. Создайте функцию, которая считает количество преподавателей, в структурном подразделении с номером N. N вводится в качестве параметра.
40. Создайте функцию, вычисляющую средний возраст студентов определенной группы
41. Создайте функцию, возвращающую самое близкую по звучанию фамилию к введенной в качестве параметра. (См. *soundex*. Для использования данной функции необходимо подключить дополнение *fuzzystrmatch* с помощью команды `CREATE EXTENSION fuzzystrmatch;`)
42. Создайте функцию, вычисляющую средний возраст студентов для каждой группы.
43. Создайте функцию, рассчитывающую среднюю зарплату преподавателей в каждом структурном подразделении
44. Создайте функцию, выводящую всех однофамильцев определенного студента. Выведите девушек и юношей с аналогичной фамилией, а также их группу.
45. Создайте функцию, рассчитывающую средний стаж преподавателей в каждом структурном подразделении.
46. Создайте функцию, вычисляющую число преподавателей в каждом структурном подразделении.
47. Создайте функцию, выводящую всех студентов именинников в определенном диапазоне. Входные параметры – дата рождения и количество дней диапазона (плюс, минус).
48. Создайте функцию, которая выведет студентов, у которых количество двоек превышает заданное значение.
49. Создайте функцию, выводящую всех преподавателей, преподающих в определенном структурном подразделении.
50. Создайте функцию, выводящую фамилию определенного преподавателя и название всех дисциплин, которые он читает в определенном семестре. Входные данные – id преподавателя, семестр.

### *Триггеры*

51. Создайте триггер запрещающий понижать оценки студентов.

52. Создайте триггер, который запрещает добавление или изменение дня рождения студента, возраст которого превышает 100 лет.
53. Создайте триггер, который запрещает добавление и изменение даты выдачи студенческого билета, у которого дата выдачи > даты по которую он действителен.
54. Создайте триггер, который запрещает изменение в структурных подразделениях
55. Создайте триггер, который выводит сообщение “Are your sure about that?” при добавлении студента John Cena.
56. Создайте триггер, выводящий сообщение об изменении оценок у студентов. Сообщение должно содержать id студента, старую и новую оценку.
57. Создайте триггер, выводящий сообщение об изменении оценок у студентов. Сообщение должно содержать фамилию студента, предмет, старую и новую оценку.
58. Создайте триггер, который не позволяет поставить оценку 2 студенту, чье имя совпадает с вашим.
59. Создайте триггер, запрещающий ставить студенту больше 4 двоек.
60. Создайте триггер, блокирующий изменение и добавление email, у которых в домене отсутствует «.ru».
61. Преподаватель в течение нескольких лет проводил занятия в командировке в другой стране. Каждый год работы зарплата повышалась на 10%. Создайте триггер, который при увеличении стажа на время работы в командировке увеличивает зарплату преподавателя.
62. Создайте триггер, не позволяющий вводить дубли в таблицу преподавателей (совпадение фамилии, имени и отчества с точностью до двух букв в каждом). Используйте функцию *Levenshtein()*.
63. Создайте триггер, не позволяющий вводить дубли в таблицу студентов (совпадение фамилии, имени и отчества с точностью до двух букв в каждом). В предупреждении выведите ФИО студента с похожими данными. Используйте функцию *Levenshtein()*.
64. Создайте триггер, сохраняющий информацию о изменении зарплаты преподавателей и дату изменения.

65. Создайте триггер, который после удаления преподавателя сохраняет информацию об этом в таблицу (ФИО и текущую дату).
66. Создать триггер, который при уменьшении заработной платы преподавателя до 0 удаляет запись о нем.
67. Создать триггер, который при выставлении пятой и более двойки удаляет студента из базы. Запись об удалении заносится в информационную таблицу.
68. Создайте триггер, который запрещает ввод латинских букв в фамилии, имени и отчестве и выводит соответствующее предупреждение.
69. Создайте триггер, сохраняющий информацию об изменениях оценок у студентов.
70. Создайте триггер, проверяющий уникальность руководителя подразделения. Учтите, что фамилия, имя, отчество руководителя в поле *head\_of\_the\_unit* могут быть переставлены. Используйте функции *position*, *regexp\_split\_to\_array*.

### **2.3. Задание 3.**

Для добавленной в 4-й лабораторной работе таблицы создайте любой триггер.

### **3. Контрольные вопросы**

1. В чем отличие между процедурой и функцией?
2. Что такое курсоры?
3. Для каких целей используют триггеры?
4. Возможно ли написать триггерную функцию на языке SQL?
5. Для чего нужны представления?

### **4. Список использованной литературы**

- [1] «Postgrespro - 40.7. Курсоры,» [В Интернете]. Available: <https://postgrespro.ru/docs/postgrespro/10/plpgsql-cursors>.
- [2] Документация к PostgreSQL 15.1, 2022.

## **Лабораторная работа №7 «Индексы, транзакции»**

### **1. Теоретическая часть**

В предыдущих лабораторных работах было показано как спроектировать БД, создать таблицы и заполнить их требуемой информацией. В данной работе рассмотрим средства, оптимизирующие и ускоряющие работу с таблицами – введем понятия индекса и транзакции.

#### **1.1. Индексы**

При работе с базами данных очень часто необходимо выполнять задачи, связанные с поиском строк в таблицах. Для ускорения подобных запросов используются индексы.

Индекс – специальная структура данных, которая связана с таблицей и создается на основе<sup>1</sup> данных, содержащихся в ней. Основная цель создания индексов – повышение производительности функционирования базы данных.  
[1]

В общем случае индексы представляют собой дополнительную структуру, содержащую значение индексируемого атрибута и указатель на данный элемент. Все записи в индексе являются упорядоченными, поэтому поиск данных значительно ускоряется.

В качестве примера можно привести алфавитный указатель, расположенный в конце книги. Обычно, указатель упорядочен по алфавиту и помимо списка определений содержит страницы, на которых располагается о них информация. Когда мы хотим найти значение некоторого слова, то обращаемся к указателю, быстро находим требуемое значение и переходим на страницу, указанную рядом с ним. Таким образом, отпадает необходимость последовательного просмотра всех страниц в книге.

В данном примере книга – это база данных, индекс – алфавитный указатель.



На практике индексы применяются, когда объемы базы данных существенно велики. Недостатком применения индексов является то, что они занимают отдельное место на диске и требуют накладных расходов для поддержания их в актуальном состоянии при выполнении обновления таблицы.

В предыдущих работах рассматривалось задание первичного ключа PRIMARY KEY. При его реализации СУБД сама автоматически создает индекс. Однако, при разработке БД очень часто требуется создавать дополнительные индексы, с учетом наиболее частых запросов.

Создание индекса происходит с помощью следующей команды, записанной в сокращенном виде:

```
CREATE [ UNIQUE ] INDEX имя_индекса ON имя_таблицы  
    ( { имя_столбца | ( выражение ) }  
[ ASC | DESC ] [ NULLS { FIRST | LAST } ] [, ...] )
```

Параметр UNIQUE отслеживает уникальность при создании индекса и контролирует добавление или изменение записи, выдавая ошибку при появлении повторяющихся значений. Указание *имя\_индекса* необязательно, PostgreSQL формирует подходящее имя по имени родительской таблицы и именам индексируемых столбцов.

При создании индекса обязательно указывается *имя столбца* или *выражение* с одним или несколькими столбцами, по которому строится индекс. Параметр *ASC* указывает порядок сортировки по возрастанию (подразумевается по умолчанию), *DESC* - по убыванию, *NULLS FIRST/ LAST* указывает, что значения NULL после сортировки оказываются перед/после остальных.

Например, создание индекса для атрибута «Номер группы» таблицы Студент происходит следующим образом.

```
CREATE INDEX id_index ON student (students_group_number);
```

Таким образом, при больших объемах базы данных, запросы, содержащие ключевое слово WHERE, будут выполняться быстрее.

## 1.2. Анализ выполнения запросов

Для разработки эффективного SQL запроса необходимо учитывать множество факторов, влияющих на его работу – применяемые планировщиком запросов алгоритмы, количество строк в таблице, используемые типы данных и т. п. Для того, чтобы получить поэтапный алгоритм выполнения запроса, составленный СУБД возможно провести анализ запроса и вывести план его выполнения. В PostgreSQL для этого существуют ключевые слова EXPLAIN ANALYZE. Например, с помощью команды

```
EXPLAIN ANALYZE SELECT * FROM students;
```

на экран выводится следующий план выполнения запроса:

```
-----  
QUERY PLAN  
-----  
Seq Scan on students (cost=0.00..12.83 rows=483 width=87) (actual time=0.010..0.036 rows=483 loops=1)  
Planning Time: 0.051 ms  
Execution Time: 0.053 ms  
(3 rows)
```

Планы запросов анализируются снизу вверх. Реальное время выполнения данного запроса – 0,053 мс. По алгоритмам Postgres время должно было составить 0,051 мс.

Более интересна первая строка. В ней указан алгоритм, по которому проводилось чтение строк – *seq scan* – последовательный поиск. Далее указывается таблица, с которой ведется работа и в скобках анализируемые параметры. Cost – «стоимость», численная оценка предполагаемого количества требуемых ресурсов для вычисления запроса. Первая оценка (0.00) – число ресурсов для вывода первой строки и последняя (12.83) – для вывода всего массива данных. Значение стоимостей берется из анализа накопленной статистики, значений сложности вычисления используемых алгоритмов и системных настроек [2]. Далее указывается приблизительное число строк, которое будет выведено в результате запроса (483) и приблизительная их

средний размер. Во вторых круглых скобках аналогично указывается предположительное время работы.

При усложнении запроса усложняется и план выполнения.

Рассмотрим запрос, в котором наложено условие фильтрации по полю с именем студента.

```
EXPLAIN ANALYZE SELECT * FROM students
WHERE first_name='Иван';
```

```

                                QUERY PLAN
-----
Seq Scan on students  (cost=0.00..14.04 rows=1 width=87) (actual time=0.047..0.047 rows=0 loops=1)
  Filter: ((first_name)::text = 'Иван'::text)
  Rows Removed by Filter: 483
Planning Time: 0.075 ms
Execution Time: 0.057 ms
(5 rows)
```

Аналогично, анализируем план снизу-вверх. Первым делом была применена операция фильтрации и с её помощью было удалено из потенциально выводимых значений 483 строки. Далее происходит последовательная выборка оставшихся строк и их пересылка клиенту. Обратите внимание, что в текущем варианте, значение rows=1, хотя на самом деле, число вернувшихся строк должно быть равно 25. Это связано с тем, что на текущий момент еще не накопилась статистика в СУБД по выполнению данного запроса и оптимизатор допустил ошибку. Для того, чтобы избежать этой ошибки, нужно еще несколько раз запустить запрос.

### 1.3. Транзакции

При работе с базами данных часто требуется выполнять несколько запросов последовательно. Например, при добавлении информации о студенте в учебную базу данных необходимо произвести две записи с помощью запроса INSERT – вставку информации в таблицу *students* и связанную с ней *student\_ids*. В случае, если произойдет сбой при выполнении одного из запросов, а второй выполнится, то в базе данных будет храниться лишь частичная информация о студенте. Для борьбы с подобными ошибками используются **транзакции**.

Транзакции – совокупность операций над базой данных, которые вместе образуют логически целостную процедуру, и могут быть либо выполнены все вместе, либо не будет выполнена ни одна из них. [1]

Для корректности работы системы транзакции должны обладать следующими свойствами:

1. Атомарность (**Atomicity**) – все операции внутри транзакции гарантированно должны выполняться или не выполняться ни одна из них.
2. Согласованность (**Consistency**) – база данных в результате выполнения транзакции переходит из одного согласованного состояния в другое
3. Изолированность (**Isolation**) – во время выполнения транзакции, другие транзакции должны по возможности минимально влиять на её ход работы
4. Долговечность (**Durability**) – после завершения работы транзакции данные должны быть надежно сохранены в базе данных

Еще одно применение транзакций – параллельная работа с базой данных нескольких пользователей. Если они одновременно попытаются изменить одну и ту же запись, то будет неизвестно, какое изменение в итоге сохранится. Попытка же заблокировать запись при вводе для остальных пользователей и организация последовательного выполнения транзакций, приведет к сильной потере производительности системы. Поэтому в СУБД PostgreSQL реализованы параллельно исполняемые транзакции. Реализация транзакций основана на многоверсионной модели (MVCC), которая предполагает, что каждый пользователь видит так называемый «снимок» данных, то есть согласованную версию БД, которую она имела на определенный момент времени. Когда параллельные транзакции вносят изменения в одну и ту же строку, на самом деле создаются отдельные версии этих строк, доступных определенной транзакции. Для конкретных задач может требоваться различная степень независимости параллельных транзакций. Поэтому, существует понятие уровня изоляции транзакции. Каждый уровень характеризуется перечнем не допустимых ошибок при параллельных транзакциях.

Рассмотрим возможные ошибки, которые могут возникнуть при параллельном выполнении транзакций:

1. Потерянное обновление. Представим ситуацию, когда две транзакции одновременно изменяют одни и те же данные. В итоге сохранено будет только одно из двух значений, без учета изменений второй транзакции.
2. «Грязное» чтение. Первая транзакция прочитала данные, которые только что изменила и еще не зафиксировала вторая транзакция. Если в итоге если вторая транзакция была отменена, то первой были прочитаны некорректные значения.
3. Неповторяющееся чтение. В ходе одной транзакции происходит два чтения одних и тех же данных. Между ними происходит запись и фиксация этих же данных другой транзакцией. Таким образом, первая транзакция прочитает два разных значения.
4. Фантомное чтение. В ходе первой транзакции происходит выборка строк из таблицы. Вторая транзакция производит изменения значений в данной таблице. Таким образом, если в рамках первой транзакции снова будет выполнен запрос на выборку, то результат будет отличаться от первого.
5. Аномалия сериализации. В данном случае результат выполнения нескольких транзакций последовательно (при любом порядке их следования) не будет совпадать с зафиксированным результатом параллельного их выполнения.

В стандарте SQL предусматривается четыре уровня изоляции. Для простоты сведем их в таблицу.

Таблица 1 Уровни изоляции транзакций

Уровень изоляции	«Грязное» чтение	Неповторяемое чтение	Фантомное чтение	Аномалия сериализации
Read uncommitted	+ (в PostgreSQL НЕ допускается)	+	+	+
Read committed	-	+	+	+
Repeatable read	-	-	+ (в PostgreSQL НЕ допускается)	+
Serializable	-	-	-	-

Обратите внимание, что в PostgreSQL некоторые уровни изоляции устроены несколько строже, чем указано в стандарте языка SQL. Таким образом, уровни Read committed и Read uncommitted в PostgreSQL совпадают.

По умолчанию, PostgreSQL использует уровень изоляции Read committed. Для просмотра текущего уровня изоляции в PostgreSQL используется команда

```
SHOW transaction_isolation;
```

В PostgreSQL транзакция определяется набором SQL-команд, окружённым командами BEGIN и COMMIT.

```
BEGIN;  
-- ...  
COMMIT;
```

Если в процессе выполнения транзакции мы решим, что не хотим фиксировать её изменения, то возможно выполнить команду ROLLBACK вместо COMMIT, чтобы все наши изменения были отменены. Для задания конкретного уровня изоляции транзакции необходимо использовать конструкцию

```
BEGIN ISOLATION LEVEL название уровня изоляции
```

Например,

```
BEGIN ISOLATION LEVEL REPEATABLE READ;
```

Рассмотрим следующий пример:

Запустим на выполнение транзакцию и выведем содержимое атрибута «Ставка» из таблицы «Трудоустройство» на экран.

```
BEGIN;  
SELECT wage_rate FROM employment;
```

```
wage_rate  
-----  
0.25  
0.75  
0.15  
0.50
```

```
0.50
1.00
1.50
0.25
```

Обновим значение ставки и выведем новые значения на экран.

```
UPDATE employment
SET wage_rate = wage_rate * 1.10;
SELECT wage_rate FROM employment;
```

```
wage_rate
-----
0.28
0.83
0.17
0.55
0.55
1.10
1.65
0.28
```

Откроем еще один экземпляр Query tool и выполним из-под нового клиента запрос на вывод ставок.

```
SELECT wage_rate FROM employment;
```

```
wage_rate
-----
0.25
0.75
0.15
0.50
0.50
1.00
1.50
0.25
```

Обратите внимание, что т.к. транзакция не была завершена, то значения ставок еще не изменились. Для её завершения выполним команду COMMIT от первого клиента

```
COMMIT
```

Повторим запрос со второго клиента, и убедимся, что число ставок обновилось.

```
wage_rate
-----
    0.28
    0.83
    0.17
    0.55
    0.55
    1.10
    1.65
    0.28
```

Вернем обратно значения ставок. Для этого выполним запрос:

```
UPDATE employment
SET wage_rate = wage_rate / 1.10;
```

```
wage_rate
-----
    0.25
    0.75
    0.15
    0.50
    0.50
    1.00
    1.50
    0.25
```

Повторим предыдущую транзакцию, только вместе команды COMMIT выполним команду ROLLBACK.

```
BEGIN;
UPDATE employment
SET wage_rate = wage_rate * 1.10;
ROLLBACK;
```

Убедимся в том, что значения не были изменены в результате транзакции.

## 2. Практическая часть

### 2.1. Задание 1.



Для глубокого анализа данного задания вам необходимо ознакомиться с рекомендованной литературой, указанной в источниках.

#### 2.1.1. Исследование производительности системы

Создайте таблицу, содержащую значения посещаемости студентом института. Таблица содержит номер студенческого билета, время его входа, выхода и сгенерированное случайное кодовое число при выходе из вуза.

```
CREATE TABLE attendance (  
    attendance_id SERIAL PRIMARY KEY,  
    generated_code VARCHAR(64),  
    person_id integer,  
    enter_time timestamp,  
    exit_time timestamp,  
    FOREIGN KEY (person_id) REFERENCES student_ids (student_id)  
);
```

С помощью следующего скрипта заполните таблицу данными.

```
do  
$$  
DECLARE  
    enter_time timestamp(0);  
    exit_time timestamp(0);  
    person_id integer;  
    enter_id VARCHAR(64);  
BEGIN  
    FOR i IN 1..1000000 LOOP  
  
        -- Генерируем случайную дату в указанном диапазоне  
  
        enter_time := to_timestamp(random() *  
            (  
                extract(epoch from '2023-12-31'::date) -  
                extract(epoch from '2023-01-01'::date)  
            )  
            + extract(epoch from '2023-01-01'::date)  
        );  
  
        -- Генерируем случайный интервал времени, который пробыл в вузе студент (не более 10 часов)
```

```

exit_time := enter_time + (floor(random() * 36000 + 1)*'1 SECOND'::interval);

    person_id := (
        SELECT student_id FROM students
        ORDER BY random()
        LIMIT 1
    );

enter_id := md5(random()::text);

    INSERT INTO attendance(generated_code, person_id, enter_time,exit_time)
    VALUES(enter_id, person_id, enter_time, exit_time);

END LOOP;
END
$$;

```

Добавьте в таблицу *attendance* одно значение, измерив время данной операции. Далее измерьте время выполнения запроса, выводящего содержимого таблицы в отсортированном виде по столбцу *generated\_code*.

Добавьте индекс на столбец *generated\_code*. Повторите предыдущие две операции. Сравните полученное время. Во сколько раз оно изменилось? Результаты вычисления занесите в таблицу.

	Время до индексирования Т <sub>б</sub>	Время после индексирования Т <sub>а</sub>	Т <sub>а</sub> / Т <sub>б</sub>
SELECT			
INSERT			

### 2.1.2. Индексы и селективность

Выполните запрос, выводящий все строки таблицы *attendance*, измерьте время его выполнения. Добавьте условие, выбрав только все записи, связанные с одним конкретным студентом. Аналогично измерьте время выполнения. Создайте индекс на атрибут *person\_id* и повторите эксперименты. Сравните

время выполнения операций до создания индекса и после. Объясните полученный результат.

	Время до индексирования $T_b$	Время после индексирования $T_a$	$T_a / T_b$
SELECT			
SELECT + WHERE			

### 2.1.3. Анализ плана выполнения запроса

Составьте запрос к таблице *attendance*, выводящий все строки в отсортированном порядке, в которых столбец *generated\_code* заканчивается символом 'a'. Проанализируйте полученный запрос и объясните результат. Используется ли в данном случае индекс?

## 2.2. Задание 2.

Предположим, что студент группы ИВТ-42 Полиграф Шариков во время зимней сессии пересдал экзамен по дисциплине «Операционные системы» на оценку 5 и пересдал экзамен по дисциплине «Базы данных» на 5. Одновременно с проставлением баллов за его успехами следила методист кафедры. Для работы с несколькими транзакциями запустите два командных окна (запросника). В первом вводите команды за преподавателя, проставляющего оценки, а во втором за методиста, просматривающего результаты.

### 2.2.1. Работа с транзакциями

В рамках транзакции измените значение оценки студента по Операционным системам и проверьте значение в первом и втором окне. Зафиксируйте изменения и вновь проверьте значения. Аналогично внесите новую оценку по Базам данных и проверьте изменения.

Преподаватель

Методист

BEGIN;	Смотрит результат до фиксации изменений преподавателем
Изменяет оценку	
Добавляет оценку	Смотрит результат после фиксации изменений преподавателем
COMMIT	

### 2.2.2. Отмена изменений транзакций

Удалите добавленное значение и верните исправленную оценку в прежнее состояние. Повторите аналогичные действия, только по окончании внесения изменений преподавателем откатите их с помощью команды ROLLBACK. Какое значение увидела методист?

### 2.2.3. Моделирование аномалий при выполнении транзакций

Повторите эксперименты в п. 0, используя различные уровни изоляции.

Преподаватель	Методист
BEGIN ISOLATION LEVEL ...	BEGIN ISOLATION LEVEL ...
Изменяет оценку	Смотрит результат до фиксации изменений преподавателем
Добавляет оценку	Смотрит результат после фиксации изменений преподавателем
COMMIT	

Внесите в таблицу в какой момент были получены ошибочные значения из-за аномалий.

Уровень изоляции	До фиксации	После фиксации
Read uncommitted		
Read committed		
Repeatable read		
Serializable		

Как вы считаете, какой уровень изоляции необходимо использовать на практике и почему?

### 2.3. Задание 3.

Проанализируйте учебную базу данных и проиндексируйте одно из полей любой таблицы. Объясните свой выбор.

### 3. Контрольные вопросы

1. За счет чего индексы ускоряют выборку данных?
2. Существуют ли случаи, когда использование индексов замедляет выборку данных?
3. Какая структура данных хранит в себе индексные записи?
4. Для чего предназначены транзакции?
5. В чем отличие между неповторяющимся и фантомным чтением?

#### **4. Список использованной литературы**

- [1] Е. П. Моргунов, PostgreSQL. Основы языка SQL, 1-е ред., Санкт-Петербург: БХВ-Петербург, 2018, стр. 336.
- [2] Г. Домбровская, Б. Новиков и А. Бейликова, Оптимизация запросов в PostgreSQL, Москва: ДМА, 2022.
- [3] «Исходный код СУБД postgres,» [В Интернете]. Available: <https://github.com/postgres/postgres>. [Дата обращения: 30 01 2023].
- [4] Документация к PostgreSQL 15.1, 2022.
- [5] Е. Рогов, PostgreSQL изнутри, 1-е ред., Москва: ДМК Пресс, 2023, стр. 662 .
- [6] Б. А. Новиков, Е. А. Горшкова и Н. Г. Графеева, Основы технологии баз данных, 2-е ред., Москва: ДМК пресс, 2020, стр. 582.

# Лабораторная работа №8

## «Разработка программы на языке C

### для работы с базой данных»

#### 1. Теоретическая часть

В современном мире базы данных редко встречаются отдельно от программного обеспечения, позволяющего с ними взаимодействовать. Подобные программы разрабатываются на разных языках программирования и служат для корректного взаимодействия между пользователем и непосредственно базой данных. В данной лабораторной работе будет рассмотрено создание программного интерфейса для работы с базой данных. Программа будет разработана на языке C.

#### 1.1. Создание проекта

##### 1.1.1. Настройка среды

При разработке программного обеспечения требуется библиотека для взаимодействия с *postgresql*. В текущей лабораторной работе будет использоваться библиотека *libpq*. Проверить ее наличие в системе можно с помощью команды:

```
dpkg -s libpq-dev | grep Version
```

В случае отсутствия библиотеки, необходимо выполнить следующие команды:

```
sudo apt-get update  
sudo apt-get -y install libpq-dev
```

Для работы с программным кодом удобно использовать программу Visual Studio Code.

Запустив Visual Studio Code, создайте новый файл (*Ctrl+N* или *File -> New File*). Сохраните его под именем *main.c* в новую директорию, в которой будут располагаться все файлы проекта.

Запишите в него следующий код:

```

1: #include <stdio.h>
2:
3: int main() {
4:     printf("I love MPSU!\n");
5:     return 0;
6: }

```

Для компиляции программы создайте файл сборки – *Makefile*. Для этого в директории проекта создайте файл *Makefile* и запишите в него следующий код:

```

7: outfile=main
8: all:
9:     gcc -o $(outfile) main.c
10: clean:
11:     rm -rf $(outfile)

```

Для запуска компиляции откройте командную строку, перейдите в директорию проекта и выполните команду *make*. Для удобства открыть её возможно с использованием программы *vscode*. Для этого в нижней панели перейдите во вкладку **TERMINAL**. Если панели нет, то вытяните её за верхний край.

После выполнения команды *make* в директории проекта должен появиться исполняемый файл *main*. Запустите его. В случае успеха, на экране должна появиться надпись “I love MPSU!”. Чтобы с помощью одной команды скомпилировать и запустить файл возможно выполнить следующую команду:

```
make && ./main
```

### 1.1.2. Функции библиотеки *libpq*

Для взаимодействия программного обеспечения с базой данных используется библиотека *libpq*. Она содержит в себе набор функций, позволяющих подключаться к базе данных, выполнять SQL запросы и производить другие действия.

Ниже приведены основные функции

Таблица 1 — Основные функции

Название функции	Описание действия
------------------	-------------------

PGconn *PQconnectdbParams( const char * const *keywords, const char * const *values, int expand_dbname);	Подключение к базе данных.
PGresult *PQexec(PGconn *conn, const char *command);	Передача команды серверу и ожидание результата.
PGresult *PQexecParams( PGconn *conn, const char *command, int nParams, const Oid *paramTypes, const char * const *paramValues, const int *paramLengths, const int *paramFormats, int resultFormat);	Передача команды серверу и ожидание результата. Имеет возможность передать параметры отдельно от текста SQL-команды.
void PQclear(PGresult *res);	Освобождение области памяти, выделенной под PGresult
int PQntuples(const PGresult *res);	Возвращает число строк (кортежей) в полученной выборке.
int PQnfields(const PGresult *res);	Возвращает число столбцов (полей) в каждой строке полученной выборки.
ExecStatusType PQresultStatus(const PGresult *res);	Возвращает статус выполнения команды.
char *PQerrorMessage(const PGconn *conn);	Возвращает сообщение об ошибке.
void PQfinish(PGconn *conn);	Закрывает соединение с сервером. Освобождает память, используемую объектом PGconn.

Более подробно об этих функциях можно прочитать в документации [1]

### 1.1.3. Разработка программного обеспечения для работы с базой данных

Измените файл *main.c*, записав в него следующий программный код.



```

1: #include <stdio.h>
2: #include <stdlib.h>
3: #include <string.h>
4: #include <libpq-fe.h>
5:
6: /*
7:      Функция выхода из программы с сообщением об ошибке.
8:  */
9: void err_exit(PGconn *conn) {
10:     PQfinish(conn); //Заккрытие соединения с сервером базы данных
11:     exit(1);        //Функция возвращает значение 1 -> программа завершилась
    аварийно
12: }
13:
14: /*
15:      Функция печати результата запроса на экран
16:      *conn -> дескриптор подключения к серверу
17:      query -> текст запроса
18:  */
19: void print_query(PGconn *conn, char* query){
20:     PGresult *res = PQexec(conn, query); //Передача запроса на сервер
21:     int rows = PQntuples(res);           //Получение числа строк в результате
    запроса
22:     int cols = PQnfields(res);           //Получение числа столбцов в результате
    запроса
23:
24:
25:     // Проверка наличия возвращенных запросом строк.
26:     // В случае их отсутствия - аварийное завершение программы
27:     if (PQresultStatus(res) != PGRES_TUPLES_OK) {
28:         printf("No data retrieved\n");
29:         PQclear(res);
30:         err_exit(conn);
31:     }
32:
33:     //Вывод результата запроса на экран
34:     for(int i=0; i<rows; i++) {
35:         for(int j=0; j<cols; j++) {
36:             printf("%s ", PQgetvalue(res, i, j));
37:         }
38:         printf("\n");
39:     }
40:     PQclear(res);
41: }
42:
43: int main() {
44:     //Подключение к серверу базы данных
45:     PGconn *conn = PQconnectdb("user=SAB password=123456 dbname=students");
46:
47:     //Проверка статуса подключения. В случае ошибки - аварийное завершение
    программы
48:     if (PQstatus(conn) == CONNECTION_BAD) {
49:         fprintf(stderr, "Connection to database failed:
    %s\n", PQerrorMessage(conn));
50:         err_exit(conn);
51:     }
52:     print_query(conn, "SELECT * FROM student LIMIT 5;");
53:     PQfinish(conn);
54:     return 0;

```

```
55: }  
56:
```

Данный код осуществляет подключение к серверу БД students, выполнение требуемого запроса (выборки пяти записей из БД) и вывод результатов на экран пользователя.

Подключение к запущенному серверу PostgreSQL происходит к базе данных – students, пользователь – SAB, пароль для подключения – 123456. Данный пользователь был создан в одной из предыдущих лабораторных работ. Если этого не было сделано, то создайте его, чтобы было возможно подключиться к базе данных. Для этого выполните следующий запрос.

```
CREATE ROLE "SAB" WITH  
    LOGIN  
    SUPERUSER  
    CREATEDB  
    CONNECTION LIMIT -1  
    PASSWORD '123456';
```

Для компиляции программы измените *Makefile*:

```
1: libpath=-I/usr/include/postgresql  
2: libs=-lpq  
3: outfile=main  
4: all:  
5:     gcc -o $(outfile) main.c $(libpath) $(libs)  
6: clean:  
7:     rm -rf $(outfile)
```

В случае успешного подключения и выполнения запроса в консоль будет выведена информация о студентах из учебной базы данных.

## 1.2. SQL-инъекции

В процессе разработки приложений уделяется большое внимание вопросам безопасности. Проблема SQL-инъекций является одной из угроз сохранению и утечки данных. Рассмотрим эту проблему на нашем примере.

Добавим в программу функцию, позволяющую выводить на экран оценку студента, по его номеру студенческого билета и названию дисциплины.

```
1:  /*
2:      Функция поиска и вывода на экран информации о студенте из базы
3:      по номеру его студенческого билета и названию дисциплины
4:      *conn -> дескриптор подключения к серверу
5:      st_id  -> номер студенческого билета
6:      f_name -> название дисциплины
7:  */
8:  void print_students_by_ID(PGconn *conn, int st_id, char* f_name)
9:  {
10:     // создание динамического массива, для помещения в него запроса
11:     char* query = (char*)malloc(512*sizeof(char));
12:     // Удаление символа окончания строки из введенной фамилии студента
13:     int pos = strlen(f_name) - 1;
14:     if (f_name[pos] == '\n')
15:         f_name[pos]='\0';
16:
17:     // Запрос, возвращающий ФИ, название дисциплины и оценку студента
18:     // значения st_id и f_name подставляются в запрос
19:     // на место %d и %s соответственно
20:     sprintf(query, "
21: SELECT  SURNAME,
22:         NAME,
23:         FIELD_NAME,
24:         MARK
25: FROM STUDENT S
26: JOIN PUBLIC.FIELD_COMPREHENSION F_C ON S.STUDENT_ID = F_C.STUDENT_ID
27: JOIN PUBLIC.FIELD F ON F.FIELD_ID = F_C.FIELD
28: WHERE S.STUDENT_ID = %d AND F.FIELD_NAME = \'%s\'", st_id, f_name);
29:
30:     PGresult *res = PQexec(conn, query);
31:
32:     if (PQresultStatus(res) != PGRES_TUPLES_OK) {
33:         printf("No data retrieved\n");
34:         PQclear(res);
35:         err_exit(conn);
36:     }
37:     // Вывод на экран найенной информации
38:     int rows = PQntuples(res);
39:     int cols = PQnfields(res);
40:
41:     for(int i=0; i<rows; i++) {
42:         for(int j=0; j<cols; j++) {
43:             printf("%s ", PQgetvalue(res, i, j));
44:         }
45:         printf("\n");
46:     }
47:     PQclear(res);
48:     free(query);
49: }
```

Для обращения к ней заменим в основной программе main() строку с вызовом функции *print\_query(...)* на последовательность команд:

```

1:     int st_id;
2:     printf("Enter student ID number: \n");
3:     scanf("%d", &st_id);
4:     getchar();
5:     printf("Enter field name: \n");
6:     fgets(query, 256, stdin);
7:
8:     print_students_by_ID(conn, st_id, query);

```

В этой части кода происходит ввод номера студенческого билета и строки, содержащей название дисциплины. (Единственным «подводным камнем» является наличие функции *getchar()* между командами ввода. Проблема заключается в том, что функция *scanf* считывает все символы до символа окончания строки, остающемся в буфере ввода. И при вызове *fgets* данный символ попадает в строку *query* и функция завершает свою работу, т.к. считывание происходит до символа окончания строки включая его. Поэтому, с помощью функции *getchar* символ окончания строки считывается и не попадает в *fgets*.) Далее следует вызов функции *print\_students\_by\_ID()*. На вход функция принимает значение дескриптора соединения, номер студенческого билета и название дисциплины.

При корректном вызове функции программа выберет из базы данных студента и выведет его оценку по дисциплине на экран.

```

Enter student ID number:
856271
Enter field name:
Базы данных
Михеев Александр Базы данных 2

```

Однако, злоумышленники при вводе требуемой информации (в нашем случае при вводе названия дисциплины) могут задать определенное значение, которое позволит им получить доступ к закрытым данным или внести изменения в БД (в нашем случае изменить оценку).

Попробуем ввести в программу вместо названия дисциплины следующее значение:

```
1' OR 1=1; UPDATE FIELD_COMPREHENSION F_C SET MARK = 5 FROM FIELD F WHERE
F.FIELD_ID = F_C.FIELD AND STUDENT_ID = 856271 AND FIELD_NAME = 'Базы данных' ; -
-
```

Строка вывода программы будет выглядеть следующим образом:

```
Enter student ID number:
856271
Enter field name:
1' OR 1=1; UPDATE FIELD_COMPREHENSION F_C SET MARK = 5 FROM FIELD F WHERE
F.FIELD_ID = F_C.FIELD AND STUDENT_ID = 856271 AND FIELD_NAME = 'Базы
данных' ;--
No data retrieved
```

Если запустить программу еще раз, введя данные студента 856271 и дисциплину «Базы данных», то оценка будет уже другая.

Михеев Александр Базы данных 5

Давайте разберемся как это произошло. В функцию *print\_students\_by\_ID()* в качестве входных аргументов передаются введенные пользователем `student_ID` и `field_name` и их значения подставляются в формируемый внутри функции запрос.

Рассмотрим получившийся в результате слияния строк SQL запрос.

```
SELECT      SURNAME,
NAME,
FIELD_NAME,
MARK
FROM STUDENT S
JOIN PUBLIC.FIELD_COMPREHENSION F_C ON S.STUDENT_ID = F_C.STUDENT_ID
JOIN PUBLIC.FIELD F ON F.FIELD_ID = F_C.FIELD
WHERE S.STUDENT_ID = 856271 AND F.FIELD_NAME =      '
1' OR 1=1;
UPDATE FIELD_COMPREHENSION F_C SET MARK = 5
FROM FIELD F
WHERE F.FIELD_ID = F_C.FIELD AND STUDENT_ID = 856271 AND FIELD_NAME = 'Базы
данных' ;
--      ,
```

Вставленная строка (выделена курсивом) будет расположена между кавычками. Выражение `F.FIELD_NAME = '1' OR 1=1` всегда истинно, поэтому для студента с заданным `student_ID` будет выполнена команда, идущая после точки с запятой в этой строке. Произойдет изменение оценки данного студента на 5 по дисциплине «Базы данных». В самом конце с помощью символа «--» будет закомментирован апостроф, завершающий строку. Таким образом с помощью внесенного SQL кода (*SQL инъекции*) возможно произвести изменение в базе данных.

### 1.3. Защита от SQL-инъекций

Самым простым способом защиты от SQL инъекций является проверка введенного значения на корректность. Например, если вводится число, то необходимо убедиться, что введенное значение именно число. Также существует возможность ограничить набор допустимых символов. При разработке графического интерфейса можно минимизировать ввод данных в поля, заменив его на выбор из вариантов.

Чаще всего в качестве защиты от SQL инъекций выбирают один из следующих способов:

#### 1. Изменение кавычек на апострофы.

Если для пользователя не так важен вводимый символ, то возможно изменить символ `'` на ```. Таким образом вредоносный код будет принят как одна длинная строка и добавлен в базу данных.

Приведем функцию, изменяющую символ апострофа.

```
1:  /*
2:      Функция изменения символа апострофа
3:      *query -> текст запроса
4:  */
5:  void correct_apostrof(char* query){
6:      for(int i = 0; i < strlen(query); ++i)
7:          if(query[i] == '\')
8:              query[i] = '`';
9:  }
```

#### 2. Экранирование кавычек

Если для пользователя важен именно символ кавычки, то его для ввода в базу данных необходимо экранировать. Напомним, что кавычки в PostgreSQL экранируются повторением данного символа. Например, если мы хотим внести в таблицу строку имя John O'Brien, то запрос на добавление будет выглядеть следующим образом:

```
INSERT INTO ApostropheTest(Name, Surname)
VALUES ('John', 'O''Brien')
```

	name character varying (20)	surname character varying (20)
1	John	O'Brien

Приведем возможный вариант программной реализации функции.

```
1:  /*
2:      Функция экранирования кавычек
3:      *query -> текст запроса
4:      *query_new -> текст запроса после изменения
5:  */
6:  void add_escape_quotes(char* query, char* query_new){
7:      int j = 0;
8:      for(int i = 0; i < strlen(query); ++i){
9:          if(query[i] == '\\'){
10:             query_new[j] = '\\';
11:             query_new[j+1] = '\\';
12:             j++;
13:          }
14:          else
15:             query_new[j] = query[i];
16:             j++;
17:      }
18:  }
```

### 3. Использование параметризованных запросов.

С помощью функции *PQexecParams* возможно отправить на сервер подготовленный запрос. Функция очень похожа на *PQexec*, но предлагает дополнительную функциональность: значения параметров могут быть указаны отдельно от самой строки-команды. Это позволяет избежать использования кавычек и экранирующих символов, что является трудоёмким методом, часто приводящим к ошибкам. *PQexecParams* позволяет включать не более одной SQL-команды в строку запроса. (В ней могут содержаться точки с запятой, однако

может присутствовать не более одной непустой команды.) Таким образом, выполнение атак с использованием SQL-инъекций становится гораздо более трудоемкой задачей.

Ниже приведен программный код реализации функции поиска оценок студента, но с использованием параметризованных запросов. Значения передаются в функцию с помощью параметров \$1 и \$2.



```

1:  /*
2:      Функция поиска информации о студенте из базы
3:      по его номеру студенческого билета и названию дисциплины
4:      с использованием параметрического запроса
5:      *conn -> дескриптор подключения к серверу
6:      st_id -> номер студенческого билета
7:      f_name -> название дисциплины
8:  */
9:  void print_students_by_ID_Params(PGconn *conn, int st_id, char* f_name)
10: {
11:     //Создание массива для задания параметров запроса
12:     const char *paramValues[2];
13:     paramValues[0] = (char*) &st_id;
14:     paramValues[1] = f_name;
15:
16:     char *query = "SELECT  SURNAME,          \
17:                   NAME,          \
18:                   FIELD_NAME,    \
19:                   MARK           \
20: FROM STUDENT S          \
21:     JOIN PUBLIC.FIELD_COMPREHENSION F_C ON S.STUDENT_ID = F_C.STUDENT_ID \
22:     JOIN PUBLIC.FIELD F ON F.FIELD_ID = F_C.FIELD                        \
23: WHERE S.STUDENT_ID=$1 AND F.FIELD_NAME=\'$2\'";
24:
25:     //Вызов параметрического запроса
26:     PGresult *res = PQexecParams(conn, query, 2, NULL, paramValues, NULL, NULL,
27: 0);
28:
29:     if (PQresultStatus(res) != PGRES_TUPLES_OK) {
30:         printf("No data retrieved\n");
31:         PQclear(res);
32:         err_exit(conn);
33:     }
34:
35:     int rows = PQntuples(res);
36:     int cols = PQnfields(res);
37:     for(int i=0; i<rows; i++) {
38:         for(int j=0; j<cols; j++) {
39:             printf("%s ", PQgetvalue(res, i, j));
40:         }
41:         printf("\n");
42:     }
43:     PQclear(res);
44: }

```

## 2. Практическая часть

### 2.1. Задание 1.

#### *Защита от SQL инъекций*

Используя функции для защиты от SQL инъекций, повторите пример из теоретической части и убедитесь в том, что она работает.

### 2.2. Задание 2.

#### *Реализация функции для взаимодействия с базой данных*

Реализуйте любой из SQL запросов, который вы выполняли в одной из предыдущих работ в виде функции на языке C, взаимодействующей с базой данных. Предусмотрите защиту от SQL инъекции любым из приведенных выше способов.

#### *Предупреждение об инъекции*

Добавьте функцию, срабатывающую в случае попытки внедрения SQL инъекции и выводящее значение, что текущему пользователю вынесено предупреждение.

#### *Использование представлений*

Замените вызов SQL запроса из программного кода на вызов представления. Объясните преимущества и недостатки такого подхода.

### **2.3. Задание 3.**

#### *Доработка программного обеспечения – добавление аутентификации*

Создайте две роли – «Ваши инициалы junior» и «Ваши инициалы admin». Junior обладает только правом чтения из базы данных, admin – полные права. Добавьте в программу поле ввода логина/пароля для подключения к базе данных от имени различных пользователей. Типы пользователей – администратор, пользователь.

#### *Доработка программного обеспечения – добавление CRUD операций*

От имени администратора сделайте возможность добавлять, изменять и удалять значения оценок у любого студента.

От имени пользователя сделайте возможность выполнять запрос на просмотр всех его оценок и поиск конкретной строки по введенным значениям фамилии, имени и номера группы. Используя разработанную функцию, попробуйте выполнить SQL инъекцию на изменение значение оценки от имени данного пользователя.

### **3. Контрольные вопросы**

1. Как возможно установить соединение с базой данных на языке C?
2. Что такое SQL инъекция?

3. Какие методы защиты от SQL инъекций вы знаете?
4. Для чего предназначены функции PQclear и PQfinish
5. В чем преимущества использования параметризованных запросов?

#### **4. Список использованной литературы**

- [1] «Документация libpq,» [В Интернете]. Available: <https://postgrespro.ru/docs/postgresql/9.6/libpq>.
- [2] «Исходный код СУБД postgres,» [В Интернете]. Available: <https://github.com/postgres/postgres>. [Дата обращения: 30 01 2023].
- [3] Документация к PostgreSQL 15.1, 2022.
- [4] Е. Рогов, PostgreSQL изнутри, 1-е ред., Москва: ДМК Пресс, 2023, стр. 662 .
- [5] Б. А. Новиков, Е. А. Горшкова и Н. Г. Графеева, Основы технологии баз данных, 2-е ред., Москва: ДМК пресс, 2020, стр. 582.
- [6] Е. П. Моргунов, PostgreSQL. Основы языка SQL, 1-е ред., Санкт-Петербург: БХВ-Петербург, 2018, стр. 336.