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 №30

**Выполнил студент: Лебедев Вячеслав Евгеньевич курс/гр. 3/147**

ФИО

**Проверил: Чернявская Анастасия Сергеевна**

ФИО

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** Информационные системы и технологии

**Дисциплина** Управление данными

**ЗАДАНИЕ НА КУРСОВОЙ ПРОЕКТ**

**Студенту** Лебедеву Вячеславу Евгеньевичу курс/гр. **3/147**

(фамилия, имя, отчество - полностью)

**Тема:** Разработка базы данных MS SQL и приложения средствами Visual Studio. Вариант №30

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: *Чернявская А.С.*

**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 MSSQL и реализация SQL-запросов | 01.04.20 – 14.04.20 |  |
| 4 | Разработка приложения средствами VisualStudio | 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 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Подпись /Ф.И.О.)

Студент Лебедев В.Е

(Подпись /Ф.И.О.)

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc57680055)

[ОСНОВНАЯ ЧАСТЬ 6](#_Toc57680056)

[1 СИСТЕМНЫЙ АНАЛИЗ 6](#_Toc57680057)

[1.1Описание предметной области 6](#_Toc57680058)

[1.2 Логическая модель «Как есть» 7](#_Toc57680059)

[1.3 Логическая модель «Как должно быть» 8](#_Toc57680060)

[1.4 Список запросов, которые должны быть реализованы в БД 9](#_Toc57680061)

[1.5 Набор требований, которым должно отвечать разрабатываемое приложение 10](#_Toc57680062)

[2 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ 11](#_Toc57680063)

[3. ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ 26](#_Toc57680064)

[3.1 Работа с таблицей «Payment table». 28](#_Toc57680065)

[3.2 Работа с таблицей «Outlet table» 28](#_Toc57680066)

[3.3 Поиск записей 29](#_Toc57680067)

[3.4 Экспорт таблицы Excel 31](#_Toc57680068)

[3.5 Отчет по сделкам 32](#_Toc57680069)

[ЗАКЛЮЧЕНИЕ 34](#_Toc57680070)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 35](#_Toc57680071)

[ПРИЛОЖЕНИЕ А. СКРИПТ БАЗЫ ДАННЫХ 36](#_Toc57680072)

[ПРИЛОЖЕНИЕ Б. ПЕРЕЧЕНЬ ФАЙЛОВ 63](#_Toc57680073)

# ВВЕДЕНИЕ

Проектирование базы данных -это упорядоченный формализованный процесс создания системы взаимосвязанных описаний, т.е. таких моделей предметной области, которые связывают (фиксируют) хранимые в базе данные с объектами предметной области, описываемыми этими данными. Прикладное назначение таких описаний состоит в том, чтобы пользователь, практически не имеющий представления об организации данных в БД (физическом размещении в памяти данных и механизмах их поиска), обращая запрос к базе, имел бы практическую возможность получить адекватную информацию о состоянии объекта предметной области.

Актуальным вопросом в настоящее время является создание автоматизированных систем, которые требуют минимальных человеческих затрат и выполняют огромные расчеты за доли секунд, выполняя которые человек затрачивает немало времени и ресурсов. Примером такой автоматизированной системы является Среда SQL Server Management Studio в которой мы и будем проектировать БД.

Данная пояснительная записка описывает процесс создания базы данных, создания приложения, работающего с этой базой данных. Ознакомление с исходным заданием, построение инфологических моделей «как есть» и «как должно быть». Затем описание процесса разработки физической модели базы данных SQL, создание представлений, хранимых процедур, наполнения базы данных и выполнения запросов. В конце 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 «СДАЧА В АРЕНДУ ТОРГОВЫХ ПОМЕЩЕНИЙ»

## 1.1Описание предметной области

Вы работаете в крупном торговом центре, сдающем в аренду коммерсантам свои торговые площади.

Вашей задачей является наведение порядка в финансовой сфере работы торгового центра.

Работа торгового центра построена следующим образом: в результате планирования вы определили некоторое количество торговых точек в пределах здания, которые могут сдаваться в аренду. Для каждой из торговых точек важными данными являются этаж, площадь, наличие кондиционера и стоимость аренды в день. Со всех потенциальных клиентов вы собираете стандартные данные (название, адрес, телефон, реквизиты, контактное лицо). При появлении потенциального клиента вы показываете ему имеющиеся свободные площади. При достижении соглашения вы оформляете договор, фиксируя в базе данных торговую точку, клиента, период (срок) аренды. Таблицы

Торговые точки (Код торговой точки, Этаж, Площадь, Наличие кондиционера, Стоимость аренды в день).

Клиенты (Код клиента, Название, Реквизиты, Адрес, Телефон, Контактное лицо).

Аренда (Код аренды, Код торговой точки, Код клиента, Дата начала, Дата окончания).

Развитие постановки задачи:

В результате эксплуатации базы данных выяснилось, что некоторые клиенты арендуют сразу несколько торговых точек. Помимо этого, вам необходимо собирать информацию о ежемесячных платежах, поступающих от арендаторов. Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы. В этапе системного анализа проводится осмысление поставленной задачи. Цель системного анализа предметной области как этапа проектирования – выделить предметную область как систему объектов и их взаимосвязей, определив при этом функционально-информационные требования к их последующему представлению в виде системы взаимосвязанных данных.

## 1.2 Логическая модель «Как есть»

Торговые точки (Код торговой точки, Этаж, Площадь, Наличие кондиционера, Стоимость аренды в день).

Клиенты (Код клиента, Название, Реквизиты, Адрес, Телефон, Контактное лицо).

Аренда (Код аренды, Код торговой точки, Код клиента, Дата начала, Дата окончания).

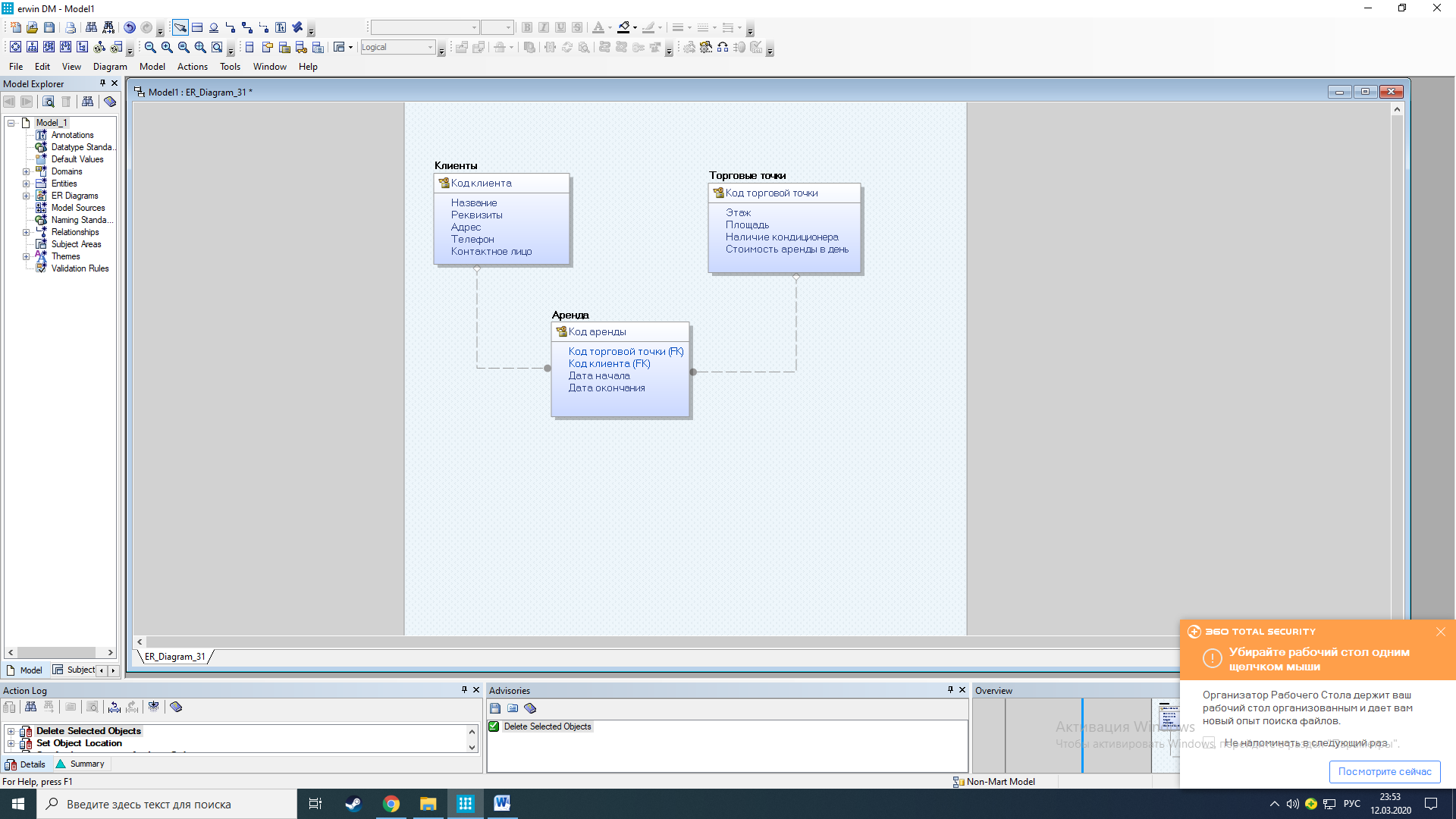


Рисунок 1.1 – Модель в нотации IDEF1x «как есть»

Исходя из проделанного анализа, мы видим, что появляются новые сущности «Договор» и «Платеж».

* В Сущность «Договор» входят атрибуты: Код договора (уникальный), Дата начала, Дата конца;
* Сущность «Платеж» содержит Код платежа, Дата платежа, Период платежа, Сумма платежа;
* Между сущностями «Торговые точки» - «Аренда», «Аренда» - «Договор» и «Договор» - «Платеж» устанавливается неспецифичная связь (многие-ко-многим);
* Между сущностями «Клиент» - «Договор» «устанавливается идентифицирующая;

## 1.3 Логическая модель «Как должно быть»

* Клиент (Код клиента, Название, Реквизиты, Адрес, Телефон, Контактное лицо);
* Торговые точки (Код торговой точки, Этаж, Площадь, Наличие кондиционера, Стоимость аренды в день);
* Договор (Код договора, Код клиента, Дата начала, Дата конца)
* Аренда (Код аренды**,** Код торговой точки, Код клиента, Дата начала, Дата окончания, Код договора)
* Платеж (Код платежа, Дата платежа, Период платежа, Сумма платежа, Код договора, Код клиента)

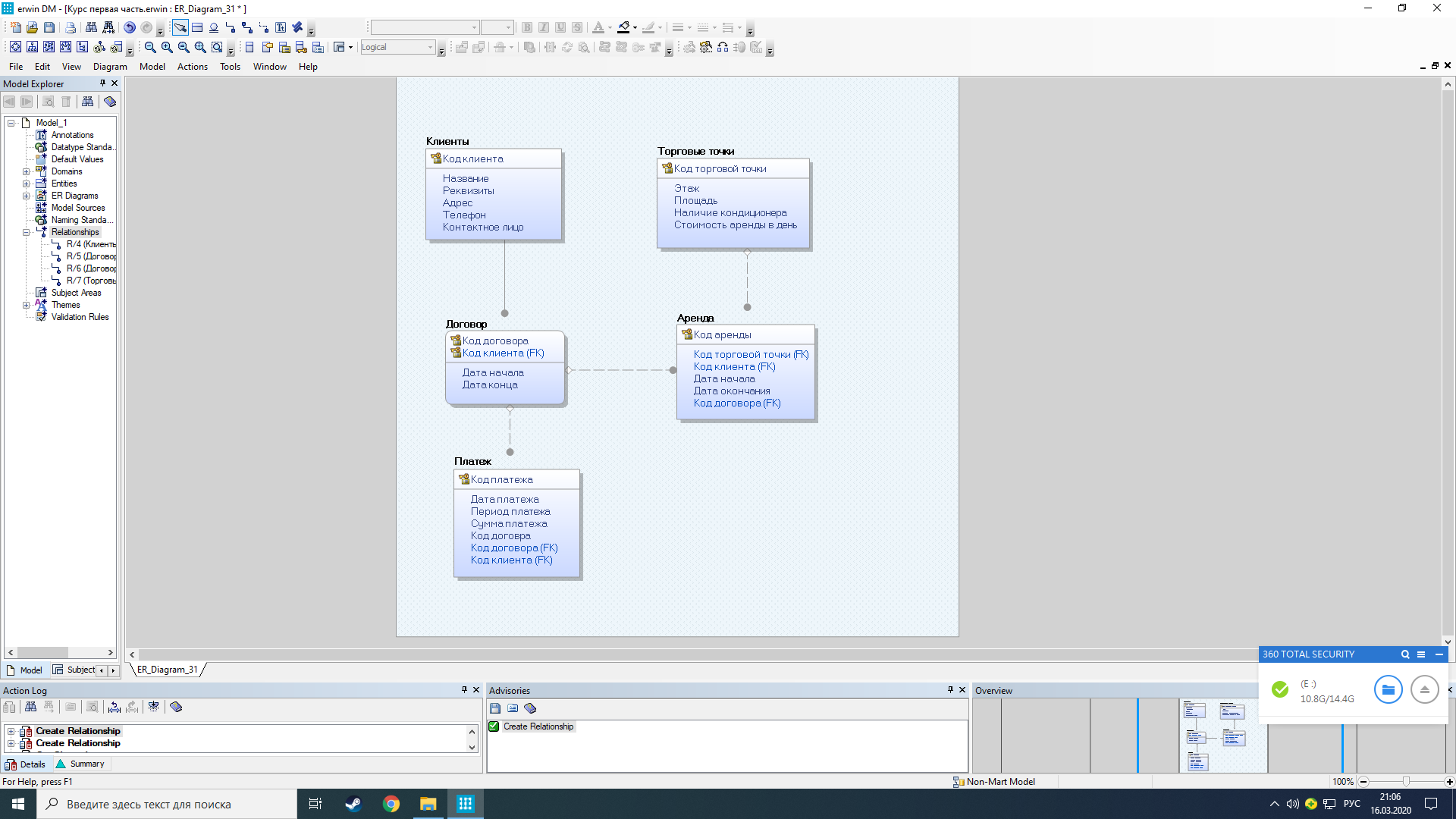


Рисунок 1.2 – Построение модели в нотации IDEF1x «Как должно быть»

Главной задачей системного анализа является разрешение проблемной ситуации, возникшей перед объектом проводимого системного исследования. Системный анализ занимается изучением проблемной ситуации, выяснением её причин, выработкой вариантов её устранения, принятием решения и организацией дальнейшего функционирования системы, разрешающего проблемную ситуацию. Начальным этапом любого системного исследования является изучение объекта проводимого системного анализа с последующей его формализацией. На этом этапе возникают задачи, в корне отличающие методологию системных исследований от методологии других дисциплин, а именно, в системном анализе решается двуединая задача. С одной стороны, необходимо формализовать объект системного исследования, с другой стороны, формализации подлежит процесс исследования системы, процесс постановки и решения проблемы.

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

1. Вывести доход за последние полгода?
2. Сколько заключено контрактов за последние 3 года?
3. Вывести список всех клиентов в алфавитном порядке?
4. Сколько в среднем стоит аренда помещения?
5. Вывести имена компаний, которые находятся выше, чем 1 этаж?
6. Вывести среднюю ежедневную плату по всем торговым помещениям?
7. Вывести среднюю площадь помещений с наличием кондиционера?
8. Вывести максимальную сумму выплат по аренде?
9. Определить количество свободных помещений под сдачу в аренду?
10. Сколько платят в день клиенты, у которых еще действует контракт?
11. Вывести самую большую площадь, сдаваемую в аренду?
12. Вывести список клиентов у которых площадь больше чем 20 кв.м?
13. Вывести максимальную арендную плату в день по действующим контрактам?
14. Вывести список компаний, в названии которых, нет ООО и телефон?
15. Вывести список клиентов и даты договоров, которые закончатся в этом году?

# 1.5 Набор требований, которым должно отвечать разрабатываемое приложение

* обеспечить ведения списка арендованных торговых точек
* обеспечить ведения списка арендаторов
* обеспечить ведения списка видов торговли
* обеспечить ведение списка ежемесячных поступлений от арендаторов
* обеспечить список количества торговых точек в торговом центре
* обеспечить фиксацию результатов сделки

# 2 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

Базы данных – это программы, которые позволяют сохранять и получать большие объемы связанной информации. Базы данных состоят из таблиц, которые содержат информацию. Когда вы создаете базу данных необходимо подумать о том, какие таблицы вам нужно создать и какие связи существуют между информацией в таблицах. Иначе говоря, вам нужно подумать о проекте вашей базы данных. Хороший проект базы данных, как было сказано ранее, обеспечит целостность данных и простоту их обслуживания.

Для качественного проектирования базы данных существуют различные методики, различные последовательности шагов или этапов, которые во многом похожи. И в целом мы можем выделить следующие этапы:

* Выделение сущностей и их атрибутов, которые будут храниться в базе данных, и формирование по ним таблиц. Атомизация сложных атрибутов на более простые.
* Определение уникальных идентификаторов (первичных ключей) объектов, которые хранятся в строках таблицы
* Определение отношений между таблицами с помощью внешних ключей
* Нормализация базы данных

Основная 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.

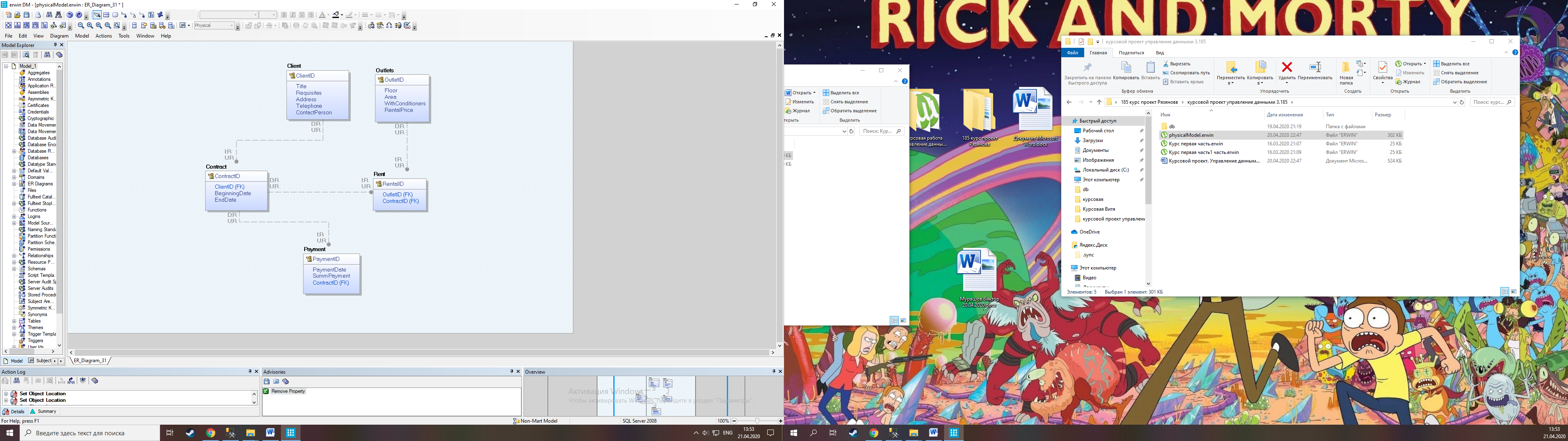


Рисунок 2.1 – Физическая модель

Физические модели данных служат для отображения моделей данных. Основными понятиями модели данных являются поле, логическая запись, логический файл. Слово «логический» введено, чтобы отличать понятия, относящиеся к логической модели данных, от понятий, относящихся к физической модели данных.

Физическая модель данных зависит от конкретной СУБД, фактически являясь отображением системного каталога. В физической модели содержится информация о всех объектах БД. Поскольку стандартов на объекты БД не существует (например, нет стандарта на типы данных), физическая модель зависит от конкретной реализации СУБД.

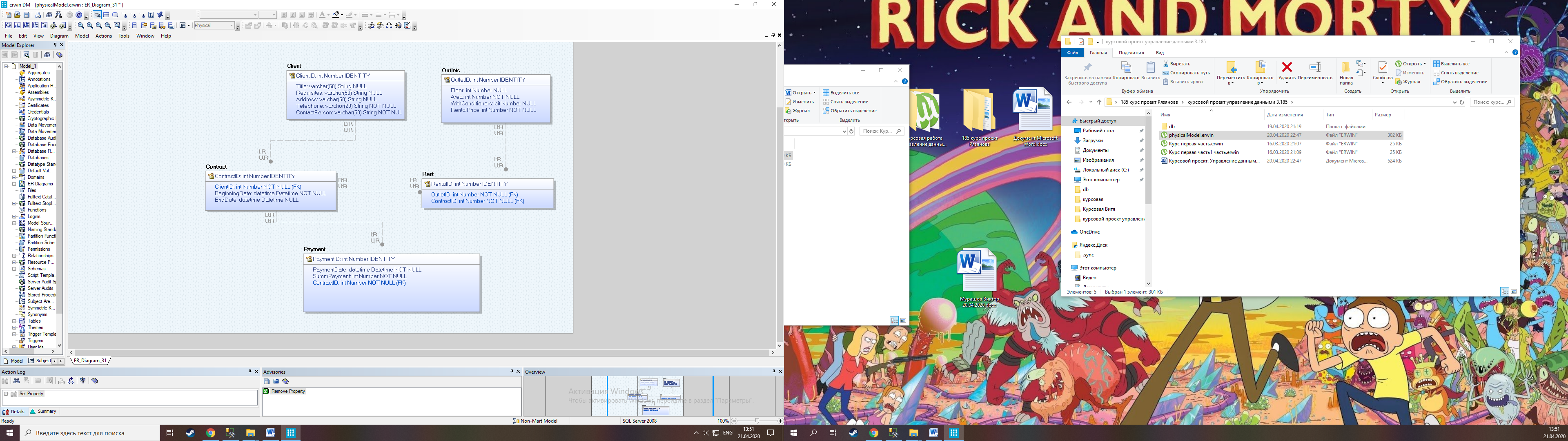


Рисунок 2.2 – Физическая модель со всеми типами данных

Физическая модель данных, как правило, создается на основе логической, поэтому каждому объекту логической модели соответствует объект физической модели (хотя соответствие может быть неоднозначным). В физической модели данных сущности логической модели данных соответствует таблица, экземпляру сущности – строка в таблице, а атрибуту – колонка таблицы. Кроме перечисленных выше объектов, физическая модель может содержать объекты, тип которых зависит от СУБД: индексы, представления, последовательности, триггеры, процедуры и т.п.

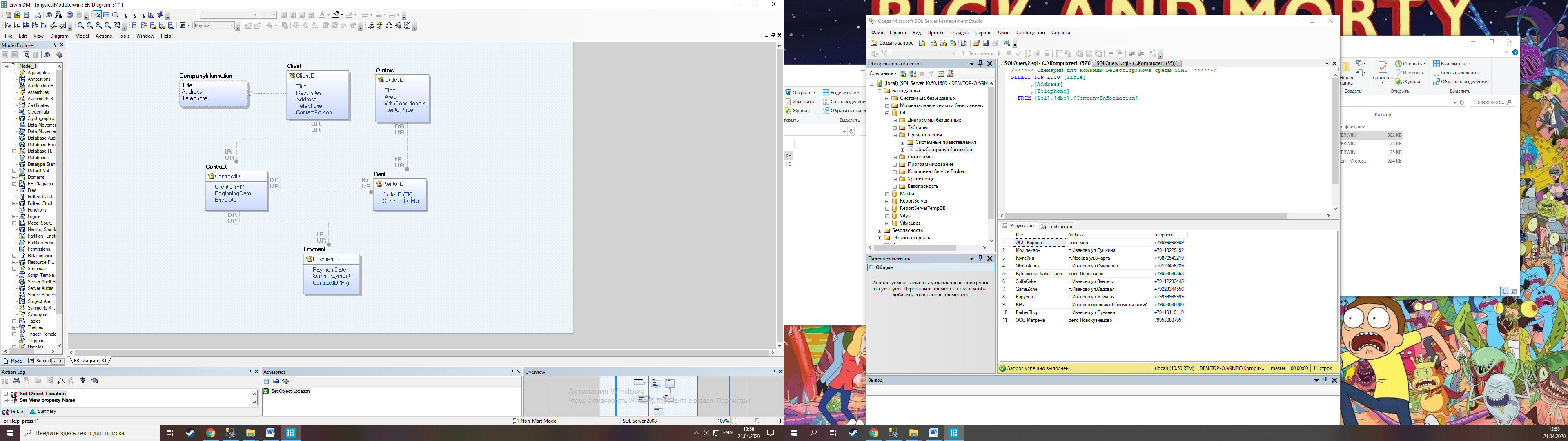


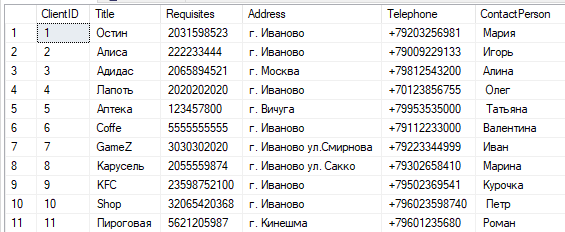
Рисунок 2.3 – Представление

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

Далее представлены таблицы базы данных и запросы по заполнению таблиц.

Запрос SQLна добавление записи в таблицу Client

INSERT INTO Client(Title, Requisites, Address, Telephone, ContactPerson)  
VALUES('Название', 'Реквизиты', 'Адрес', 'Телефон', 'Контактное лицо')



2.4 – Таблица Client

Запрос SQLна добавление записи в таблицу Contract

INSERT INTO Contract(ClientID, BeginningDate, EndDate)  
 VALUES('Код клиента', 'Дата начала', 'Дата конца')

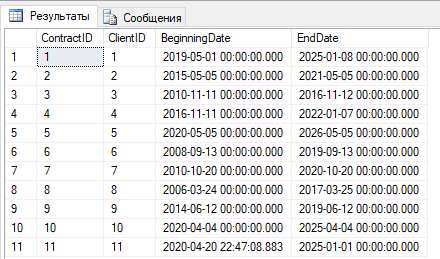


Рисунок 2.5 – Таблица Contract

Запрос SQL на добавление записи в таблицу Outlets

INSERT INTO Outlets(Floor, Area, RentalPrice, WithConditioners)  
 VALUES(Этаж, Площадь, Стоимость аренды, Наличие кондиционера)



Рисунок 2.6 – Таблица Outlets

Запрос SQLна добавление записи в таблицу Payment

INSERT INTO Payment(PaymentDate, SummPayment, ContractID)  
 VALUES('Дата платежа', Сумма платежа, Контактное лицо)



Рисунок 2.7 – Таблица Payment

Запрос SQL на добавление записи в таблицу Rent

INSERT INTO Rent (OutletID, ContractID)

VALUES(Код торговой точки, Контактное лицо)

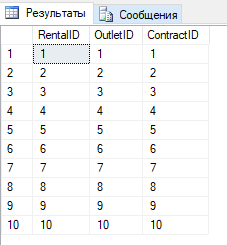


Рисунок 2.8 – Таблица Rent

Теперь проверим работу представлений.

Представления - это таблицы чье содержание выбирается или получается из других таблиц. Они работают в запросах и операторах DML точно также как и основные таблицы, но не содержат никаких собственных данных. Представления - подобны окнам, через которые вы просматриваете информацию (как она есть, или в другой форме, как вы потом увидите), которая фактически хранится в базовой таблице. Представление - это фактически запрос, который выполняется всякий раз, когда представление становится темой команды.

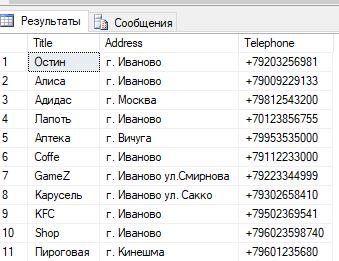


Рисунок 2.9 – Вывод представления

Создадим ограничение для таблицы Contract. Нам нужно чтобы нельзя было добавить запись с неверной датой создания и неверной датой окончания, так же что дата начала должна быть раньше даты конца.

Создание ограничений:  
 ALTER TABLE Contract add constraint ValidDates CHECK (BeginningDate < EndDate)  
 Проверка ограничения ValidDates:  
INSERT INTO Contract (ClientID, BeginningDate, EndDate)  
VALUES (5, '01.01.2023', '01.01.2013')  
 Вывод:  
 Конфликт инструкции INSERT с ограничением CHECK "ValidDates". Конфликт произошел в базе данных "Rent", таблица "dbo.Contract ", column 'BeginningDate'.  
Выполнение данной инструкции было прервано.

Запрос на создание триггера:

CREATETRIGGERSetDefaulConditionerONOutlets

INSTEADOFINSERT

AS

DECLARE

@OutletIDASint,

@FloorASint,

@AreaASint,

@WithConditionersASbit,

@RentalPriceASint

SELECT@OutletID=OutletID,@Floor=Floor,@Area=Area,@WithConditioners=WithConditioners,@RentalPrice=RentalPrice

FROMinserted

IF@WithConditionersisNULL

BEGIN

INSERTINTOOutlets(WithConditioners,Floor,Area,RentalPrice)

VALUES(0 ,@Floor,@Area,@RentalPrice)

END

IF@WithConditionersisnotNull

BEGIN

INSERTINTOOutlets(WithConditioners,Floor,Area,RentalPrice)

VALUES(@WithConditioners,@Floor,@Area,@RentalPrice)

END

Пробный запрос:  
INSERTINTOOutlets(Area,Floor,RentalPrice)

values(10,1,500)

(Он уже есть это последняя запись в таблице)

Процедура:

CREATEPROCEDURECreateClientContract

(

@CLIENT\_Titlevarchar(50),

@CLIENT\_Requisitesvarchar(50),

@CLIENT\_Addressvarchar(50),

@CLIENT\_Telephonevarchar(20),

@CLIENT\_ContactPersonvarchar(50),

@CONTRACT\_EndDatedate

)

AS

BEGIN

INSERTINTOClient(Title,Requisites,Address,Telephone,ContactPerson)

VALUES (@CLIENT\_Title,@CLIENT\_Requisites,@CLIENT\_Address,@CLIENT\_Telephone,@CLIENT\_ContactPerson)

INSERTINTOContract (BeginningDate,EndDate,ClientID)

VALUES (GETDATE(),@CONTRACT\_EndDate,SCOPE\_IDENTITY())

END

RETURN

## Теперь реализуем список запросов.

1. Вывести доход за последние полгода?

SELECT SUM(p.SummPayment)

FROM Payment p

WHERE p.PaymentDate BETWEEN DATEADD(M,6,GETDATE()) AND GETDATE()

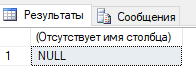


Рисунок 2.10 – Вывод запроса

2. Сколько заключено контрактов, за последние 3 года?

SELECT COUNT(c.BeginningDate)FROM Contract c

WHERE c.BeginningDate BETWEEN DATEADD (yy, -3, GETDATE()) AND GETDATE()

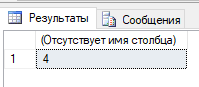


Рисунок 2.11 – Вывод запроса

3. Вывести список всех клиентов в алфавитном порядке?

SELECT\* FROM Client ORDER BY Title

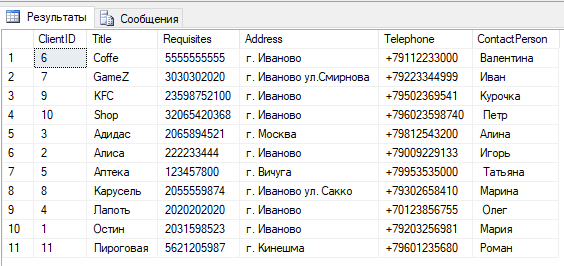


Рисунок 2.12 – Вывод запроса

4. Сколько в среднем стоит аренда помещения?

SELECT AVG(o.RentalPrice)

FROM Outlets o

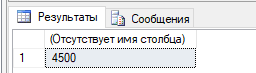


Рисунок 2.13 – Вывод запроса

5. Вывести имена компаний, которые находятся выше, чем 1 этаж?

SELECT cl.Title,o.Floor

FROM Outlets o JOIN Rent r ON o.OutletID=r.OutletID

JOIN Contract c ON c.ContractID=r.ContractID

JOIN Client cl ON cl.ClientID=c.ClientID

WHERE o.Floor> 1

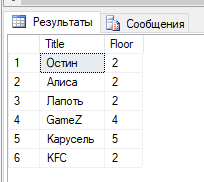


Рисунок 2.14 – Вывод запроса

6. Вывести среднюю ежедневную плату по всем торговым помещениям?

SELECT AVG(o.RentalPrice) average

FROM Outlets o

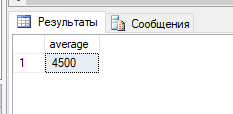


Рисунок 2.15 – Вывод запроса

7. Вывести среднюю площадь помещений с наличием кондиционера?

SELECT AVG(Area) avgArea

FROM Outlets o

WHERE o.WithConditioners= 1

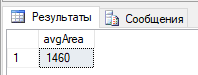


Рисунок 2.16 – Вывод запроса

1. Вывести максимальную сумму выплат по аренде?

select DISTINCT MAX(p.SummPayment) AS MAX

FROM Payment p

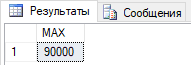


Рисунок 2.17 – Вывод запроса

9. Определить количество свободных помещений под сдачу в аренду?

SELECT COUNT(o.OutletID) AS count

FROM Outlets o

WHERE o.OutletID NO TIN (

SELECT o.OutletID

FROM Outlets o

JOIN Rent r ON o.OutletID=r.OutletID)

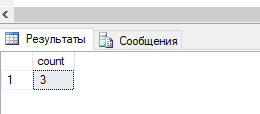


Рисунок 2.18 – Вывод запроса

10. Сколько платят в день клиенты, у которых еще действует контракт?

SELECT o.RentalPrice,cl.ContactPerson,c.BeginningDate

FROM Outlets o JOIN Rent r ON o.OutletID=r.OutletID

JOIN Contract c ON c.ContractID=r.ContractID

JOIN Clientcl ON cl.ClientID=c.ClientID

WHERE c.EndDate>GETDATE()

GROUPBY cl.ContactPerson,o.RentalPrice,c.BeginningDate

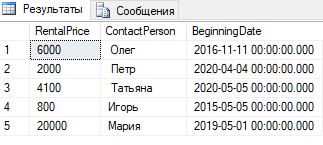


Рисунок 2.19 – Вывод запроса

11. Вывести самую большую площадь, сдаваемую в аренду?

SELECT MAX(o.Area) MaxArea

FROM Outlets o JOIN Rent r ON o.OutletID=r.OutletID

JOIN Contract c ON c.ContractID=r.ContractID

WHERE c.EndDate>GETDATE()

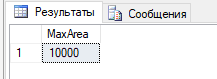


Рисунок 2.20 – Вывод запроса

12. Вывести список клиентов у которых площадь больше чем 20 кв.м?

SELECT cl.ContactPerson,o.Area

FROM Outlets o JOIN Rent r ON o.OutletID=r.OutletID

JOIN Contract c ON c.ContractID=r.ContractID

JOIN Client cl ON cl.ClientID=c.ClientID

WHERE o.Area> 20

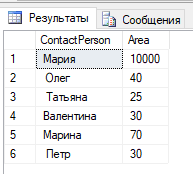


Рисунок 2.21 – Вывод запроса

13. Вывести максимальную арендную плату в день по действующим контрактам?

SELECT MAX(o.RentalPrice)AS Max

FROM Outlets o JOIN Rent r ON o.OutletID=r.OutletID

JOIN Contract c ON c.ContractID=r.ContractID

WHERE c.EndDate>GETDATE()

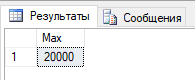


Рисунок 2.22 – Вывод запроса

14. Вывести список компаний, в названии которых нет ООО и телефон?

SELECT Title,Telephone

FROM Client cl

WHERE cl.Title Not LIKE '%ООО%'

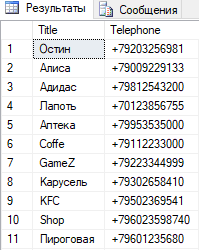


Рисунок 2.23 – Вывод запроса

15. Вывести список клиентов и даты договоров, которые закончатся в этом году?

SELECT cl.ContactPerson,c.BeginningDate,c.EndDate

FROM Contract c

JOIN Client cl ON c.ClientID=cl.ClientID

WHERE YEAR (c.EndDate)=YEAR (GETDATE()) AND c.EndDate>GETDATE()

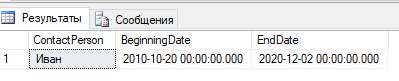


Рисунок 2.24 – Вывод запроса

На данном этапе мы спроектировали БД с использование Erwin. Для этого создали логическую модель данных. Описали предметную область, определили сущности, зависимости между сущностями, задали первичные ключи. Затем был переход от этапа логической модели данных к этапу описания физической модели данных. Также были написаны запросы и представлены фрагменты вывода данных запросов.

# 3. ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ

На данном этапе необходимо спроектировать приложение-ассистент для быстрой работы с базой данных. Проектирование приложения основывалось по следующим требованиям:

* обеспечить ведения списка арендованных торговых точек
* обеспечить ведения списка арендаторов
* обеспечить ведения списка видов торговли
* обеспечить ведение списка ежемесячных поступлений от арендаторов
* обеспечить список количества торговых точек в торговом центре
* обеспечить фиксацию результатов сделки.

Далее мы рассмотрим разработанное приложение.

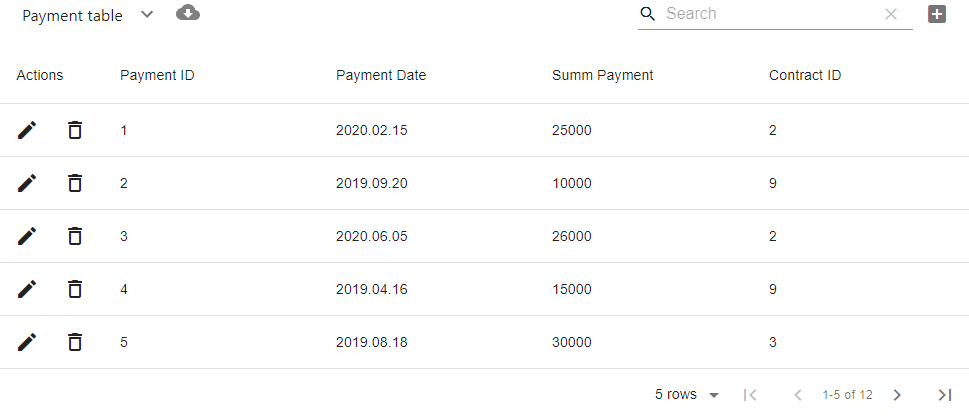


Рисунок 3.1 –«Payment table»

Записи, представленные в таблицах, можно без труда изменить, добавить либо удалить. Запись можно изменить, нажав на кнопку «карандаш». Чтобы удалить, необходимо нажать на «корзину» и запись удалится. В левом нижнем углу представлен «ползунок», с помощью него, можно переходить между записями.

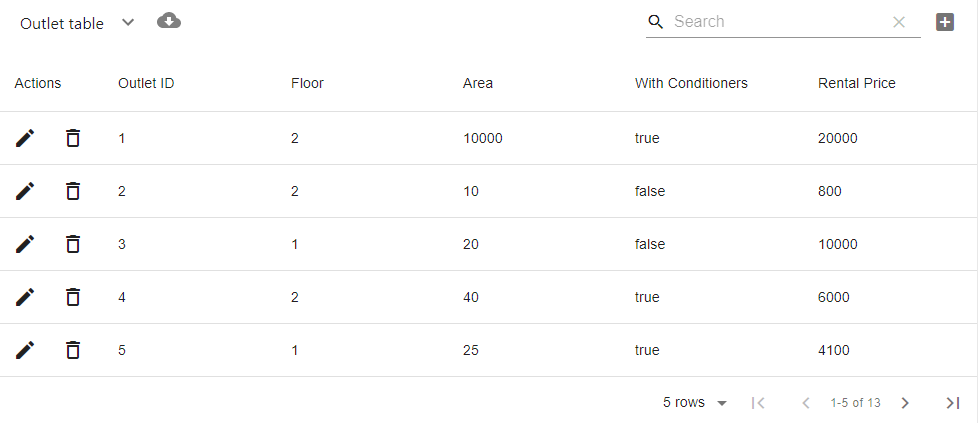


Рисунок 3.2 – «Outlet table»

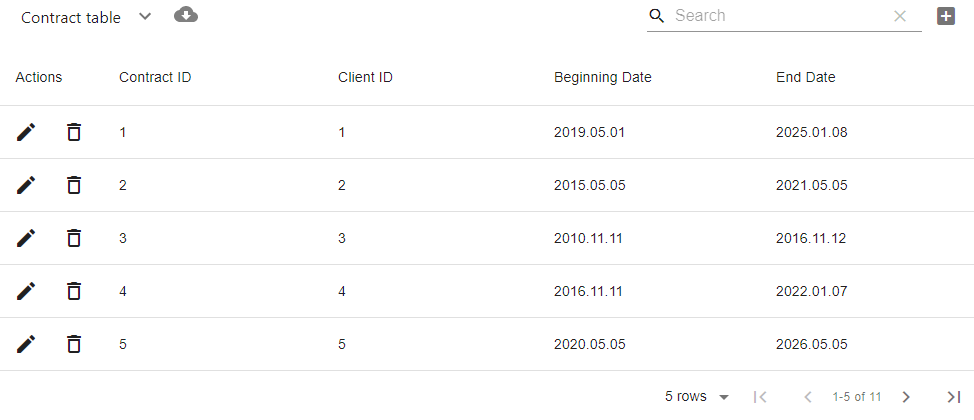


Рисунок 3.3 – «Contract table»

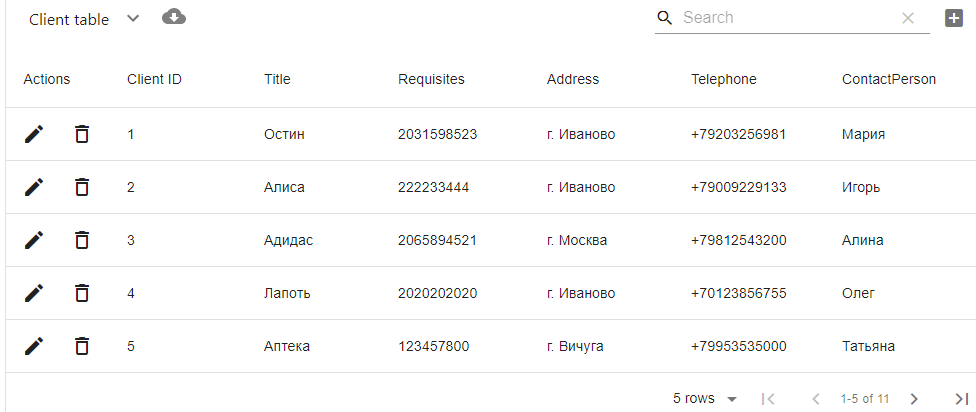


Рисунок 3.4 – «Client table»

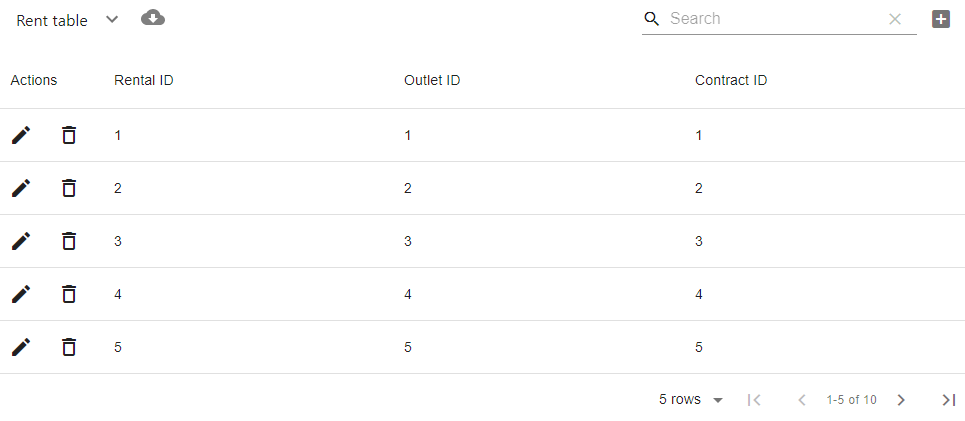


Рисунок 3.5 – «Rent table»

## 3.1 Работа с таблицей «Payment table».

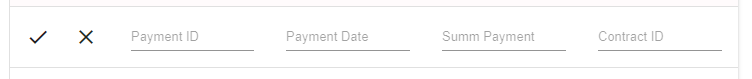
Нажав на «+» в правом верхнем углу можно добавить новую запись 

Рисунок 3.6 – Добавление записи

Кнопка «Корзина» удаляет выбранную запись из таблицы

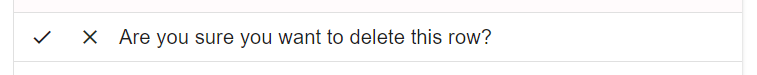


Рисунок 3.7 – Удаление записи

## 3.2 Работа с таблицей «Outlet table»

Кнопка «+» в правом верхнем углу добавляет новую запись в таблицу

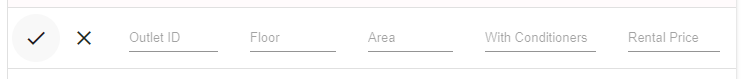


Рисунок 3.8 – Добавление записи

Кнопка «Карандаш» загружает в форму «Outlet table» данные строки, чтобы ее изменить.



Рисунок 3.9 – Изменение записи

Кнопка «Корзина» удаляет выбранную запись из таблицы

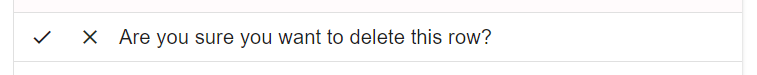


Рисунок 3.10 – Удаление записи

## 3.3 Поиск записей

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



Рисунок 3.11 – Окно поиска

Также можно использовать фильтрацию, для этого необходимо сначала выбрать нужную таблицу, затем нажать на кнопку “стрелка вниз”.

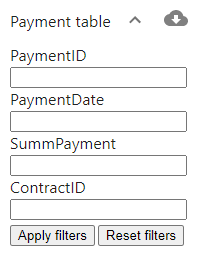


Рисунок 3.12 – Фильтры

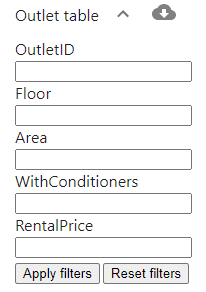


Рисунок 3.13 – Фильтры

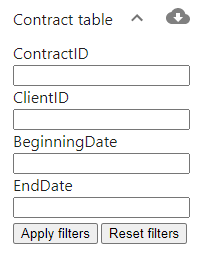


Рисунок 3.14 – Фильтры

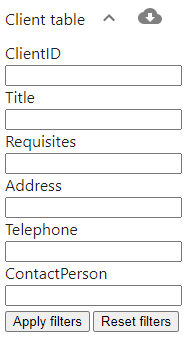


Рисунок 3.15 – Окно фильтров

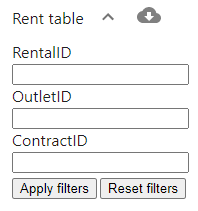


Рисунок 3.16 – Окно фильтров

## 3.4 Экспорт таблицы Excel

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



Рисунок 3.17 –Выгрузка в Microsoft Excel

## 3.5 Отчет по сделкам

Для просмотра выплаты по каждому клиенту, нужно перейти в таблицу Payment. В выпадающем списке можно выбрать имя клиента, по которому мы хотим увидеть графики его выплат.

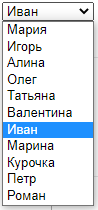


Рисунок 3.18-Выпадающий список

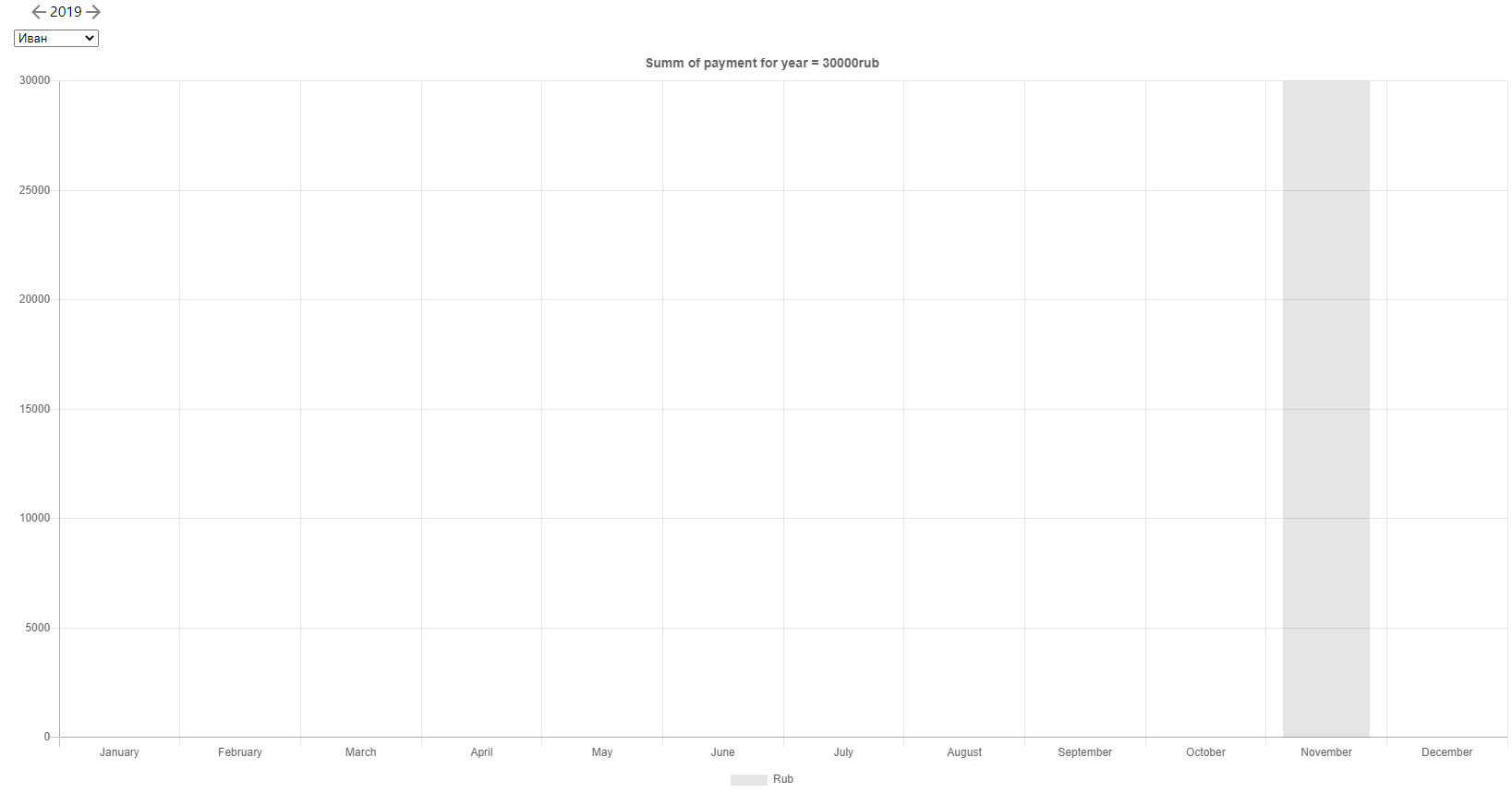


Рисунок 3.19 –График отчета по сделкам за 2019 год

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта по заданию «Бюро по трудоустройству» 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.

Также, было спроектировано приложение-ассистент для работы с базой данных. В приложении были реализованы функции просмотра, добавления, изменения и удаления записей базы данных, поиска записей, фильтрация для записей и экспорт таблиц.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Разработка приложений баз данных: лабораторный практикум / Э.Г. Галиаскаров и др.; Иван. гос. хим.-технол. ун-т. – Иваново, 2015. – 112 с.
2. 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.
3. Учебный сайт <https://metanit.com/sql/postgresql/>
4. Учебник <http://www.sql-tutorial.ru/>
5. Татьяна Карпова, курс «Базы данных: модели, разработка, реализация» <https://www.intuit.ru/studies/courses/1001/297/info> Сайт «НОУ ИНТУИТ»
6. Проектирование баз данных: лабораторный практикум / Э.Г. Галиаскаров, А.Ю. Крылов; Иван. гос. хим.-технол. ун-т.- Иваново, 2012.- 96 с.
7. Упражнения по SQL<http://sql-ex.ru/>

# ПРИЛОЖЕНИЕ А. СКРИПТ БАЗЫ ДАННЫХ

CREATE TABLE [Client]

(

[ClientID] int NOT NULL IDENTITY ( 1,1 ) ,

[Title] varchar(50) NULL ,

[Requisites] varchar(50) NULL ,

[Address] varchar(50) NULL ,

[Telephone] varchar(20) NOT NULL ,

[ContactPerson] varchar(50) NOT NULL

)

ALTER TABLE [Client]

ADD CONSTRAINT [XPKClient] PRIMARY KEY CLUSTERED ([ClientID] ASC)

CREATE TABLE [Contract]

(

[ContractID] int NOT NULL IDENTITY ( 1,1 ) ,

[ClientID] int NOT NULL ,

[BeginningDate] datetime NOT NULL ,

[EndDate] datetime NULL

)

ALTER TABLE [Contract]

ADD CONSTRAINT [XPKContract] PRIMARY KEY CLUSTERED ([ContractID] ASC)

CREATE TABLE [Outlets]

(

[OutletID] int NOT NULL IDENTITY ( 1,1 ) ,

[Floor] int NULL ,

[Area] int NOT NULL ,

[WithConditioners] bit NULL ,

[RentalPrice] int NOT NULL

)

ALTER TABLE [Outlets]

ADD CONSTRAINT [XPKOutlets] PRIMARY KEY CLUSTERED ([OutletID] ASC)

CREATE TABLE [Payment]

(

[PaymentID] int NOT NULL IDENTITY ( 1,1 ) ,

[PaymentDate] datetime NOT NULL ,

[SummPayment] int NOT NULL ,

[ContractID] int NOT NULL

)

ALTER TABLE [Payment]

ADD CONSTRAINT [XPKPayment] PRIMARY KEY CLUSTERED ([PaymentID] ASC)

CREATE TABLE [Rent]

(

[RentalID] int NOT NULL IDENTITY ( 1,1 ) ,

[OutletID] int NOT NULL ,

[ContractID] int NOT NULL

)

ALTER TABLE [Rent]

ADD CONSTRAINT [XPKRent] PRIMARY KEY CLUSTERED ([RentalID] ASC)

CREATE VIEW [CompanyInformation]([Title],[Address],[Telephone])

AS

SELECT [Title],[Address],[Telephone]

FROM [Client]

ALTER TABLE [Contract]

ADD CONSTRAINT [R\_4] FOREIGN KEY ([ClientID]) REFERENCES [Client]([ClientID])

ALTER TABLE [Payment]

ADD CONSTRAINT [R\_5] FOREIGN KEY ([ContractID]) REFERENCES [Contract]([ContractID])

ALTER TABLE [Rent]

ADD CONSTRAINT [R\_6] FOREIGN KEY ([ContractID]) REFERENCES [Contract]([ContractID])

ALTER TABLE [Rent]

ADD CONSTRAINT [R\_7] FOREIGN KEY ([OutletID]) REFERENCES [Outlets]([OutletID])

CREATE TRIGGER tD\_Client ON Client FOR DELETE AS

/\* erwin Builtin Trigger \*/

/\* DELETE trigger on Client \*/

BEGIN

DECLARE @errno int,

@severity int,

@state int,

@errmsg varchar(255)

/\* erwin Builtin Trigger \*/

/\* Client Contract on parent delete restrict \*/

/\* ERWIN\_RELATION:CHECKSUM="00010398", PARENT\_OWNER="", PARENT\_TABLE="Client"

CHILD\_OWNER="", CHILD\_TABLE="Contract"

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

FK\_CONSTRAINT="R\_4", FK\_COLUMNS="ClientID" \*/

IF EXISTS (

SELECT \* FROM deleted,Contract

WHERE

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

Contract.ClientID = deleted.ClientID

)

BEGIN

SELECT @errno = 30001,

@errmsg = 'Cannot delete Client because Contract exists.'

GOTO error

END

CREATE TRIGGER tU\_Client ON Client FOR UPDATE AS

/\* erwin Builtin Trigger \*/

/\* UPDATE trigger on Client \*/

BEGIN

DECLARE @numrows int,

@nullcnt int,

@validcnt int,

@insClientID int,

@errno int,

@severity int,

@state int,

@errmsg varchar(255)

SELECT @numrows = @@rowcount

/\* erwin Builtin Trigger \*/

/\* Client Contract on parent update restrict \*/

/\* ERWIN\_RELATION:CHECKSUM="000126d4", PARENT\_OWNER="", PARENT\_TABLE="Client"

CHILD\_OWNER="", CHILD\_TABLE="Contract"

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

FK\_CONSTRAINT="R\_4", FK\_COLUMNS="ClientID" \*/

IF

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

UPDATE(ClientID)

BEGIN

IF EXISTS (

SELECT \* FROM deleted,Contract

WHERE

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

Contract.ClientID = deleted.ClientID

)

BEGIN

SELECT @errno = 30005,

@errmsg = 'Cannot update Client because Contract exists.'

GOTO error

END

END

CREATE TRIGGER tD\_Contract ON Contract FOR DELETE AS

/\* erwin Builtin Trigger \*/

/\* DELETE trigger on Contract \*/

BEGIN

DECLARE @errno int,

@severity int,

@state int,

@errmsg varchar(255)

/\* erwin Builtin Trigger \*/

/\* Contract Rent on parent delete restrict \*/

/\* ERWIN\_RELATION:CHECKSUM="0001e4a2", PARENT\_OWNER="", PARENT\_TABLE="Contract"

CHILD\_OWNER="", CHILD\_TABLE="Rent"

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

FK\_CONSTRAINT="R\_6", FK\_COLUMNS="ContractID" \*/

IF EXISTS (

SELECT \* FROM deleted,Rent

WHERE

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

Rent.ContractID = deleted.ContractID

)

BEGIN

SELECT @errno = 30001,

@errmsg = 'Cannot delete Contract because Rent exists.'

GOTO error

END

/\* erwin Builtin Trigger \*/

/\* Contract Payment on parent delete restrict \*/

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

CHILD\_OWNER="", CHILD\_TABLE="Payment"

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

FK\_CONSTRAINT="R\_5", FK\_COLUMNS="ContractID" \*/

IF EXISTS (

SELECT \* FROM deleted,Payment

WHERE

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

Payment.ContractID = deleted.ContractID

)

BEGIN

SELECT @errno = 30001,

@errmsg = 'Cannot delete Contract because Payment exists.'

GOTO error

END

CREATE TRIGGER tI\_Contract ON Contract FOR INSERT AS

/\* erwin Builtin Trigger \*/

/\* INSERT trigger on Contract \*/

BEGIN

DECLARE @numrows int,

@nullcnt int,

@validcnt int,

@errno int,

@severity int,

@state int,

@errmsg varchar(255)

SELECT @numrows = @@rowcount

/\* erwin Builtin Trigger \*/

/\* Client Contract on child insert restrict \*/

/\* ERWIN\_RELATION:CHECKSUM="0001643e", PARENT\_OWNER="", PARENT\_TABLE="Client"

CHILD\_OWNER="", CHILD\_TABLE="Contract"

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

FK\_CONSTRAINT="R\_4", FK\_COLUMNS="ClientID" \*/

IF

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

UPDATE(ClientID)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,Client

WHERE

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

inserted.ClientID = Client.ClientID

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

IF @validcnt + @nullcnt != @numrows

BEGIN

SELECT @errno = 30002,

@errmsg = 'Cannot insert Contract because Client does not exist.'

GOTO error

END

END

CREATE TRIGGER tU\_Contract ON Contract FOR UPDATE AS

/\* erwin Builtin Trigger \*/

/\* UPDATE trigger on Contract \*/

BEGIN

DECLARE @numrows int,

@nullcnt int,

@validcnt int,

@insContractID int,

@errno int,

@severity int,

@state int,

@errmsg varchar(255)

SELECT @numrows = @@rowcount

/\* erwin Builtin Trigger \*/

/\* Contract Rent on parent update restrict \*/

/\* ERWIN\_RELATION:CHECKSUM="000365d4", PARENT\_OWNER="", PARENT\_TABLE="Contract"

CHILD\_OWNER="", CHILD\_TABLE="Rent"

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

FK\_CONSTRAINT="R\_6", FK\_COLUMNS="ContractID" \*/

IF

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

UPDATE(ContractID)

BEGIN

IF EXISTS (

SELECT \* FROM deleted,Rent

WHERE

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

Rent.ContractID = deleted.ContractID

)

BEGIN

SELECT @errno = 30005,

@errmsg = 'Cannot update Contract because Rent exists.'

GOTO error

END

END

/\* erwin Builtin Trigger \*/

/\* Contract Payment on parent update restrict \*/

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

CHILD\_OWNER="", CHILD\_TABLE="Payment"

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

FK\_CONSTRAINT="R\_5", FK\_COLUMNS="ContractID" \*/

IF

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

UPDATE(ContractID)

BEGIN

IF EXISTS (

SELECT \* FROM deleted,Payment

WHERE

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

Payment.ContractID = deleted.ContractID

)

BEGIN

SELECT @errno = 30005,

@errmsg = 'Cannot update Contract because Payment exists.'

GOTO error

END

END

/\* erwin Builtin Trigger \*/

/\* Client Contract on child update restrict \*/

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

CHILD\_OWNER="", CHILD\_TABLE="Contract"

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

FK\_CONSTRAINT="R\_4", FK\_COLUMNS="ClientID" \*/

IF

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

UPDATE(ClientID)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,Client

WHERE

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

inserted.ClientID = Client.ClientID

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

IF @validcnt + @nullcnt != @numrows

BEGIN

SELECT @errno = 30007,

@errmsg = 'Cannot update Contract because Client does not exist.'

GOTO error

END

END

CREATE TRIGGER tD\_Outlets ON Outlets FOR DELETE AS

/\* erwin Builtin Trigger \*/

/\* DELETE trigger on Outlets \*/

BEGIN

DECLARE @errno int,

@severity int,

@state int,

@errmsg varchar(255)

/\* erwin Builtin Trigger \*/

/\* Outlets Rent on parent delete restrict \*/

/\* ERWIN\_RELATION:CHECKSUM="0001030d", PARENT\_OWNER="", PARENT\_TABLE="Outlets"

CHILD\_OWNER="", CHILD\_TABLE="Rent"

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

FK\_CONSTRAINT="R\_7", FK\_COLUMNS="OutletID" \*/

IF EXISTS (

SELECT \* FROM deleted,Rent

WHERE

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

Rent.OutletID = deleted.OutletID

)

BEGIN

SELECT @errno = 30001,

@errmsg = 'Cannot delete Outlets because Rent exists.'

GOTO error

END

CREATE TRIGGER tU\_Outlets ON Outlets FOR UPDATE AS

/\* erwin Builtin Trigger \*/

/\* UPDATE trigger on Outlets \*/

BEGIN

DECLARE @numrows int,

@nullcnt int,

@validcnt int,

@insOutletID int,

@errno int,

@severity int,

@state int,

@errmsg varchar(255)

SELECT @numrows = @@rowcount

/\* erwin Builtin Trigger \*/

/\* Outlets Rent on parent update restrict \*/

/\* ERWIN\_RELATION:CHECKSUM="00011f71", PARENT\_OWNER="", PARENT\_TABLE="Outlets"

CHILD\_OWNER="", CHILD\_TABLE="Rent"

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

FK\_CONSTRAINT="R\_7", FK\_COLUMNS="OutletID" \*/

IF

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

UPDATE(OutletID)

BEGIN

IF EXISTS (

SELECT \* FROM deleted,Rent

WHERE

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

Rent.OutletID = deleted.OutletID

)

BEGIN

SELECT @errno = 30005,

@errmsg = 'Cannot update Outlets because Rent exists.'

GOTO error

END

END

CREATE TRIGGER tI\_Payment ON Payment FOR INSERT AS

/\* erwin Builtin Trigger \*/

/\* INSERT trigger on Payment \*/

BEGIN

DECLARE @numrows int,

@nullcnt int,

@validcnt int,

@errno int,

@severity int,

@state int,

@errmsg varchar(255)

SELECT @numrows = @@rowcount

/\* erwin Builtin Trigger \*/

/\* Contract Payment on child insert restrict \*/

/\* ERWIN\_RELATION:CHECKSUM="0001565e", PARENT\_OWNER="", PARENT\_TABLE="Contract"

CHILD\_OWNER="", CHILD\_TABLE="Payment"

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

FK\_CONSTRAINT="R\_5", FK\_COLUMNS="ContractID" \*/

IF

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

UPDATE(ContractID)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,Contract

WHERE

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

inserted.ContractID = Contract.ContractID

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

IF @validcnt + @nullcnt != @numrows

BEGIN

SELECT @errno = 30002,

@errmsg = 'Cannot insert Payment because Contract does not exist.'

GOTO error

END

END

CREATE TRIGGER tU\_Payment ON Payment FOR UPDATE AS

/\* erwin Builtin Trigger \*/

/\* UPDATE trigger on Payment \*/

BEGIN

DECLARE @numrows int,

@nullcnt int,

@validcnt int,

@insPaymentID int,

@errno int,

@severity int,

@state int,

@errmsg varchar(255)

SELECT @numrows = @@rowcount

/\* erwin Builtin Trigger \*/

/\* Contract Payment on child update restrict \*/

/\* ERWIN\_RELATION:CHECKSUM="000159ac", PARENT\_OWNER="", PARENT\_TABLE="Contract"

CHILD\_OWNER="", CHILD\_TABLE="Payment"

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

FK\_CONSTRAINT="R\_5", FK\_COLUMNS="ContractID" \*/

IF

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

UPDATE(ContractID)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,Contract

WHERE

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

inserted.ContractID = Contract.ContractID

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

IF @validcnt + @nullcnt != @numrows

BEGIN

SELECT @errno = 30007,

@errmsg = 'Cannot update Payment because Contract does not exist.'

GOTO error

END

END

CREATE TRIGGER tI\_Rent ON Rent FOR INSERT AS

/\* erwin Builtin Trigger \*/

/\* INSERT trigger on Rent \*/

BEGIN

DECLARE @numrows int,

@nullcnt int,

@validcnt int,

@errno int,

@severity int,

@state int,

@errmsg varchar(255)

SELECT @numrows = @@rowcount

/\* erwin Builtin Trigger \*/

/\* Outlets Rent on child insert restrict \*/

/\* ERWIN\_RELATION:CHECKSUM="000289a8", PARENT\_OWNER="", PARENT\_TABLE="Outlets"

CHILD\_OWNER="", CHILD\_TABLE="Rent"

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

FK\_CONSTRAINT="R\_7", FK\_COLUMNS="OutletID" \*/

IF

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

UPDATE(OutletID)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,Outlets

WHERE

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

inserted.OutletID = Outlets.OutletID

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

IF @validcnt + @nullcnt != @numrows

BEGIN

SELECT @errno = 30002,

@errmsg = 'Cannot insert Rent because Outlets does not exist.'

GOTO error

END

END

/\* erwin Builtin Trigger \*/

/\* Contract Rent on child insert restrict \*/

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

CHILD\_OWNER="", CHILD\_TABLE="Rent"

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

FK\_CONSTRAINT="R\_6", FK\_COLUMNS="ContractID" \*/

IF

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

UPDATE(ContractID)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,Contract

WHERE

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

inserted.ContractID = Contract.ContractID

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

IF @validcnt + @nullcnt != @numrows

BEGIN

SELECT @errno = 30002,

@errmsg = 'Cannot insert Rent because Contract does not exist.'

GOTO error

END

END

CREATE TRIGGER tU\_Rent ON Rent FOR UPDATE AS

/\* erwin Builtin Trigger \*/

/\* UPDATE trigger on Rent \*/

BEGIN

DECLARE @numrows int,

@nullcnt int,

@validcnt int,

@insRentalID int,

@errno int,

@severity int,

@state int,

@errmsg varchar(255)

SELECT @numrows = @@rowcount

/\* erwin Builtin Trigger \*/

/\* Outlets Rent on child update restrict \*/

/\* ERWIN\_RELATION:CHECKSUM="0002a2bc", PARENT\_OWNER="", PARENT\_TABLE="Outlets"

CHILD\_OWNER="", CHILD\_TABLE="Rent"

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

FK\_CONSTRAINT="R\_7", FK\_COLUMNS="OutletID" \*/

IF

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

UPDATE(OutletID)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,Outlets

WHERE

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

inserted.OutletID = Outlets.OutletID

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

IF @validcnt + @nullcnt != @numrows

BEGIN

SELECT @errno = 30007,

@errmsg = 'Cannot update Rent because Outlets does not exist.'

GOTO error

END

END

/\* erwin Builtin Trigger \*/

/\* Contract Rent on child update restrict \*/

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

CHILD\_OWNER="", CHILD\_TABLE="Rent"

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

FK\_CONSTRAINT="R\_6", FK\_COLUMNS="ContractID" \*/

IF

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

UPDATE(ContractID)

BEGIN

SELECT @nullcnt = 0

SELECT @validcnt = count(\*)

FROM inserted,Contract

WHERE

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

inserted.ContractID = Contract.ContractID

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

IF @validcnt + @nullcnt != @numrows

BEGIN

SELECT @errno = 30007,

@errmsg = 'Cannot update Rent because Contract does not exist.'

GOTO error

END

END

# ПРИЛОЖЕНИЕ Б. ПЕРЕЧЕНЬ ФАЙЛОВ

1. Пояснительная записка к курсовому проекту в папке doc Лебедев 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 Rent.mdf
5. Папки с исходным кодом приложения Server и Client