Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное

образовательное учреждение высшего образования

**ИВАНОВСКИЙ ГОСУДАРСТВЕННЫЙ ХИМИКО-ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ**

**Факультет техники, управления и цифровой инфраструктуры**

**Кафедра информационных технологий и цифровой экономики**

**КУРСОВОЙ ПРОЕКТ**

**ПО ДИСЦИПЛИНЕ**

**УПРАВЛЕНИЕ ДАННЫМИ**

**Тема:** Разработка базы данных MS SQL

и приложения средствами Visual Studio.

Вариант №12

**Выполнил студент: Долгова Анастасия Игоревна курс/гр. 3/185**

ФИО

**Проверил: Куваев Александр Евгеньевич**

Иваново 2020

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное

образовательное учреждение высшего образования

**ИВАНОВСКИЙ ГОСУДАРСТВЕННЫЙ ХИМИКО-ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ**

**Факультет** ТУиЦИ

**Кафедра** ИТиЦЭ

**Направление** 09.03.02 Информационные системы и технологии

**Профиль** Анализ данных и цифровые финансовые технологии

**Дисциплина** Управление данными

**ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ**

**Студенту** Долговой Анастасии Игоревне курс/гр. **3/185**

(фамилия, имя, отчество - полностью)

**Тема:** Разработка базы данных MS SQL и приложения средствами Visual Studio. Вариант №12

Исходные данные: *законодательные и подзаконные нормативные акты; энциклопедическая и справочная литература; статистические и аналитические материалы; монографии; данные профессиональных периодических изданий; интернет-ресурсы; самостоятельно собранные первичные материалы.*

Содержание работы:

Введение: *цель и задачи работы; предмет и объект исследования; обоснование структуры работы.*

Теоретическая часть должна содержать: *системный анализ предметной области, инфологическую модель БД, список запросов, которые должны быть реализованы в БД, список требований к приложению*

В практической части необходимо: *сгенерировать БД, реализовать SQL-запросы, разработать программное приложение к БД.*

Заключение*: основные выводы по проекту, обобщение основных предложений и рекомендаций.*

Руководитель работы:  *Куваев А.Е.*

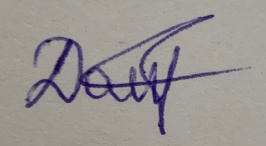
**Календарный план-график**

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Наименование этапов выполнения курсовой работы | Срок выполнения этапов | Примечание |
| 1 | Исследование предметной области и составление логической модели БД | 17.02.20 – 17.03.20 |  |
| 2 | Создание физической модели БД | 18.03.20 – 31.03.20 |  |
| 3 | Генерация БД средствами MS SQL и реализация SQL-запросов | 01.04.20 – 14.04.20 |  |
| 4 | Разработка приложения средствами Visual Studio | 15.04.20 – 12.05.20 |  |
| 5 | Оформление пояснительной записки | 13.05.20 – 17.05.20 |  |
| 6 | Защита курсового проекта | 18.05.20 - 23.05.20 |  |

Дата выдачи задания 17.02.2020

Дата представления законченной работы 23.05.2020

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Куваев А.Е.

 (Подпись /Ф.И.О.)

Студент Долгова А.И.

(Подпись /Ф.И.О.)

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc42722730)

[1 СИСТЕМНЫЙ АНАЛИЗ 7](#_Toc42722731)

[1.1 Описание предметной области 7](#_Toc42722732)

[1.2 Логическая модель «Как есть» 8](#_Toc42722733)

[1.3 Логическая модель «Как должно быть» 8](#_Toc42722734)

[1.4 Список запросов, которые должны быть реализованы в БД 9](#_Toc42722735)

[2 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ 11](#_Toc42722736)

[3. ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ 25](#_Toc42722737)

[3.1 Таблица «Виды работ» 27](#_Toc42722738)

[3.2 Таблица «Работники» 28](#_Toc42722739)

[3.3 Таблица «Итоговые выплаты» 29](#_Toc42722740)

[3.4 Таблица «Работы» 29](#_Toc42722741)

[3.5 Таблица «Работники и работы» 30](#_Toc42722742)

[3.6 Поиск записей 30](#_Toc42722743)

[3.7 Экспорт таблицы Excel 33](#_Toc42722744)

[ЗАКЛЮЧЕНИЕ 34](#_Toc42722745)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 35](#_Toc42722746)

[ПРИЛОЖЕНИЕ А. СКРИПТ БАЗЫ ДАННЫХ 36](#_Toc42722747)

[ПРИЛОЖЕНИЕ Б. ПЕРЕЧЕНЬ ФАЙЛОВ 57](#_Toc42722748)

# ВВЕДЕНИЕ

Под базой данных понимается объективная форма представления и организации совокупности данных, систематизированная таким способом, чтобы эти данные могли быть найдены и обработаны с помощью ЭВМ.

Система управления базой данных - это совокупность языковых и программных средств, предназначенных для создания, ведения и коллективного использования БД.

Проектирование БД представляет собой сложный трудоемкий процесс отображения предметной области во внутреннюю модель данных. В процессе проектирования разрабатывается модели разных уровней архитектуры БД, проверяется возможность отображения объектов одной модели объектами другой модели.

При проектировании базы данных решаются две основных проблемы:

* Каким образом отобразить объекты предметной области в абстрактные объекты модели данных, чтобы это отображение не противоречило семантике предметной области, и было по возможности лучшим (эффективным, удобным и т.д.)? Часто эту проблему называют проблемой логического проектирования баз данных.
* Как обеспечить эффективность выполнения запросов к базе данных, т.е. каким образом, имея в виду особенности конкретной СУБД, расположить данные во внешней памяти, создание каких дополнительных структур (например, индексов) потребовать и т.д.? Эту проблему называют проблемой физического проектирования баз данных.

Данная пояснительная записка описывает процесс создание базы данных, организации логики базы данных, создания приложения, работающего с этой базой данных.

В первый этап разработки входит ознакомление с исходным заданием, построение инфологических моделей «как есть» и «как должно быть». Также в этот этап входит составление модели данных в нотации IDEF1x, составления вопросов и требований того, что должна делать система.

Во второй этап разработки входит описание процесса разработки физической модели базы данных SQL, создание представлений, хранимых процедур, наполнения базы данных и выполнения запросов.

В третий этап разработки входит описание работы разработанного приложения для работы с базой данных, - работа с таблицами, добавление, удаление и редактирование записей таблиц, поиск записей в базе данных и отчетность.

ОСНОВНАЯ ЧАСТЬ

# 1 СИСТЕМНЫЙ АНАЛИЗ

ЗАДАНИЕ «РАСПРЕДЕЛЕНИЕ ДОПОЛНИТЕЛЬНЫХ ОБЯЗАННОСТЕЙ»

## 1.1 Описание предметной области

Вы работаете в коммерческой компании и занимаетесь распределением дополнительных обязанностей. Вашей задачей является отслеживание хода их выполнения. Компания имеет штат сотрудников, каждый из которых получает определенный оклад. Время от времени возникает потребность в выполнении некоторой дополнительной работы, не входящей в круг основных должностных обязанностей сотрудников. Для наведения порядка в этой сфере деятельности вы проклассифицировали все виды дополнительных работ, определив сумму оплаты по факту их выполнения. При возникновении дополнительной работы определенного вида вы назначаете ответственного, фиксируя дату начала. По факту окончания вы фиксируете дату и выплачиваете дополнительную сумму к зарплате с учетом вашей классификации.

Таблицы:

Сотрудники (Код работника, Фамилия, Имя, Отчество, Оклад, Зарплата).

Виды работ (Код вида, Описание, Оплата за работу).

Работы (Код работы, Код вида, Код работника, Дата начала, Дата окончания ).

Развитие постановки задачи:

Теперь ситуация изменилась. Несложный анализ показал, что нужно не просто подразделять станки по видам, а иметь информацию о том, сколько раз ремонтировался тот или иной конкретный станок.

Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы.

В этапе системного анализа проводится осмысление поставленной задачи. Далее необходимо составить инфологические модели

## 1.2 Логическая модель «Как есть»

Сотрудники (Код работника, Фамилия, Имя, Отчество, Оклад, Зарплата).

Виды работ (Код вида, Описание, Оплата за работу).

Работы (Код работы, Код вида, Код работника, Дата начала, Дата окончания ).

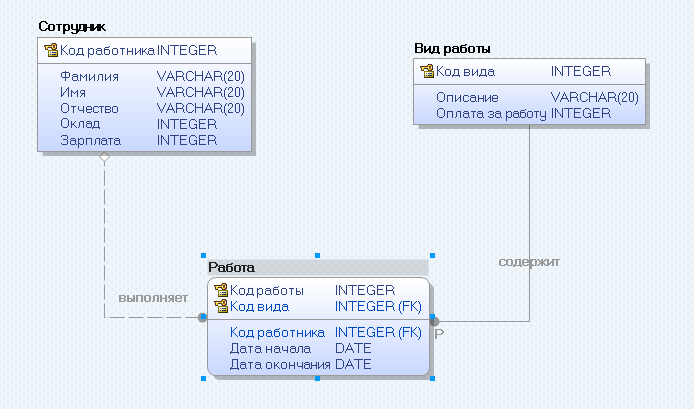


Рисунок 1.1 – Построение модели «как есть» в нотации IDEF1x

Исходя из анализа развития постановки задачи можно сделать вывод, что:

* Появляется новая сущность «Зарплата»;

## 1.3 Логическая модель «Как должно быть»

Теперь ситуация изменилась. Выяснилось, что некоторые из дополнительных работ являются достаточно трудоемкими и, в то же время, срочными, что требует привлечения к их выполнению нескольких сотрудников. Также оказалось, что длительность работ в каждом конкретном случае различна. Соответственно, нужно заранее планировать длительность работы и количество сотрудников, занятых ее выполнением.

Построение модели в нотации IDEF1x выполнено в программе Erwin Data Modeler 2018 R1.

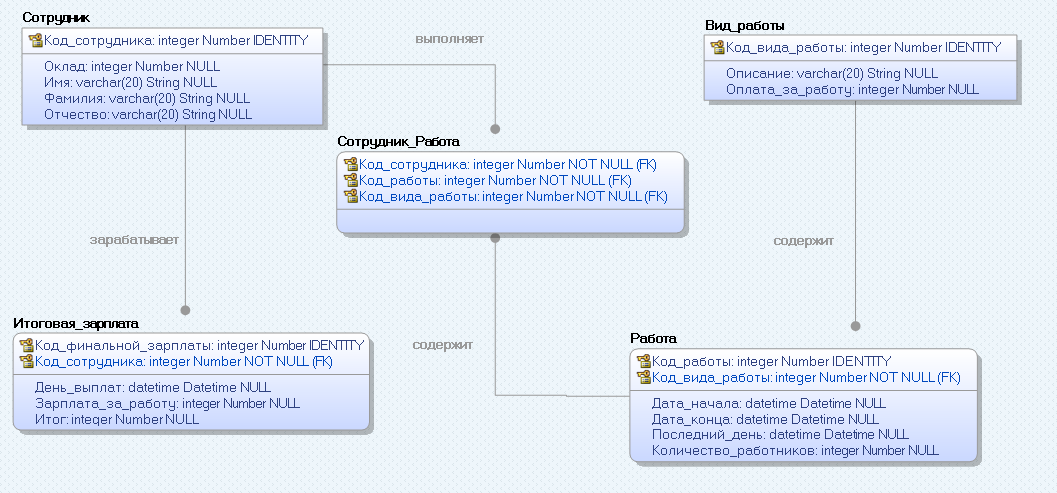


Рисунок 1.2 – Построение модели в нотации IDEF1x

## 1.4 Список запросов, которые должны быть реализованы в БД

1. Вывести список всех сотрудников компании и их зарплат.
2. Вывести список выполненных работ.
3. Вывести список самых срочных работ.
4. Вывести список уже начатых, но еще не выполненных работ.
5. Сколько в компании сотрудников с одинаковыми именами.
6. Сколько в базе данных работ, которые были выполнены в тот же день, когда начались.
7. Вывести список всех видов работ, отсортировать по оплате за работу.
8. Вывести список сотрудников, занятых одной какой-либо работой.
9. Вывести список самых трудных (максимальное количество необходимых сотрудников) работ.
10. Вывести список работ, которые выполнялись дольше всего.
11. Вывести список работ, выполненных в 2019 году.
12. Вывести список сотрудников, у которых оплата за доп. работу оказалась больше оклада.
13. Вывести список работ, которые были сданы позже положенного срока.
14. Вывести список работ, выполненных одним сотрудником.
15. Вывести список однотипных работ.
16. Вывести список сотрудников, которые не выполнили свою работу вовремя.

1.5 Набор требований, которым должно отвечать разрабатываемое приложение

* Обеспечить ведение списка работников.
* Обеспечить ведение списка работ.
* Обеспечить ведение списка видов работ.
* Обеспечить вывод выполненных, невыполненных и просроченных работ.
* Обеспечить возможность начислить зарплату за определенный период всем сотрудникам.
* Обеспечить возможность выгрузки данных в электронную таблицу Excel.

# 2 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

Главная задача данного этапа заключается в преобразовании инфологической модели в физическую модель базы данных, преобразовании ее в схему базы данных для MS SQL Server 2008, используя Transact-SQL, проверки корректности сделанных преобразований и соответствию исходных требований, а также проектированию приложения к базам данных.

Физическая модель представлена на рисунке 2.1.

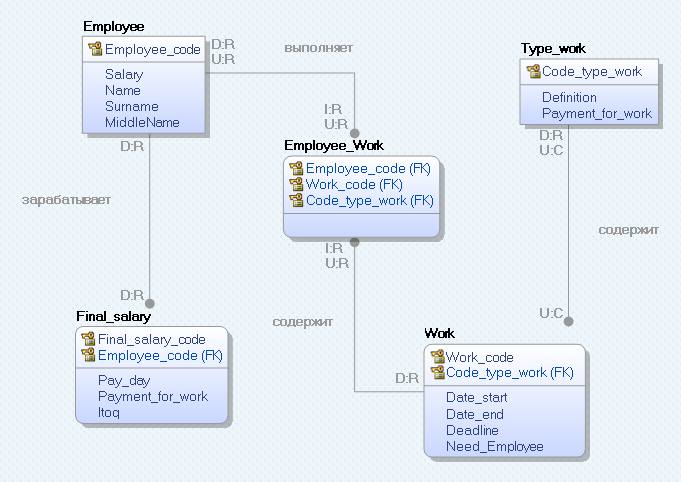


Рисунок 2.1 – Физическая модель

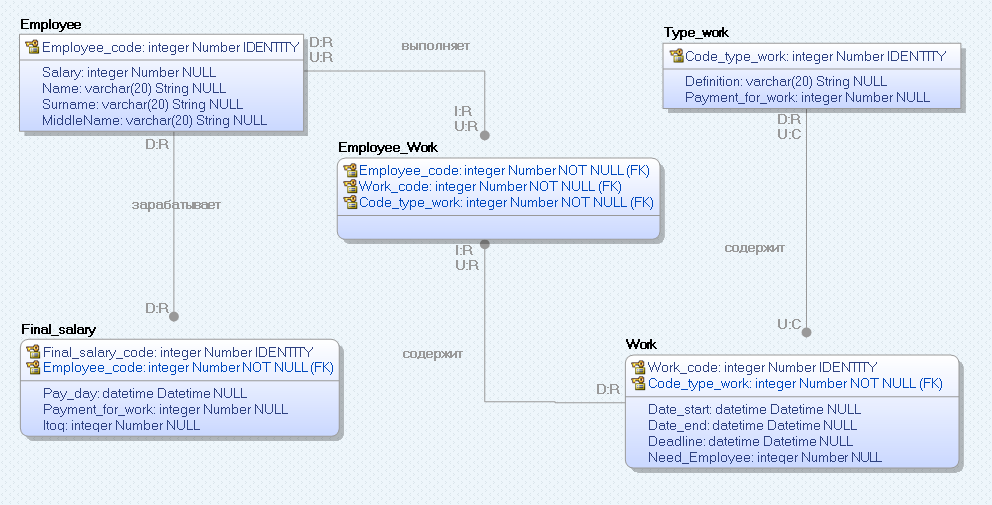


Рисунок 2.2 – Физическая модель со всеми типами данных

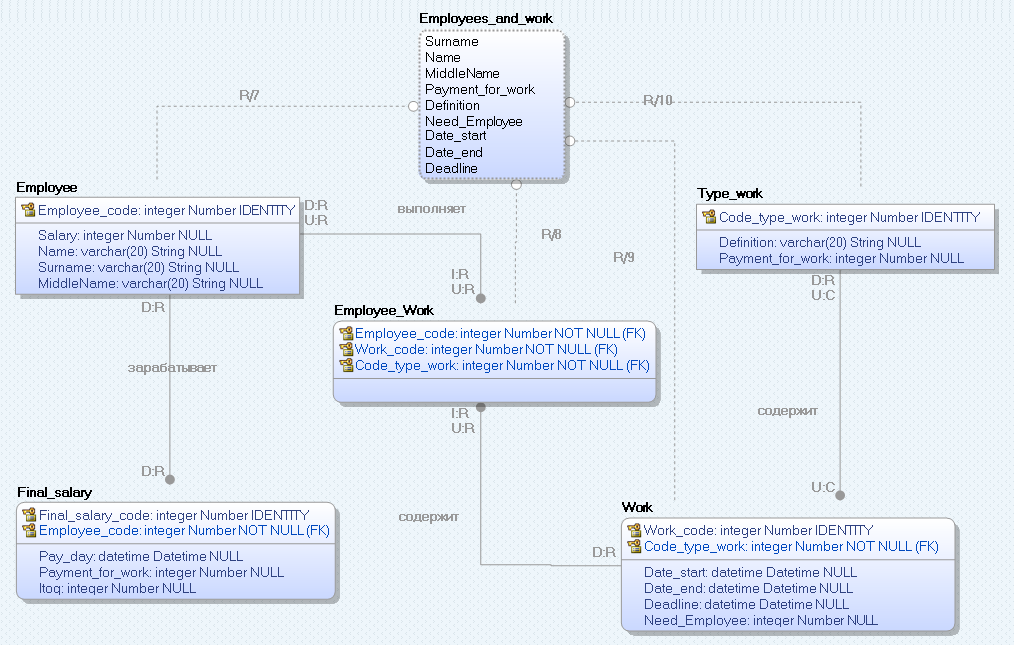


Рисунок 2.3 – Представление Employees\_and\_work

После создания физической модели переносим ее в базу данных SQL с помощью Forward Engineering и заполним ее.

Далее представлены таблицы базы данных и запросы по заполнению таблиц.

Запрос SQL на добавление записи в таблицу Employee

INSERT INTO Employee(Salary, Surname, Name, MiddleName) values('оплата', 'фамилия', 'имя', 'отчество')

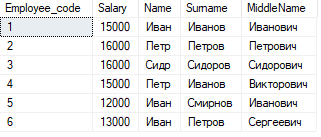


Рисунок 2.4 – Таблица Employee

Запрос SQL на добавление записи в таблицу Type\_work:

INSERT INTO Type\_work(Definition, Payment\_for\_work) values('описание работы', 'оплата')

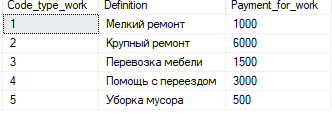


Рисунок 2.5 – Таблица Type\_work

Таблица Final\_salary заполнилась автоматически:

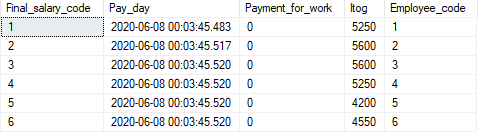


Рисунок 2.6 – Таблица Final\_salary

Запрос SQL на добавление записи в таблицу work

INSERT INTO Work(Date\_start,Date\_end, Deadline, Need\_Employee, Code\_type\_work)

values('дата начала', 'дата окончания', 'крайний срок', 'количество людей', 'код работы')

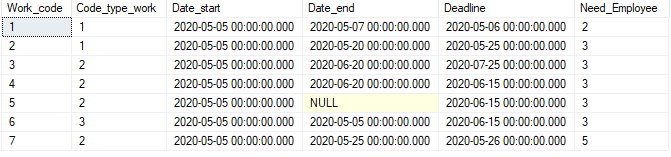


Рисунок 2.7 – Таблица Work

Запрос SQL на добавление записи в таблицу Employee\_work

INSERT INTO Employee\_Work(Work\_code, Employee\_code, Code\_type\_work) values('код работы','код работника','код типа работы')

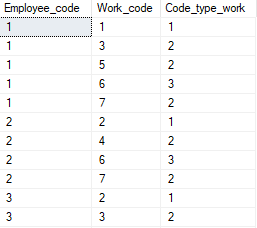


Рисунок 2.8 – Таблица Employee\_Work

Далее проверяем работу представления.

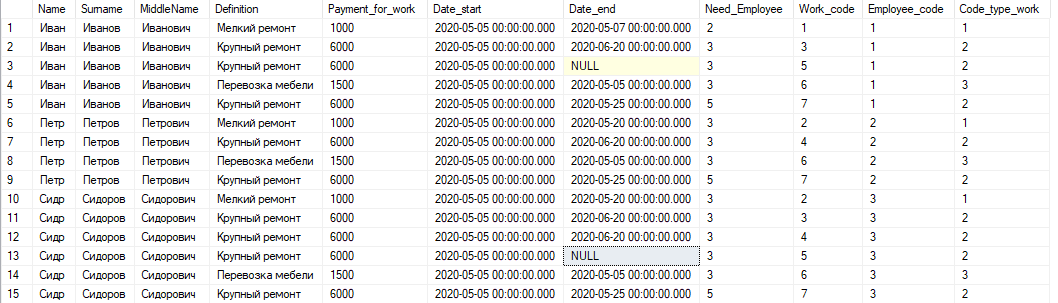


Рисунок 2.9 – Вывод представления Employee\_and\_Work

Создадим ограничение для таблицы Final\_salary. Нам нужно чтобы нельзя было добавить запись с зарплатой за работу не меньше нуля и итоговой зарплаты не меньше нуля.

Создание ограничения:

ALTER TABLE Final\_salary

ADD CONSTRAINT check\_not\_negative CHECK (Payment\_for\_work >= 0 AND Itog >= 0)

Проверка ограничения:

INSERT INTO Deal (DateOfDeal,Commission,PositionID,SeekerID)

VALUES ('01.01.2023',1,1,1)

Далее создаем хранимую процедуру SetSetItogSalary, которая создает нового работодателя и вакансию от него, запрос на создание процедуры представлен ниже.

Create procedure SetSetItogSalary (@date\_start DateTime, @date\_end DateTime)

as

begin

declare @a int = 1;

While @a <= (select max(Employee\_code) from employee)

begin

begin

insert into Final\_salary (Employee\_code,Pay\_day,Itog)

values (@a, @date\_end, (select salary from Employee where Employee\_code = @a))

end;

set @a = @a + 1

end;

Declare @b int = 1

while @b <= (select MAX(Final\_salary\_code) from Final\_salary)

begin

begin

declare @c int = (select sum(eaw.Payment\_for\_work)

from Employees\_and\_work eaw join Final\_salary fs on eaw.Employee\_code = fs.Employee\_code

where fs.Final\_salary\_code = @b and eaw.Date\_end is not null and (Eaw.Date\_start >= @date\_start) and (Eaw.Date\_end <= @date\_end))

if @c is null

set @c = 0;

end;

begin

update Final\_salary

set Payment\_for\_work = @c

where Pay\_day = @date\_end and Final\_salary.Final\_salary\_code = @b

update Final\_salary

set Itog = ((select Employee.Salary from employee, Final\_salary where employee.Employee\_code = Final\_salary.Employee\_code and Final\_salary.Final\_salary\_code = @b) + Final\_salary.Payment\_for\_work)

where Pay\_day = @date\_end and Final\_salary.Final\_salary\_code = @b

end;

set @b = @b + 1

end;

end;

Далее создадим триггер sign\_on\_bonus. Был написан триггер для нашей Базы данных для первоначального заполнения таблицы Final\_salary(в ней формируется итоговая зарплата сотрудника за месяц). В нашей организации при приеме человека на работу, ему выплачиваются подъемные в размере 35% от его зарплаты(Salary).

Запрос на создание триггера:

Create trigger sign\_on\_bonus

on Employee after insert

as

Insert into Final\_salary

(

Payment\_for\_work,

Pay\_day,

Final\_salary.Itog,

Employee\_code

)

values

(

0,

GETDATE(),

(select (Salary\*0.35) from inserted),

(select Employee\_code from inserted)

)

После проверки триггера выполним запросы, которые были даны ранее (1 этап: Системный анализ):

1. Вывести список всех сотрудников компании и их зарплат.

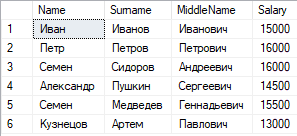
SELECT Name, Surname, Salary FROM Employee

Рисунок 2.10 Результаты выполнения запроса.

1. Вывести список выполненных работ.

SELECT Date\_start, Date\_end, Deadline, tw.Definition

FROM Work w JOIN

Type\_work tw on tw.Code\_type\_work = w.Code\_type\_work

WHERE Date\_end is not null

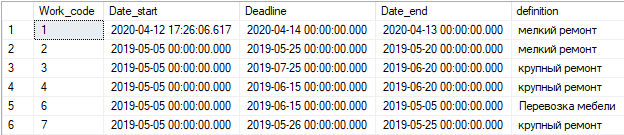


Рисунок 2.11 Результаты выполнения запроса.

1. Вывести список самых срочных работ.

SELECT Date\_start, Date\_end, Deadline, tw.Definition

FROM Work w Join

Type\_work tw on tw.Code\_type\_work = w.Code\_type\_work

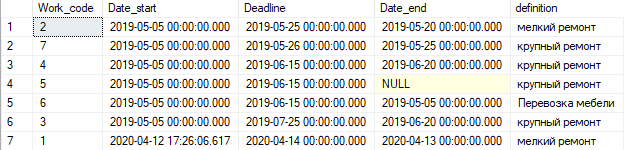
order by Deadline

Рисунок 2.12 Результаты выполнения запроса.

1. Вывести список уже начатых, но еще не выполненных работ.

SELECT tw.Definition, Date\_start, Date\_end, Deadline

FROM Work w Join

Type\_work tw ON tw.Code\_type\_work = w.Code\_type\_work

WHERE Date\_start is not null and Date\_end is null



Рисунок 2.13 Результаты выполнения запроса.

1. Сколько в компании сотрудников с одинаковыми именами.

SELECT Count(name) AS Count, Name

FROM Employee

GROUP BY Name

HAVING count(\*)>1;

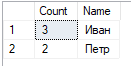


Рисунок 2.14 Результаты выполнения запроса.

1. Сколько в базе данных работ, которые были выполнены в тот же день, когда начались.

SELECT COUNT(\*) AS Count

FROM Work

WHERE Date\_start = Date\_end



Рисунок 2.15 Результаты выполнения запроса.

1. Вывести список всех видов работ, отсортировать по оплате за работу.

SELECT \*

FROM Type\_work

ORDER BY (Payment\_for\_work)

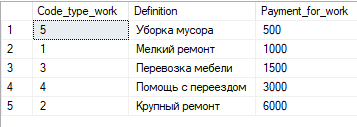


Рисунок 2.16 Результаты выполнения запроса.

1. Вывести список сотрудников, занятых больше чем одной работой.

SELECT distinct Employee\_code,Name

FROM Employees\_and\_work

GROUP BY Employee\_code,Name

Having Count(Work\_code)>1

ORDER BY Employee\_code

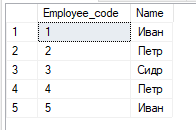


Рисунок 2.17 Результаты выполнения запроса.

1. Вывести список самых трудных (максимальное количество необходимых сотрудников) работ.

SELECT top 3 \*

FROM Work

ORDER BY (Need\_Employee) desc

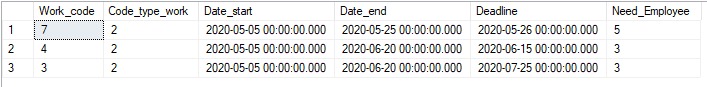


Рисунок 2.18 Результаты выполнения запроса.

1. Вывести список работ, которые выполнялись дольше всего.

SELECT top 3 tw.Definition, Date\_start, Date\_end, Deadline

FROM Work w Join

Type\_work tw ON tw.Code\_type\_work = w.Code\_type\_work

WHERE Date\_end is not null

ORDER BY DATEDIFF(second, Date\_start, Date\_end) desc

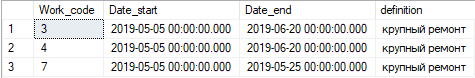


Рисунок 2.19 Результаты выполнения запроса.

1. Вывести список работ, выполненных в 2019 году.

SELECT tw.Definition, Date\_start, Date\_end, Deadline

FROM Work w Join

Type\_work tw ON tw.Code\_type\_work = w.Code\_type\_work

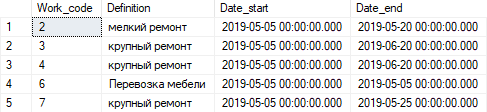
WHERE Date\_start>='01.01.2020' AND Date\_end<='31.12.2020'

Рисунок 2.20 Результаты выполнения запроса.

1. Вывести список сотрудников, у которых оплата за доп. работу оказалась больше оклада.

SELECT Final\_salary\_code, E.Employee\_code, E.Name, E.Surname, Payment\_for\_work, Salary

FROM Final\_salary fs, Employee E

WHERE fs.Employee\_code = E.Employee\_code and Payment\_for\_work > e.Salary



Рисунок 2.21 Результаты выполнения запроса.

Данный запрос выводит пустую таблицу, потому что никто из работников не получил за доп. работу больше, чем его собственный оклад. Следующий запрос выведет всех сотрудников с их окладами и зарплатой за доп. работу.

SELECT Final\_salary\_code, E.Employee\_code, E.Name, E.Surname, Payment\_for\_work, Salary

FROM Final\_salary fs, Employee E

WHERE fs.Employee\_code = E.Employee\_code

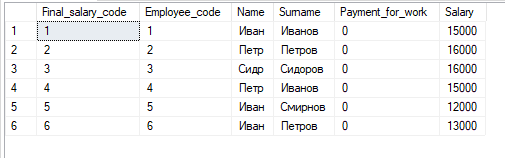


Рисунок 2.22 Результаты выполнения запроса.

1. Вывести список работ, которые были сданы позже положенного срока.

SELECT tw.Definition, Date\_start, Date\_end, Deadline

FROM Work w Join

Type\_work tw ON tw.Code\_type\_work = w.Code\_type\_work

WHERE Date\_end > Deadline or (GETDATE() > deadline and Date\_end is null)



Рисунок 2.23 Результаты выполнения запроса.

1. Вывести список работ, выполненных одним сотрудником.

SELECT distinct Work\_code,Count(Employee\_code) Number\_of\_Employee

FROM Employees\_and\_work

WHERE Date\_end is not null

GROUP BY Work\_code

HAVING Count(Employee\_code) = 1



Рисунок 2.24 Результаты выполнения запроса.

Данный запрос возвращает пустую таблицу, потому что все сотрудники работали в командах, это показывает следующий запрос:

SELECT distinct Work\_code,Count(Employee\_code) Number\_of\_Employee

FROM Employees\_and\_work

WHERE Date\_end is not null

GROUP BY Work\_code

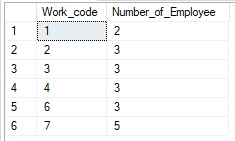


Рисунок 2.25 Результаты выполнения запроса.

1. Вывести список однотипных работ. Сгруппируем работы по типу и отсортируем. Получим список типов работ и список самих работ

SELECT Distinct TW.Code\_type\_work, Definition, W.Work\_code

FROM Type\_work TW, Work W

WHERE TW.Code\_type\_work = W.Code\_type\_work

GROUP BY Definition, TW.Code\_type\_work, W.Work\_code

ORDER BY TW.Code\_type\_work

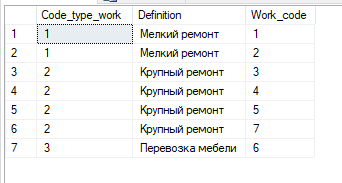


Рисунок 2.26 Результаты выполнения запроса.

1. Вывести список сотрудников, которые не выполнили свою работу вовремя.

SELECT Distinct EW.Employee\_code, EW.Name, Ew.Surname, EW.Work\_code, EW.Definition

FROM Work W join Employees\_and\_work EW on W.Work\_code = EW.Work\_code

WHERE W.Date\_end > W.Deadline or (GETDATE() > W.deadline and W.Date\_end is null)

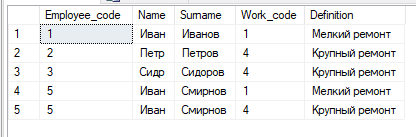


Рисунок 2.27 Результаты выполнения запроса.

# 3. ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ

На данном этапе мы проектируем приложение для быстрой работы с базой данных.

Нажимая кнопки в левой части приложения, можно переключаться между таблицами.

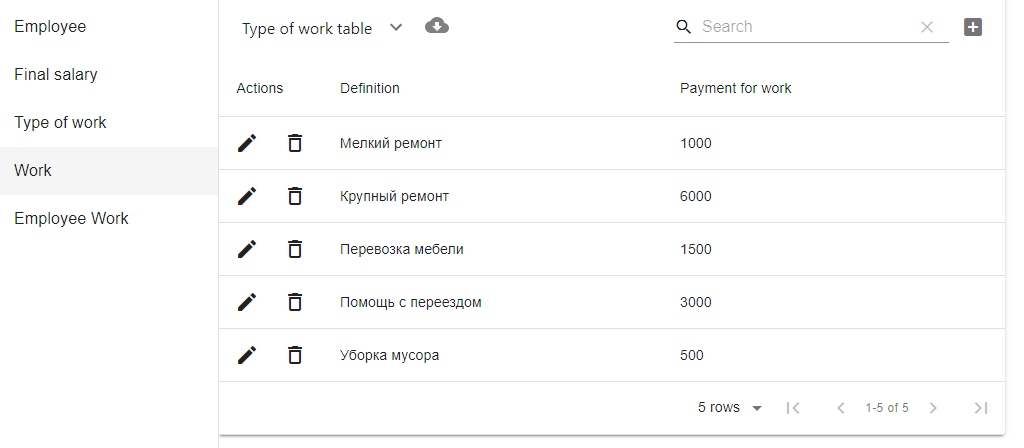


Рисунок 3.1 – Таблица «Типы работ»

Окно подключения к базе данных, которое нужно вызывать лишь при запуске приложения на новом компьютере, т. к. система автоматически подключается к базе, если запуск приложения на этом компьютере уже был. Система проверит текущие настройки подключения к базе данных и, если обнаружит проблемы, сообщит о них пользователю и предложит выполнить настройку подключения к базе данных, вызвав окно подключения.

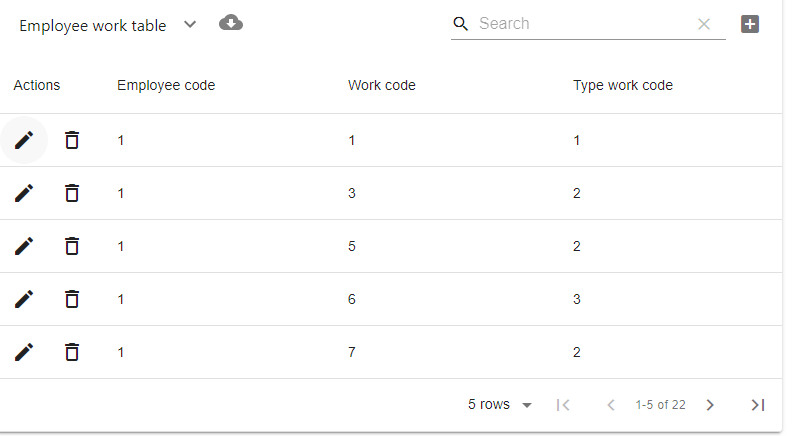


Рисунок 3.2 – Таблица «Работники и работы»

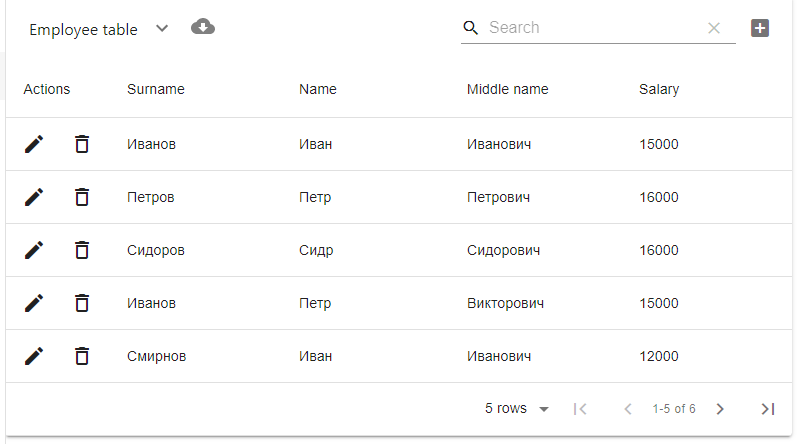


Рисунок 3.3 – Таблица «Работники»

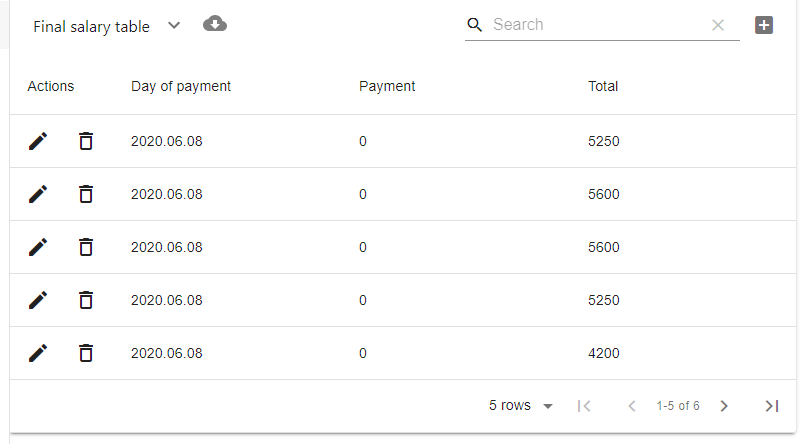


Рисунок 3.4 – Таблица «Итоговые выплаты»

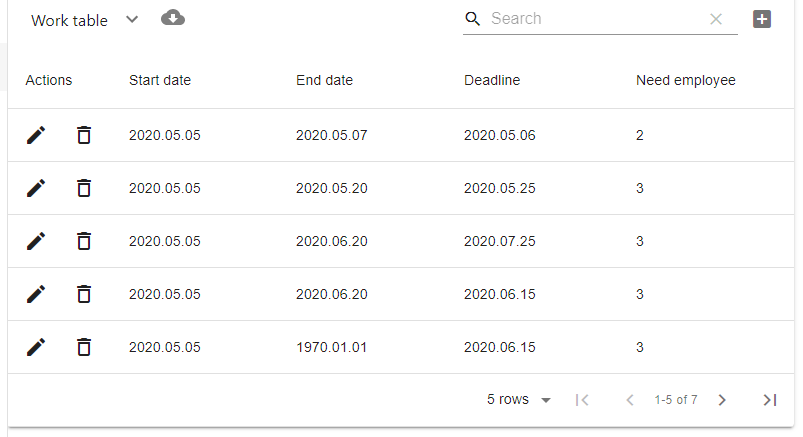


Рисунок 3.5 – Таблица «Работы»

Далее рассмотрим подробнее вид окон и работу с разными таблицами:

## 3.1 Таблица «Виды работ»

Кнопка «+» в правом верхнем углу добавляет новую запись в таблицу (Рисунок 3.6).

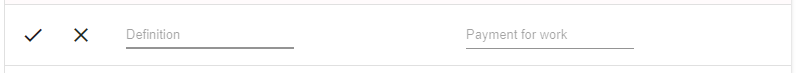


Рисунок 3.6 – Добавление типа работы

Кнопка в виде «Карандаша» загружает в форму «Типы работы» данные строки, чтобы ее изменить (Рисунок 3.7).

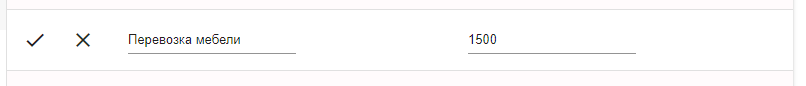


Рисунок 3.7 – Изменение типа работы

Кнопка в виде «Корзина» удаляет выбранную запись из таблицы (Рисунок 3.8).



Рисунок 3.8 – Удаление типа работы

## 3.2 Таблица «Работники»

Кнопка «+» в правом верхнем углу добавляет новую запись в таблицу (Рисунок 3.9).



Рисунок 3.9 – Добавление работника

Кнопка в виде «Карандаша» загружает в форму «Работники» данные строки, чтобы ее изменить (Рисунок 3.10).



Рисунок 3.10 – Изменение работника

Кнопка в виде «Корзина» удаляет выбранную запись из таблицы (Рисунок 3.11).



Рисунок 3.11 – Удаление работника

## 3.3 Таблица «Итоговые выплаты»

Кнопка «+» в правом верхнем углу добавляет новую запись в таблицу (Рисунок 3.12).

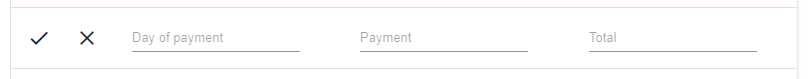


Рисунок 3.12 – Добавление итоговых выплат

Кнопка в виде «Карандаша» загружает в форму «Итоговых выплат» данные строки, чтобы ее изменить (Рисунок 3.13).



Рисунок 3.13 – Изменение итоговых выплат

Кнопка в виде «Корзины» удаляет выбранную запись из таблицы (Рисунок 3.14).



Рисунок 3.14 – Удаление итоговых выплат

## 3.4 Таблица «Работы»

Кнопка «+» в правом верхнем углу добавляет новую запись в таблицу (Рисунок 3.15).



Рисунок 3.15 – Добавление работы

Кнопка в виде «Карандаша» загружает в форму «Работы» данные строки, чтобы ее изменить (Рисунок 3.16).

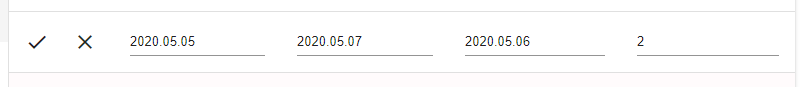


Рисунок 3.16 – Изменение работы

Кнопка в виде «Корзины» удаляет выбранную запись из таблицы (Рисунок 3.17).



Рисунок 3.17 – Удаление работы

## 3.5 Таблица «Работники и работы»

Кнопка «+» в правом верхнем углу добавляет новую запись в таблицу (Рисунок 3.18).

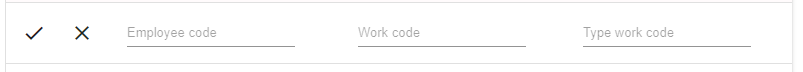


Рисунок 3.18 – Добавление

Кнопка в виде «Карандаша» загружает в форму «Работники и работы» данные строки, чтобы ее изменить (Рисунок 3.19).



Рисунок 3.19 – Изменение

Кнопка в виде «Корзины» удаляет выбранную запись из таблицы (Рисунок 3.20).



Рисунок 3.20 – Удаление

## 3.6 Поиск записей

Для поиска записей в базе данных, необходимо выбрать нужную таблицу, затем при заполнении строки поиска записи, будут отображаться результаты поиска. Строка поиска представлено на рисунке 3.21.



Рисунок 3.21 – Окно поиска

Для поиска записей в базе данных, также можно использовать фильтрацию, для этого необходимо сначала выбрать нужную таблицу, затем нажать на кнопку в виде “стрелки вниз”. Окна фильтров для разных таблиц представлено на рисунках.

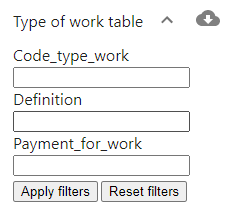


Рисунок 3.22 – Фильтры для типов работ

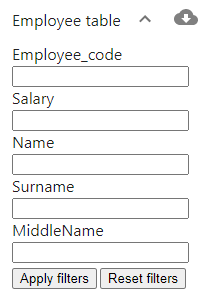


Рисунок 3.23 – Фильтры для работников

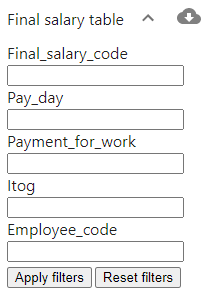


Рисунок 3.24 – Фильтры для итоговых выплат

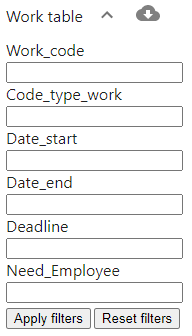


Рисунок 3.25 – Окно фильтров работ

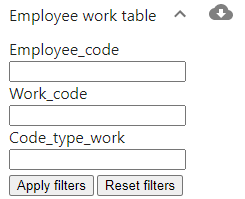


Рисунок 3.26 – Окно фильтров для таблицы работников и работ.

Рассмотрим пример использования фильтра. Найдем всех, у кого зарплата выше 15000 у.е. и имя ‘Иван’ (Рисунок 3.27 и Рисунок 3.28).

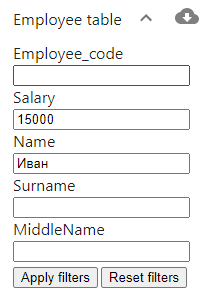


Рисунок 3.27 – Поиск работников с именем Иван и зарплатой выше 15000

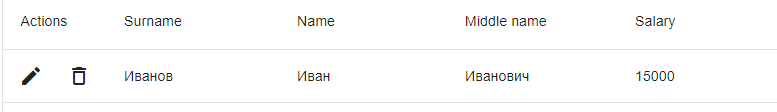


Рисунок 3.28 – Результат фильтрации

## 3.7 Экспорт таблицы Excel

Для выгрузки таблицы Excel необходимо сначала выбрать таблицу для сохранения, затем нажать на кнопку в виде ‘облака’, после этого сразу загрузиться, и откроется файл excel с нужной таблицей (Рисунок 3.30).

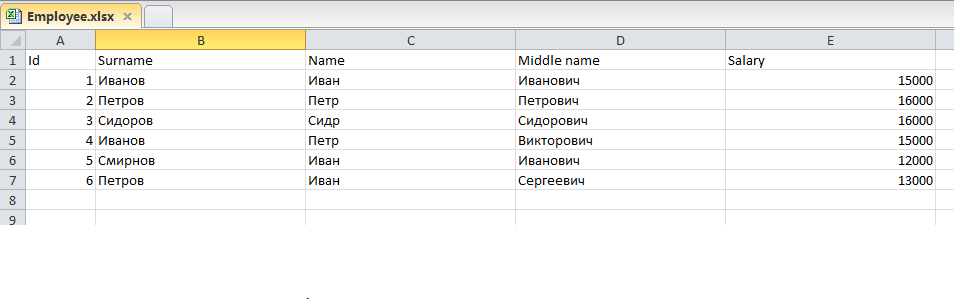


Рисунок 3.29 – Результат экспорта в Microsoft Excel

## 

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта по заданию «Распределение дополнительных обязанностей» были выполнены различные задачи по анализу поставленной задачи, создания физической модели.

На основе физической модели была создана реляционная база данных, в которой дополнительно были созданы триггеры, ограничения, хранимые процедуры для корректной работы базы данных.

Помимо этого, было спроектировано приложение для работы с данной базой данных. В этом приложении были реализованы функции просмотра, добавления, изменения и удаления записей базы данных, поиска записей, фильтрация для записей и экспорт таблиц

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Требования к оформлению квалификационных работ: метод. указания для студентов по направлению 230200 «Информационные системы» / Сост.: А. П. Власов, Н.А. Марчук: Иван. гос. хим.-технол. ун-т. – Иваново, 2010, 35 с.
2. Проектирование баз данных: лабораторный практикум / Э.Г. Галиаскаров, А.Ю. Крылов; Иван. гос. хим.-технол. ун-т.- Иваново, 2012.- 96 с.
3. Разработка приложений баз данных: лабораторный практикум / Э.Г. Галиаскаров и др.; Иван. гос. хим.-технол. ун-т. – Иваново, 2015. – 112 с.
4. Сайт обмена знаниями и решения проблем StackOverflow [Электронный ресурс] URL <https://stackoverflow.com/> (дата обращения 15.04.2020)
5. Упражнения по SQL [Электронный ресурс] URL <http://sql-ex.ru/> (дата обращения 15.04.2020)
6. Татьяна Карпова, курс «Базы данных: модели, разработка, реализация» Сайт «НОУ ИНТУИТ» [Электронный ресурс] URL <https://www.intuit.ru/studies/courses/1001/297/info> (дата обращения 01.04.2020)
7. Сайт с документацией по react, React [Электронный ресурс] URL <https://ru.reactjs.org/> (дата обращения 15.04.20)

# ПРИЛОЖЕНИЕ А. СКРИПТ БАЗЫ ДАННЫХ

CREATE TABLE [Employee]

(

[Employee\_code] integer NOT NULL IDENTITY ( 1,1 ) ,

[Salary] integer NULL ,

[Name] varchar(20) NULL ,

[Surname] varchar(20) NULL ,

[MiddleName] varchar(20) NULL

)

CREATE TABLE [Employee\_Work]

(

[Employee\_code] integer NOT NULL ,

[Work\_code] integer NOT NULL ,

[Code\_type\_work] integer NOT NULL

)

CREATE TABLE [Final\_salary]

(

[Final\_salary\_code] integer NOT NULL IDENTITY ( 1,1 ) ,

[Pay\_day] datetime NULL ,

[Payment\_for\_work] integer NULL ,

[Itog] integer NULL ,

[Employee\_code] integer NOT NULL

)

CREATE TABLE [Type\_work]

(

[Code\_type\_work] integer NOT NULL IDENTITY ( 1,1 ) ,

[Definition] varchar(20) NULL ,

[Payment\_for\_work] integer NULL

)

CREATE TABLE [Work]

(

[Work\_code] integer NOT NULL IDENTITY ( 1,1 ) ,

[Code\_type\_work] integer NOT NULL ,

[Date\_start] datetime NULL ,

[Date\_end] datetime NULL ,

[Deadline] datetime NULL ,

[Need\_Employee] integer NULL

)

create view Employees\_and\_work

as

select

E.Name, E.Surname, E.MiddleName,

TW.Definition, TW.Payment\_for\_work,

W.Date\_start, W.Date\_end, W.Need\_Employee,

EW.Work\_code, EW.Employee\_code, EW.Code\_type\_work

from

Employee E, Type\_work TW, Work W, Employee\_Work EW

where

(EW.Work\_code = W.Work\_code) and

(EW.Employee\_code = E.Employee\_code) and

(TW.Code\_type\_work = W.Code\_type\_work)

ALTER TABLE [Employee\_Work]

ADD CONSTRAINT [содержит] FOREIGN KEY ([Work\_code],[Code\_type\_work]) REFERENCES [Work]([Work\_code],[Code\_type\_work])

ALTER TABLE [Final\_salary]

ADD CONSTRAINT [зарабатывает] FOREIGN KEY ([Employee\_code]) REFERENCES [Employee]([Employee\_code])

ALTER TABLE [Work]

ADD CONSTRAINT [содержит] FOREIGN KEY ([Code\_type\_work]) REFERENCES [Type\_work]([Code\_type\_work])

ON UPDATE CASCADE

CREATE TRIGGER tD\_Employee ON Employee FOR DELETE AS

/\* erwin Builtin Trigger \*/

/\* DELETE trigger on Employee \*/

BEGIN

DECLARE @errno int,

@severity int,

@state int,

@errmsg varchar(255)

/\* erwin Builtin Trigger \*/

/\* Employee Final\_salary on parent delete restrict \*/

/\* ERWIN\_RELATION:CHECKSUM="00022ad4", PARENT\_OWNER="", PARENT\_TABLE="Employee"

CHILD\_OWNER="", CHILD\_TABLE="Final\_salary"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="зарабатывает", FK\_COLUMNS="Employee\_code" \*/

IF EXISTS (

SELECT \* FROM deleted,Final\_salary

WHERE

/\* %JoinFKPK(Final\_salary,deleted," = "," AND") \*/

Final\_salary.Employee\_code = deleted.Employee\_code

)

BEGIN

SELECT @errno = 30001,

@errmsg = 'Cannot delete Employee because Final\_salary exists.'

GOTO error

END

/\* erwin Builtin Trigger \*/

/\* Employee Employee\_Work on parent delete restrict \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="Employee"

CHILD\_OWNER="", CHILD\_TABLE="Employee\_Work"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="выполняет", FK\_COLUMNS="Employee\_code" \*/

IF EXISTS (

SELECT \* FROM deleted,Employee\_Work

WHERE

/\* %JoinFKPK(Employee\_Work,deleted," = "," AND") \*/

Employee\_Work.Employee\_code = deleted.Employee\_code

)

BEGIN

SELECT @errno = 30001,

@errmsg = 'Cannot delete Employee because Employee\_Work exists.'

GOTO error

END

CREATE TRIGGER tU\_Employee ON Employee FOR UPDATE AS

/\* erwin Builtin Trigger \*/

/\* UPDATE trigger on Employee \*/

BEGIN

DECLARE @numrows int,

@nullcnt int,

@validcnt int,

@insEmployee\_code integer,

@errno int,

@severity int,

@state int,

@errmsg varchar(255)

SELECT @numrows = @@rowcount

/\* erwin Builtin Trigger \*/

/\* Employee Employee\_Work on parent update restrict \*/

/\* ERWIN\_RELATION:CHECKSUM="00013dfc", PARENT\_OWNER="", PARENT\_TABLE="Employee"

CHILD\_OWNER="", CHILD\_TABLE="Employee\_Work"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="выполняет", FK\_COLUMNS="Employee\_code" \*/

IF

/\* %ParentPK(" OR",UPDATE) \*/

UPDATE(Employee\_code)

BEGIN

IF EXISTS (

SELECT \* FROM deleted,Employee\_Work

WHERE

/\* %JoinFKPK(Employee\_Work,deleted," = "," AND") \*/

Employee\_Work.Employee\_code = deleted.Employee\_code

)

BEGIN

SELECT @errno = 30005,

@errmsg = 'Cannot update Employee because Employee\_Work exists.'

GOTO error

END

END

CREATE TRIGGER tI\_Employee\_Work ON Employee\_Work FOR INSERT AS

/\* erwin Builtin Trigger \*/

/\* INSERT trigger on Employee\_Work \*/

BEGIN

DECLARE @numrows int,

@nullcnt int,

@validcnt int,

@errno int,

@severity int,

@state int,

@errmsg varchar(255)

SELECT @numrows = @@rowcount

/\* erwin Builtin Trigger \*/

/\* Work Employee\_Work on child insert restrict \*/

/\* ERWIN\_RELATION:CHECKSUM="0002e019", PARENT\_OWNER="", PARENT\_TABLE="Work"

CHILD\_OWNER="", CHILD\_TABLE="Employee\_Work"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="содержит", FK\_COLUMNS="Work\_code""Code\_type\_work" \*/

IF

/\* %ChildFK(" OR",UPDATE) \*/

UPDATE(Work\_code) OR

UPDATE(Code\_type\_work)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,Work

WHERE

/\* %JoinFKPK(inserted,Work) \*/

inserted.Work\_code = Work.Work\_code and

inserted.Code\_type\_work = Work.Code\_type\_work

/\* %NotnullFK(inserted," IS NULL","select @nullcnt = count(\*) from inserted where"," and") \*/

IF @validcnt + @nullcnt != @numrows

BEGIN

SELECT @errno = 30002,

@errmsg = 'Cannot insert Employee\_Work because Work does not exist.'

GOTO error

END

END

/\* erwin Builtin Trigger \*/

/\* Employee Employee\_Work on child insert restrict \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="Employee"

CHILD\_OWNER="", CHILD\_TABLE="Employee\_Work"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="выполняет", FK\_COLUMNS="Employee\_code" \*/

IF

/\* %ChildFK(" OR",UPDATE) \*/

UPDATE(Employee\_code)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,Employee

WHERE

/\* %JoinFKPK(inserted,Employee) \*/

inserted.Employee\_code = Employee.Employee\_code

/\* %NotnullFK(inserted," IS NULL","select @nullcnt = count(\*) from inserted where"," and") \*/

IF @validcnt + @nullcnt != @numrows

BEGIN

SELECT @errno = 30002,

@errmsg = 'Cannot insert Employee\_Work because Employee does not exist.'

GOTO error

END

END

CREATE TRIGGER tU\_Employee\_Work ON Employee\_Work FOR UPDATE AS

/\* erwin Builtin Trigger \*/

/\* UPDATE trigger on Employee\_Work \*/

BEGIN

DECLARE @numrows int,

@nullcnt int,

@validcnt int,

@insEmployee\_code integer,

@insWork\_code integer,

@insCode\_type\_work integer,

@errno int,

@severity int,

@state int,

@errmsg varchar(255)

SELECT @numrows = @@rowcount

/\* erwin Builtin Trigger \*/

/\* Work Employee\_Work on child update restrict \*/

/\* ERWIN\_RELATION:CHECKSUM="0002e2d7", PARENT\_OWNER="", PARENT\_TABLE="Work"

CHILD\_OWNER="", CHILD\_TABLE="Employee\_Work"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="содержит", FK\_COLUMNS="Work\_code""Code\_type\_work" \*/

IF

/\* %ChildFK(" OR",UPDATE) \*/

UPDATE(Work\_code) OR

UPDATE(Code\_type\_work)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,Work

WHERE

/\* %JoinFKPK(inserted,Work) \*/

inserted.Work\_code = Work.Work\_code and

inserted.Code\_type\_work = Work.Code\_type\_work

/\* %NotnullFK(inserted," IS NULL","select @nullcnt = count(\*) from inserted where"," AND") \*/

IF @validcnt + @nullcnt != @numrows

BEGIN

SELECT @errno = 30007,

@errmsg = 'Cannot update Employee\_Work because Work does not exist.'

GOTO error

END

END

/\* erwin Builtin Trigger \*/

/\* Employee Employee\_Work on child update restrict \*/

/\* ERWIN\_RELATION:CHECKSUM="00000000", PARENT\_OWNER="", PARENT\_TABLE="Employee"

CHILD\_OWNER="", CHILD\_TABLE="Employee\_Work"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="выполняет", FK\_COLUMNS="Employee\_code" \*/

IF

/\* %ChildFK(" OR",UPDATE) \*/

UPDATE(Employee\_code)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,Employee

WHERE

/\* %JoinFKPK(inserted,Employee) \*/

inserted.Employee\_code = Employee.Employee\_code

/\* %NotnullFK(inserted," IS NULL","select @nullcnt = count(\*) from inserted where"," AND") \*/

IF @validcnt + @nullcnt != @numrows

BEGIN

SELECT @errno = 30007,

@errmsg = 'Cannot update Employee\_Work because Employee does not exist.'

GOTO error

END

END

CREATE TRIGGER tD\_Final\_salary ON Final\_salary FOR DELETE AS

/\* erwin Builtin Trigger \*/

/\* DELETE trigger on Final\_salary \*/

BEGIN

DECLARE @errno int,

@severity int,

@state int,

@errmsg varchar(255)

/\* erwin Builtin Trigger \*/

/\* Employee Final\_salary on child delete restrict \*/

/\* ERWIN\_RELATION:CHECKSUM="0001657c", PARENT\_OWNER="", PARENT\_TABLE="Employee"

CHILD\_OWNER="", CHILD\_TABLE="Final\_salary"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="зарабатывает", FK\_COLUMNS="Employee\_code" \*/

IF EXISTS (SELECT \* FROM deleted,Employee

WHERE

/\* %JoinFKPK(deleted,Employee," = "," AND") \*/

deleted.Employee\_code = Employee.Employee\_code AND

NOT EXISTS (

SELECT \* FROM Final\_salary

WHERE

/\* %JoinFKPK(Final\_salary,Employee," = "," AND") \*/

Final\_salary.Employee\_code = Employee.Employee\_code

)

)

BEGIN

SELECT @errno = 30010,

@errmsg = 'Cannot delete last Final\_salary because Employee exists.'

GOTO error

END

CREATE TRIGGER tD\_Type\_work ON Type\_work FOR DELETE AS

/\* erwin Builtin Trigger \*/

/\* DELETE trigger on Type\_work \*/

BEGIN

DECLARE @errno int,

@severity int,

@state int,

@errmsg varchar(255)

/\* erwin Builtin Trigger \*/

/\* Type\_work Work on parent delete restrict \*/

/\* ERWIN\_RELATION:CHECKSUM="00010ac4", PARENT\_OWNER="", PARENT\_TABLE="Type\_work"

CHILD\_OWNER="", CHILD\_TABLE="Work"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="содержит", FK\_COLUMNS="Code\_type\_work" \*/

IF EXISTS (

SELECT \* FROM deleted,Work

WHERE

/\* %JoinFKPK(Work,deleted," = "," AND") \*/

Work.Code\_type\_work = deleted.Code\_type\_work

)

BEGIN

SELECT @errno = 30001,

@errmsg = 'Cannot delete Type\_work because Work exists.'

GOTO error

END

CREATE TRIGGER tU\_Type\_work ON Type\_work FOR UPDATE AS

/\* erwin Builtin Trigger \*/

/\* UPDATE trigger on Type\_work \*/

BEGIN

DECLARE @numrows int,

@nullcnt int,

@validcnt int,

@insCode\_type\_work integer,

@errno int,

@severity int,

@state int,

@errmsg varchar(255)

SELECT @numrows = @@rowcount

/\* erwin Builtin Trigger \*/

/\* Type\_work Work on parent update cascade \*/

/\* ERWIN\_RELATION:CHECKSUM="00018991", PARENT\_OWNER="", PARENT\_TABLE="Type\_work"

CHILD\_OWNER="", CHILD\_TABLE="Work"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="содержит", FK\_COLUMNS="Code\_type\_work" \*/

IF

/\* %ParentPK(" OR",UPDATE) \*/

UPDATE(Code\_type\_work)

BEGIN

IF @numrows = 1

BEGIN

SELECT @insCode\_type\_work = inserted.Code\_type\_work

FROM inserted

UPDATE Work

SET

/\* %JoinFKPK(Work,@ins," = ",",") \*/

Work.Code\_type\_work = @insCode\_type\_work

FROM Work,inserted,deleted

WHERE

/\* %JoinFKPK(Work,deleted," = "," AND") \*/

Work.Code\_type\_work = deleted.Code\_type\_work

END

ELSE

BEGIN

SELECT @errno = 30006,

@errmsg = 'Cannot cascade Type\_work update because more than one row has been affected.'

GOTO error

END

END

CREATE TRIGGER tD\_Work ON Work FOR DELETE AS

/\* erwin Builtin Trigger \*/

/\* DELETE trigger on Work \*/

BEGIN

DECLARE @errno int,

@severity int,

@state int,

@errmsg varchar(255)

/\* erwin Builtin Trigger \*/

/\* Work Employee\_Work on parent delete restrict \*/

/\* ERWIN\_RELATION:CHECKSUM="000138a4", PARENT\_OWNER="", PARENT\_TABLE="Work"

CHILD\_OWNER="", CHILD\_TABLE="Employee\_Work"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="содержит", FK\_COLUMNS="Work\_code""Code\_type\_work" \*/

IF EXISTS (

SELECT \* FROM deleted,Employee\_Work

WHERE

/\* %JoinFKPK(Employee\_Work,deleted," = "," AND") \*/

Employee\_Work.Work\_code = deleted.Work\_code AND

Employee\_Work.Code\_type\_work = deleted.Code\_type\_work

)

BEGIN

SELECT @errno = 30001,

@errmsg = 'Cannot delete Work because Employee\_Work exists.'

GOTO error

END

CREATE TRIGGER tU\_Work ON Work FOR UPDATE AS

/\* erwin Builtin Trigger \*/

/\* UPDATE trigger on Work \*/

BEGIN

DECLARE @numrows int,

@nullcnt int,

@validcnt int,

@insWork\_code integer,

@insCode\_type\_work integer,

@errno int,

@severity int,

@state int,

@errmsg varchar(255)

SELECT @numrows = @@rowcount

/\* erwin Builtin Trigger \*/

/\* Type\_work Work on child update cascade \*/

/\* ERWIN\_RELATION:CHECKSUM="00013f0e", PARENT\_OWNER="", PARENT\_TABLE="Type\_work"

CHILD\_OWNER="", CHILD\_TABLE="Work"

P2C\_VERB\_PHRASE="", C2P\_VERB\_PHRASE="",

FK\_CONSTRAINT="содержит", FK\_COLUMNS="Code\_type\_work" \*/

IF

/\* %ChildFK(" OR",UPDATE) \*/

UPDATE(Code\_type\_work)

BEGIN

INSERT Type\_work (Code\_type\_work)

SELECT Code\_type\_work

FROM inserted

WHERE

/\* %NotnullFK(inserted," IS NOT NULL AND") \*/

NOT EXISTS (

SELECT \* FROM Type\_work

WHERE

/\* %JoinFKPK(inserted,Type\_work," = "," AND") \*/

inserted.Code\_type\_work = Type\_work.Code\_type\_work

)

END

# ПРИЛОЖЕНИЕ Б. ПЕРЕЧЕНЬ ФАЙЛОВ

1. Пояснительная записка к курсовому проекту в папке doc Долгова пояснительная записка.docx
2. Инфологическая модель в папке erwin logicalModel.erwin
3. Физическая модель базы данных в папке erwin physicalModel.erwin
4. Файлы базы данных в папке db Works.mdf
5. Папки с исходным кодом приложения Server и Client