Федеральное государственное бюджетное образовательное

учреждение высшего профессионального образования

**Национальный исследовательский университет**

**«МЭИ»**

**Кафедра Вычислительных Машин Систем и Сетей**

**Отчет по**

**Типовому расчету**

**«Проектирование базы данных»**

**Выполнил**: студент гр. А-08-17

Челышев Э. А.

**Проверил**:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Москва 2019 НИУ МЭИ**

1. **Проектирование базы данных**

Проектирование, получение ER-диаграммы и физической модели (пп. 4-5) были выполнены с использованием приложения PowerDesigner.

**ER-диаграмма:**

****

Поясню некоторые этапы проектирования БД.

Выполнена 1НФ, так как все векторные значения разбиты на части (таблица **Address**, столбцы FirstName, LastName, MiddleName в таблице **Workers**).

Выполнена 2НФ, так как первичные ключи во всех таблицах, кроме таблицы, реализующей связь много-ко-многим, являются простыми. Таблица **PU** имеет составной ключ, равный всей таблице.

Выполнена 3НФ, так как все возможные транзитивные связи вынесены в отдельные таблицы. Так, например, таблица **Cadr** содержит информацию о должностях и окладах, так как в техническом задании прослеживается прямая связь между должностью и окладом работника.

Имеется связь «много-ко-многим», разрешенная введением дополнительной таблицы.

Создан ряд индексов, их выбор определялся исходя из предположения, что именно по выбранным столбцам в выбранных таблицах будет максимальное число запросов. Например, все эти индексы будут использоваться при вызове представлений, созданных в п.4.

Физическая модель получена для языка Transact-SQL (MS SQL SERVER).

create database Tip;

go

use Tipovoy;

go

/\*==============================================================\*/

/\* Table: Address \*/

/\*==============================================================\*/

create table Address (

id\_address int not null identity(1,1),

Region varchar(30) ,

Town varchar(20) ,

Street varchar(30) ,

House tinyint ,

Corpus varchar(2) ,

Flat int ,

constraint PK\_ADDRESS primary key (id\_address)

)

go

/\*==============================================================\*/

/\* Table: Cadr \*/

/\*==============================================================\*/

create table Cadr (

id\_cadr int not null identity(1,1),

Doljnost varchar(100) ,

Salary money ,

constraint PK\_CADR primary key (id\_cadr)

)

go

/\*==============================================================\*/

/\* Table: Devises \*/

/\*==============================================================\*/

create table Devises (

id\_dev int not null identity(1,1),

id\_ws int not null,

device varchar(25) ,

quantity int ,

constraint PK\_DEVISES primary key (id\_dev)

)

go

/\*==============================================================\*/

/\* Table: Materials \*/

/\*==============================================================\*/

create table Materials (

id\_mat int not null identity(1,1),

id\_ws int not null,

material varchar(25) ,

quantity float ,

measurerment varchar(5) ,

constraint PK\_MATERIALS primary key (id\_mat)

)

go

/\*==============================================================\*/

/\* Table: PU \*/

/\*==============================================================\*/

create table PU (

id\_proj int not null,

id\_usl int not null,

constraint PK\_PU primary key (id\_proj, id\_usl)

)

go

/\*==============================================================\*/

/\* Table: Projects \*/

/\*==============================================================\*/

create table Projects (

id\_pr int not null identity(1,1),

Pr\_Name varchar(30) ,

id\_zak int not null,

constraint PK\_PROJECTS primary key (id\_pr)

)

go

/\*==============================================================\*/

/\* Table: Uslugi \*/

/\*==============================================================\*/

create table Uslugi (

id\_usl int not null identity(1,1),

id\_ws int not null,

Usl\_Name varchar(30) null,

Min\_Cost money null,

constraint PK\_USLUGI primary key (id\_usl)

)

go

/\*==============================================================\*/

/\* Table: Workers \*/

/\*==============================================================\*/

create table Workers (

id\_w int not null identity(1,1),

LastName varchar(20) null,

FirstName varchar(20) null,

MiddleName varchar(20) null,

Graphic\_smen varchar(10) null,

id\_ws int not null,

id\_c int not null,

Brigada int null,

id\_a int not null,

Phone bigint null,

Email varchar(25) null,

constraint PK\_WORKERS primary key (id\_w)

)

go

/\*==============================================================\*/

/\* Table: Workshops \*/

/\*==============================================================\*/

create table Workshops (

id\_ws int not null identity(1,1),

WS\_Name varchar(30) ,

constraint PK\_WORKSHOPS primary key (id\_ws)

)

go

/\*==============================================================\*/

/\* Table: Zakazchick \*/

/\*==============================================================\*/

create table Zakazchick (

id\_z int not null identity(1,1),

Z\_Name varchar(30) ,

Region varchar(30) ,

Town varchar(20) ,

Street varchar(30) ,

House tinyint ,

Corpus varchar(2) ,

Flat int ,

constraint PK\_ZAKAZCHICK primary key (id\_z)

)

go

alter table Devises

add constraint FK\_DEVISES\_RELATIONS\_WORKSHOP foreign key (id\_ws)

references Workshops (id\_ws)

go

alter table Materials

add constraint FK\_MATERIAL\_RELATIONS\_WORKSHOP foreign key (id\_ws)

references Workshops (id\_ws)

go

alter table PU

add constraint FK\_PU\_RELATIONS\_PROJECTS foreign key (id\_proj)

references Projects (id\_pr)

go

alter table PU

add constraint FK\_PU\_RELATIONS\_USLUGI foreign key (id\_usl)

references Uslugi (id\_usl)

go

alter table Projects

add constraint FK\_PROJECTS\_RELATIONS\_ZAKAZCHI foreign key (id\_zak)

references Zakazchick (id\_z)

go

alter table Uslugi

add constraint FK\_USLUGI\_RELATIONS\_WORKSHOP foreign key (id\_ws)

references Workshops (id\_ws)

go

alter table Workers

add constraint FK\_WORKERS\_RELATIONS\_WORKSHOP foreign key (id\_ws)

references Workshops (id\_ws)

go

alter table Workers

add constraint FK\_WORKERS\_RELATIONS\_CADR foreign key (id\_c)

references Cadr (id\_cadr)

go

alter table Workers

add constraint FK\_WORKERS\_RELATIONS\_ADDRESS foreign key (id\_a)

references Address (id\_address)

go

create index w\_cadr on Workers (

id\_c ASC

)

go

create index w\_ws on Workers (

id\_ws ASC

)

go

create index w\_brig on Workers (

Brigada ASC

)

go

create index m\_ws on Materials (

id\_ws ASC

)

go

create index d\_ws on Devises (

id\_ws ASC

)

go

1. **Создание скалярной и табличной функций**

Данная скалярная функция производит вычисление суммарной стоимости проекта по его идентификатору. Используется в представлении **PROJ**.

CREATE FUNCTION Contr\_Cost(@id int)

RETURNS money

AS

BEGIN

DECLARE @sum money

SELECT @sum = SUM(RES.Min\_Cost) FROM (Select PU.id\_proj, PU.id\_usl, Uslugi.Min\_Cost from PU

left join Uslugi on PU.id\_usl=Uslugi.id\_usl) AS RES WHERE RES.id\_proj = @id;

IF(@sum IS NULL) SET @sum = 0

RETURN @sum

END

Данная табличная функция выводит информацию о работниках отдельного цеха. Использована для реализации представлений по цехам

CREATE FUNCTION workers\_in\_ws

(

@id INT

)

RETURNS TABLE

AS

RETURN

(SELECT t1.LastName+' '+t1.FirstName+' ' +t1.MiddleName AS 'ФИО', t1.Graphic\_smen AS 'График смен',

t1.Brigada AS '№ бригады', t1.Phone AS 'Телефон',t1.Email AS 'E-mail',

t2.Doljnost AS 'Должность', t3.Region AS 'Регион', t3.Town AS 'Город',

t3.Street AS 'Улица', t3.House AS 'Дом', t3.Corpus AS 'Корпус', t3.Flat AS 'Кв.' from

Workers t1 join Cadr t2 on t1.id\_c=t2.id\_cadr join Address t3 on id\_a=id\_address where t1.id\_ws=@id)

GO

Данная табличная функция выводит информацию об оборудовании отдельного цеха.

CREATE FUNCTION devices\_in\_ws

(

@id INT

)

RETURNS TABLE

AS

RETURN

(SELECT id\_dev AS 'id', device AS 'Оборудование', quantity AS 'Количество' FROM Devises WHERE id\_ws = @id )

Go

Данная табличная функция выводит информацию о материалах отдельного цеха.

CREATE FUNCTION materials\_in\_ws

(

@id INT

)

RETURNS TABLE

AS

RETURN

(SELECT id\_mat AS 'id', material AS 'Материал', quantity AS 'Количество', measurement AS 'ед. изм.' FROM Materials WHERE id\_ws = @id )

Go

1. **Создание процедур для манипуляции данными**

Для манипуляции данными таблиц были созданы процедуры. Именно с их помощью осуществляется модификация данных через клиент-серверное приложение. Приведу некоторые процедуры модификации данных.

Процедура удаления работника. В случае, если адрес удаляемого работника больше никем не используется, данные об адресе удаляются.

create proc DEL\_WORKER(@id\_w int)

AS

BEGIN

declare @adr int;

SET @adr= (SELECT id\_a from Workers WHERE id\_w=@id\_w)

delete from Workers where id\_w=@id\_w;

declare @cnt int;

SET @cnt = (SELECT COUNT(\*) from Workers WHERE id\_a=@adr)

IF @cnt=0

BEGIN

Delete FROM Address WHERE id\_address=@adr;

ENDEND

Go

Процедура изменения данных о работнике. Вместе с этим изменяется информация об адресе работника.

create proc UPD\_WORKER(@id int, @LN varchar(20), @FN varchar(20),@MN varchar(20), @GS varchar(10), @id\_ws int, @id\_c int, @B int, @p bigint, @E varchar(25), @R varchar(30),@T varchar(20), @S varchar(30), @H tinyint, @C varchar(2), @F int)

as

begin

DECLARE @id\_a int;

SET @id\_a=(SELECT id\_a FROM Workers WHERE id\_w=@id);

DECLARE @cnt int;

SET @cnt=(SELECT COUNT(\*) FROM Workers WHERE id\_a=@id\_a )

IF (SELECT COUNT(\*) FROM Address WHERE Region=@R AND Town=@T AND Street=@S AND House=@H AND Corpus=@C AND Flat=@F)=0

BEGIN

IF @cnt>1

BEGIN

insert into Address(Region, Town, Street, House, Corpus, Flat) values(@R, @T, @S, @H, @C, @F);

SET @id\_a=(SELECT TOP(1) id\_address FROM Address ORDER BY id\_address DESC)

END

ELSE BEGIN

update Address set Region=@R,Town=@T, Street=@S, House=@H, Corpus=@C, Flat=@F WHERE id\_address=@id\_a

END

END

SET @id\_a = (SELECT id\_address FROM Address WHERE Region=@R AND Town=@T AND Street=@S AND House=@H AND Corpus=@C AND Flat=@F)

update Workers

set LastName=@LN, FirstName=@FN, MiddleName=@MN,

Graphic\_smen=@GS, id\_ws=@id\_ws, id\_c=@id\_c,

Brigada=@B, id\_a=@id\_a, Phone=@p, Email=@E

WHERE id\_w=@id

End

Go

Процедура вставки нового работника и данных о его адресе

create proc INS\_WORKER @LN varchar(20), @FN varchar(20),@MN varchar(20), @GS varchar(10), @id\_ws int, @id\_c int, @B int, @p bigint, @E varchar(25), @R varchar(30),@T varchar(20), @S varchar(30), @H tinyint, @C varchar(2), @F int) as

Begin

DECLARE @pattern nvarchar(100)='([\w\.\-]+)\@([\w]+\.)([\w]+)';

DECLARE @flag int;

DECLARE @id\_a int;

IF (SELECT COUNT(\*) FROM Address WHERE Region=@R AND Town=@T AND Street=@S AND House=@H AND Corpus=@C AND Flat=@F)=0

BEGIN

INSERT INTO Address(Region, Town, Street, House, Corpus, Flat) values(@R, @T, @S, @H, @C, @F);

SET @id\_a=(SELECT TOP(1) id\_address FROM Address ORDER BY id\_address DESC);

END

ELSE BEGIN

SET @id\_a=(SELECT id\_address FROM Address WHERE Region=@R AND Town=@T AND Street=@S AND House=@H AND Corpus=@C AND Flat=@F)

END

IF (dbo.Regex\_Check(@pattern,@E)=1)

begin

insert into Workers(LastName, FirstName, MiddleName, Graphic\_smen, id\_ws, id\_c, Brigada, id\_a, Phone, Email) values

(@LN, @FN, @MN, @GS, @id\_ws, @id\_c, @B, @id\_a, @p, @E)

End

Else begin

insert into Workers(LastName, FirstName, MiddleName, Graphic\_smen, id\_ws, id\_c, Brigada, id\_a, Phone, Email) values

(@LN, @FN, @MN, @GS, @id\_ws, @id\_c, @B, @id\_a, @p, 'no')

End

End

1. **Пользователи и представления для них**

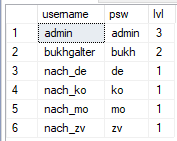
Была создана таблица **Avtor**, содержащая информацию о пользователях, их паролях и уровнях допуска. Уровень допуска определяет, в какую из форм клиент-серверного приложения должен быть направлен пользователь.

use Tip

go

create table Avtor(username varchar(20) primary key, psw varchar(20), lvl int);

go



Было создано несколько представлений для разных пользователей. Приведу некоторые из них.

Информация о работниках, оборудовании и материалах монтажного цеха. Использованы функции из п.2.

CREATE VIEW FOR\_NACH\_MO\_WORKERS AS

SELECT \* FROM workers\_in\_ws(4)

go

CREATE VIEW FOR\_NACH\_MO\_DEV AS SELECT \* from devices\_in\_ws(1);

go

CREATE VIEW FOR\_NACH\_MO\_MAT AS SELECT \* from materials\_in\_ws(1);

Go

Представление, выводящее информацию о работниках, их заработной плате и их личные данные.

CREATE VIEW ZARPLATA AS

SELECT t1.id\_w AS 'id', t1.LastName+' '+t1.FirstName+' ' +t1.MiddleName AS 'ФИО', t1.Graphic\_smen AS 'График смен', t1.Brigada AS '№ бригады',

t1.Phone AS 'Телефон',t1.Email AS 'E-mail', t2.Doljnost AS 'Должность',

(SELECT WS\_Name FROM Workshops WHERE Workshops.id\_ws= t1.id\_ws) AS 'Цех', t2.Salary AS 'Зарплата', t3.Region AS 'Регион',

t3.Town AS 'Город', t3.Street AS 'Улица', t3.House AS 'Дом', t3.Corpus AS 'Корпус', t3.Flat AS 'Квартира' from

Workers t1 join Cadr t2 on t1.id\_c=t2.id\_cadr join Address t3 on id\_a=id\_address;

Go

Представление, выводящее информацию о проектах и используемых ими услугах.

CREATE VIEW TBL AS SELECT (SELECT Pr\_Name from Projects WHERE id\_pr=PU.id\_proj) AS 'Проект',

(SELECT Usl\_Name from Uslugi WHERE id\_usl=PU.id\_usl) AS 'Услуга',

(SELECT Min\_Cost FROM Uslugi WHERE id\_usl=PU.id\_usl) AS 'Стоимость' FROM PU;

1. **Код CLR-функции (С#, использованы регулярные выражения)**

using System;

using System.Data;

using System;

using System.Data;

using System.Data.SqlClient;

using System.Data.SqlTypes;

using Microsoft.SqlServer.Server;

using System.Text.RegularExpressions;

public partial class UserDefinedFunctions

{

[Microsoft.SqlServer.Server.SqlFunction]

public static SqlBoolean SqlFunction1(SqlString format, SqlString str)

{

Regex pattern = new Regex(format.ToString());

return (SqlBoolean)pattern.IsMatch(str.ToString());

}

}

Была создана функция **Regex\_Check**:

CREATE FUNCTION Regex\_Check(@pattern nvarchar(100), @str nvarchar(100))

RETURNS bit

AS EXTERNAL NAME clr.UserDefinedFunctions.SqlFunction1;

GO

1. **Триггер.**

В случае удаления единственного работника в своей профессии транзакция отклоняется.

CREATE TRIGGER Trigger\_on\_DELETE ON Workers

AFTER DELETE

AS

DECLARE @number int;

DECLARE @id\_c int;

SET @id\_c =(SELECT id\_c FROM DELETED);

SET @number=(SELECT COUNT(\*) FROM Workers WHERE id\_c=@id\_c);

IF (@number<1)

BEGIN

RAISERROR ('Невозможно удалить последнего специалиста в какой-либо области', 14, 1);

ROLLBACK TRANSACTION;

END;

GO

1. **Создание клиент-серверного приложения**

Для создания клиент-серверного приложения был выбран язык программирования **С#**, как один из наиболее удобных инструментов ООП-разработки.

Были созданы четыре формы: форма авторизации, форма начальника цеха, форма бухгалтера и форма администратора.

При попытке авторизации приложение направляет запрос в базу данных, после получения которого либо перенаправляет на нужную форму, либо отказывает в доступе.

public int Valid()

{

bool flag;

int lvl = 0;

string search = "SELECT \* FROM dbo.Avtor WHERE username= '" + user.Text + "' AND psw='"+psw.Text+"'";

using (connection.CreateCommand())

{

connection.Open();

SqlCommand cmd = new SqlCommand(search, connection);

SqlDataReader dataReader = cmd.ExecuteReader();

using (dataReader)

{

flag = dataReader.Read();

if (flag)

{

lvl = Convert.ToInt32(dataReader["lvl"].ToString());

}

else

MessageBox.Show("Ошибка авторизации! \n Попробуйте еще!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

connection.Close();

}

return lvl;

}

С использованием объекта управления **DataGridView** были отображены данные представлений. По событию нажатия на соответствующую кнопку происходит вызов процедур модификации данных, созданных в п. 3.

Для подключения к базе данных было создано соединение:

SqlConnection connection = new SqlConnection(@"Data Source=DESKTOP-172R4OA;Initial Catalog=Tip;Integrated Security=True");

Для отправки запросов была создана функция **Command**:

public DataTable Command(string str)

{

try

{

using (connection.CreateCommand())

{

SqlDataAdapter sqlDA = new SqlDataAdapter(str, connection);

DataTable dtbl = new DataTable();

sqlDA.Fill(dtbl);

return dtbl;

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Произошла ошибка при попытке модификации данных!\n" + "Обратитесь к специалисту, администрирующему систему!");

return null;

}

}

Рассмотрим модификацию данных в клиент-серверном приложении на примере таблицы **Devises**.

Добавление:

using (connection.CreateCommand())

{

string search= "exec dbo.INS\_DEV "+Convert.ToString(comboBox1.SelectedIndex+1)+", '"+textBox1.Text + "', " + textBox2.Text;

Command(search);

dataGridView2.DataSource = Command("select \* from dbo.DEV");

}

Изменение:

int row\_number = dataGridView2.CurrentRow.Index;

int cell\_number = dataGridView2.CurrentCell.ColumnIndex;

string id = dataGridView2.Rows[row\_number].Cells[0].Value.ToString();

string ws = dataGridView2.Rows[row\_number].Cells[1].Value.ToString();

switch (ws)

{

case "Монтажный":

ws ="1";

break;

case "Звуковой":

ws = "2";

break;

case "Костюмерный":

ws = "3";

break;

case "Декорационный":

ws = "4";

break;

case "Складские помещения":

ws = "5";

break;

case "Павильоны":

ws = "6";

break;

case "Бухгалтерия":

ws = "7";

break;

case "Дирекция":

ws = "8";

break;

}

string dev = dataGridView2.Rows[row\_number].Cells[2].Value.ToString();

string quant = dataGridView2.Rows[row\_number].Cells[3].Value.ToString();

string search = "exec dbo.UPD\_DEV " +id + ", " +ws+ ", '" + dev+ "', " + quant;

Command(search);

dataGridView2.DataSource = Command("select \* from dbo.DEV");

Удаление:

int row\_number = dataGridView2.CurrentRow.Index;

int cell\_number = dataGridView2.CurrentCell.ColumnIndex;

string id = dataGridView2.Rows[row\_number].Cells[0].Value.ToString();

string search = "exec dbo.DEL\_DEV " + id;

Command(search);

dataGridView2.DataSource = Command("select \* from dbo.DEV");