

##### МИНОБРНАУКИ РОССИИ

##### Федеральное государственное бюджетное образовательное учреждение высшего образования

#### «МИРЭА—Российский технологический университет»

**РТУ МИРЭА**

#### Институт комплексной безопасности и специального приборостроения

##### (наименование института, филиала)

### Кафедра КБ-2 «Прикладные информационные технологии»

##### (наименование кафедры)

Курсовая работа

### по дисциплине Клиент-серверные базы данных

##### (наименование дисциплины)

**Тема курсовой работы** Создание клиент-серверного приложения для работы с базой данных\_

**Студент группы** БСБО-01-18 **\_Левченко Дмитрий Олегович**

##### (учебная группа, фамилия, имя, отчество студента) (подпись студента)

#### Руководитель курсовой работы

##### (должность, звание, ученая степень) (подпись руководителя)

**Рецензент** (при наличии)

##### (должность, звание, ученая степень) (подпись рецензента)

##### Работа представлена к защите « » 20 г.

##### Допущен к защите « » \_20 г.

### Москва 2021



##### МИНОБРНАУКИ РОССИИ

##### Федеральное государственное бюджетное образовательное учреждение высшего образования

#### «МИРЭА—Российский технологический университет»

**РТУ МИРЭА**

#### Институт комплексной безопасности и специального приборостроения

##### (наименование института, филиала)

##### Кафедра КБ-2 «Прикладные информационные технологии» (наименование кафедры)

##### Утверждаю

##### Заведующий кафедрой:

##### (Ф.И.О.) (подпись)

##### « » 20 г.

**ЗАДАНИЕ**

## на выполнение курсовой работы

### по дисциплине Клиент-серверные базы данных

##### (наименование дисциплины)

**Тема курсовой работы** Создание клиент-серверного приложения для работы с базой данных

**Студент группы \_БСБО-01-18 Левченко Дмитрий Олегович**

##### (учебная группа, фамилия, имя, отчество студента) (подпись студента)

#### Исходные данные Перечень вопросов, подлежащих обработке, и обязательного графического материала:

##### Создание клиента для работы с БД Реализация прав доступа в клиентской части Создание БД, таблиц, функций, процедур, триггеров индексов, транзакций Реализация в клиенте запросов согласно заданию Добавление индексов для ускорения запросов

#### Срок предоставление к защите курсовой работы до « » 20 г.

#### Задание на курсовую работу выдал

##### (Ф.И.О. руководителя) (подпись руководителя)

##### « » 20 г.

**Задание на курсовую работу получил** Левченко Д.О.

##### (Ф.И.О. исполнителя) (подпись исполнителя)

**СОДЕРЖАНИЕ**

[Введение 4](#_TOC_250006)

[Постановка задачи 8](#_TOC_250005)

[Схема базы данных 10](#_TOC_250004)

[Выполненные задания 12](#_TOC_250003)

[Листинг программы 18](#_TOC_250002)

[Демонстрация работы приложения 28](#_TOC_250001)

[Используемая литература 37](#_TOC_250000)

# ВВЕДЕНИЕ

### В основе решения многих задач лежит обработка информации. Для облегчения обработки информации создаются информационные системы (ИС). Автоматизированными называют ИС, в которых применяют технические средства.

### База данных (БД) представляет собой совокупность специальным образом организованных данных, хранимых в памяти вычислительной системы и отображающих состояние объектов, и их взаимосвязей в рассматриваемой предметной области.

### Базы данных бывают централизованными, которые хранятся на одном компьютере, и распределенными, которые хранятся на нескольких компьютерах некоторой сети.

### Логическую структуру хранимых в базе данных называют моделью представления данных. К основным моделям представления данных относятся следующие: иерархическая, сетевая, реляционная, постреляционная, многомерная и объектно-ориентированная.

### Система управления базами данных (СУБД) – это комплекс языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями. Обычно СУБД различают по используемой модели данных. Так, СУБД, основанные на использовании реляционной модели данных, называют реляционными СУБД.

### Приложение представляет собой программу или комплекс программ, обеспечивающих автоматизацию обработки информации для прикладной задачи. Приложения, разработанные в среде СУБД, часто называют приложениями СУБД, а приложения, разработанные вне СУБД, – внешними приложениями.

### Сервером определенного ресурса в компьютерной сети называется компьютер или программа, управляющие этим ресурсом, клиентом — компьютер или программа, использующие этот ресурс. В качестве ресурса компьютерной сети могут выступать, к примеру, базы данных, файловые системы, службы печати, почтовые службы. Тип сервера определяется видом ресурса, которым он управляет. Например, если управляемым ресурсом является база данных, то соответствующий сервер называется сервером базы данных.

### Достоинством организации информационной системы по архитектуре клиент-сервер является удачное сочетание централизованного хранения, обслуживания и коллективного доступа к общей корпоративной информации с

### индивидуальной работой пользователей над персональной информацией. Архитектура клиент-сервер допускает различные варианты реализации.

### Важнейшим достоинством применения БД в информационных системах является обеспечение независимости данных от прикладных программ. Это дает возможность пользователям не заниматься проблемами представления данных на физическом уровне: размещения данных в памяти, методов доступа к ним и т. д.

### Для работы с хранящейся в базе данных информацией СУБД предоставляет программам и пользователям следующие два типа языков:

### язык описания данных – высокоуровневый непроцедурный язык декларативного типа, предназначенный для описания логической структуры данных;

### язык манипулирования данными – совокупность конструкций, обеспечивающих выполнение основных операций по работе с данными: ввод, модификацию и выборку данных по запросам.

### SQL (Structured Query Language) – структурированный язык запросов – сочетает в себе свойства языков обоих типов – описания и манипулирования данными.

### Механизм транзакций используется в СУБД для поддержания целостности данных в базе. Транзакцией называется некоторая неделимая последовательность операций над данными БД, которая отслеживается СУБД от начала и до завершения. Если по каким-либо причинам (сбои и отказы оборудования, ошибки в программном обеспечении, включая приложение) транзакция остается незавершенной, то она отменяется.

### Ведение журнала изменений в БД выполняется СУБД для обеспечения надежности хранения данных в базе при наличии аппаратных сбоев и отказов, а также ошибок в программном обеспечении.

### Обеспечение целостности БД составляет необходимое условие успешного функционирования БД, особенно для случая использования БД в сетях. Целостность БД есть свойство базы данных, означающее, что в ней содержится полная, непротиворечивая и адекватно отражающая предметную область информация. Поддержание целостности БД включает проверку целостности и ее восстановление в случае обнаружения противоречий в базе данных. Целостное состояние БД описывается с помощью ограничений целостности в виде условий, которым должны удовлетворять хранимые в базе данные.

### Нормальная форма — свойство отношения в реляционной модели данных, характеризующее его с точки зрения избыточности, потенциально приводящей к логически ошибочным результатам выборки или изменения данных. Нормальная форма определяется как совокупность требований, которым должно удовлетворять отношение.

### Процесс преобразования отношений базы данных к виду, отвечающему нормальным формам, называется нормализацией. Нормализация предназначена для приведения структуры БД к виду, обеспечивающему минимальную логическую избыточность, и не имеет целью уменьшение или увеличение производительности работы или же уменьшение, или увеличение физического объёма базы данных. Конечной целью нормализации является уменьшение потенциальной противоречивости, хранимой в базе данных информации. Как отмечает К. Дейт, общее назначение процесса нормализации заключается в следующем:

### исключение некоторых типов избыточности;

### устранение некоторых аномалий обновления;

### разработка проекта базы данных, который является достаточно

### «качественным» представлением реального мира, интуитивно понятен и может служить хорошей основой для последующего расширения;

### упрощение процедуры применения необходимых ограничений целостности.

### Устранение избыточности производится, как правило, за счёт декомпозиции отношений таким образом, чтобы в каждом отношении хранились только первичные факты (то есть факты, не выводимые из других хранимых фактов).

## Первая нормальная форма

### Переменная отношения находится в первой нормальной форме (1НФ) тогда и только тогда, когда в любом допустимом значении отношения каждый его кортеж содержит только одно значение для каждого из атрибутов.

### В реляционной модели отношение всегда находится в первой нормальной форме по определению понятия отношение. Что же касается различных таблиц, то они могут не быть правильными представлениями отношений и, соответственно, могут не находиться в 1НФ.

## Вторая нормальная форма

### Переменная отношения находится во второй нормальной форме тогда и только тогда, когда она находится в первой нормальной форме и каждый не

### ключевой атрибут неприводимо (функционально полно) зависит от ее потенциального ключа.

## Третья нормальная форма

### Переменная отношения находится в третьей нормальной форме тогда и только тогда, когда она находится во второй нормальной форме, и отсутствуют транзитивные функциональные зависимости неключевых атрибутов от ключевых.

# ПОСТАНОВКА ЗАДАЧИ

### Разработать клиент-серверное приложение, серверная часть которой реализована на PostgreSQL (*или MS SQL, Oracle*), представляющая собой модель предметной области в соответствии с вариантом задания. В рамках заданной предметной области реализовать заданную (по варианту) схему отношений, т.е. выделить сущности и их атрибуты, так чтобы связи между сущностями соответствовали представленной схеме. В рамках курсовой работы необходимо на стороне сервера реализовать и использовать при демонстрации приложения следующие компоненты:

### Постоянные таблицы и связи между ними, количество таблиц и наличие связей должно соответствовать заданию, допускается увеличение числа таблиц и их полей для более адекватного представления предметной области;

### В приложении реализовать не менее пяти запросов, включая (для демонстрации навыков работы)

### Составной многотабличный запрос с параметром, включающий соединение таблиц и CASE-выражение;

### На основе обновляющего представления (многотабличного VIEW), в котором критерий упорядоченности задает пользователь при выполнении;

### Запрос, содержащий коррелированные и некоррелированные подзапросы в разделах SELECT, FROM и WHERE (в каждом хотя бы по одному);

### Многотабличный запрос, содержащий группировку записей, агрегативные функции и параметр, используемый в разделе HAVING;

### Запрос, содержащий предикат ANY(SOME) или ALL;

### Создать индексы для увеличения скорости выполнения запросов;

### В таблице (в соответствии с вариантом) предусмотреть поле, которое заполняется автоматически по срабатыванию триггера при добавлении, обновлении и удалении данных, иметь возможность продемонстрировать работу триггера при работе приложения. Триггеры должны обрабатывать только те записи, которые были добавлены, изменены или удалены в ходе текущей операции (транзакции).

### Операции добавления, удаления и обновления реализовать в виде хранимых процедур (с параметрами) хотя бы для одной таблицы; для остальных допустимо использовать возможности связывания полей ввода в приложении с полями БД;

### Реализовать отдельную хранимую процедуру, состоящую из нескольких отдельных операций в виде единой транзакции, которая при определенных условиях может быть зафиксирована или откатана;

### В триггере или хранимой процедуре реализовать курсор на обновления отдельных данных;

### В запросе (из пункта 2 или в дополнительном к тому перечню) использовать собственную скалярную функцию, а в хранимой процедуре – векторную (или табличную) функцию. Функции сохранить в базе данных

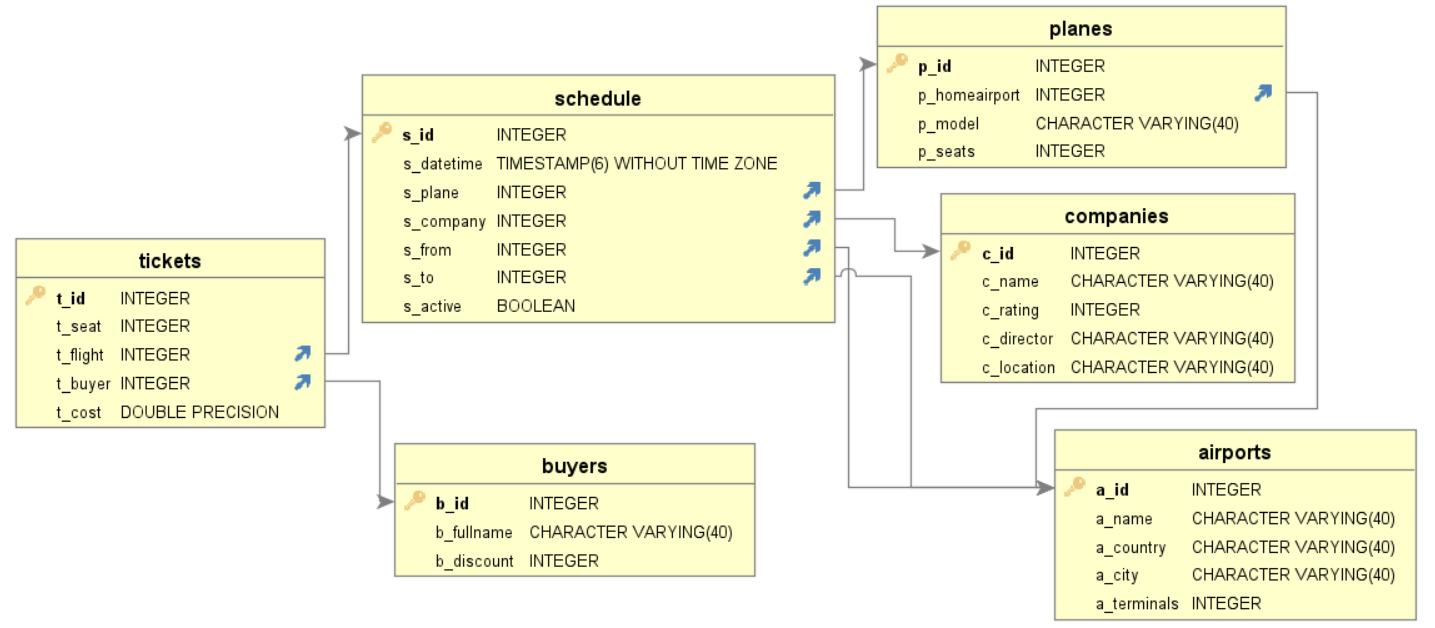
### Распределение прав пользователей: предусмотреть не менее двух пользователей с разным набором привилегий. Каждый набор привилегий оформить в виде роли.

### Клиент должен обеспечивать добавление, модификацию и удаление данных по всей предметной области. Добавление, редактирование данных хотя бы по одной таблице производить в отдельном окне.

### Запрещено в качестве вводимых данных, в том числе для связи таблиц, указывать значения первичных и внешних ключей – для обеспечения ссылочной целостности пользователь должен выбирать значения из справочника, а соответствующие значения должны подставляться программно (тем или иным способом – автоматически).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 30 | Продажа билетов на самолет | Ведение БД по расписанию и самолетовылетам. Продажа и возврат билетов | 3 | 3 |

# СХЕМА БАЗЫ ДАННЫХ



CREATE TABLE IF NOT EXISTS public.Airports (

A\_ID INT

GENERATED ALWAYS AS IDENTITY

NOT NULL

UNIQUE

PRIMARY KEY,

A\_Name VARCHAR(40),

A\_Country VARCHAR(40),

A\_City VARCHAR(40),

A\_Terminals INT

DEFAULT 0

);

CREATE INDEX IX\_Airports\_A\_ID ON Airports(A\_ID);

ALTER TABLE Airports CLUSTER ON IX\_Airports\_A\_ID;

CLUSTER Airports;

CREATE TABLE IF NOT EXISTS public.Planes (

P\_ID INT

GENERATED ALWAYS AS IDENTITY

NOT NULL

UNIQUE

PRIMARY KEY,

P\_HomeAirport INT

DEFAULT 0,

P\_Model VARCHAR(40),

P\_Seats INT

DEFAULT 0,

CONSTRAINT FK\_Planes\_Airport\_A\_ID

FOREIGN KEY(P\_HomeAirport)

REFERENCES public.Airports(A\_ID)

ON DELETE SET NULL

);

CREATE INDEX IX\_Planes\_P\_HomeAirport ON Planes(P\_HomeAirport);

ALTER TABLE Planes CLUSTER ON IX\_Planes\_P\_HomeAirport;

CLUSTER Planes;

CREATE TABLE IF NOT EXISTS public.Companies (

C\_ID INT

GENERATED ALWAYS AS IDENTITY

NOT NULL

UNIQUE

PRIMARY KEY,

C\_Name VARCHAR(40),

C\_Rating INT

DEFAULT 0,

C\_Director VARCHAR(40),

C\_Location VARCHAR(40)

);

CREATE INDEX IX\_Companies\_C\_Name ON Companies(C\_Name);

ALTER TABLE Companies CLUSTER ON IX\_Companies\_C\_Name;

CLUSTER Companies;

CREATE INDEX IX\_Companies\_C\_Rating ON Companies(C\_Rating);

CREATE TABLE IF NOT EXISTS public.Schedule (

S\_ID INT

GENERATED ALWAYS AS IDENTITY

NOT NULL

UNIQUE

PRIMARY KEY,

S\_DateTime TIMESTAMP,

S\_Plane INT

DEFAULT 0,

S\_Company INT

DEFAULT 0,

S\_From INT

DEFAULT 0,

S\_To INT

DEFAULT 0,

S\_Active BOOLEAN

DEFAULT true,

CONSTRAINT FK\_Schedule\_Planes\_P\_ID

FOREIGN KEY(S\_Plane)

REFERENCES public.Planes(P\_ID)

ON DELETE SET NULL,

CONSTRAINT FK\_Schedule\_Companies\_C\_ID

FOREIGN KEY(S\_Company)

REFERENCES public.Companies(C\_ID)

ON DELETE SET NULL,

CONSTRAINT FK\_Schedule\_Airport\_A\_ID\_From

FOREIGN KEY(S\_From)

REFERENCES public.Airports(A\_ID)

ON DELETE SET NULL,

CONSTRAINT FK\_Schedule\_Airport\_A\_ID\_To

FOREIGN KEY(S\_To)

REFERENCES public.Airports(A\_ID)

ON DELETE SET NULL

);

CREATE INDEX IX\_Schedule\_S\_Plane ON Schedule(S\_Plane);

CREATE INDEX IX\_Schedule\_S\_Company ON Schedule(S\_Company);

CREATE INDEX IX\_Schedule\_S\_From ON Schedule(S\_From);

CREATE INDEX IX\_Schedule\_S\_To ON Schedule(S\_To);

CREATE INDEX IX\_Schedule\_S\_ID ON Schedule(S\_ID);

ALTER TABLE Schedule CLUSTER ON IX\_Schedule\_S\_ID;

CLUSTER Schedule;

CREATE TABLE IF NOT EXISTS public.Buyers (

B\_ID INT

GENERATED ALWAYS AS IDENTITY

NOT NULL

UNIQUE

PRIMARY KEY,

B\_FullName VARCHAR(40),

B\_Discount INT

DEFAULT 0

);

CREATE INDEX IX\_Buyers\_B\_FullName ON Buyers(B\_FullName);

ALTER TABLE Buyers CLUSTER ON IX\_Buyers\_B\_FullName;

CLUSTER Buyers;

CREATE TABLE IF NOT EXISTS public.Tickets (

T\_ID INT

GENERATED ALWAYS AS IDENTITY

NOT NULL

UNIQUE

PRIMARY KEY,

T\_Seat INT

DEFAULT 0,

T\_Flight INT

DEFAULT 0,

T\_Buyer INT

DEFAULT 0,

T\_Cost FLOAT

DEFAULT 0,

CONSTRAINT FK\_Tickets\_Schedule\_S\_ID

FOREIGN KEY(T\_Flight)

REFERENCES public.Schedule(S\_ID)

ON DELETE SET NULL,

CONSTRAINT FK\_Tickets\_Buyers\_B\_ID

FOREIGN KEY(T\_Buyer)

REFERENCES public.Buyers(B\_ID)

ON DELETE SET NULL

);

CREATE INDEX IX\_Tickets\_T\_Flight ON Tickets(T\_Flight);

ALTER TABLE Tickets CLUSTER ON IX\_Tickets\_T\_Flight;

CLUSTER Tickets;

CREATE INDEX IX\_Tickets\_T\_Buyer ON Tickets(T\_Buyer);

delete from Airports;

delete from Planes;

delete from Companies;

delete from Schedule;

delete from Buyers;

delete from Tickets;

alter sequence Airports\_A\_ID\_seq restart;

alter sequence Planes\_P\_ID\_seq restart;

alter sequence Companies\_C\_ID\_seq restart;

alter sequence Schedule\_S\_ID\_seq restart;

alter sequence Buyers\_B\_ID\_seq restart;

alter sequence Tickets\_T\_ID\_seq restart;

--Airports

select \* from airports\_insert('Аэропорт Хитроу', 'Великобритания', 'Лондон', '5');

select \* from airports\_insert('Аэропорт Париж — Шарль-де-Голль', 'Франция', 'Париж', '3');

select \* from airports\_insert('Международный аэропорт имени Ататюрка', 'Турция', 'Стамбул', '2');

select \* from airports\_insert('Аэропорт Франкфурт-на-Майне', 'Германия', 'Франкфурт', '3');

select \* from airports\_insert('Амстердамский аэропорт Схипхол', 'Нидерланды', 'Амстердам', '1');

select \* from airports\_insert('Аэропорт Мадрид-Барахас', 'Испания', 'Мадрид', '4');

select \* from airports\_insert('Аэропорт Фьюмичино', 'Италия', 'Рим', '5');

select \* from airports\_insert('Международный аэропорт Шереметьево', 'Россия', 'Москва', '8');

--Planes

select \* from planes\_insert('1', 'Airbus A380-800', '700');

select \* from planes\_insert('1', 'Boeing 747-8', '605');

select \* from planes\_insert('6', 'Boeing 747-400', '660');

select \* from planes\_insert('7', 'Boeing 777-300', '550');

select \* from planes\_insert('7', 'Airbus A340-600', '475');

select \* from planes\_insert('8', 'Boeing 777-200', '440');

select \* from planes\_insert('8', 'Airbus A350-900', '380');

select \* from planes\_insert('8', 'Airbus A333-300', '335');

--Companies

select \* from companies\_insert('American Airlines', '96', 'Дуг Паркер', 'США');

select \* from companies\_insert('Emirates', '98', 'Тим Кларк', 'ОАЭ');

select \* from companies\_insert('Ryanair', '90', 'Майкл Коули', 'Ирландия');

select \* from companies\_insert('Lufthansa', '94', 'Карстен Шпор', 'Германия');

select \* from companies\_insert('Аэрофлот', '92', 'Савельев В.Г.', 'Россия');

--Schedule

select \* from schedule\_insert('2021-02-12 04:05:00', '1', '1', '1', '6', true);

select \* from schedule\_insert('2021-02-12 14:16:00', '2', '3', '2', '4', true);

select \* from schedule\_insert('2021-02-11 02:56:00', '3', '2', '2', '3', true);

select \* from schedule\_insert('2021-02-11 06:15:00', '4', '1', '7', '1', true);

select \* from schedule\_insert('2021-02-11 18:29:00', '5', '4', '5', '2', true);

select \* from schedule\_insert('2021-02-10 01:01:00', '6', '4', '1', '8', true);

select \* from schedule\_insert('2021-02-10 21:44:00', '7', '3', '8', '6', true);

select \* from schedule\_insert('2021-02-09 17:37:00', '8', '5', '3', '7', true);

--Buyers

select \* from buyers\_insert('Буков Игорь Матвеевич', '0');

select \* from buyers\_insert('Качурина Наталия Тимуровна', '10');

select \* from buyers\_insert('Лысов Денис Гаврилович', '0');

select \* from buyers\_insert('Калмыкова Анфиса Елизаровна', '0');

select \* from buyers\_insert('Кахманова Григорий Артемович', '0');

select \* from buyers\_insert('Ямковой Демьян Валерьянович', '5');

select \* from buyers\_insert('Мамонова Екатерина Филипповна', '0');

select \* from buyers\_insert('Ижутин Игорь Куприянович', '7');

--Tickets

call tickets\_insert('101', '1', '1', '500');

call tickets\_insert('21', '2', '2', '2000');

call tickets\_insert('112', '3', '3', '3000');

call tickets\_insert('251', '4', '4', '4000');

call tickets\_insert('111', '5', '5', '5000');

call tickets\_insert('47', '6', '6', '6000');

call tickets\_insert('310', '7', '7', '7000');

call tickets\_insert('57', '8', '8', '8000');

# ВЫПОЛНЕННЫЕ ЗАДАНИЯ

### В приложении реализовать не менее пяти запросов, включая (для демонстрации навыков работы)

### Составной многотабличный запрос с параметром, включающий соединение таблиц и CASE-выражение;

CREATE FUNCTION public.request\_2a(\_requested\_id integer) RETURNS TABLE(\_id integer, \_fullName varchar(40), \_discount integer, \_final\_Cost float)

LANGUAGE plpgsql

AS $$

begin

return query

select Tickets.T\_ID, Buyers.B\_FullName, Buyers.B\_Discount,

case

WHEN Buyers.B\_Discount = 0

then Tickets.T\_Cost

else

calculate\_discount(Tickets.T\_Cost, Buyers.B\_Discount)

end \_cost

from Tickets

inner join Buyers

on Tickets.T\_Buyer = Buyers.B\_ID

where Tickets.T\_ID = \_requested\_id;

end

$$;

### На основе обновляющего представления (многотабличного VIEW), в котором критерий упорядоченности задает пользователь при выполнении;

CREATE VIEW public.request\_2b\_by\_name\_view AS

SELECT Companies.C\_Name, COUNT(Schedule.S\_Company) FROM Schedule

INNER JOIN Companies ON Schedule.S\_Company = Companies.C\_ID

GROUP BY Companies.C\_Name

ORDER BY Companies.C\_Name;

CREATE VIEW public.request\_2b\_by\_count\_view AS

SELECT Companies.C\_Name, COUNT(Schedule.S\_Company) as \_count FROM Schedule

INNER JOIN Companies ON Schedule.S\_Company = Companies.C\_ID

GROUP BY Companies.C\_Name

ORDER BY \_count DESC;

### Запрос, содержащий коррелированные и некоррелированные подзапросы в разделах SELECT, FROM и WHERE (в каждом хотя бы по одному)

### Кореллированный WHERE и некореллированный SELECT:

### CREATE OR REPLACE FUNCTION public.request\_2c1() RETURNS TABLE(\_company varchar(40), \_from VARCHAR(40), \_to VARCHAR(40), "Avg Price" INT)

### LANGUAGE plpgsql

### AS $$

### begin

### return query

### SELECT

### C\_Name,

### aFrom.\_Country,

### aTo.\_Country,

### (SELECT AVG(T\_Cost) FROM Tickets WHERE T\_Flight = S\_ID)::INT

### FROM

### Schedule

### INNER JOIN (SELECT \* FROM Companies WHERE C\_Name = 'American Airlines') s ON C\_ID = S\_Company

### INNER JOIN (SELECT \* FROM airports\_select()) aFrom ON aFrom.\_ID = S\_From

### INNER JOIN (SELECT \* FROM airports\_select()) aTo ON aTo.\_ID = S\_To;

### end;

### $$;

### Кореллированный FOR и некореллированный WHERE:

### CREATE OR REPLACE FUNCTION public.request\_2c2() RETURNS TABLE(\_name varchar(40), \_terminals integer)

### LANGUAGE plpgsql

### AS $$

### begin

### return query

### select \_airports.A\_Name, \_airports.A\_Terminals FROM Planes,

### LATERAL (SELECT A\_Name, A\_Terminals FROM Airports WHERE A\_ID = Planes.P\_HomeAirport) as \_airports

### WHERE \_airports.A\_Terminals > (SELECT AVG(A\_Terminals) FROM Airports)

### GROUP BY \_airports.A\_Name, \_airports.A\_Terminals;

### end

### $$;

### Кореллированный WHERE и некореллированный SELECT:

### CREATE OR REPLACE FUNCTION public.request\_2c3() RETURNS TABLE(\_model varchar(40), \_seats INT, "Min Seats" INT, "Avg Seats" INT, "Max Seats" INT)

### LANGUAGE plpgsql

### AS $$

### begin

### return query

### SELECT P\_Model, P\_Seats,

### (SELECT MIN(P\_Seats) FROM Planes) minseats,

### (SELECT AVG(P\_Seats) FROM Planes)::INT avgseats,

### (SELECT MAX(P\_Seats) FROM Planes) maxseats

### FROM

### Planes

### WHERE

### EXISTS(SELECT 1 FROM Schedule WHERE S\_Plane = P\_ID);

### end

### $$;

### Многотабличный запрос, содержащий группировку записей, агрегативные функции и параметр, используемый в разделе HAVING;

CREATE OR REPLACE FUNCTION public.request\_2d(\_amount INT) RETURNS TABLE(\_country varchar(40), \_airport varchar(40), \_amount bigint)

LANGUAGE plpgsql

AS $$

begin

return query

SELECT A\_Country, A\_Name, COUNT(P\_ID)

FROM Planes

INNER JOIN public.Airports ON A\_ID = P\_HomeAirport

GROUP BY A\_Country, A\_Name

HAVING COUNT(P\_ID) >= \_amount;

end

$$;

### Запрос, содержащий предикат ANY(SOME) или ALL;

CREATE OR REPLACE FUNCTION public.request\_2e() RETURNS TABLE(\_name varchar(40), \_rating integer, \_location varchar(40))

LANGUAGE plpgsql

AS $$

begin

return query

SELECT C\_Name, C\_Rating, C\_Location

FROM Companies

WHERE C\_Rating >= ANY(SELECT AVG(C\_Rating) FROM Companies GROUP BY C\_Location);

end

$$;

### Создать индексы для увеличения скорости выполнения запросов;

###### (см. пункт Схема базы данных)

### В таблице (в соответствии с вариантом) предусмотреть поле, которое заполняется автоматически по срабатыванию триггера при добавлении, обновлении и удалении данных, иметь возможность продемонстрировать работу триггера при работе приложения. Триггеры должны обрабатывать только те записи, которые были добавлены, изменены или удалены в ходе текущей операции (транзакции).

CREATE FUNCTION public.schedule\_data\_check() RETURNS trigger

LANGUAGE plpgsql

AS $$

BEGIN

IF (TG\_OP = 'INSERT') THEN

IF (NEW.S\_DateTime < clock\_timestamp()) THEN

RETURN NULL;

ELSE

RETURN NEW;

END IF;

END IF;

IF (TG\_OP = 'UPDATE') THEN

IF (NEW.S\_DateTime < OLD.S\_DateTime) THEN

NEW.S\_DateTime = OLD.S\_DateTime;

END IF;

RETURN NEW;

END IF;

IF (TG\_OP = 'DELETE') THEN

UPDATE public.Schedule SET S\_Active = false WHERE S\_ID = OLD.S\_ID;

RETURN NULL;

END IF;

END;

$$;

CREATE TRIGGER schedule\_trigger BEFORE INSERT OR UPDATE OR DELETE ON public.Schedule FOR EACH ROW EXECUTE FUNCTION public.schedule\_data\_check();

### Операции добавления, удаления и обновления реализовать в виде хранимых процедур (с параметрами) хотя бы для одной таблицы; для остальных допустимо использовать возможности связывания полей ввода в приложении с полями БД;

--Airports

CREATE FUNCTION public.airports\_select() RETURNS TABLE(\_id integer, \_name varchar(40), \_country varchar(40), \_city varchar(40), \_terminals integer)

LANGUAGE plpgsql

AS $$

begin

return query

select \* from Airports order by A\_ID;

end

$$;

CREATE FUNCTION public.airports\_insert(\_name varchar(40), \_country varchar(40), \_city varchar(40), \_terminals integer) RETURNS integer

LANGUAGE plpgsql

AS $$

begin

insert into Airports(A\_Name, A\_Country, A\_City, A\_Terminals)

values(\_name, \_country, \_city, \_terminals);

if found then

return 1;

else

return 0;

end if;

end

$$;

CREATE FUNCTION public.airports\_update(\_id integer, \_name varchar(40), \_country varchar(40), \_city varchar(40), \_terminals integer) RETURNS integer

LANGUAGE plpgsql

AS $$

begin

update Airports

set

A\_Name = \_name,

A\_Country = \_country,

A\_City = \_city,

A\_Terminals = \_terminals

where

A\_ID = \_id;

if found then

return 1;

else

return 0;

end if;

end

$$;

CREATE FUNCTION public.airports\_delete(\_id integer) RETURNS integer

LANGUAGE plpgsql

AS $$

begin

delete from Airports

where

A\_ID = \_id;

if found then

return 1;

else

return 0;

end if;

end

$$;

--Planes

CREATE FUNCTION public.planes\_select() RETURNS TABLE(\_id integer, \_homeAirport integer, \_model varchar(40), \_seats integer)

LANGUAGE plpgsql

AS $$

begin

return query

select \* from Planes order by P\_ID;

end

$$;

CREATE FUNCTION public.planes\_insert(\_homeAirport integer, \_model varchar(40), \_seats integer) RETURNS integer

LANGUAGE plpgsql

AS $$

begin

insert into Planes(P\_HomeAirport, P\_Model, P\_Seats)

values(\_homeAirport, \_model, \_seats);

if found then

return 1;

else

return 0;

end if;

end

$$;

CREATE FUNCTION public.planes\_update(\_id integer, \_homeAirport integer, \_model varchar(40), \_seats integer) RETURNS integer

LANGUAGE plpgsql

AS $$

begin

update Planes

set

P\_HomeAirport = \_homeAirport,

P\_Model = \_model,

P\_Seats = \_seats

where

P\_ID = \_id;

if found then

return 1;

else

return 0;

end if;

end

$$;

CREATE FUNCTION public.planes\_delete(\_id integer) RETURNS integer

LANGUAGE plpgsql

AS $$

begin

delete from Planes

where

P\_ID = \_id;

if found then

return 1;

else

return 0;

end if;

end

$$;

--Companies

CREATE FUNCTION public.companies\_select() RETURNS TABLE(\_id integer, \_name varchar(40), \_rating integer, \_director varchar(40), \_location varchar(40))

LANGUAGE plpgsql

AS $$

begin

return query

select \* from Companies order by C\_ID;

end

$$;

CREATE FUNCTION public.companies\_insert(\_name varchar(40), \_rating integer, \_director varchar(40), \_location varchar(40)) RETURNS integer

LANGUAGE plpgsql

AS $$

begin

insert into Companies(C\_Name, C\_Rating, C\_Director, C\_Location)

values(\_name, \_rating, \_director, \_location);

if found then

return 1;

else

return 0;

end if;

end

$$;

CREATE FUNCTION public.companies\_update(\_id integer, \_name varchar(40), \_rating integer, \_director varchar(40), \_location varchar(40)) RETURNS integer

LANGUAGE plpgsql

AS $$

begin

update Companies

set

C\_Name = \_name,

C\_Rating = \_rating,

C\_Director = \_director,

C\_Location = \_location

where

C\_ID = \_id;

if found then

return 1;

else

return 0;

end if;

end

$$;

CREATE FUNCTION public.companies\_delete(\_id integer) RETURNS integer

LANGUAGE plpgsql

AS $$

begin

delete from Companies

where

C\_ID = \_id;

if found then

return 1;

else

return 0;

end if;

end

$$;

--Schedule

CREATE FUNCTION public.schedule\_select() RETURNS TABLE(\_id integer, \_dateTime timestamp, \_plane integer, \_company integer, \_from integer, \_to integer, \_active boolean)

LANGUAGE plpgsql

AS $$

begin

return query

select \* from Schedule order by S\_ID;

end

$$;

CREATE FUNCTION public.schedule\_insert(\_dateTime timestamp, \_plane integer, \_company integer, \_from integer, \_to integer, \_active boolean) RETURNS integer

LANGUAGE plpgsql

AS $$

begin

insert into Schedule(S\_DateTime, S\_Plane, S\_Company, S\_From, S\_To, S\_Active)

values(\_dateTime, \_plane, \_company, \_from, \_to, \_active);

if found then

return 1;

else

return 0;

end if;

end

$$;

CREATE FUNCTION public.schedule\_update(\_id integer, \_dateTime timestamp, \_plane integer, \_company integer, \_from integer, \_to integer, \_active boolean) RETURNS integer

LANGUAGE plpgsql

AS $$

begin

update Schedule

set

S\_DateTime = \_dateTime,

S\_Plane = \_plane,

S\_Company = \_company,

S\_From = \_from,

S\_To = \_to,

S\_Active = \_active

where

S\_ID = \_id;

if found then

return 1;

else

return 0;

end if;

end

$$;

CREATE FUNCTION public.schedule\_delete(\_id integer) RETURNS integer

LANGUAGE plpgsql

AS $$

begin

delete from Schedule

where

S\_ID = \_id;

if found then

return 1;

else

return 0;

end if;

end

$$;

--Buyers

CREATE FUNCTION public.buyers\_select() RETURNS TABLE(\_id integer, \_fullName varchar(40), \_discount integer)

LANGUAGE plpgsql

AS $$

begin

return query

select \* from Buyers order by B\_ID;

end

$$;

CREATE FUNCTION public.buyers\_insert(\_fullName varchar(40), \_discount integer) RETURNS integer

LANGUAGE plpgsql

AS $$

begin

insert into Buyers(B\_FullName, B\_Discount)

values(\_fullName, \_discount);

if found then

return 1;

else

return 0;

end if;

end

$$;

CREATE FUNCTION public.buyers\_update(\_id integer, \_fullName varchar(40), \_discount integer) RETURNS integer

LANGUAGE plpgsql

AS $$

begin

update Buyers

set

B\_FullName = \_fullName,

B\_Discount = \_discount

where

B\_ID = \_id;

if found then

return 1;

else

return 0;

end if;

end

$$;

CREATE FUNCTION public.buyers\_delete(\_id integer) RETURNS integer

LANGUAGE plpgsql

AS $$

begin

delete from Buyers

where

B\_ID = \_id;

if found then

return 1;

else

return 0;

end if;

end

$$;

--Tickets

CREATE FUNCTION public.tickets\_select() RETURNS TABLE(\_id integer, \_seat integer, \_flight integer, \_buyer integer, \_cost float)

LANGUAGE plpgsql

AS $$

begin

return query

select \* from Tickets order by T\_ID;

end

$$;

CREATE FUNCTION public.tickets\_update(\_id integer, \_seat integer, \_flight integer, \_buyer integer, \_cost float) RETURNS integer

LANGUAGE plpgsql

AS $$

begin

update Tickets

set

T\_Seat = \_seat,

T\_Flight = \_flight,

T\_Buyer = \_buyer,

T\_Cost = \_cost

where

T\_ID = \_id;

if found then

return 1;

else

return 0;

end if;

end

$$;

CREATE FUNCTION public.tickets\_delete(\_id integer) RETURNS integer

LANGUAGE plpgsql

AS $$

begin

delete from Tickets

where

T\_ID = \_id;

if found then

return 1;

else

return 0;

end if;

end

$$;

### Реализовать отдельную хранимую процедуру, состоящую из нескольких отдельных операций в виде единой транзакции, которая при определенных условиях может быть зафиксирована или откатана;

CREATE PROCEDURE public.tickets\_insert(\_seat integer, \_flight integer, \_buyer integer, \_cost float)

LANGUAGE plpgsql

AS $$

declare

\_amount\_of\_seats integer;

begin

SELECT P\_Seats FROM Planes INTO \_amount\_of\_seats WHERE P\_ID = (SELECT S\_Plane FROM Schedule WHERE S\_ID = \_flight);

insert into Tickets(T\_Seat, T\_Flight, T\_Buyer, T\_Cost)

values(\_seat, \_flight, \_buyer, \_cost);

if (\_seat >= \_amount\_of\_seats) then

rollback;

end if;

end

$$;

### В триггере или хранимой процедуре реализовать курсор на обновления отдельных данных;

CREATE OR REPLACE PROCEDURE public.RefreshSchedule()

LANGUAGE plpgsql

AS $$

DECLARE \_id INT;

\_date TIMESTAMP;

\_active BOOLEAN;

cur CURSOR FOR SELECT S\_ID, S\_DateTime, S\_Active FROM public.Schedule WHERE S\_Active = true;

x Schedule%rowtype;

BEGIN

FOR x IN cur loop

IF (x.S\_DateTime < clock\_timestamp()) THEN

UPDATE public.Schedule SET s\_active = false WHERE s\_id = x.S\_ID;

END IF;

END loop;

END

$$;

### В запросе (из пункта 2 или в дополнительном к тому перечню) использовать собственную скалярную функцию, а в хранимой процедуре – векторную (или табличную) функцию. Функции сохранить в базе данных;

###### ---Скалярная функция---

CREATE FUNCTION public.calculate\_discount(\_base\_cost float, \_discount integer) RETURNS FLOAT

LANGUAGE plpgsql

AS $$

begin

return \_base\_cost \* ((100.0 - \_discount) / 100.0);

end

$$;

###### ---Векторная функция---

###### Любая %table\_name%\_select() функция

### Распределение прав пользователей: предусмотреть не менее двух пользователей с разным набором привилегий. Каждый набор привилегий оформить в виде роли.

CREATE ROLE manager;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE public.airports TO manager;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE public.buyers TO manager;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE public.companies TO manager;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE public.planes TO manager;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE public.schedule TO manager;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE public.tickets TO manager;

GRANT USAGE ON SCHEMA public TO manager;

CREATE USER george WITH

IN ROLE manager

ENCRYPTED PASSWORD 'floyd';

CREATE ROLE ticketseller;

GRANT SELECT ON TABLE public.airports TO ticketseller;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE public.buyers TO ticketseller;

GRANT SELECT ON TABLE public.companies TO ticketseller;

GRANT SELECT ON TABLE public.planes TO ticketseller;

GRANT SELECT, INSERT, UPDATE ON TABLE public.schedule TO ticketseller;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE public.tickets TO ticketseller;

GRANT USAGE ON SCHEMA public TO ticketseller;

CREATE USER cassa1 WITH

IN ROLE ticketseller

ENCRYPTED PASSWORD 'cassacassacassa';

CREATE USER cassa2 WITH

IN ROLE ticketseller

ENCRYPTED PASSWORD 'cassacassa?';

# ЛИСТИНГ ПРОГРАММЫ

**MainWindow.xaml.cs**

using System;

using System.Windows;

using System.Windows.Controls;

using Npgsql;

using System.Data;

namespace DB\_Client

{

public partial class MainWindow : Window

{

private NpgsqlConnection connection;

private Runner w;

public MainWindow()

{

InitializeComponent();

w = new Runner();

w.Show();

w.SendGrid(DataBaseGrid);

}

private void ConnectButton\_Click(object sender, RoutedEventArgs e)

{

if (connection != null)

{

connection.CloseAsync();

w.RefreshConnection(null);

}

try

{

NpgsqlConnectionStringBuilder builder = new NpgsqlConnectionStringBuilder();

builder.Database = "postgres";

builder.Host = AddressField.Text;

builder.Username = UsernameField.Text;

builder.Password = PasswordField.Password;

connection = new NpgsqlConnection(builder.ConnectionString);

connection.Open();

w.RefreshConnection(connection);

w.ReloadRoutines();

} catch (NpgsqlException)

{

Console.WriteLine("Connection failed");

}

}

private DataTable SendQuery(string request)

{

DataTable table = new DataTable();

try

{

NpgsqlCommand command = new NpgsqlCommand(request, connection);

NpgsqlDataAdapter adapter = new NpgsqlDataAdapter(command);

adapter.Fill(table);

} catch (NpgsqlException)

{

Console.WriteLine("Command failed");

} catch (InvalidCastException)

{

Console.WriteLine("Connect first");

}

return table;

}

private void TableButton\_Click(object sender, RoutedEventArgs e)

{

DataTable table = SendQuery("SELECT \* FROM public." + ((string)((Button)sender).Content).ToLower() + "\_select()");

DataBaseGrid.ItemsSource = table.DefaultView;

}

}

**ParameterEntry.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace DB\_Client

{

class ParameterEntry

{

public string Type { get; set; }

public string Name { get; set; }

public string Text { get; set; }

public string getText()

{

return '\'' + Text + '\'';

}

public ParameterEntry(string type, string name)

{

Type = type;

Name = name;

}

}

}

**ParameterViewModel.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Collections.ObjectModel;

namespace DB\_Client

{

class ParameterViewModel

{

private static ObservableCollection<ParameterEntry> \_Parameters;

public static ObservableCollection<ParameterEntry> Parameters

{

get { return \_Parameters; }

set { \_Parameters = value; }

}

public ParameterViewModel()

{

Parameters = new ObservableCollection<ParameterEntry>();

}

}

}

**RoutineEntry.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Collections.ObjectModel;

namespace DB\_Client

{

class RoutineEntry

{

public string Name { get; set; }

public int Id { get; set; }

public char Type { get; set; } // P / F

public ObservableCollection<ParameterEntry> Parameters { get; set; }

public RoutineEntry(int id, string name, char type)

{

Id = id;

Name = name;

Type = type;

Parameters = new ObservableCollection<ParameterEntry>();

}

}

}

**RoutineViewModel.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Collections.ObjectModel;

namespace DB\_Client

{

class RoutineViewModel

{

private static ObservableCollection<RoutineEntry> \_Routines;

public static ObservableCollection<RoutineEntry> Routines

{

get { return \_Routines; }

set { \_Routines = value; }

}

private static RoutineEntry \_Routine;

public static RoutineEntry Routine

{

get { return \_Routine; }

set { \_Routine = value; }

}

public RoutineViewModel()

{

Routines = new ObservableCollection<RoutineEntry>();

}

}

}

**Runner.xaml.cs**

using System;

using System.Windows;

using System.Windows.Controls;

using System.Collections.ObjectModel;

using System.Data;

using Npgsql;

namespace DB\_Client

{

public partial class Runner : Window

{

private NpgsqlConnection connection;

private DataGrid grid;

private ObservableCollection<RoutineEntry> \_Routines;

private ObservableCollection<RoutineEntry> Routines

{

get { return \_Routines; }

set { \_Routines = value; }

}

public Runner()

{

InitializeComponent();

Routines = new ObservableCollection<RoutineEntry>();

comboBox.ItemsSource = Routines;

}

public void RefreshConnection(NpgsqlConnection \_connection)

{

connection = \_connection;

}

public void SendGrid(DataGrid \_grid)

{

grid = \_grid;

}

private void ComboBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

try

{

ObservableCollection<ParameterEntry> parameters = ((RoutineEntry)e.AddedItems[0]).Parameters;

ArgsItemControl.ItemsSource = parameters;

} catch (IndexOutOfRangeException)

{

((ComboBox)sender).SelectedIndex = -1;

ArgsItemControl.ItemsSource = null;

}

}

public void ReloadRoutines()

{

try

{

if (connection != null)

{

DataTable procs = GetRoutines();

Routines = new ObservableCollection<RoutineEntry>();

for (int i = 0; i < procs.Rows.Count; i++)

{

Routines.Add(new RoutineEntry(i, procs.Rows[i].Field<string>(0), procs.Rows[i].Field<string>(1) == "PROCEDURE" ? 'P' : 'F'));

DataTable parameters = GetParameters(Routines[i].Name);

for (int j = 0; j < parameters.Rows.Count; j++)

{

Routines[i].Parameters.Add(new ParameterEntry(parameters.Rows[j].Field<string>(0), parameters.Rows[j].Field<string>(1)));

}

}

comboBox.ItemsSource = Routines;

}

} catch (NpgsqlException exception)

{

Console.WriteLine("Can't load routines.\n" + exception.Message);

}

}

private DataTable GetRoutines()

{

NpgsqlCommand command = new NpgsqlCommand("select concat(specific\_schema, '.', routine\_name) as procedure\_name, routine\_type from information\_schema.routines " +

"where routine\_schema not in ('pg\_catalog', 'information\_schema') order by procedure\_name;", connection);

NpgsqlDataAdapter adapter = new NpgsqlDataAdapter(command);

DataTable table = new DataTable();

adapter.Fill(table);

return table;

}

private DataTable GetParameters(string routineName)

{

routineName = routineName.Split('.')[1];

NpgsqlCommand command = new NpgsqlCommand("select p.data\_type, p.parameter\_name " +

"from information\_schema.routines r left join information\_schema.parameters p on r.specific\_schema = p.specific\_schema and r.specific\_name = p.specific\_name " +

"where r.routine\_schema not in ('pg\_catalog', 'information\_schema') AND p.parameter\_mode = 'IN' AND r.routine\_name = \'" + routineName + "\' order by p.ordinal\_position;", connection);

NpgsqlDataAdapter adapter = new NpgsqlDataAdapter(command);

DataTable table = new DataTable();

adapter.Fill(table);

return table;

}

private void ExecuteButton\_Click(object sender, RoutedEventArgs e)

{

string query;

if (((RoutineEntry)comboBox.SelectedItem).Type == 'P')

{

query = "exec " + ((RoutineEntry)comboBox.SelectedItem).Name + "(";

} else

{

query = "SELECT \* FROM " + ((RoutineEntry)comboBox.SelectedItem).Name + "(";

}

if (ArgsItemControl.Items.Count > 0)

{

query += ((ParameterEntry)ArgsItemControl.Items[0]).getText();

}

for (int i = 1; i < ArgsItemControl.Items.Count; i++)

{

query += ", " + ((ParameterEntry)ArgsItemControl.Items[i]).getText();

}

query += ")";

try

{

NpgsqlCommand command = new NpgsqlCommand(query, connection);

if (((RoutineEntry)comboBox.SelectedItem).Type == 'F')

{

NpgsqlDataAdapter adapter = new NpgsqlDataAdapter(command);

DataTable table = new DataTable();

adapter.Fill(table);

grid.ItemsSource = table.DefaultView;

}

} catch (NpgsqlException)

{

Console.WriteLine("Routine failed: " + query);

}

}

}

# ДЕМОНСТРАЦИЯ РАБОТЫ ПРИЛОЖЕНИЯ

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# ИСПОЛЬЗУЕМАЯ ЛИТЕРАТУРА

### Аллен, Г. Тейлор SQL для чайников / Аллен Г. Тейлор. - М.: Диалектика, Вильямс, 2015. - 416 c.

### Бен, Форта SQL за 10 минут / Форта Бен. - М.: Диалектика / Вильямс, 2015.

### - 0 c.

### Бьюли, Алан Изучаем SQL / Алан Бьюли. - М.: Символ-плюс, 2014. - 0 c.

### Грабер, Мартин SQL для простых смертных / Мартин Грабер. - М.: ЛОРИ, 2014. - 378 c.

### Гудсон, Джон Практическое руководство по доступу к данным (+ DVD- ROM) / Джон Гудсон , Роб Стюард. - М.: БХВ-Петербург, 2013. - 304 c.

### Дейт, К. Дж. SQL и реляционная теория. Как грамотно писать код на SQL

### / К.Дж. Дейт. - М.: Символ-плюс, 2016. - 0 c.

### Джеймс, Р. Грофф SQL. Полное руководство / Джеймс Р. Грофф, Пол Н. Вайнберг, Эндрю Дж. Оппель. - М.: Вильямс, 2014. - 960 c.