МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

дисциплина «Базы данных»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 5

«Проектирование и создание реляционной базы данных в заданной предметной области, работа с ней с использованием *ADO*.*NET*»

Вариант №5

Выполнил: студент гр. ИТИ-22

Карпенко Д.Е.

Принял: преподаватель-стажёр

Карась О.В.

Гомель 2023

**Цель работы:** получение навыков проектирования и создания БД *MS SQL Server*, а также выполнения операций с ней из внешнего программного кода с использованием библиотек доступа к данным Microsoft *ADO.NET*.

**Задание:**

1. Спроектировать c использованием метода *ER*-диаграмм и, используя средства *Transact SQL*, создать БД *MS SQL Server*, соответствующую своему варианту задания.

Структура созданной БД должна быть нормализована – таблицы должны удовлетворять требованиям третьей нормальной формы.

Должны быть назначены первичные и внешние ключи и установлены ограничения целостности данных.

2. Написать хранимые процедуры с параметрами для вставки данных во все таблицы базы данных.

3. Создать консольное приложение на языке *С*#, которое используя объекты *ADO*.*NET* (*Connection*, *Command*, *Parameter*, *DataReader*, *Transaction*), будет выполнять:

3.1. Инициализацию не менеe трех таблиц созданной базы данных начальным набором записей.

При выборе таблиц для заполнения тестовыми наборами руководствоваться следующим:

* выбранные таблицы должны содержать основную информацию, касающиеся предметной области приложения;
* таблицы должны быть связаны непосредственно;
* не менее, чем одна таблица должна находиться в схеме базы данных на стороне отношения «многие»;
* таблицы на стороне отношения «один» должны содержать не менее 100 записей;
* таблицы на стороне отношения «многие» должны содержать не менее 10000 записей.

Заполнение таблиц данными осуществлять в следующем порядке:

* сначала генерируются данные для таблиц, находящихся в схеме базы данных на стороне отношения «один» (таблицы «справочники»),
* потом – таблицы, находящиеся на стороне отношения «многие» («оперативные» таблицы).

Выполнения операций вставки данных организовать внутри транзакции. Собственно вставку данных внутри транзакции организовать путем циклических вызовов через соответствующий объект *Command ADO*.*NET*.

3.2. Вывод на консоль результатов выполнения запросов на выборку или сообщений о результатах выполнения операций изменения данных (вставки, удаления, обновления) согласно варианту задания.

Вариант задания представлен в таблице 1.

Таблица 1 – Вариант задания

|  |  |  |  |
| --- | --- | --- | --- |
| Вариант | Предметная область | Примерный минимальный перечень информации для хранения в таблицах | Запросы и операции по обработке информации |
| 4 | Аэропорт | 1. Сотрудники (ФИО, Возраст, Пол, Адрес, Телефон, Паспортные данные, Должность). 2. Должности (Наименование должности, Оклад, Обязанности, Требования). 3. Самолёты (Марка, Вместимость, Грузоподъемность, Тип, Технические характеристики, Дата выпуска, Налётано часов, Дата последнего ремонта). 4. Типы самолётов (Наименование, Назначение, Ограничения). 5. Рейсы (Дата, Время, Пункт вылета, Пункт назначения, Самолет, Время полёта). 6. Билеты (ФИО пассажира, Паспортные данные, Должность, Место, Рейс, Цена). | 1. Данные о сотрудниках и должностях, на которых они работают. 2. Информация о самолетах и пилотах, которые на них летали. 3. Список рейсов, которые выполняются только внутри страны. 4. Параметрический запрос для отображения информации о сотрудниках, работающих на определенной должности. 5. Параметрический запрос для отображения информации о самолётах заданного типа. 6. Вычислить количество сделанных рейсов каждым пилотом. 7. Перекрестный запрос с информацией об общей стоимости проданных билетов для различных рейсов на каждую дату. 8. Добавление нового рейса. 9. Обновление данных о рейсе. 10. Удаление данных о рейсе. |

**Ход работы**

На рисунке 1 представлена *ER*-диаграмма структуры БД.

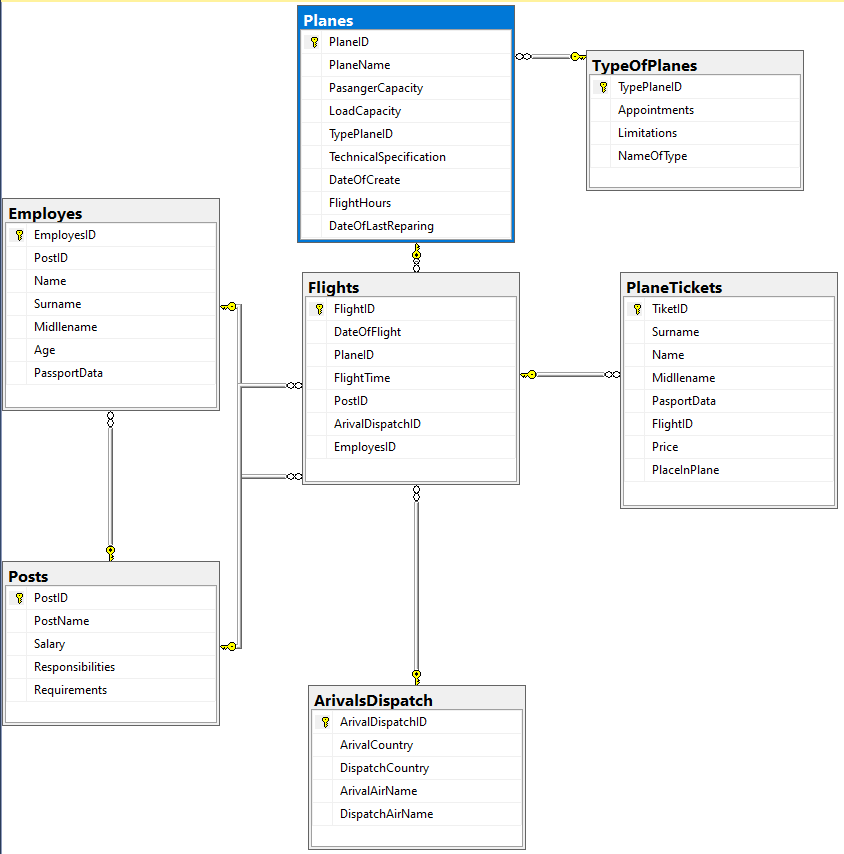
****

Рисунок 1 – *ER*-диаграмми

На рисунке 2 представлена структура таблицы «*ArivalsDispatch*»

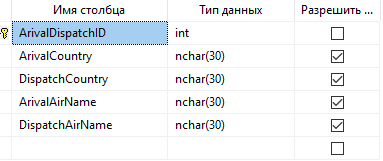


Рисунок 2 – Таблица «*ArivalsDispatch*»

На рисунке 3 представлена структура таблицы «*EmployesID*»

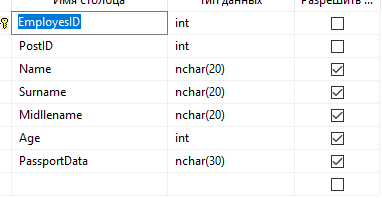


Рисунок 3 – Таблица «*EmployesID*»

На рисунке 4 представлена структура таблицы «*Flights*»

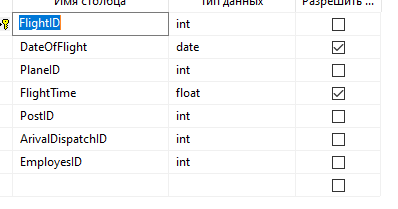
****

Рисунок 4 – Таблица «*Flights*»

На рисунке 5 представлена структура таблицы «*Planes*»

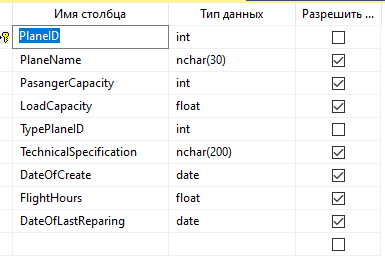


Рисунок 5 – Таблица «*Planes*»

На рисунке 6 представлена структура таблицы «*PlaneTikets*»

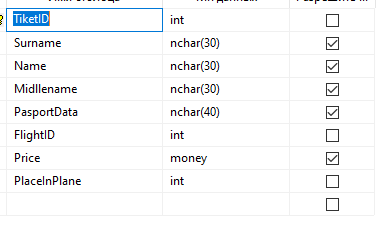


Рисунок 6 – Таблица «*PlaneTikets*»

На рисунке 7 представлена структура таблицы «*Posts*»

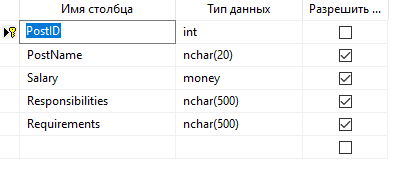


Рисунок 7 – Таблица «*Posts*»

На рисунке 8 представлена структура таблицы «*TypeOfPlanes*»

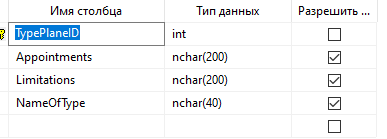


Рисунок 8 – Таблица «*TypeOfPlanes*»

На рисунке 8 представлена структура БД «*Airlines*»

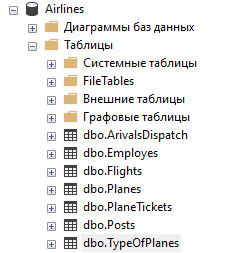


Рисунок 9 – Структура БД «*Airlines*»

Хранимые процедуры представлены в приложении А.

На рисунке 10 представлен фрагмент кода, который добавляет начальные значения в таблицы на стороне отношений «один» с помощью *Transaction*.

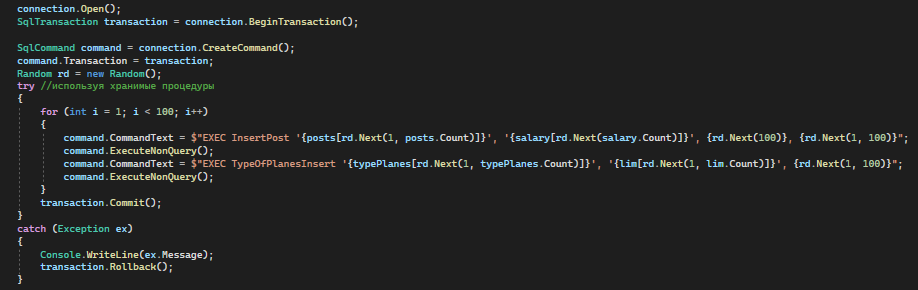


Рисунок 10 – Добавление начальных значений

На рисунке 11 представлен фрагмент кода, который добавляет начальные значения в таблицы на стороне отношений «многие» с помощью *Transaction*.

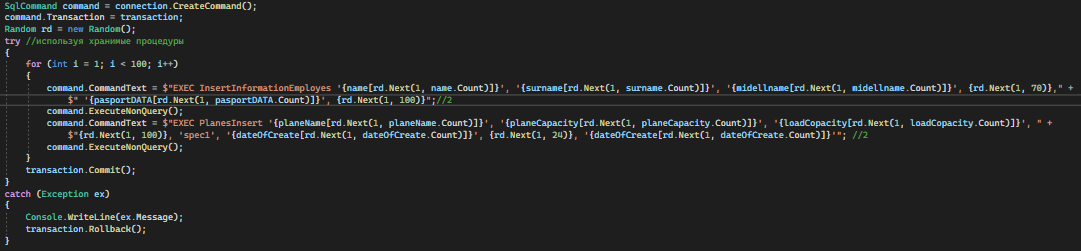
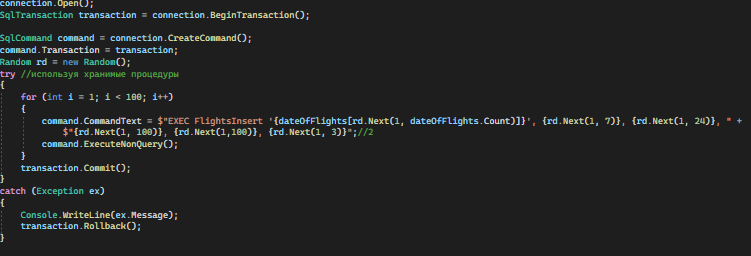
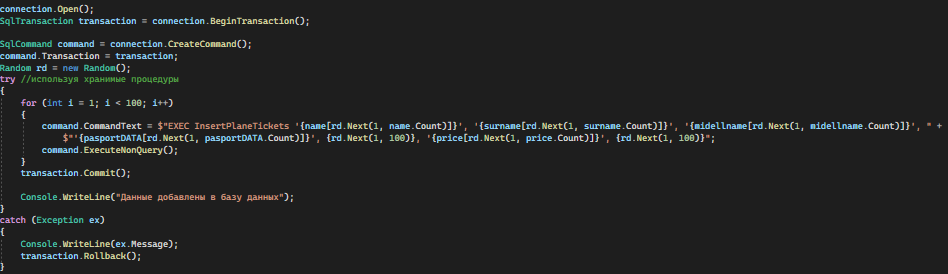
  

Рисунок 11 – Добавление начальных значений

На рисунке 12 представлены исходные данные заполненной таблицы «*ArivalDispatch*». Данная таблица является вспомогательной и в задании к ЛР её нету, в следствии этого она заполнена на 7 строк.

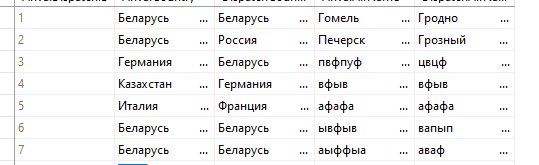


Рисунок 12 – Данные таблицы « *ArivalDispatch* »

На рисунке 13 представлены исходные данные заполненной таблицы «*Employes*».

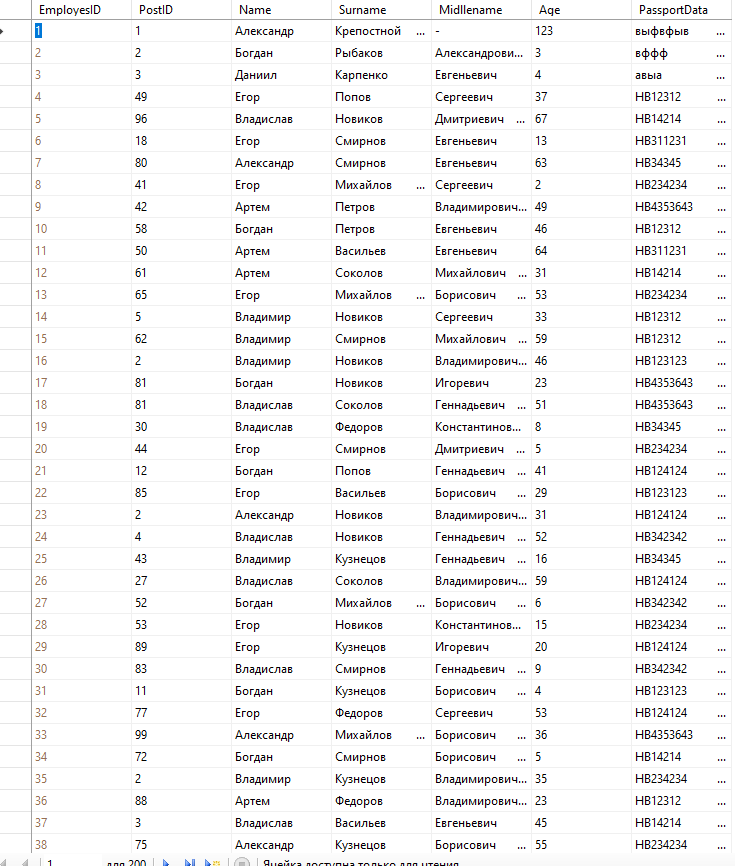


Рисунок 13 – Данные таблицы «*Employes*»

На рисунке 14 представлены исходные данные заполненных таблиц «*Flights*» и «*Planes*».

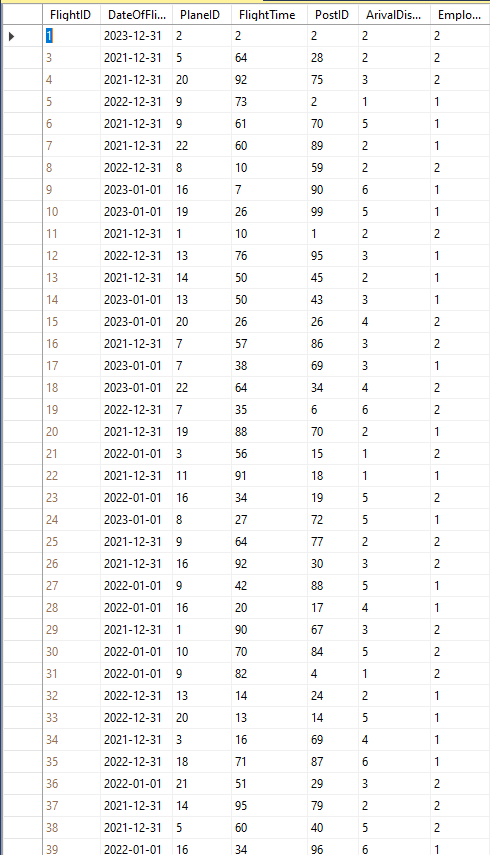
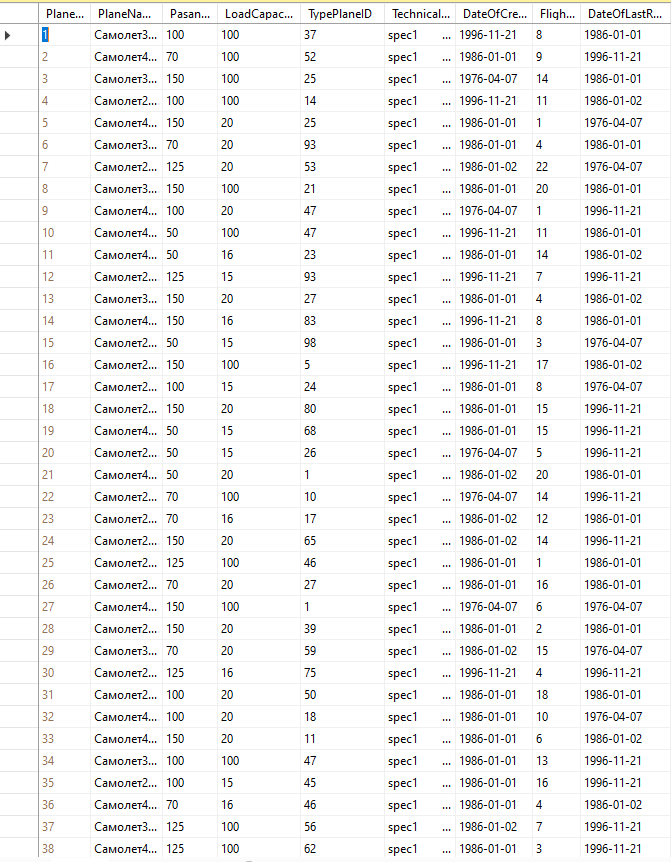
 

Рисунок 14 – Данные таблиц «*Flights*» и «*Planes*»

На рисунке 15 представлены исходные данные заполненной таблицы «*PlaneTkets*» и «*Posts*».

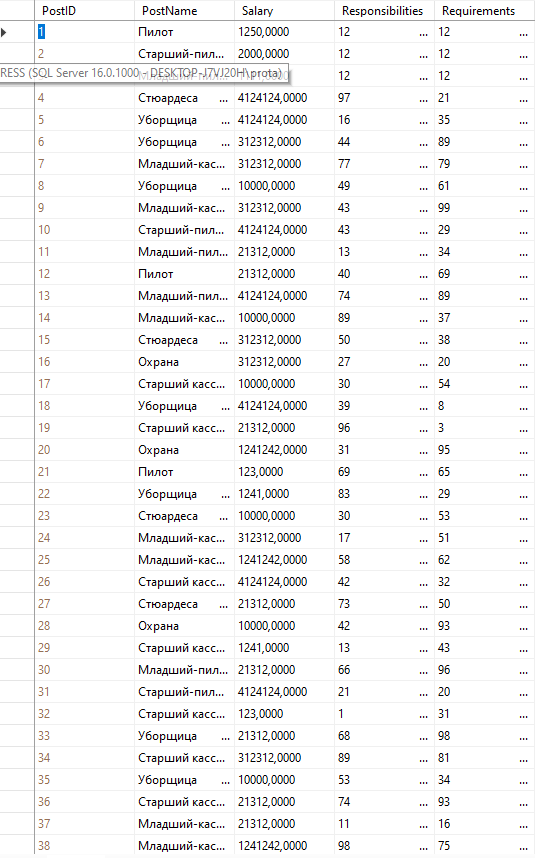
 

Рисунок 15 – Данные таблиц «*PlaneTkets*» и «*Posts*»

На рисунке 16 представлены исходные данные заполненной таблицы «*TypesOfPlanes*».

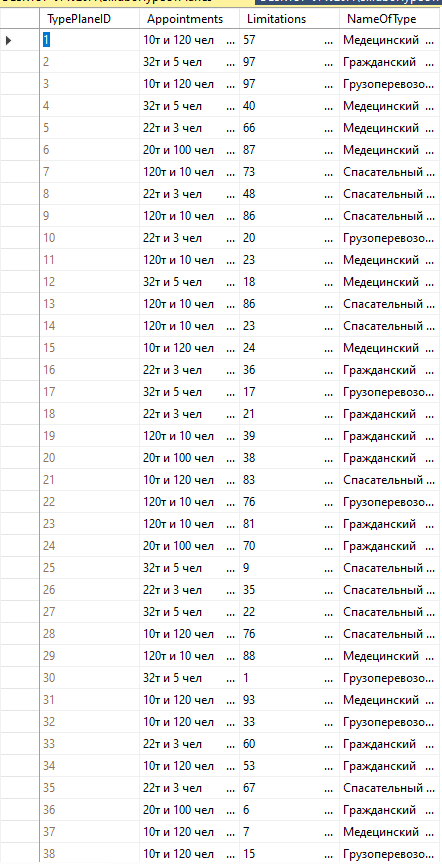


Рисунок 16 – Данные таблиц «*TypesOfPlanes*»

На рисунке 17 представлено главное меню консольного приложения с подключенной *SQL*.

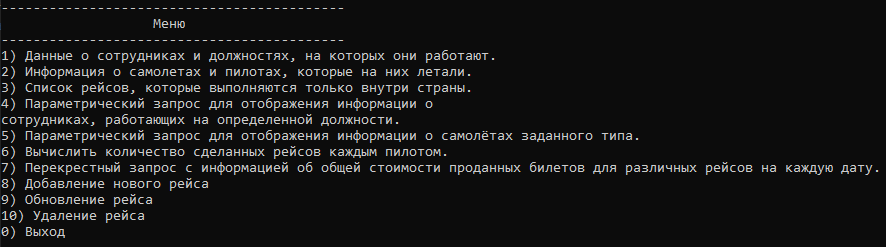


Рисунок 17 – Главное меню

На рисунке 18 представлен вывод данных о сотрудниках и их должностях, на которых они работают.

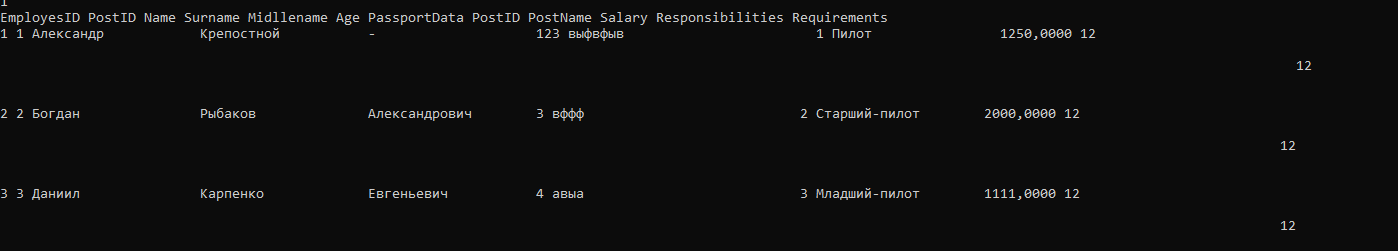


Рисунок 18 – Данные о сотрудниках и их должностях, на которых они работают

На рисунке 19 представлена информация о самолетах и пилотах, которые на них летали.

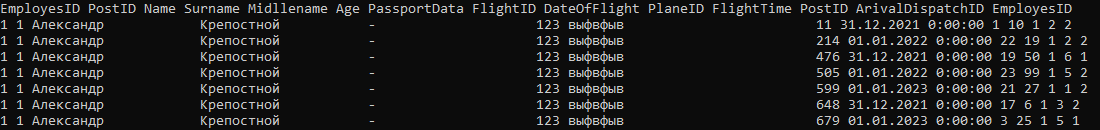


Рисунок 19 – Информация о самолетах и пилотах, которые на них летали

На рисунке 20 представлен список рейсов, которые выполняются только внутри страны.

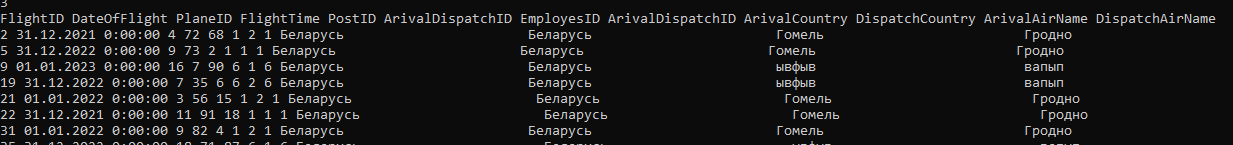


Рисунок 20 – Список рейсов, которые выполняются только внутри страны

На рисунке 21 представлен параметрический запрос для отображения информации осотрудниках, работающих на определенной должности.

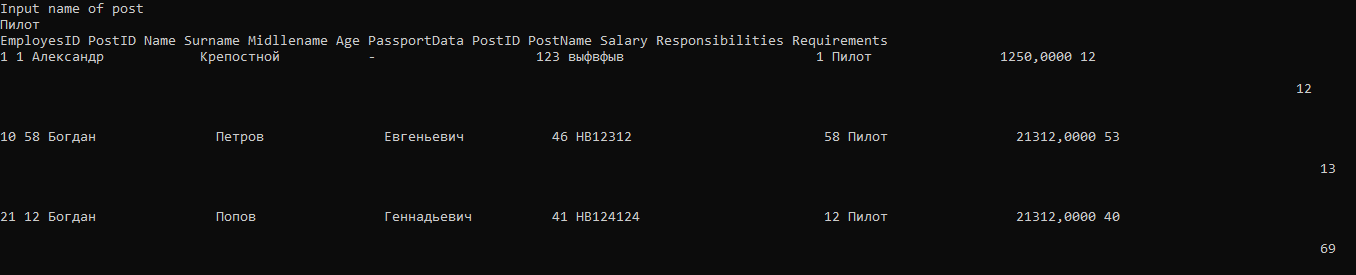


Рисунок 21 – Параметрический запрос для отображения информации о

сотрудниках, работающих на определенной должности

На рисунке 22 представлен параметрический запрос для отображения информации о самолётах заданного типа.

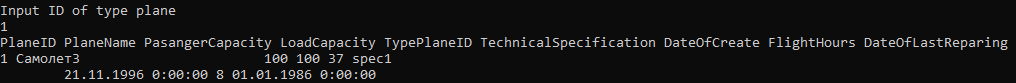


Рисунок 22 – Параметрический запрос для отображения информации о самолётах заданного типа.

На рисунке 23 предствлен список с количество сделанных рейсов каждым пилотом.

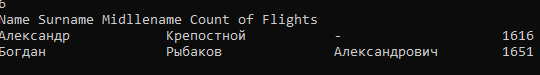


Рисунок 23 – Список с количество сделанных рейсов каждым пилотом

На рисунке 24 представлен список с перекрестным запросом с информацией об общей стоимости проданных билетов для различных рейсов на каждую дату.

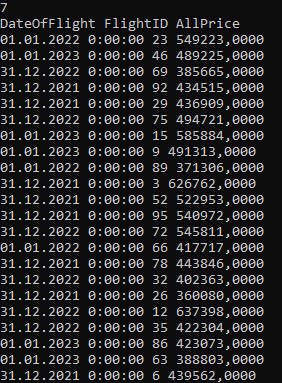
****

Рисунок 24 – Перекрестный запрос с информацией об общей стоимости проданных билетов для различных рейсов на каждую дату

На рисунке 25 представлено добавление нового рейса.

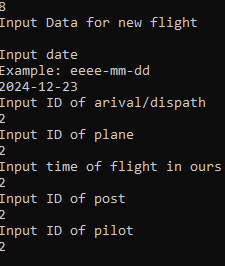


Рисунок 25 – Добавление нового рейса

На рисунке 26 представлено обновление рейса.

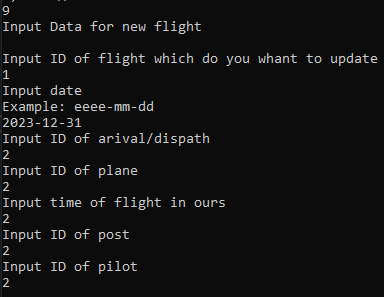


Рисунок 26 – Обновление рейса

На рисунке 27 представлено удаление рейса по его айди.



Рисунок 27 – Удаление рейса по его айди

**Вывод**: в результате выполнения лабораторной работы были получены знания и навыки в проектировании и создании БД *MS SQL Server*, а также выполнения операций с ней из внешнего программного кода с использованием библиотек доступа к данным *Microsoft* *ADO.NET*.

**ПРИЛОЖЕНИЕ А**

**Текст скриптов**

**Файл *EmployesInsert.sql*:**

CREATE PROCEDURE InsertInformationEmployes(@Name nchar(20),@Surname nchar(20), @Midellname nchar(20), @Age int, @PasportData nchar(30), @PostID int)

AS

INSERT INTO Employees([Name], Surname, Midllename, Age, PassportData, PostID)

VALUES (@Name, @Surname, @Midellname, @Age, @PasportData, @PostID)

**Файл *FlightsInsert.sql*:**

CREATE PROCEDURE FlightsInsert(@DateOfFlight date, @ArivalDispatchID int, @PlaneID int, @FlightTime float, @PostID int)

AS

INSERT INTO Flights (DateOfFlight, ArivalDispatchID,PlaneID, FlightTime, PostID)

VALUES (@DateOfFlight, @ArivalDispatchID, @PlaneID, @FlightTime, @PostID)

**Файл *PlanesInsert.sql*:**

CREATE PROCEDURE PlanesInsert(@PlaneName nchar(40), @PasangerCapacity int, @LoadCapacity float, @TypePlaneID int, @TechnicalSpecification nchar(200),

@DateOfCreate date, @FlightHours float, @DateOfLastReparing date)

AS

INSERT INTO Planes (PlaneName, PasangerCapacity, LoadCapacity, TypePlaneID, TechnicalSpecification, DateOfCreate, FlightHours, DateOfLastReparing)

VALUES (@PlaneName, @PasangerCapacity, @LoadCapacity, @TypePlaneID, @TechnicalSpecification, @DateOfCreate, @FlightHours, @DateOfLastReparing)

**Файл *InsertPlaneTickets.sql*:**

CREATE PROCEDURE InsertPlaneTickets(@Surname nchar(30), @Name nchar(30), @Medellname nchar(30), @PasportData nchar(40), @FlightID int, @Price money, @PlaceInPlane int)

AS

INSERT INTO PlaneTickets(Surname, [Name], Midllename, PasportData, FlightID, Price, PlaceInPlane)

VALUES (@Surname, @Name, @Medellname, @PasportData, @FlightID, @Price, @PlaceInPlane)

**Файл *InsertPost.sql*:**

CREATE PROCEDURE InsertPost(@NameOfPost nchar(20), @Salary money, @Responsibilities nchar(500), @Requirements nchar(500))

AS

INSERT INTO Posts(PostName, Salary, Responsibilities, Requirements)

VALUES (@NameOfPost, @Salary, @Responsibilities, @Requirements)

**Файл *TypeOfPlanesInsert.sql*:**

CREATE PROCEDURE TypeOfPlanesInsert(@NameOfType nchar(30), @Appointments nchar(500), @Limitations nchar(500))

AS

INSERT INTO TypeOfPlanes(NameOfType, Appointments, Limitations)

VALUES (@NameOfType, @Appointments, @Limitations)

**Файл *DeleteFlights.sql*:**

CREATE PROCEDURE DeleteFlights(@FlightID int)

AS

DELETE FROM Flights

WHERE FlightID = @FlightID

**Файл *UpdateFlights.sql*:**

CREATE PROCEDURE UpdateFlights(@FlightID int, @DateOfFlight date,@ArivalDispatchID int,@PlaneID int, @FlightTime float, @PostID int, @EmployesID int)

AS

UPDATE Flights

SET DateOfFlight = @DateOfFlight, PlaneID = @PlaneID, FlightTime = @FlightTime, PostID = @PostID, EmployesID = @EmployesID, ArivalDispatchID = @ArivalDispatchID

WHERE @FlightID = FlightID

**Файл *CMDInterface.cs*:**

using BisnesLogic;

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

namespace Laba\_BD\_5

{

public class CMDInterface

{

public static string path = @"Data Source= DESKTOP-J7VJ20H\SQLEXPRESS;Initial Catalog=Airlines;Integrated Security=True";

private static Logic Logic = new Logic();

private static List<string> table = new List<string>();

static int stop = 1;

static void Main(string[] args)

{

using (SqlConnection conn = new SqlConnection(path))

{

Logic logic = new Logic(conn, path);

Logic = logic;

try

{

Console.WriteLine("База данных подключена!\n\*Нажмите любую клавишу\*\n");

Console.ReadKey();

Console.Clear();

while (stop != 0)

{

Menu();

}

}

catch (Exception e)

{

Console.WriteLine("База данных отключена!\n");

Console.WriteLine("Error: " + e);

Console.WriteLine(e.StackTrace);

}

finally

{

conn.Close();

conn.Dispose();

}

}

}

static List<string> flight = new List<string>();

static void Menu()

{

Console.WriteLine("-------------------------------------------");

Console.WriteLine(" Меню ");

Console.WriteLine("-------------------------------------------");

Console.WriteLine("1) Данные о сотрудниках и должностях, на которых они работают.");

Console.WriteLine("2) Информация о самолетах и пилотах, которые на них летали.");

Console.WriteLine("3) Список рейсов, которые выполняются только внутри страны.");

Console.WriteLine("4) Параметрический запрос для отображения информации о \n" +

"сотрудниках, работающих на определенной должности.");

Console.WriteLine("5) Параметрический запрос для отображения информации о самолётах заданного типа.");

Console.WriteLine("6) Вычислить количество сделанных рейсов каждым пилотом.");

Console.WriteLine("7) Перекрестный запрос с информацией об общей стоимости проданных билетов для различных рейсов на каждую дату.");

Console.WriteLine("8) Добавление нового рейса");

Console.WriteLine("9) Обновление рейса");

Console.WriteLine("10) Удаление рейса");

Console.WriteLine("0) Выход");

int a;

do

{

try

{

a = Convert.ToInt32(Console.ReadLine());

}

catch

{

a = -1;

}

}

while (a < 0 || a > 10);

switch (a)

{

case 0:

Console.WriteLine("Программа завершена!");

stop=0;

break;

case 1:

table = Logic.GetEmployes();

for (int i = 0; i < 100; i++)

{

Console.WriteLine(table[i]);

}

break;

case 2:

table = Logic.GetInfAboutPilots();

for (int i = 0; i < 100; i++)

{

Console.WriteLine(table[i]);

}

break;

case 3:

table = Logic.GetFlightsWhichInCountry();

for (int i = 0; i < 100; i++)

{

Console.WriteLine(table[i]);

}

break;

case 4:

Console.WriteLine("Input name of post");

string namePost = Console.ReadLine();

table = Logic.GetAllEmployesWhithConcretNameOfPost(namePost);

for (int i = 0; i < 100; i++)

{

Console.WriteLine(table[i]);

}

break;

case 5:

Console.WriteLine("Input ID of type plane");

int IDPlane = Convert.ToInt32(Console.ReadLine());

table = Logic.GetAllPlanesOfOneType(IDPlane);

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

{

Console.WriteLine(table[i]);

}

break;

case 6:

table = Logic.GetCountOfAllFlightsOfAllPilots();

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

{

Console.WriteLine(table[i]);

}

break;

case 7:

table = Logic.GetAllInfOfSales();

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

{

Console.WriteLine(table[i]);

}

break;

case 8:

flight = new List<string>();

Console.WriteLine("Input Data for new flight\n");

Console.WriteLine("Input date \nExample: eeee-mm-dd");

flight.Add(Console.ReadLine());

Console.WriteLine("Input ID of arival/dispath");

flight.Add(Console.ReadLine());

Console.WriteLine("Input ID of plane");

flight.Add(Console.ReadLine());

Console.WriteLine("Input time of flight in ours");

flight.Add(Console.ReadLine());

Console.WriteLine("Input ID of post");

flight.Add(Console.ReadLine());

Console.WriteLine("Input ID of pilot");

flight.Add(Console.ReadLine());

Logic.ExecSetFlight(flight.ToArray());

break;

case 9:

flight = new List<string>();

Console.WriteLine("Input Data for new flight\n");

Console.WriteLine("Input ID of flight which do you whant to update");

flight.Add(Console.ReadLine());

Console.WriteLine("Input date \nExample: eeee-mm-dd");

flight.Add(Console.ReadLine());

Console.WriteLine("Input ID of arival/dispath");

flight.Add(Console.ReadLine());

Console.WriteLine("Input ID of plane");

flight.Add(Console.ReadLine());

Console.WriteLine("Input time of flight in ours");

flight.Add(Console.ReadLine());

Console.WriteLine("Input ID of post");

flight.Add(Console.ReadLine());

Console.WriteLine("Input ID of pilot");

flight.Add(Console.ReadLine());

Logic.ExecUpdateFlights(flight.ToArray());

break;

case 10:

Console.WriteLine("Input ID of flight which do you whant to delete");

int id = Convert.ToInt32(Console.ReadLine());

Logic.ExecDeleteOneFlight(id);

break;

default:

Console.Write("Error!");

break;

}

}

}

}

**Файл *Logic.cs*:**

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

namespace BisnesLogic

{

public class Logic

{

string path;

SqlConnection conn;

public Logic() { }

public Logic(SqlConnection connection, string path)

{

this.conn = connection;

this.path = path;

SetDataOne();

SetSecondDATA();

SetTherdDATA();

SetForthDATA();

}

public void AbsExec(string command)

{

using (SqlConnection connection = new SqlConnection(path))

{

connection.Open();

SqlTransaction transaction = connection.BeginTransaction();

SqlCommand cmmnd = connection.CreateCommand();

cmmnd.Transaction = transaction;

cmmnd.CommandText = command;

cmmnd.ExecuteNonQuery();

transaction.Commit();

}

}

private List<string> AbsTabel(string command)

{

List<string> list = new List<string>();

string str = null;

using (SqlConnection conn = new SqlConnection(path))

{

conn.Open();

SqlCommand cmmnd = new SqlCommand(command, conn);

SqlDataReader reader = cmmnd.ExecuteReader();

if (reader.HasRows) // если есть данные

{

int countOfCols = reader.FieldCount;

for (int i = 0; i < countOfCols; i++)

{

str = str + reader.GetName(i) + " ";

}

list.Add(str);

while (reader.Read())

{

string row = null;

for (int i = 0; i < countOfCols; i++)

{

row = row + Convert.ToString(reader.GetValue(i)) + " ";

}

list.Add(row);

}

}

}

return list;

}

//1. Данные о сотрудниках и должностях, на которых они работают.

public List<string> GetEmployes()

{

string command = "SELECT \* FROM Employes INNER JOIN Posts ON Posts.PostID = Employes.PostID";

return AbsTabel(command);

}

//2. Информация о самолетах и пилотах, которые на них летали.

public List<string> GetInfAboutPilots()

{

string command = "SELECT \* FROM Employes, Flights WHERE Flights.PostID = Employes.PostID";

return AbsTabel(command);

}

//3. Список рейсов, которые выполняются только внутри страны.

public List<string> GetFlightsWhichInCountry()

{

string country = "Беларусь";

string command = "SELECT \* FROM Flights, ArivalsDispatch WHERE Flights.ArivalDispatchID = ArivalsDispatch.ArivalDispatchID AND" +

" ArivalsDispatch.ArivalCountry = \'" + country + "\' AND ArivalsDispatch.DispatchCountry = \'" + country + "\'";

return AbsTabel(command);

}

//4. Параметрический запрос для отображения информации о сотрудниках, работающих на определенной должности.

public List<string> GetAllEmployesWhithConcretNameOfPost(string nameOfPost)

{

string command = "SELECT \* FROM Employes, Posts WHERE Employes.PostID = Posts.PostID AND Posts.PostName = \'" + nameOfPost + "\'";

return AbsTabel(command);

}

//5. Параметрический запрос для отображения информации о самолётах заданного типа.

public List<string> GetAllPlanesOfOneType(int IDPlane)

{

string command = "SELECT \* FROM Planes WHERE Planes.PlaneID = " + IDPlane;

return AbsTabel(command);

}

//6. Вычислить количество сделанных рейсов каждым пилотом.

public List<string> GetCountOfAllFlightsOfAllPilots()

{

string comand = "SELECT Employes.Name, Employes.Surname, Employes.Midllename, Count(Flights.FlightID) as 'Count of Flights' " +

"FROM Employes, Flights " +

"WHERE Employes.EmployesID = Flights.EmployesID " +

"GROUP BY Employes.[Name], Employes.Surname, Employes.Midllename";

return AbsTabel(comand);

}

//7. Перекрестный запрос с информацией об общей стоимости проданных билетов для различных рейсов на каждую дату.

public List<string> GetAllInfOfSales()

{

string comand = "SELECT Flights.DateOfFlight, Flights.FlightID, SUM(PlaneTickets.Price) AS AllPrice " +

"FROM PlaneTickets JOIN Flights ON PlaneTickets.FlightID = Flights.FlightID " +

"GROUP BY Flights.DateOfFlight, Flights.FlightID;";

return AbsTabel(comand);

}

//8

public void ExecSetFlight(string[] flight)

{

string command = "Exec FlightsInsert " + "\'" + flight[0] + "\'" + ", " + flight[1] + ", " + flight[2] + ", " + flight[3] + ", " + flight[4] + ", " + flight[5];

AbsExec(command);

}

//9 @FlightID int, @DateOfFlight date,@ArivalDispatchID int,@PlaneID int, @FlightTime float, @PostID int, @EmployesID int

public void ExecUpdateFlights(string[] flight)

{

string command = "Exec UpdateFlights " + flight[0] + ", \'" + flight[1] + "\', " + flight[2] + ", " + flight[3] + ", " + flight[4] + ", " + flight[5] + ", " + flight[6];

AbsExec(command);

}

//10

public void ExecDeleteOneFlight(int id)

{

string delete = "DELETE FROM PlaneTickets \r\nWHERE FlightID = " + id;

AbsTabel(delete);

delete = "UPDATE PlaneTickets \r\nSET FlightID = NULL \r\nWHERE FlightID = " + id;

string command = "Exec DeleteFlights " + id;

AbsExec(command);

}

void SetDataOne()

{

using (SqlConnection connection = new SqlConnection(path))

{

List<string> posts = new List<string> { "Кассир", "Младший-кассир", "Старший кассир", "Пилот", "Старший-пилот", "Младший-пилот", "Стюардеса", "Охрана", "Уборщица" }; //post

List<string> salary = new List<string> { "10000", "21312", "312312", "123", "1241", "4124124", "4124124", "1241242" }; //post

List<string> name = new List<string> { "Даниил", "Александр", "Владислав", "Егор", "Артем", "Богдан", "Владимир" }; //Emploeys

List<string> surname = new List<string> { "Иванов", "Смирнов", "Кузнецов", "Попов", "Васильев", "Петров", "Соколов", "Михайлов", "Новиков", "Федоров" };

List<string> midellname = new List<string> { "Александрович", "Борисович", "Владимирович", "Геннадьевич", "Дмитриевич", "Евгеньевич", "Игоревич", "Константинович", "Михайлович", "Сергеевич" };

string[] age = new string[70];

for (int i = 1; i < age.Length; i++)

{

age[i - 1] = i.ToString();

}

List<string> pasportDATA = new List<string> { "HB124124", "HB124124", "HB14214", "HB311231", "HB34345", "HB12312", "HB342342", "HB234234", "HB4353643", "HB123123" }; //Employes

List<string> typePlanes = new List<string> { "Военный", "Гражданский", "Грузоперевозочный", "Медецинский", "Спасательный" }; //TypePlanes

List<string> lim = new List<string> { "2т и 10 чел", "120т и 10 чел", "22т и 3 чел", "10т и 120 чел", "32т и 5 чел", "20т и 100 чел" };

List<string> planeName = new List<string> { "Самолет1", "Самолет2", "Самолет3", "Самолет4" }; //Planes

List<string> planeCapacity = new List<string> { "40", "50", "70", "100", "125", "150" };

List<string> loadCopacity = new List<string> { "15", "15", "16", "20", "100" };

//typePlane Id 1-100

//tehcnilcalSpecs

List<string> dateOfCreate = new List<string> { "2000-03-22", "1986-01-01", "1986-01-01", "1986-01-02", "1976-04-07", "1996-11-21" };

//flightHours

List<string> dateOfLastRepair = new List<string> { "2000-03-22", "1986-01-01", "1986-01-01", "1986-01-02", "1976-04-07", "1996-11-21" }; //Planes

List<string> ArivalCountry = new List<string> { "Белорусь", "Россия" };//ArivalsDispatch

List<string> DispatchCountry = new List<string> { "Белорусь", "Белорусь", "Белорусь", "Германия", "Россия", "Германия", "Чехия", };

List<string> ArivalAirName = new List<string> { "Городской" };

List<string> DispatchAirName = new List<string> { "Сельский", "Крепостной", "WestEnd", "Doechland" };//ArivalsDispatch

List<string> dateOfFlights = new List<string> { "2021-01-01", "2022-01-01", "2023-01-01", "2022-12-31", "2021-12-31", "2021-12-31" };

//Flights

//PlaneID

//FlightTime

//PostID

//ArivalDispatchID

//EmployesID

//tikets

//name

//surname

//midellname

//PasportDATA

//FlightID

List<string> price = new List<string> { "123123", "123", "3124", "31231", "2131", "32131" };

//place in plane

connection.Open();

SqlTransaction transaction = connection.BeginTransaction();

SqlCommand command = connection.CreateCommand();

command.Transaction = transaction;

Random rd = new Random();

try //используя хранимые процедуры

{

for (int i = 1; i < 100; i++)

{

command.CommandText = $"EXEC InsertPost '{posts[rd.Next(1, posts.Count)]}', '{salary[rd.Next(salary.Count)]}', {rd.Next(100)}, {rd.Next(1, 100)}";

command.ExecuteNonQuery();

command.CommandText = $"EXEC TypeOfPlanesInsert '{typePlanes[rd.Next(1, typePlanes.Count)]}', '{lim[rd.Next(1, lim.Count)]}', {rd.Next(1, 100)}";

command.ExecuteNonQuery();

}

transaction.Commit();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

transaction.Rollback();

}

}

}

void SetSecondDATA()

{

using (SqlConnection connection = new SqlConnection(path))

{

List<string> posts = new List<string> { "Кассир", "Младший-кассир", "Старший кассир", "Пилот", "Старший-пилот", "Младший-пилот", "Стюардеса", "Охрана", "Уборщица" }; //post

List<string> salary = new List<string> { "10000", "21312", "312312", "123", "1241", "4124124", "4124124", "1241242" }; //post

List<string> name = new List<string> { "Даниил", "Александр", "Владислав", "Егор", "Артем", "Богдан", "Владимир" }; //Emploeys

List<string> surname = new List<string> { "Иванов", "Смирнов", "Кузнецов", "Попов", "Васильев", "Петров", "Соколов", "Михайлов", "Новиков", "Федоров" };

List<string> midellname = new List<string> { "Александрович", "Борисович", "Владимирович", "Геннадьевич", "Дмитриевич", "Евгеньевич", "Игоревич", "Константинович", "Михайлович", "Сергеевич" };

string[] age = new string[70];

for (int i = 1; i < age.Length; i++)

{

age[i - 1] = i.ToString();

}

List<string> pasportDATA = new List<string> { "HB124124", "HB124124", "HB14214", "HB311231", "HB34345", "HB12312", "HB342342", "HB234234", "HB4353643", "HB123123" }; //Employes

List<string> typePlanes = new List<string> { "Военный", "Гражданский", "Грузоперевозочный", "Медецинский", "Спасательный" }; //TypePlanes

List<string> lim = new List<string> { "2т и 10 чел", "120т и 10 чел", "22т и 3 чел", "10т и 120 чел", "32т и 5 чел", "20т и 100 чел" };

List<string> planeName = new List<string> { "Самолет1", "Самолет2", "Самолет3", "Самолет4" }; //Planes

List<string> planeCapacity = new List<string> { "40", "50", "70", "100", "125", "150" };

List<string> loadCopacity = new List<string> { "15", "15", "16", "20", "100" };

//typePlane Id 1-100

//tehcnilcalSpecs

List<string> dateOfCreate = new List<string> { "2000-03-22", "1986-01-01", "1986-01-01", "1986-01-02", "1976-04-07", "1996-11-21" };

//flightHours

List<string> dateOfLastRepair = new List<string> { "2000-03-22", "1986-01-01", "1986-01-01", "1986-01-02", "1976-04-07", "1996-11-21" }; //Planes

List<string> dateOfFlights = new List<string> { "2021-01-01", "2022-01-01", "2023-01-01", "2022-12-31", "2021-12-31", "2021-12-31" };

connection.Open();

SqlTransaction transaction = connection.BeginTransaction();

SqlCommand command = connection.CreateCommand();

command.Transaction = transaction;

Random rd = new Random();

try //используя хранимые процедуры

{

for (int i = 1; i < 100; i++)

{

command.CommandText = $"EXEC InsertInformationEmployes '{name[rd.Next(1, name.Count)]}', '{surname[rd.Next(1, surname.Count)]}', '{midellname[rd.Next(1, midellname.Count)]}', {rd.Next(1, 70)}," +

$" '{pasportDATA[rd.Next(1, pasportDATA.Count)]}', {rd.Next(1, 100)}";//2

command.ExecuteNonQuery();

command.CommandText = $"EXEC PlanesInsert '{planeName[rd.Next(1, planeName.Count)]}', '{planeCapacity[rd.Next(1, planeCapacity.Count)]}', '{loadCopacity[rd.Next(1, loadCopacity.Count)]}', " +

$"{rd.Next(1, 100)}, 'spec1', '{dateOfCreate[rd.Next(1, dateOfCreate.Count)]}', {rd.Next(1, 24)}, '{dateOfCreate[rd.Next(1, dateOfCreate.Count)]}'"; //2

command.ExecuteNonQuery();

}

transaction.Commit();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

transaction.Rollback();

}

}

}

void SetTherdDATA()

{

using (SqlConnection connection = new SqlConnection(path))

{

List<string> dateOfFlights = new List<string> { "2021-01-01", "2022-01-01", "2023-01-01", "2022-12-31", "2021-12-31", "2021-12-31" };

connection.Open();

SqlTransaction transaction = connection.BeginTransaction();

SqlCommand command = connection.CreateCommand();

command.Transaction = transaction;

Random rd = new Random();

try //используя хранимые процедуры

{

for (int i = 1; i < 100; i++)

{

command.CommandText = $"EXEC FlightsInsert '{dateOfFlights[rd.Next(1, dateOfFlights.Count)]}', {rd.Next(1, 7)}, {rd.Next(1, 24)}, " +

$"{rd.Next(1, 100)}, {rd.Next(1,100)}, {rd.Next(1, 3)}";//2

command.ExecuteNonQuery();

}

transaction.Commit();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

transaction.Rollback();

}

}

}

void SetForthDATA()

{

using (SqlConnection connection = new SqlConnection(path))

{

List<string> name = new List<string> { "Даниил", "Александр", "Владислав", "Егор", "Артем", "Богдан", "Владимир" }; //Emploeys

List<string> surname = new List<string> { "Иванов", "Смирнов", "Кузнецов", "Попов", "Васильев", "Петров", "Соколов", "Михайлов", "Новиков", "Федоров" };

List<string> midellname = new List<string> { "Александрович", "Борисович", "Владимирович", "Геннадьевич", "Дмитриевич", "Евгеньевич", "Игоревич", "Константинович", "Михайлович", "Сергеевич" };

string[] age = new string[70];

for (int i = 1; i < age.Length; i++)

{

age[i - 1] = i.ToString();

}

List<string> pasportDATA = new List<string> { "HB124124", "HB124124", "HB14214", "HB311231", "HB34345", "HB12312", "HB342342", "HB234234", "HB4353643", "HB123123" }; //Employes

//typePlane Id 1-100

//tehcnilcalSpecs

List<string> dateOfCreate = new List<string> { "2000-03-22", "1986-01-01", "1986-01-01", "1986-01-02", "1976-04-07", "1996-11-21" };

//flightHours

List<string> dateOfLastRepair = new List<string> { "2000-03-22", "1986-01-01", "1986-01-01", "1986-01-02", "1976-04-07", "1996-11-21" }; //Planes

List<string> dateOfFlights = new List<string> { "2021-01-01", "2022-01-01", "2023-01-01", "2022-12-31", "2021-12-31", "2021-12-31" };

List<string> price = new List<string> { "123123", "123", "3124", "31231", "2131", "32131" };

//place in plane

connection.Open();

SqlTransaction transaction = connection.BeginTransaction();

SqlCommand command = connection.CreateCommand();

command.Transaction = transaction;

Random rd = new Random();

try //используя хранимые процедуры

{

for (int i = 1; i < 100; i++)

{

command.CommandText = $"EXEC InsertPlaneTickets '{name[rd.Next(1, name.Count)]}', '{surname[rd.Next(1, surname.Count)]}', '{midellname[rd.Next(1, midellname.Count)]}', " +

$"'{pasportDATA[rd.Next(1, pasportDATA.Count)]}', {rd.Next(1, 100)}, '{price[rd.Next(1, price.Count)]}', {rd.Next(1, 100)}";

command.ExecuteNonQuery();

}

transaction.Commit();

Console.WriteLine("Данные добавлены в базу данных");

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

transaction.Rollback();

}

}

}

}

}