Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет МГТУ им. Н.Э. Баумана)

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»

Лабораторные работы №1-6

Курс «Базы данных»

Выполнил:

студент группы №ИУ5-41Б

Цыпышев Т.А.

Проверил:

Ковалева Н.А.

2024 г.

Лабораторная работа №1.

«Создание базы данных в СУБД PostgreSQL. Основы программирования на языке SQL.»

Цель работы:

Получить теоретические и практические навыки создания базы данных в СУБД PostgreSQL, изучить основные понятия и операторы, научиться работать в среде pgAdmin, сформировать знания и умения по программирования на языке SQL, приобрести практические навыки работы со средствами языка SQL для обновления, удаления и вставки данных в БД.

Ход работы:

Создадим таблицы из задания:

```
CREATE TABLE customers(
 IdCustomer INT PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
 CompanyName VARCHAR(100),
 LastName VARCHAR(50) NOT NULL,
 FirstName VARCHAR(50) NOT NULL,
 Adress VARCHAR(250),
 City VARCHAR(50),
 IndexCode VARCHAR(20),
 Phone VARCHAR(20),
 EMail VARCHAR(255)
);
ALTER TABLE customers ADD CONSTRAINT unique_last_firstname UNIQUE (LastName, FirstName);

CREATE TABLE orders (
 IdOrder INT PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
 IdCustomer INT NOT NULL REFERENCES customers(IdCustomer),
 OrderDate date NOT NULL default current_date,
 ShipDate date,
 PaidDate date,
 Status char(1) CHECK (Status IN ('C', 'P', 'A'))
);

CREATE TABLE products (
 IdProduct INT PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
 PrName VARCHAR(150) NOT NULL,
 PrPrice money,
 InStock INT,
 ReOrder BOOLEAN,
 Description TEXT
```

```
);

CREATE TABLE items (
 ItemID INT PRIMARY KEY GENERATED BY DEFAULT AS IDENTITY,
 OrderID INT NOT NULL REFERENCES orders(IdOrder),
 ProductID INT NOT NULL REFERENCES products(IDProduct),
 Quantity INT NOT NULL,
 Total MONEY
);
```

Заполним таблицы данными:

```
-- Вставка данных в таблицу customers
INSERT INTO customers (CompanyName, LastName, FirstName, Adress, City,
IndexCode, Phone, EMail)
VALUES
('ABC Corporation', 'Smith', 'John', '123 Main St', 'New York', '10001',
'555-1234', 'john@abccorp.com'),
('XYZ Enterprises', 'Johnson', 'Emily', '456 Elm St', 'Los Angeles', '90001',
'555-5678', 'emily@xyz.com'),
('LMN Industries', 'Williams', 'Michael', '789 Oak St', 'Chicago', '60601',
'555-9012', 'michael@lmnindustries.com'),
('DEF Limited', 'Brown', 'Jessica', '101 Pine St', 'Houston', '77002',
'555-3456', 'jessica@def.com'),
('GHI Inc.', 'Jones', 'Daniel', '202 Maple St', 'Miami', '33101', '555-7890',
'daniel@ghiinc.com');

-- Вставка данных в таблицу orders
INSERT INTO orders (IdCustomer, OrderDate, ShipDate, PaidDate, Status)
VALUES
(1, '2024-03-01', '2024-03-03', '2024-03-04', 'C'),
(2, '2024-03-02', '2024-03-04', '2024-03-05', 'P'),
(3, '2024-03-03', '2024-03-05', '2024-03-06', 'A'),
(4, '2024-03-04', '2024-03-06', '2024-03-07', 'C'),
(5, '2024-03-05', '2024-03-07', '2024-03-08', 'P');

-- Вставка данных в таблицу products
INSERT INTO products (PrName, PrPrice, InStock, ReOrder, Description)
VALUES
('Product A', 10.99, 100, TRUE, 'Description of Product A'),
('Product B', 20.49, 75, FALSE, 'Description of Product B'),
('Product C', 15.29, 50, TRUE, 'Description of Product C'),
('Product D', 8.99, 120, FALSE, 'Description of Product D'),
('Product E', 25.99, 90, TRUE, 'Description of Product E');
```

```
-- Вставка данных в таблицу items
INSERT INTO items (OrderID, ProductID, Quantity, Total)
VALUES
(1, 1, 5, 54.95),
(2, 3, 3, 45.87),
(3, 2, 2, 40.98),
(4, 4, 4, 35.96),
(5, 5, 1, 25.99);
```

Получим таблицы:

customers

	idcustomer [PK] integer	companyname character varying (100)	lastname character varying (50)	firstname character varying (50)	adress character varying (250)	city character varying (50)	indexcode character varying (20)	phone character varying (20)	email character varying (255)
1	1	ABC Corporation	Smith	John	123 Main St	New York	10001	555-1234	john@abccorp.com
2	2	XYZ Enterprises	Johnson	Emily	456 Elm St	Los Angeles	90001	555-5678	emily@xyz.com
3	3	LMN Industries	Williams	Michael	789 Oak St	Chicago	60601	555-9012	michael@lmnindustries.com
4	4	DEF Limited	Brown	Jessica	101 Pine St	Houston	77002	555-3456	jessica@def.com
5	5	GHI Inc.	Jones	Daniel	202 Maple St	Miami	33101	555-7890	daniel@ghiinc.com

orders

	idorder [PK] integer	idcustomer integer	orderdate date	shipdate date	paiddate date	status character
1	1	1	2024-03-01	2024-03-03	2024-03-04	C
2	2	2	2024-03-02	2024-03-04	2024-03-05	P
3	3	3	2024-03-03	2024-03-05	2024-03-06	A
4	4	4	2024-03-04	2024-03-06	2024-03-07	C
5	5	5	2024-03-05	2024-03-07	2024-03-08	P
6	6	5	2024-03-05	2024-03-07	2024-03-08	P

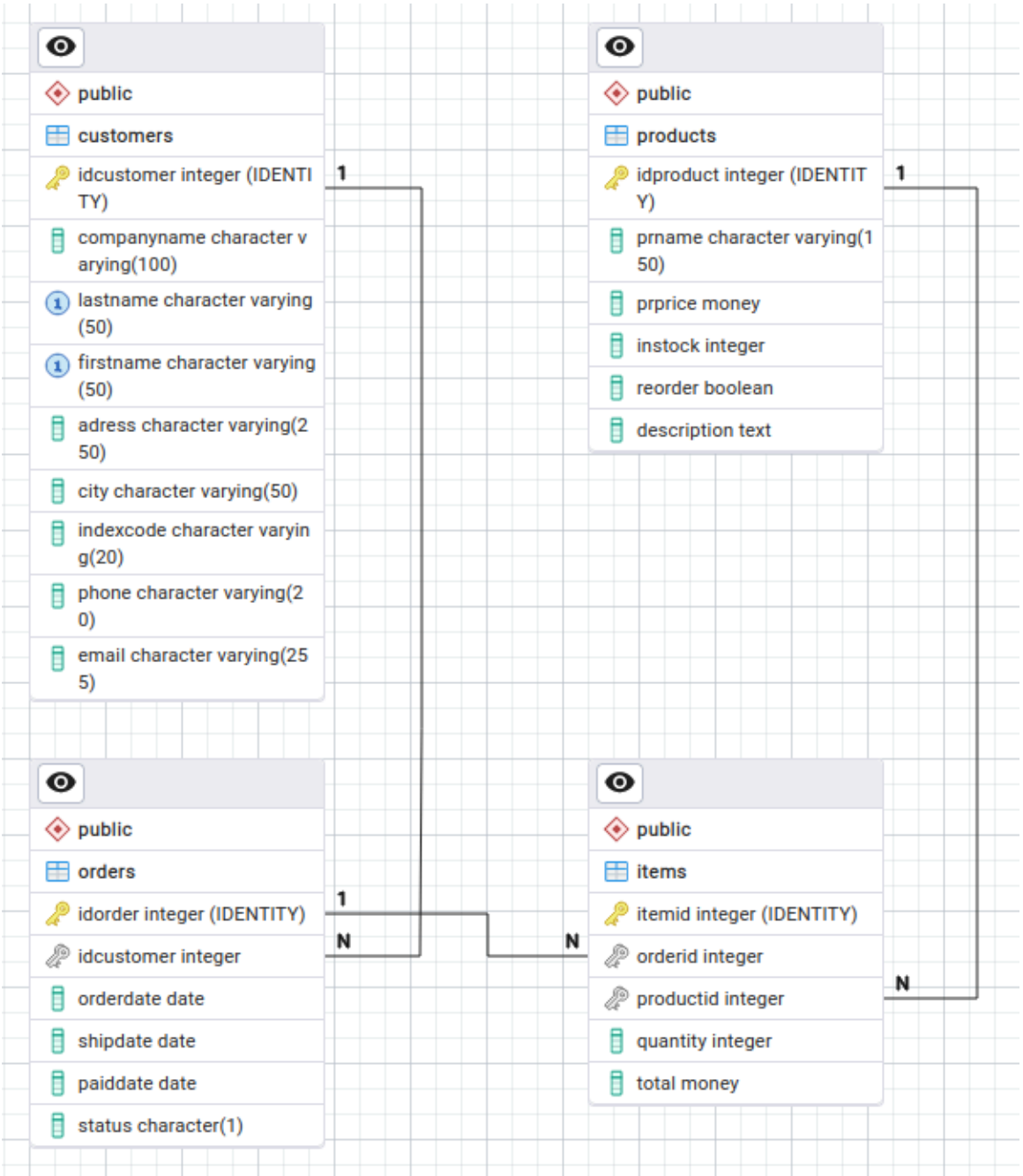
products

	idproduct [PK] integer	prname character varying (150)	prprice money	instock integer	reorder boolean	description text
1	1	Product A	$10.99	100	true	Description of Product A
2	2	Product B	$20.49	75	false	Description of Product B
3	3	Product C	$15.29	50	true	Description of Product C
4	4	Product D	$8.99	120	false	Description of Product D
5	5	Product E	$25.99	90	true	Description of Product E
6	6	Product F	$25.99	90	true	Description of Product F

items

	itemid [PK] integer	orderid integer	productid integer	quantity integer	total money
1	1	1	1	5	$54.95
2	2	2	3	3	$45.87
3	3	3	2	2	$40.98
4	4	4	4	4	$35.96
5	5	5	5	1	$25.99
6	7	5	6	1	$27.99

Получим EDR-модель:



Вывод:

Мы научились создавать базы данных в СУБД PostgreSQL, освоив как теоретические, так и практические аспекты этого процесса. Мы изучили основные понятия и операторы SQL, что позволило нам эффективно управлять данными. Мы также освоили работу в среде pgAdmin, которая облегчает администрирование и взаимодействие с PostgreSQL. Полученные знания и умения по программированию на языке SQL помогли нам уверенно выполнять операции обновления, удаления и вставки данных в базу данных.

Лабораторная работа №2.

«Основы программирования на языке SQL. SELECT для СУБД PostgreSQL.»

Цель работы:

Сформировать знания и умения по программированию на языке SQL, приобрести практические навыки работы со средствами языка SQL для выборки и редактирования данных в БД.

Ход работы:

Создадим к базе данных SELECT-запросы следующих видов:

```
-- Общие запроы, включающие различные типы SELECT-запросов
SELECT * FROM customers; -- (a) выбор всех данных из таблицы customers

SELECT FirstName, LastName, City FROM customers; -- (b) выбор некоторых столбцов таблицы customers

SELECT * FROM customers ORDER BY LastName ASC; -- (c) сортировка данных по фамилии

SELECT * FROM customers LIMIT 5; -- (d) ограничение выборки до первых 5 строк

SELECT * FROM customers WHERE LastName = 'Smith'; -- (e) выборка по фамилии 'Smith'

SELECT * FROM orders WHERE OrderDate BETWEEN '2024-03-01' AND '2024-03-03'; -- (f) выборка заказов между указанными датами

SELECT * FROM customers WHERE IdCustomer IN (SELECT IdCustomer FROM orders WHERE Status = 'C'); -- (g) выборка клиентов, у которых есть заказы со статусом 'C'

SELECT * FROM customers WHERE EMail LIKE '%@abccorp.com'; -- (h) выборка клиентов с почтовым адресом на домене 'abccorp.com'

SELECT * FROM customers WHERE Phone IS NULL; -- (i) выборка клиентов без указанного номера телефона

SELECT COUNT(*) AS TotalCustomers FROM customers; -- (j) подсчет общего числа клиентов

SELECT IdCustomer, COUNT(*) AS TotalOrders FROM orders GROUP BY IdCustomer
```

```
HAVING COUNT(*) > 2; -- (k) подсчет числа заказов для каждого клиента, у которого их больше 2

    SELECT o.OrderDate, c.FirstName, c.LastName
    FROM orders o
    JOIN customers c ON o.IdCustomer = c.IdCustomer; -- (l) выборка данных из нескольких таблиц с использованием соединения по предикату

    SELECT DISTINCT City FROM customers; -- (m) выбор уникальных значений городов клиентов

    SELECT * FROM customers WHERE EXISTS (SELECT * FROM orders WHERE customers.IdCustomer = orders.IdCustomer); -- (n) выборка клиентов, у которых есть заказы

    SELECT FirstName, LastName,
       CASE
           WHEN City = 'New York' THEN 'East Coast'
           WHEN City = 'Los Angeles' THEN 'West Coast'
           ELSE 'Other'
       END AS Region
    FROM customers; -- (o) использование функции CASE для определения региона клиентов
```

Задание по варианту:

1. Получить информацию о покупателях (компания, фамилия, имя, адрес, телефон, город), которые совершили заказ со статусом «P». Список отсортировать по городу и фамилиям.

```
SELECT c.CompanyName, c.LastName, c.FirstName, c.Adress, c.Phone, c.City
FROM customers c
INNER JOIN orders o ON c.IdCustomer = o.IdCustomer
WHERE o.Status = 'P'
ORDER BY c.City, c.LastName;
```

2. Получить информацию о количестве покупателей в каждом из городов, считать только оплаченные заказы. Список отсортировать по количеству покупателей.

```
SELECT c.City, COUNT(DISTINCT c.IdCustomer) AS TotalCustomers
FROM customers c
INNER JOIN orders o ON c.IdCustomer = o.IdCustomer
WHERE o.Status = 'P'
GROUP BY c.City
```

```
ORDER BY TotalCustomers DESC;
```

Вывод:

Мы сформировали знания и умения по программированию на языке SQL, что позволило нам уверенно взаимодействовать с базами данных. Мы приобрели практические навыки работы со средствами SQL, которые необходимы для эффективной выборки и редактирования данных. Эти навыки включают написание сложных запросов, а также использование различных операторов и функций для манипуляции данными.

Лабораторная работа №3.

«Использование представлений в СУБД PostgreSQL.»

Цель работы:

Получить теоретические и практические навыки создания представлений данных в СУБД PostgreSQL.

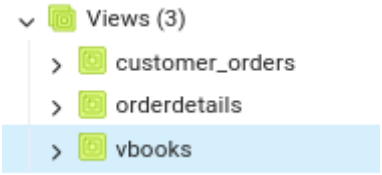
Ход работы:

Добавляем представление и проверяем его работу:

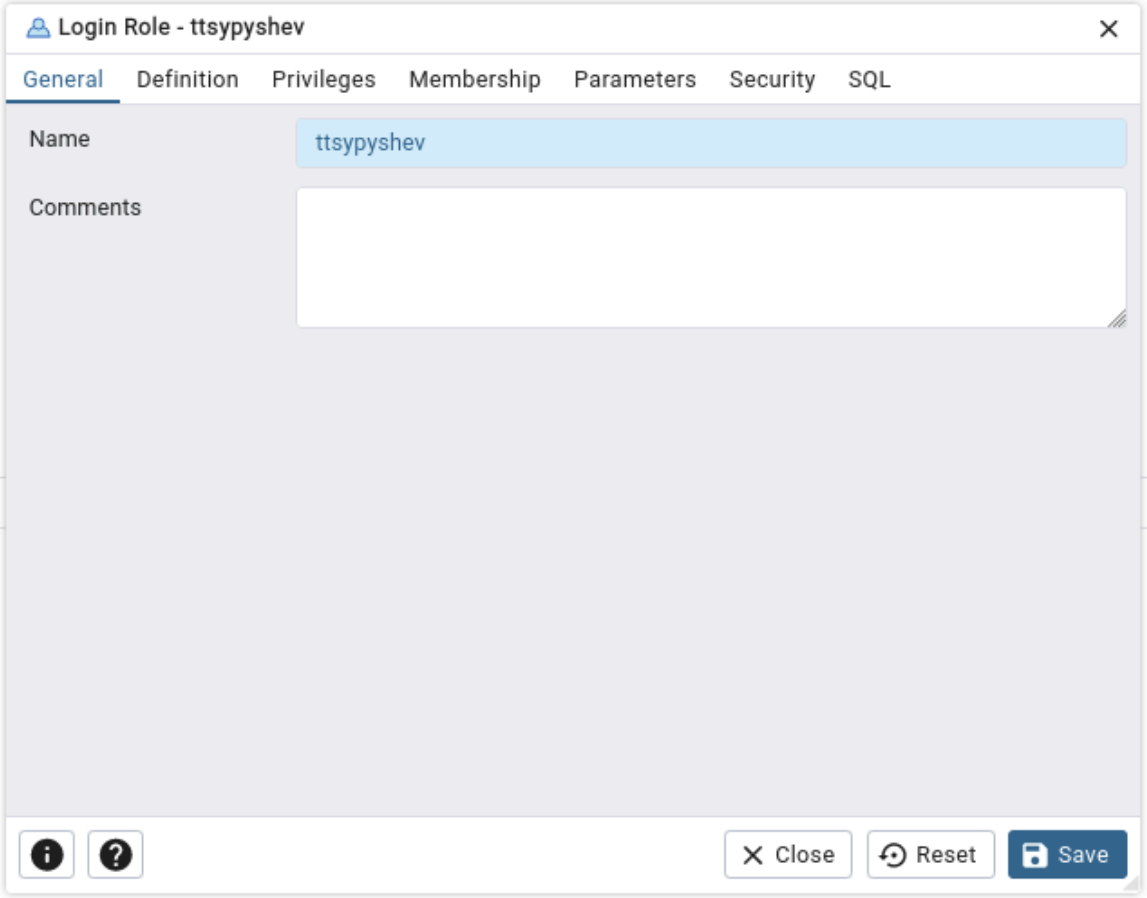
```
--создаём таблицу
CREATE TABLE BOOKS (
   BookID INT PRIMARY KEY,
   Bookname VARCHAR(255),
   Author VARCHAR(255),
   Publisher VARCHAR(255)
);

--заполняем талицу данными
INSERT INTO BOOKS (Bookname, Author, Publisher) VALUES
(1, 'Война и мир', 'Л. Толстой', 'BHV'),
(2, 'Анна Каренина', 'Л. Толстой', 'BHV'),
(3, 'Преступление и наказание', 'Ф. Достоевский', 'Эксмо');

--создаем представление
CREATE VIEW vBOOKS AS SELECT bookid, Bookname, Author, Publisher FROM BOOKS
WHERE Publisher =’BHV’
--вставляем запись
INSERT INTO vBOOKS VALUES (4, ‘Война и мир’,’Л. Толстой’,’BHV’)

--вставляем запись с издательством, отсутствующем в представлении
INSERT INTO vBOOKS VALUES (5, ‘Война и мир’,’Л. Толстой’,’Эксмо’)
```

```
--добавляем опциюWITH CHECK OPTION
ALTER VIEW vBOOKS SET (check_option='cascaded');

--вставляем запись с издательством, отсутствующем в представлении
INSERT INTO vBOOKS VALUES (6, 'Война и мир','Л. Толстой','Эксмо')
```

В pgAdmin представление выглядит вот так:



Создадим роль с использованием графического интерфейса:



С помощью скрипта она будет выглядеть вот так:

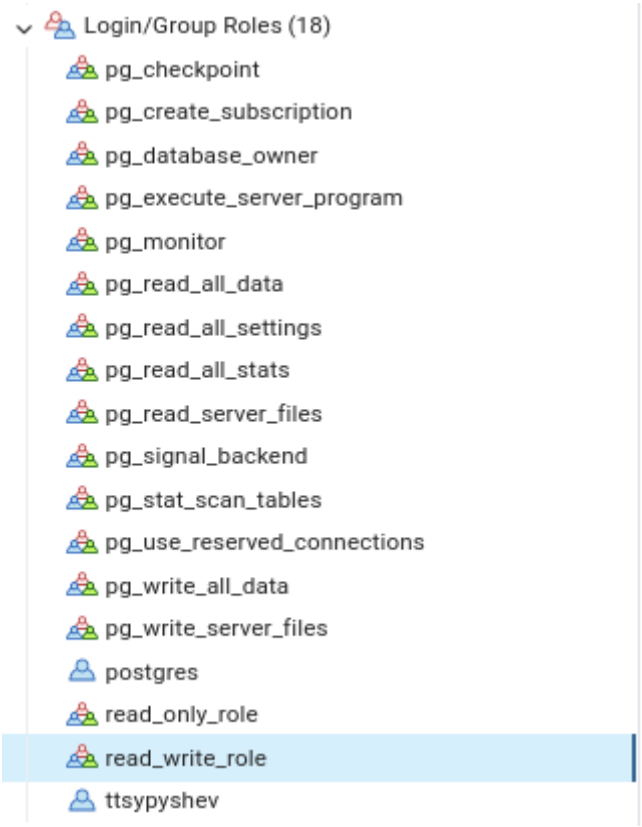
```
-- Role: ttsypyshev
-- DROP ROLE IF EXISTS ttsypyshev;

CREATE ROLE ttsypyshev WITH
 LOGIN
 NOSUPERUSER
```

```
	INHERIT
	NOCREATEDB
	NOCREATEROLE
	NOREPLICATION
	NOBYPASSRLS
	ENCRYPTED PASSWORD
'SCRAM-SHA-256$4096:TsE3dzPrzhUyfT8MdL0Wdg==$pYpgZSaPTuZleHUMdN3rwgD388w+EvQbKp6A8+P/
vzo=:ga5EaR5tDrCWJ2zDOlT5x4ca1bgDJbF9MzWCRuvRSjY=';

	GRANT pg_monitor, read_write_role TO ttsypyshev;
```

Создадим роли и научимся с ними работать через SQL-запросы:

```
	-- Создание роли "read_only_role" с доступом только на чтение
	CREATE ROLE "read_only_role";

	-- Создание роли "read_write_role" с доступом на чтение и запись
	CREATE ROLE "read_write_role";

	-- Предоставление роли "read_only_role" права на SELECT на таблице person
	GRANT SELECT ON TABLE person TO read_only_role;

	-- Предоставление роли "read_write_role" прав на SELECT, INSERT, UPDATE и
DELETE на таблице person
	GRANT SELECT, INSERT, UPDATE, DELETE on TABLE person TO read_write_role;

	-- Назначение роли "read_only_role" пользователю ttsypyshev
	GRANT read_only_role TO ttsypyshev;

	-- Отзыв роли "read_only_role" у пользователя ttsypyshev
	REVOKE read_only_role FROM ttsypyshev;

	-- Назначение роли "read_write_role" пользователю ttsypyshev
	GRANT read_write_role TO ttsypyshev;

	-- Предоставление пользователю ttsypyshev всех привилегий на таблицу person
	GRANT ALL PRIVILEGES ON TABLE person TO ttsypyshev;

	-- Изменение пароля пользователя ttsypyshev на '4321'
	ALTER USER ttsypyshev PASSWORD '4321';
```

В pgAdmin роли выглядят выглядит вот так:



Вывод:

Мы приобрели теоретические и практические навыки создания представлений данных в СУБД PostgreSQL. Это включает понимание концепции представлений, их преимуществ и применения в различных сценариях. Мы научились создавать и управлять представлениями, что позволяет эффективно организовывать и отображать данные из баз данных.

Лабораторная работа №4.

«Создание оконного приложения для работы с базой данных»

Цель работы:

Создать в среде Qt Creator на языке Qt C++ оконное приложение для работы с базой данных.

Ход работы:

Создадим проект “Tset1” для того, что бы ознакомится с интерфейсом и функциями программы:

mainwindow.h

```
#ifndef MAINWINDOW_H
```

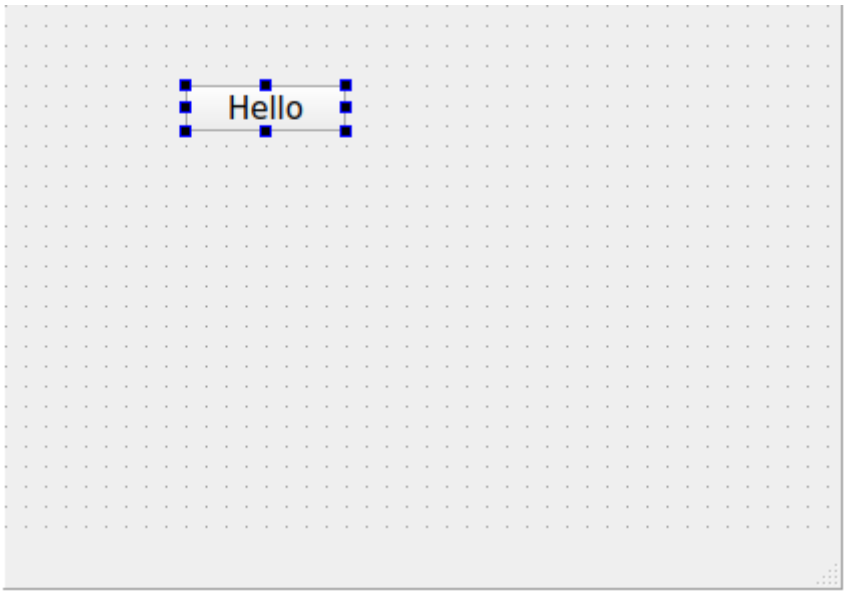
```
#define MAINWINDOW_H

#include <QMainWindow>
#include <QPushButton>
#include <QLineEdit>

namespace Ui {
class MainWindow;
}
class MainWindow : public QMainWindow
{
  Q_OBJECT
public:
  explicit MainWindow(QWidget *parent = 0);
  ~MainWindow();
private slots:
  void on_btnHello_clicked();
  void funHi(bool);
private:
  Ui::MainWindow *ui;
  QPushButton *btnHi;
  QLineEdit *leFio;
};

#endif // MAINWINDOW_H
```

mainwindow.cpp

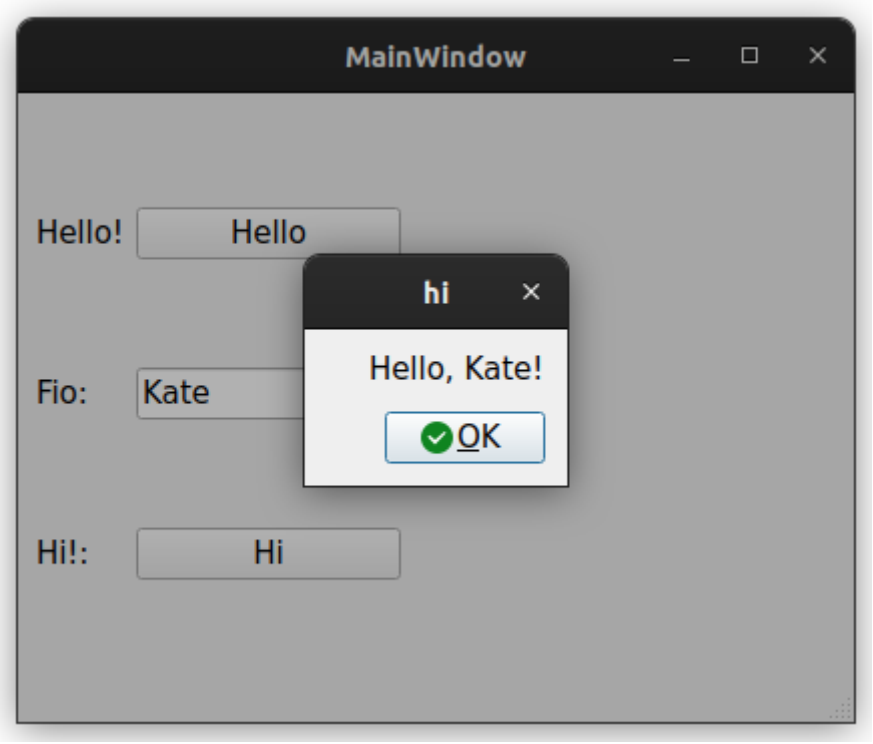
```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "QMessageBox"

#include <QFormLayout>
#include <QLabel>
#include <QDebug>

MainWindow::MainWindow(QWidget *parent) :
  QMainWindow(parent),
  ui(new Ui::MainWindow)
{
  ui->setupUi(this);
  btnHi = new QPushButton("Hi");
  leFio = new QLineEdit();
```

```
    QLayout *layout = this->layout();
    if( layout==0 )
    {
        layout = new QFormLayout();
        this->setLayout(layout);
    }
    QWidget *w = new QWidget(this);
    QGridLayout *formLayout = new QGridLayout();
    formLayout->addWidget(new QLabel(tr("Hello!")),0,0 );
    formLayout->addWidget(ui->btnHello,0,1 );
    formLayout->addWidget(new QLabel(tr("Fio:")),1,0 );
    formLayout->addWidget(leFio,1,1 );
    formLayout->addWidget(new QLabel(tr("Hi!:")),2,0 );
    formLayout->addWidget(btnHi,2,1 );
    w->setLayout(formLayout);
    w->setFixedSize(200,300);
    formLayout->setSizeConstraint(QLayout::SetDefaultConstraint);
    layout->addWidget(w);
    connect(btnHi,SIGNAL(clicked(bool)),this,SLOT(funHi(bool)));
}

MainWindow::~MainWindow()
{
    delete leFio;
    delete btnHi;
    delete ui;
}

void MainWindow::on_btnHello_clicked()
{
    QMessageBox::about(this,"info","Hello!");
}

void MainWindow::funHi(bool flag)
{
    QString str = "Hello, " + leFio->text() + "!";
    QMessageBox::about(this,"hi",str);
}
```

mainwindow.ui



Результат выполнения программы:



Создадим проект “lab4” для того, что бы ознакомится с возможностью подключения и доступными функциями для работы с базами данных PostgresSQL:

mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QSqlDatabase>
#include <QSqlQuery>
```

```
#include <QTableWidget>
#include <QMessageBox>
#include <QSqlError>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
  Q_OBJECT
public:
  MainWindow(QWidget *parent = nullptr);
  ~MainWindow();
  QSqlDatabase dbconn;
private slots:
  void dbconnect();
  void selectAll();
  void add();
  void del();
  void edit();
private:
  Ui::MainWindow *ui;
};

#endif // MAINWINDOW_H
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent) :
  QMainWindow(parent),
  ui(new Ui::MainWindow)
{
  ui->setupUi(this);
  connect(ui->btnConnect,SIGNAL(clicked(bool)),this, SLOT(dbconnect()));
  connect(ui->btnSelectAll, SIGNAL(clicked(bool)), this, SLOT(selectAll()));
  connect(ui->btnAdd, SIGNAL(clicked(bool)), this, SLOT(add()));
  connect(ui->btnDel, SIGNAL(clicked(bool)), this, SLOT(del()));
  connect(ui->btnEdit, SIGNAL(clicked(bool)), this, SLOT(edit()));
```

```
  // Количество столбцов
  ui->twOrg->setColumnCount(4);
  // Возможность прокрутки
  ui->twOrg->setAutoScroll(true);
  // Режим выделения ячеек - только одна строка
  ui->twOrg->setSelectionMode(QAbstractItemView::SingleSelection);
  ui->twOrg->setSelectionBehavior(QAbstractItemView::SelectRows);
  // Заголовки таблицы
  ui->twOrg->setHorizontalHeaderItem(0,new QTableWidgetItem("Abbr"));
  ui->twOrg->setHorizontalHeaderItem(1,new QTableWidgetItem("Title"));
  ui->twOrg->setHorizontalHeaderItem(2,new QTableWidgetItem("City"));
  ui->twOrg->setHorizontalHeaderItem(3,new QTableWidgetItem("INN"));
  // Последний столбец растягивается при изменении размера формы
  ui->twOrg->horizontalHeader()->setStretchLastSection(true);
  // Запрет на изменение ячеек таблицы при отображении
  ui->twOrg->setEditTriggers(QAbstractItemView::NoEditTriggers);
}

MainWindow::~MainWindow()
{
  if( dbconn.isOpen())
     dbconn.close();
  delete ui;
}

void MainWindow::dbconnect()
{
  if(!dbconn.isOpen())
  {
     // Если соединение не открыто, то вывести список доступных драйверов БД
     // (вывод в поле teResult, метод append добавляет строки).
     ui->teResult->append("SQL drivers:");
     ui->teResult->append(QSqlDatabase::drivers().join(","));
     // Создать глобальную переменную для установки соединения с БД
     dbconn=QSqlDatabase::addDatabase("QPSQL");
     // Установить параметры соединения: имя БД, адрес хоста, логин и пароль пользователя, порт
(если отличается от стандартного)
     dbconn.setDatabaseName("dbtest");
     dbconn.setHostName("localhost");
     dbconn.setUserName("postgres");
     dbconn.setPassword("postgres");
     // Открыть соединение и результат вывести в окно вывода
```

```
        if( dbconn.open() )
            ui->teResult->append("Connect is open...");
        else
        {
            ui->teResult->append("Error of connect:");
            ui->teResult->append(dbconn.lastError().text());
        }
    }
    else
        // Если соединение уже открыто, то сообщить об этом
        ui->teResult->append("Connect is already open...");
}

void MainWindow::selectAll()
{
    // Очистить содержимое компонента
    ui->twOrg->clearContents();
    // Если соединение не открыто, то вызвать нашу функцию для открытия
    // если подключиться не удалось, то вывести сообщение об ошибке и
    // выйти из функции
    if( !dbconn.isOpen() )
    {
        dbconnect();
        if( !dbconn.isOpen() )
        {
            QMessageBox::critical(this,"Error",dbconn.lastError().text());
            return;
        }
    }
    // Создать объект запроса с привязкой к установленному соединению
    QSqlQuery query(dbconn);
    // Создать строку запроса на выборку данных
    QString sqlstr = "select * from org";
    // Выполнить запрос и поверить его успешность
    bool query_success = query.exec(sqlstr);
    // Если запрос активен (успешно завершен),
    // то вывести сообщение о прочитанном количестве строк в окно вывода
    // и установить количество строк для компонента таблицы
    if( query_success )
        ui->twOrg->setRowCount( query.size());
    else
        ui->twOrg->setRowCount( 0);
```

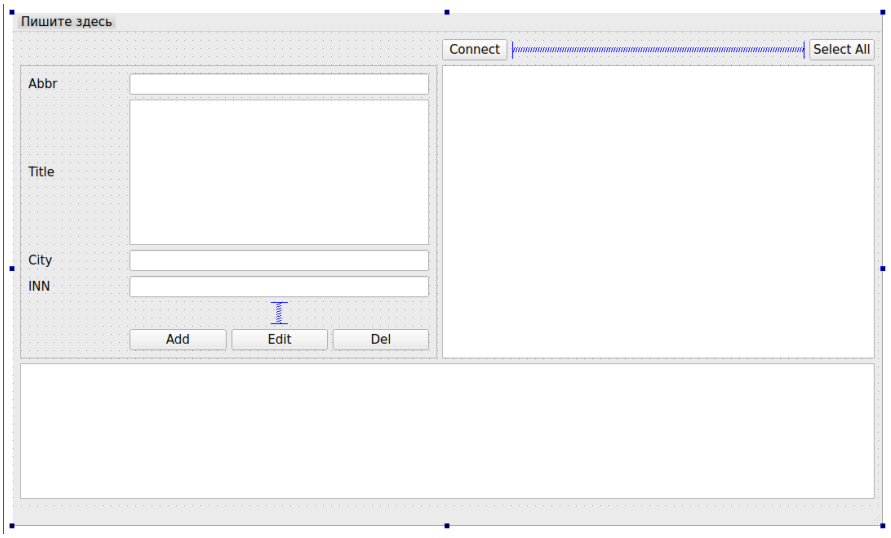
```
  ui->teResult->append( QString("Read %1 rows").arg(query.size()));
  // Прочитать в цикле все строки результата (курсора)
  // и вывести их в компонент таблицы
  int i=0;
  while(query.next())
  {
     ui->twOrg->setItem(i,0,new
                  QTableWidgetItem(query.value("abbr").toString()));
     ui->twOrg->setItem(i,1,new
                  QTableWidgetItem(query.value("title").toString()));
     ui->twOrg->setItem(i,2,new
                  QTableWidgetItem(query.value("city").toString()));
     ui->twOrg->setItem(i,3,new
                  QTableWidgetItem(query.value("inn").toString()));
     i++;
  }
}

void MainWindow::add()
{
  // Подключиться к БД
  if( !dbconn.isOpen() )
  {
     dbconnect();
     if( !dbconn.isOpen() )
     {
        QMessageBox::critical(this,"Error",dbconn.lastError().text());
        return;
     }
  }
  QSqlQuery query(dbconn);
  // Создать строку запроса
  QString sqlstr = "insert into org(abbr,title,city,inn) values(?,?,?,?)";
  // Подготовить запрос
  query.prepare(sqlstr);
  // Передать параметры из полей ввода в запрос
  query.bindValue(0,ui->leAbbr->text());
  query.bindValue(1,ui->teTitle->toPlainText());
  query.bindValue(2,ui->leCity->text());
  // Если тип поля отличается от строкового, то преобразовать его
  query.bindValue(3,ui->leInn->text().toLongLong());
  // Выполнить запрос
```

```
    if( !query.exec() )
    {
        ui->teResult->append( query.lastQuery());
        QMessageBox::critical(this,"Error",query.lastError().text());
        return;
    }
    // Если запрос выполнен, то вывести сообщение одобавлении строки
    ui->teResult->append( QString("AddRead %1 rows").arg(query.numRowsAffected()) );
    // и обновить записи в компоненте таблицы
    selectAll();
}

void MainWindow::del()
{
    // Подключение к БД
    if( !dbconn.isOpen() )
    {
        dbconnect();
        if( !dbconn.isOpen() )
        {
            QMessageBox::critical(this,"Error",dbconn.lastError().text());
            return;
        }
    }
    // Получить номер выбранной строки в компоненте таблицы
    int currow = ui->twOrg->currentRow();
    // Если он меньше 0 (строка не выбрана), то
    // сообщение об ошибке и выход из функции
    if( currow < 0 )
    {
        QMessageBox::critical(this,"Error","Not selected row!");
        return;
    }
    // Спросить у пользователя подтверждение удаления записи
    // Используется статический метод QMessageBox::question
    // для задания вопроса, который возвращает код нажатой кнопки
    if( QMessageBox::question(this,"Delete","Delete row?",
                              QMessageBox::Cancel,QMessageBox::Ok)==QMessageBox::Cancel)
        return;
    // Создать объект запроса
    QSqlQuery query(dbconn);
    // Создать строку запроса.
```

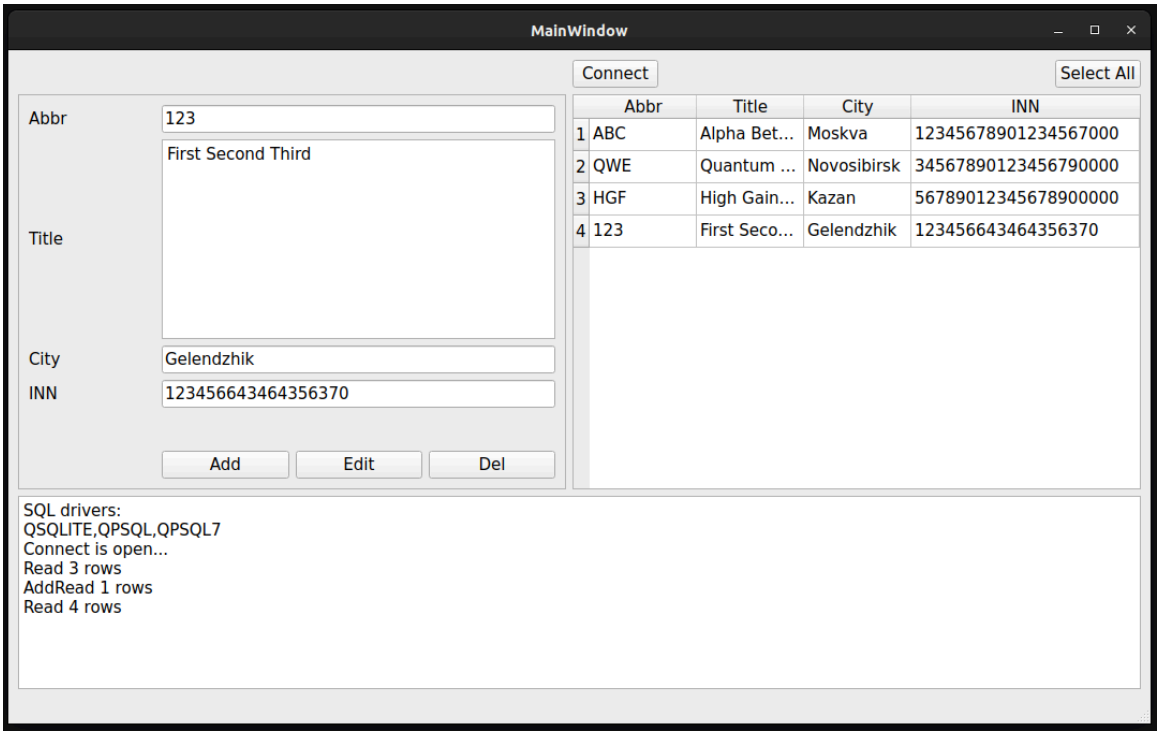
```
    // Вместо подготовки запроса и передачи параметров значение параметра
    // конкатенируется со строкой запроса
    // Обратите,что строковое значение помещается в одинарные кавычки
    // Значение выбирается из компонента таблицы методом item(row,col)
    QString sqlstr = "delete from org where abbr = '"
            + ui->twOrg->item(currow,0)->text() + "'";
    // Выполнить строку запроса и проверить его успешность
    if( !query.exec(sqlstr) )
    {
        ui->teResult->append( query.lastQuery());
        QMessageBox::critical(this,"Error",query.lastError().text());
        return;
    }
    // Вывести сообщение об удалении строки
    ui->teResult->append( QString("Del %1 rows").arg(query.numRowsAffected()) );
    // Обновить содержимое компонента таблицы
    selectAll();
}

void MainWindow::edit()
{
    // Подключение к БД
    if (!dbconn.isOpen())
    {
        dbconnect();
        if (!dbconn.isOpen())
        {
            QMessageBox::critical(this, "Error", dbconn.lastError().text());
            return;
        }
    }

    // Получить номер выбранной строки в компоненте таблицы
    int currow = ui->twOrg->currentRow();
    // Если он меньше 0 (строка не выбрана), то
    // сообщение об ошибке и выход из функции
    if (currow < 0)
    {
        QMessageBox::critical(this, "Error", "Not selected row!");
        return;
    }
```

```
    // Создать объект запроса
    QSqlQuery query(dbconn);
    // Создать строку запроса
    QString sqlstr = "update org set abbr = ?, title = ?, city = ?, inn = ? where abbr = ?";
    // Подготовить запрос
    query.prepare(sqlstr);
    // Передать параметры из полей ввода в запрос
    query.bindValue(0, ui->leAbbr->text());
    query.bindValue(1, ui->teTitle->toPlainText());
    query.bindValue(2, ui->leCity->text());
    // Если тип поля отличается от строкового, то преобразовать его
    query.bindValue(3, ui->leInn->text().toLongLong());
    // Передать старое значение abbr для условия where
    query.bindValue(4, ui->twOrg->item(currow, 0)->text());

    // Выполнить запрос
    if (!query.exec())
    {
        ui->teResult->append(query.lastQuery());
        QMessageBox::critical(this, "Error", query.lastError().text());
        return;
    }

    // Если запрос выполнен, то вывести сообщение о редактировании строки
    ui->teResult->append(QString("Edited %1 rows").arg(query.numRowsAffected()));
    // Обновить содержимое компонента таблицы
    selectAll();
}
```

mainwindow.ui



Результат выполнения программы:



Вывод:

Мы создали оконное приложение на языке Qt C++ в среде Qt Creator для работы с базой данных. В процессе разработки мы изучили основы работы с Qt и интеграции его с СУБД. Мы освоили создание пользовательского интерфейса и подключение его к базе данных, что позволило реализовать основные функции взаимодействия с данными. Полученные знания и опыт дают нам уверенность в создании подобных приложений в будущем.

Лабораторная работа №5.

«Использование пользовательских процедур, функций и триггеров в PostgreSQL.»

Цель работы:

Изучить пользовательские процедуры, функции и триггеры в базах данных, приобрести практические навыки создания хранимых процедур и триггеров в PostgreSQL.

Ход работы:

Создадим и заполним данными таблицу:

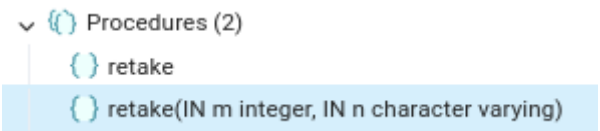
```
-- Создание таблицы "Студенты" с указанными столбцами
CREATE TABLE Студенты (
    id INT PRIMARY KEY,
```

```
 Имя VARCHAR(255) NOT NULL,
 Оценка_1 INT,
 Оценка_2 INT,
 Оценка_3 INT,
 Группа VARCHAR(255)
);

-- Вставка данных в таблицу "Студенты"
INSERT INTO Студенты (id, Имя, Оценка_1, Оценка_2, Оценка_3, Группа) VALUES
 (1, 'Иван Иванов', 5, 2, 3, '18У135'),
 (2, 'Мария Петрова', 4, 5, 5, '18У136'),
 (3, 'Алексей Сидоров', 5, 4, 5, '18У137'),
 (4, 'Ольга Смирнова', 3, 3, 2, '18У138'),
 (5, 'Дмитрий Козлов', 2, 5, 4, '18У139');
```

Создадим и изучим работу процедуры:

```
-- Создание хранимой процедуры "retake" с параметрами m и n
CREATE PROCEDURE retake (m INT, n VARCHAR(255))
LANGUAGE SQL
AS $$
   -- Обновление значения в столбце "Оценка_2" для студентов с указанной группой
   UPDATE Студенты SET Оценка_2 = m WHERE Группа = n;
$$;

-- Вызов процедуры "retake" с передачей аргументов
CALL retake(5, '18У137');

-- Вывод всех данных из таблицы "Студенты"
SELECT * FROM Студенты;
```

В интерфейсе pgAdmin она будет выглядеть вот так:

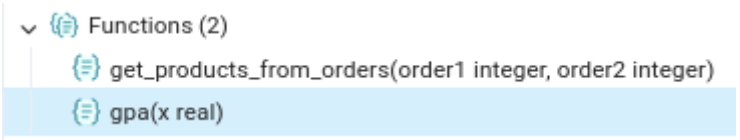


Создадим и изучим работу функции:

```
-- Создание функции "gpa" с одним параметром x типа real и возвращающей набор записей
CREATE FUNCTION gpa (x real)
RETURNS SETOF record
LANGUAGE SQL
AS $$
```

```
    -- Выборка всех студентов, у которых средний балл выше x
    SELECT * FROM Студенты WHERE (Оценка_1 + Оценка_2 + Оценка_3) / 3 > x;
$$;

-- Вызов функции "gpa" с аргументом 3
SELECT gpa(3);
```

В интерфейсе pgAdmin она будет выглядеть вот так:



Задание по варианту:

1. Вывести список товаров (ID, описание, количество, сумма, дата доставки), которые были заказаны в двух заданных заказах. Номера заказов вводятся как параметры.

```
CREATE OR REPLACE FUNCTION get_products_from_orders (
    order1 INT,
    order2 INT
)
RETURNS TABLE (
    itemid INT,
    description TEXT,
    quantity INT,
    total NUMERIC,
    shipdate DATE
)
LANGUAGE SQL
AS $$
    SELECT
        i.itemid,
        p.description,
        i.quantity,
        i.total,
        o.shipdate
    FROM items i
    JOIN orders o ON i.orderid = o.idorder
    JOIN products p ON i.productid = p.idproduct
    WHERE i.orderid IN (order1, order2);
$$;

select * from items
SELECT * FROM get_products_from_orders(4, 5);
```

2. Подсчитать количество каждого товара в заказах города. Название города вводится как параметр.

```
SELECT
   c.city AS Город,
   p.prname AS Товар,
   SUM(i.quantity) AS Количество
FROM
   items i
JOIN
   orders o ON i.orderid = o.idorder
JOIN
   customers c ON o.idcustomer = c.idcustomer
JOIN
   products p ON i.productid = p.idproduct
GROUP BY
   c.city, p.prname
ORDER BY
   c.city, p.prname;
```

Вывод:

Мы изучили пользовательские процедуры, функции и триггеры в базах данных, что расширило наше понимание их использования и преимуществ. Мы приобрели практические навыки создания хранимых процедур и триггеров в PostgreSQL, что позволило нам автоматизировать различные операции и улучшить производительность баз данных. Эти знания помогают нам создавать более эффективные и надежные приложения, взаимодействующие с PostgreSQL.

Лабораторная работа №6.

«Запросы DCL. Резервное копирование.»

Цель работы:

Получить теоретические и практические навыки импорта и экспорта данных в PostgreSQL, а также создания полной версии бекапа и части данных.

Ход работы:

Создадим новую таблицу:

```
-- Создание таблицы "people"
CREATE TABLE people (
   id SERIAL PRIMARY KEY,
```

```
    name VARCHAR(255),
    surname VARCHAR(255),
    age INT
);
```

Получим .csv:

COPY people TO '/home/people1.csv' DELIMITER ',' CSV HEADER;

Импортируем .csv:

COPY people FROM '/home/people2.csv' DELIMITER ',' CSV HEADER;

Проверяем правильность данных:

	id [PK] integer	name character varying (255)	surname character varying (255)	age integer
1	1	Иван	Иванов	30
2	2	Петр	Петров	25
3	3	Мария	Сидорова	35
4	4	Елена	Кузнецова	40

Создадим резервную копию с помощью pg_dump:

pg_dump postgres -h localhost -p 15432 -U postgres > backup.sql

Создадим новую базу данных и загрузим туда нашу резервную копию:

psql lab6 < backup.sql -U postgres -h localhost -p 5432

Вывод:

Мы получили теоретические и практические навыки импорта и экспорта данных в PostgreSQL, что позволяет нам эффективно управлять данными между различными системами. Мы научились создавать полные версии бекапа, а также частичные копии данных, обеспечивая надежное восстановление базы данных в случае сбоя. Эти навыки обеспечивают безопасность и целостность данных, а также удобство их перемещения и архивации.