

**Системы управления базами данных: учебно-методическое пособие  
(электронное издание)**

**Автор/составитель:**

**О.И. Евдошенко**  
доцент кафедры САПРиМ  
АГАСУ

**Астрахань, 2020**

## Содержание

Введение .....	3
1. Теоретический материал.....	4
1.1. Введение в базы данных. Основные понятия и определения .....	4
1.2. Операции реляционной алгебры .....	9
1.4. СУБД PostgreSQL. Язык SQL .....	22
1.4.1. Определение данных .....	24
1.4.2. Модификация данных .....	27
1.4.3. Запросы.....	28
2. Практические задания.....	32
2.1. Создание базы данных в СУБД PostgreSQL .....	32
2.2. Выборка данных из базы данных посредством SQL-запросов.....	38
2.3. Использование агрегатных функций в SQL-запросах .....	42
2.4. Группировка данных посредством SQL-запросов .....	45
2.5. Создание представлений.....	50
2.6. Создание функций.....	52
2.7. Подзапросы. Предложение HAVING, CASE в SQL-запросах .....	56
Список литературы.....	59

## Введение

Базы данных играют огромную роль в современном информационном мире. Умение грамотно проектировать и разрабатывать базы данных является одним из ключевых требований к специалисту в области информационных технологий. Согласно профессиональным стандартам, такой специалист должен уметь обеспечивать функционирование баз данных (БД) и управлять развитием этих БД (профессиональный стандарт 06.011 «Администратор баз данных», утвержден Министерством труда и социальной защиты).

В данном учебно-методическом пособии дается общее представление о базах данных, моделях организации данных (сетевая, иерархическая, реляционная, объектно-ориентированная). Приведены описания основных и вспомогательных операций реляционной алгебры, которые являются инструментом для манипулирования табличными данными в реляционных базах данных, по каждой операции автор приводит иллюстрированные примеры. Описывается терминология нормализации баз данных для начинающих, основные сведения о которой полезны при обсуждении структуры реляционной базы данных.

Пособие включает практическую часть, при выполнении заданий которой студент овладеет фундаментальными навыками работы в реляционной СУБД PostgreSQL и познакомится с языком SQL, являющимся фактическим стандартом работы с реляционными базами данных. Выполнение заданий позволит студенту закрепить полученные теоретические знания и получить практические навыки работы с основными конструкциями языка SQL для определения и манипулирования данными.

В первой главе учебно-методического пособия приведен теоретический материал. Глава состоит из трех параграфов: «Введение в базы данных. Основные понятия и определения», «Операции реляционной алгебры» и «Введение в нормализацию отношений».

Во второй главе приведены практические задания. Каждый параграф представляет собой лабораторную работу, которую студенту необходимо выполнить в СУБД PostgreSQL. Каждая работа включает в себя текст задания и образец его выполнения.

Данное учебно-методическое пособие предназначено для студентов, обучающихся по направлениям подготовки, входящих в укрупненную группу 09.00.00 «Информатика и вычислительная техника». Кроме того, данное пособие может быть интересно для студентов других направлений подготовки, желающих самостоятельно получить практический опыт разработки реляционных баз данных.

# 1. Теоретический материал

## 1.1. Введение в базы данных. Основные понятия и определения

**База данных (БД)** — совокупность взаимосвязанных хранящихся вместе с отношениями между ними устойчивых данных при наличии такой минимальной избыточности, которая допускает их независимое использование оптимальным образом для одного или нескольких приложений.

**База данных** – это совокупность взаимосвязанных структурированных данных, относящихся к определенной *предметной области* и организованных так, чтобы обеспечить независимость данных от программ обработки. Фактически база данных – это *модель предметной области* (ПО).

Под **предметной областью (ПО)** принято понимать некоторую область человеческой деятельности или область реального мира, подлежащих изучению для организации управления и автоматизации, например, отдел кадров, вуз и.т.д. В каждой предметной области можно выделить сущности. **Сущность** – это реальных процесс или объект предметной области (см. рисунок 1.1).

Сущности бывают *базовые* и *зависимые*.

Для каждого типа сущности необходимо определить имя.

Атрибуты: характеристики сущностей. Атрибуты бывают:

1. *Идентифицирующие и описательные атрибуты.*
2. *Составные и простые атрибуты.*
3. *Однозначные и многозначные атрибуты.*
4. *Обязательные и необязательные.*

Для каждого атрибута необходимо определить название, указать тип данных и описать ограничения целостности – множество значений, которые может принимать данный атрибут.

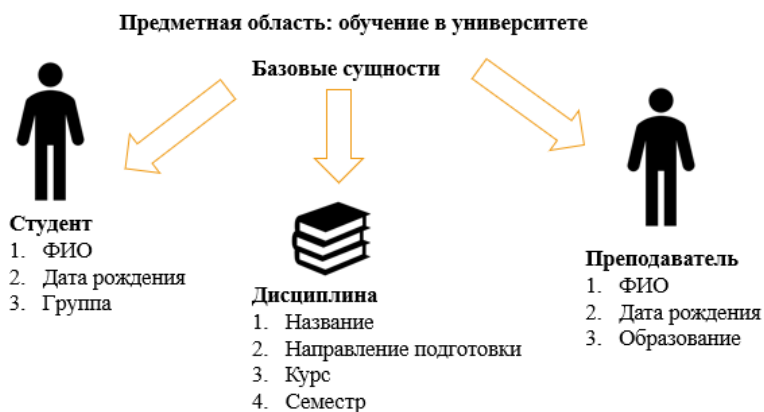


Рисунок 1.1 – Сущности и атрибуты сущностей предметной области университет

Между сущностями любой предметной области имеются связи.

**Связь** – это осмысленная ассоциация между сущностями. Для связи указывается название, вид (факультативная или обязательная), степень (1:1, 1:M или M:M) и кардинальность (унарная, бинарная, тернарная или n-арная).

**Кардинальность** - определяется количеством сущностей, которые охвачены данной связью (см. рисунок 1.2).

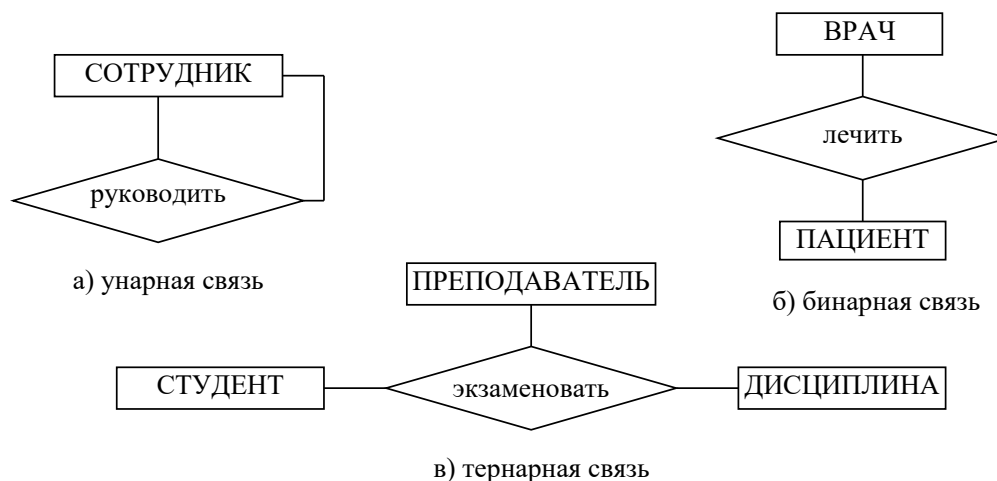


Рисунок 1.2 – Определение кардинальности

Если объект одного типа оказывается по необходимости связанным с объектом другого типа, то между этими типами объектов существует **обязательная связь** (обозначается двойной линией). Иначе связь является **факультативной**.

Для следующих требований связи между сущностями предметной области будут выглядеть как на рисунке 1.3:

1. С сотрудником может быть заключен только один договор
2. Сотрудник может работать только в одном отделе, при этом в отделе может работать много сотрудников.
3. Сотрудник может участвовать в нескольких проектах, в одном проекте может участвовать несколько сотрудников



Рисунок 1.3 – Связи между сущностями (предметная область - проектная деятельность)

Для предметной области: университет (см. рисунок 1.4). Базовые сущности: студент, дисциплина, преподаватель. Зависимые сущности: урок.



Рисунок 1.4 – Связи между сущностями (предметная область – университет)

Основополагающими понятиями в концепции баз данных являются обобщенные категории «данные» и «модель данных».

Понятие «**данные**» в концепции баз данных — это набор конкретных значений, параметров, характеризующих объект, условие, ситуацию или любые другие факторы. Для обработки данных они должны иметь определенную структуру. Поэтому центральным понятием в области баз данных является понятие модели.

**Модель данных** — это некоторая абстракция, которая, будучи приложима к конкретным данным, позволяет пользователям и разработчикам трактовать их уже как информацию, то есть сведения, содержащие не только данные, но и взаимосвязь между ними.

С помощью модели данных могут быть представлены объекты предметной области и взаимосвязи между ними. В зависимости от вида организации данных различают следующие важнейшие модели БД:

- иерархическую
- сетевую
- реляционную
- объектно-ориентированную

В **иерархической БД** данные представляются в виде древовидной структуры. Подобная структура БД удобна для работы с данными, упорядоченными иерархически. При

оперировании данными со сложными логическими связями иерархическая модель оказывается слишком громоздкой. В данной модели используются специальные термины (см. таблицу 1.1). Пример иерархической модели представлен на рисунке 1.5.

Таблица 1.1 - Термины иерархической модели

<b>Атрибут</b> (элемент данных, поле)	наименьшая единица структуры данных. Обычно каждому элементу при описании базы данных присваивается уникальное имя. По этому имени к нему обращаются при обработке. Элемент данных также часто называют полем.
<b>Запись</b> (сегмент)	именованная совокупность атрибутов. Использование записей позволяет за одно обращение к базе получить некоторую логически связанную совокупность данных. Именно записи изменяются, добавляются и удаляются. Тип записи определяется составом ее атрибутов. <i>Экземпляр записи</i> - конкретная запись с конкретным значением элементов.
<b>Групповое отношение</b>	<i>иерархическое отношение</i> между записями двух типов. Родительская запись (владелец группового отношения) называется исходной записью, а дочерние записи (члены группового отношения) - подчиненными. Иерархическая база данных может хранить только такие древовидные структуры.

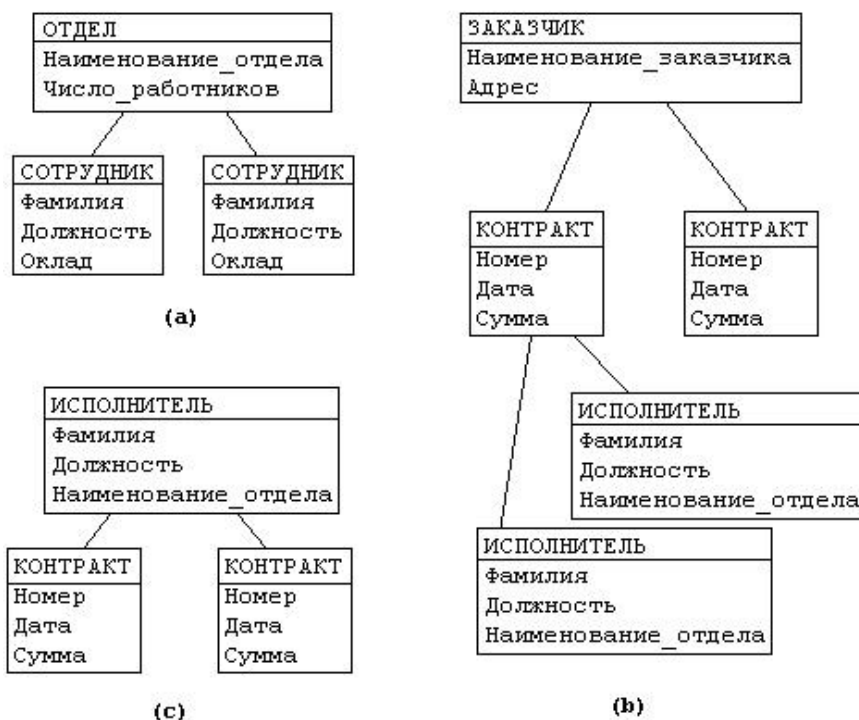


Рисунок 1.5 – Иерархическая модель данных

В **сетевой БД** данные организуются в виде графа. Недостатком сетевой структуры является жесткость структуры и сложность ее организации. Пример сетевой модели представлен на рисунке 1.6.

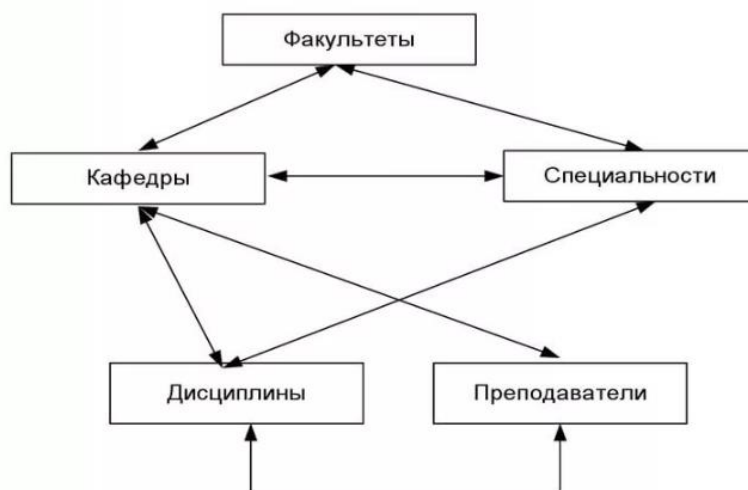


Рисунок 1.6 – Сетевая модель данных

**Реляционная БД (РБД)** получила свое название от английского термина relation (отношение). Была предложена в 70-м году сотрудником фирмы IBM Эдгаром Коддом. Реляционная БД представляет собой совокупность таблиц, связанных отношениями. Достоинствами реляционной модели данных являются простота, гибкость структуры. Кроме того, ее удобно реализовывать на компьютере. Большинство современных БД для персональных компьютеров являются реляционными. В данной модели используются специальные термины (см. таблицу 1.2). Пример таблицы реляционной БД представлен на рисунке 1.7.

Таблица 1.2 – Термины реляционной модели

Домен	Совокупность допустимых значений (тип данных)
Кортеж	Строка таблицы
Кардинальность	Количество строк
Атрибут	Поле, столбец таблицы
Степень отношения	Количество полей (столбцов)
Первичный ключ (FK)	Уникальный ключ; атрибут, который идентифицирует кортеж
Внешний ключ (PK)	Служит для организации связи между таблицами
Схема отношения	Строка заголовков столбцов таблицы



Целое	Строка	Строка	Целое		Типы данных
Номер	Имя	Должность	Деньги		Домены
Табельный номер	Имя	Должность	Оклад	Премия	Атрибуты
2934	Иванов	Инженер	1120	400	Кортежи
2935	Петров	Вед. инженер	1440	500	
2936	Климов	Бухгалтер	920	350	

Рисунок 1.7 – Пример таблицы реляционной базы данных

**Объектно-ориентированные БД (ООБД)** объединяют сетевую и реляционную модели и используются для создания крупных БД с данными сложной структуры. В данной модели используются специальные термины (см. таблицу 1.3).

Таблица 1.3 – Термины объектно-ориентированной модели

ООБД	РБД
Класс	Отношение
Экземпляр класса (объект)	Кортеж
Атрибут класса	Столбец (атрибут)
Иерархия классов	Иерархия отношений
Подкласс	Отношения "потомок"
Суперкласс	Отношения "предок"
Методы	Правила преобразования данных

## 1.2. Операции реляционной алгебры

**Реляционная алгебра (РА)** базируется на теории множеств и является основой логики работы баз данных. Любой запрос к БД можно записать в виде некоторой формулы РА или некоторого выражения.

Операции РА применяются к отношениям и в результате применения операций РА получаются отношения (таблицы). Различают унарные и бинарные операции РА: унарные применяются к одному отношению (таблице), бинарные – к двум.

**Существует пять основных операций РА:**

- ✓ проекция;
- ✓ селекция;
- ✓ декартово произведение;
- ✓ объединение;
- ✓ разность;

**и три вспомогательных операции РА, которые могут быть выражены через основные:**

- ✓ пересечение;
- ✓ соединение;
- ✓ деление.

### **Проекция ( $\pi$ ).**

Это унарная операция (выполняемая над одним отношением), служащая для выбора подмножества атрибутов из отношения R. Она уменьшает степень отношения и может уменьшить кардинальность отношения за счёт исключения одинаковых кортежей (см. рисунок 1.8 и рисунок 1.9).



Рисунок 1.8 – Выполнение проекции с отношением «Товары»

**SQL-запрос операции:** SELECT DISTINCT Товар, Категория FROM Товары

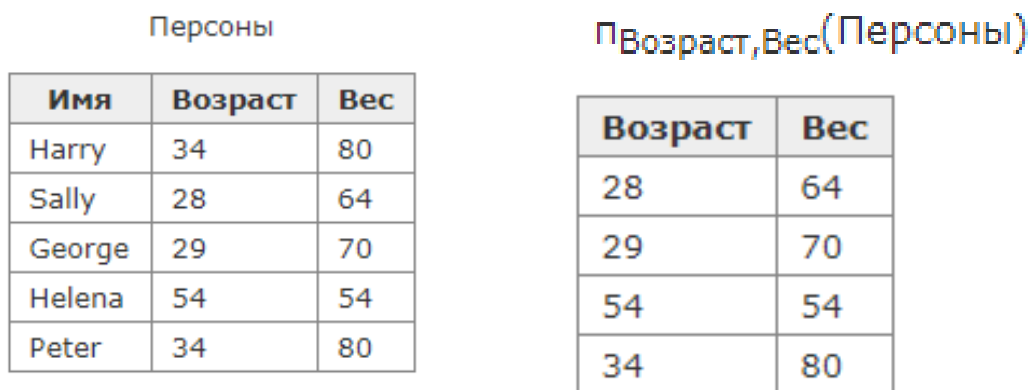



Рисунок 1.9 – Выполнение проекции с отношением «Персоны»

**Селекция, выборка (select,  $\sigma$ ).**

Это унарная операция, результатом которой является подмножество кортежей исходного отношения, соответствующих условиям, которые накладываются на значения определённых атрибутов (см. рисунок 1.10 и рисунок 1.11).

**Дата поступления = 10.02.2015**

Товар	Категория	Дата поступления
Монитор	ПК	10.02.2015
Кефир	Молочные продукты	11.02.2015
Кефир	Молочные продукты	12.02.2015
Монитор	ПК	10.02.2015



Товар	Категория	Дата поступления
Монитор	ПК	10.02.2015
Монитор	ПК	10.02.2015

Рисунок 1.10 – Выполнение селекции с отношением «Товары»

**SQL-запрос операции:**

SELECT \* FROM Товары WHERE Дата\_поступления = '10.02.2015'

Код	Название	Компания	Цена
1	Баклажаны	ООО «Томаты»	250
2	Кофе	ООО «Бразилия»	300
3	Грейфрут	ООО «Лето»	100
4	Капуста	ООО «Овощи»	130

$\sigma(\text{Цена} > 130)$  Продукты

Код	Название	Компания	Цена
1	Баклажаны	ООО «Томаты»	250
2	Кофе	ООО «Бразилия»	300

Рисунок 1.11 – Выполнение селекции товаров с ценой больше 130

**Декартово произведение (cartesian product).**

Это бинарная операция над разносхемными отношениями, соответствующая определению декартова произведения для РМД: в результате получается отношение, схема которого включает все атрибуты исходных отношений. Результирующее отношение содержит все возможные комбинации кортежей исходных отношений (см. рисунок 1.12).

Товар	Категория	Дата поступления	Название поставщика	Адрес
Монитор	ПК	10.02.2015	ООО «Грант»	г. Астрахань
Кефир	Молочные продукты	12.02.2015	ОАО «Молочник»	г. Волгоград

↓

Товар	Категория	Дата поступления	Название поставщика	Адрес
Монитор	ПК	10.02.2015	ООО «Грант»	г. Астрахань
Монитор	ПК	10.02.2015	ОАО «Молочник»	г. Волгоград
Кефир	Молочные продукты	12.02.2015	ООО «Грант»	г. Астрахань
Кефир	Молочные продукты	12.02.2015	ОАО «Молочник»	г. Волгоград

Рисунок 1.12 – Выполнение операции «Декартово произведение»

**SQL-запрос операции:** SELECT \* FROM Товары, Поставщики

### Объединение (union).

Объединением двух односхемных отношений R и S называется отношение, которое включает в себя все кортежи исходных отношений без повторов (см. рисунок 1.13).

Товар	Категория	Дата поступления
Клавиатура	ПК	08.02.2015
Кефир	Молочные продукты	12.02.2015

Товар	Категория	Дата поступления
Монитор	ПК	10.02.2015
Кефир	Молочные продукты	12.02.2015
Клавиатура	ПК	08.02.2015

Рисунок 1.13 – Выполнение операции «Объединение»

### SQL-запрос операции:

SELECT Товар, Категория, Дата поступления FROM Товары

*UNION*

SELECT Товар, Категория, Дата поступления FROM Товары на складе

### Разность (excerpt).

Разность возвращает отношение, содержащее все кортежи, которые принадлежат первому из заданных отношений и не принадлежат второму (см. рисунок 1.14).



Рисунок 1.14 – Выполнение операции «Разность»

#### SQL-запрос операции:

SELECT Товар, Категория, Дата поступления FROM Товары

*EXCEPT*

SELECT Товар, Категория, Дата поступления FROM Товары\_на\_складе

#### Пересечение (intersect).

Пересечение двух односхемных отношений R и S есть подмножество кортежей, принадлежащих обоим отношениям (см. рисунок 1.15).



Рисунок 1.15 – Выполнение операции «Пересечение»

#### SQL-запрос операции:

SELECT Товар, Категория, Дата поступления FROM Товары

*INTERSECT*

SELECT Товар, Категория, Дата поступления FROM Товары\_на\_складе

#### Соединение (join).

Эта операция определяет подмножество декартова произведения двух разносхемных отношений. Кортеж декартова произведения входит в результирующее отношение, если для атрибутов разных исходных отношений выполняется некоторое условие F.

Соединение по атрибуту «Отдел» первого и второго отношения (см. рисунок 1.16).

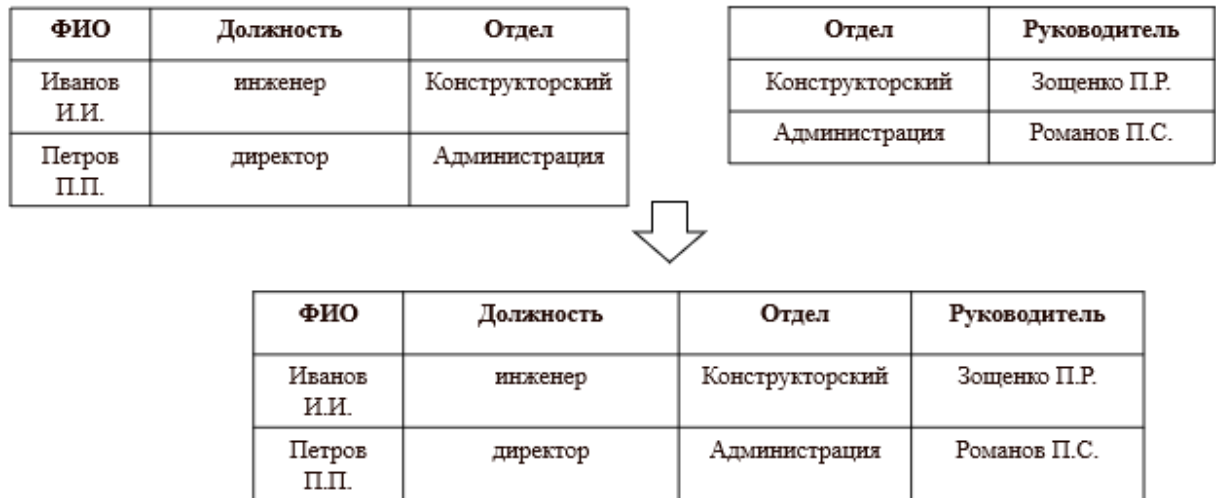


Рисунок 1.16 – Выполнение операции «Соединение»

#### SQL-запрос операции:

SELECT \* FROM Сотрудники, Отделы WHERE Сотрудники.Отдел=Отделы.Отдел

#### Деление

Из первого отношения извлекаются значения строк, для которых присутствуют все комбинации значений из второй таблицы (см. рисунок 1.17).

Отношение A

S	P
S1	P1
S1	P2
S1	P3
S1	P4
S2	P1
S2	P3
S3	P2
S3	P3

Отношение B

P
P1

Отношение B1

P
P2
P3

Отношение B2

P
P1
P2
P3

Результат:

A/B	A/B1	A/B2
S1	S1	S1
S2	S3	

Рисунок 1.17 – Выполнение операции «Деление»

Примеры реляционных выражений и их результатов.

### Пример 1:

$\pi_{\text{Компания}} \sigma_{\text{Цена} \leq 130} (\text{Продукты})$

Код	Название	Компания	Цена
1	Баклажаны	ООО «Томаты»	250
2	Кофе	ООО «Бразилия»	300
3	Грейфрут	ООО «Лето»	100
4	Капуста	ООО «Овощи»	130

Результат:

Компания
ООО «Лето»
ООО «Овощи»

### Пример 2:

$\pi_{(\text{Покупатель})} \sigma_{(\text{Продукты.Код} = \text{Покупатели.Код} \ \& \ \text{Цена} < 130)} (\text{Продукты} \times \text{Покупатели})$

Код	Название	Компания	Цена
1	Баклажаны	ООО «Томаты»	250
2	Кофе	ООО «Бразилия»	300
3	Грейфрут	ООО «Лето»	100
4	Капуста	ООО «Овощи»	130

Код	Покупатель
1	Пятерочка
3	Окей
2	Пятерочка
2	Магнит

Результат:

Покупатель
Окей

**SELECT** Покупатель  
**FROM** Продукты, Покупатели  
**WHERE** Продукты.Код=Покупатели.Код AND Цена<130

Пример 3:

**Отношение А**

<i>Табельный номер</i>	<b>Фамилия</b>	<b>Зарплата</b>
1	Иванов	1000
2	Петров	2000
3	Сидоров	3000

**Отношение В**

<i>Табельный номер</i>	<b>Фамилия</b>	<b>Зарплата</b>
1	Иванов	1000
2	Пушников	2500
4	Сидоров	3000

SELECT Табельный номер, Фамилия, Зарплата FROM А  
*EXCEPT*  
 SELECT Табельный номер, Фамилия, Зарплата FROM В

<i>Табельный номер</i>	<b>Фамилия</b>	<b>Зарплата</b>
2	Петров	2000
3	Сидоров	3000

SELECT Фамилия, Зарплата FROM А  
*EXCEPT*  
 SELECT Фамилия, Зарплата FROM В

<b>Фамилия</b>	<b>Зарплата</b>
Петров	2000

SELECT Табельный номер, Фамилия, Зарплата FROM А  
*INTERSECT*  
 SELECT Табельный номер, Фамилия, Зарплата FROM В

<b>Табельный номер</b>	<b>Фамилия</b>	<b>Зарплата</b>
1	Иванов	1000

SELECT Фамилия, Зарплата FROM А  
*INTERSECT*



SELECT Фамилия, Зарплата FROM B

Фамилия	Зарплата
Иванов	1000
Сидоров	3000

Задания для закрепления изученного материала:

Исходные отношения с данными

A.FIO	A.YEAR	A.JOB	A.CHAIR
Иванов И.И.	1960	Доцент	23
Козлов К.К.	1959	Доцент	25
Петров П.П.	1960	Ст.преп.	24
Лютикова Л.Л.	1977	Ассистент	24
Иванов И.И.	1960	Доцент	24

B.FIO	B.YEAR	B.JOB	B.CHAIR
Иванов И.И.	1960	Доцент	24
Сидоров С.С.	1953	Проф.	22
Николаев П.С.	1945	Проф.	22
Иванов И.И.	1960	Доцент	23
Козлов К.К.	1959	Доцент	25

C.JOB	C.Pay
Зав. каф	50000
Проф.	35000
Доцент	20000
Ст.преп.	15000
Ассистент	5000

Реляционные выражения:

- $\pi(\text{FIO}) \sigma(\text{A.JOB}=\text{C.JOB} \ \& \ \text{YEAR}=1960 \ \& \ \text{PAY}=20000) \ A \times C$
- $A \text{ MINUS } B$
- $A \text{ INTERSECT } B$
- $\pi(\text{FIO}) \sigma(\text{YEAR}=1959) \ A \text{ INTERSECT } B$

5.  $\pi(\text{FIO}) \sigma(\text{D.JOB}=\text{C.JOB} \ \& \ \text{YEAR}=1977 \ \& \ \text{PAY}=5000) \ \text{D}(\text{A MINUS B}) \times \text{C}$
6.  $\pi(\text{A.FIO}, \text{A.JOB}) \sigma(\text{A.JOB}=\text{C.JOB} \ \& \ \text{PAY}=20000) \ \text{MINUS} \ \pi(\text{A.FIO}, \text{A.JOB}) \sigma(\text{A.CHAIR}=25)$

### 1.3. Нормализация отношений

Различают следующие проблемы в базах данных:

1. избыточность данных;
2. аномалии обновления;
3. аномалии удаления;
4. аномалии ввода.

**Избыточность данных** характеризуется наличием в кортежах отношений повторяющейся информации. Многократное дублирование данных приводит к неоправданному увеличению занимаемого объема внешней памяти.

**Аномалии обновления**, прежде всего, связаны с избыточностью данных, что приводит к проблемам при их изменении. При изменении повторяющихся данных придется многократно изменять их значения, однако, если изменения будут внесены не во все кортежи, возникнет несоответствие информации, которое называется аномалией обновления.

**Аномалии удаления** могут возникать при удалении записей из ненормализованных таблиц и характеризуются вероятностью удаления не всех дублированных кортежей.

**Аномалии ввода** возникают при добавлении в таблицу новых записей, обычно в поля с ограничениями NOT NULL (не пустые). Когда в отношение на данный момент времени невозможно ввести однозначную информацию.

Нормализация схемы отношения выполняется путём декомпозиции схемы. Декомпозиция отношения не должна приводить к потере зависимостей между атрибутами сущностей.

Для декомпозиции должна существовать операция реляционной алгебры, применение которой позволит восстановить исходное отношение.

#### Первая нормальная форма (1НФ).

Отношение приведено к 1НФ, если все его атрибуты простые и каждая ее строка содержит только одно значение для каждого атрибута (см. рисунок 1.18).



Рисунок 1.18 – Приведение к 1НФ

Приведем отношение СТУДЕНТ (№\_зачетки, Фамилия, Группа, Факультет, Семестр, Предмет, Преподаватель, Вид\_Работы, Оценка) к 3НФ. Данное отношение уже находится в 1НФ, так как каждый атрибут содержит одно значение.

№ зачетки	Семестр	Предмет	Фамилия	Группа	Факультет	Преподаватель	Вид_работы	Оценка
01	1	Химия	Панов	Г1	Ф1	Сомов	Экз	Отл
01	1	Физика	Панов	Г1	Ф1	Петров	Экз	Отл
01	1	История	Панов	Г1	Ф1	Львов	Экз	Отл
02	1	Химия	Туров	Г2	Ф1	Сомов	Экз	Хор
02	1	Физика	Туров	Г2	Ф1	Петров	Экз	Отл
02	1	История	Туров	Г2	Ф1	Львов	Экз	Хор

**Первичным ключом** в отношении СТУДЕНТ является группа атрибутов (составной ключ):

{№\_зачетки, Семестр, Предмет}.

### Вторая нормальная форма (2НФ).

Отношение находится во 2НФ, если оно приведено к 1НФ и каждый неключевой атрибут функционально полно зависит от составного ключа.

*Функциональная зависимость.* Поле В таблицы функционально зависит от поля А той же таблицы в том и только в том случае, когда в любой заданный момент времени для каждого из различных значений поля А обязательно существует только одно из различных значений поля В. Отметим, что здесь допускается, что поля А и В могут быть составными.

*Полная функциональная зависимость.* Поле В находится в полной функциональной зависимости от составного поля А, если оно функционально зависит от А и не зависит функционально от любого подмножества поля А.

Для того чтобы привести отношение ко 2НФ, нужно:

1. построить его проекцию, исключив атрибуты, которые не находятся в функционально полной зависимости от составного ключа;
2. построить дополнительные проекции на часть составного ключа и атрибуты, функционально зависящие от этой части ключа.

Для предметной области справедливы следующие функциональные зависимости:

$F_1 = \text{№\_зачетки} \rightarrow \text{Фамилия, Группа, Факультет}$

$F_2 = \text{№\_зачетки, Семестр, Предмет} \rightarrow \text{Преподаватель, Вид\_Работы, Оценка}$

$F_3 = \text{№\_зачетки, Семестр, Предмет} \rightarrow \text{Фамилия, Группа, Факультет}$

$F_4 = \text{№\_зачетки, Семестр, Предмет} \rightarrow \text{Оценка}$

$F_5 = \text{Предмет} \rightarrow \text{Преподаватель}$

$F_6 = \text{Семестр, Предмет} \rightarrow \text{Вид\_Работы}$

$F_7 = \text{Группа} \rightarrow \text{Факультет}.$

Функциональная зависимость  $F_3$  является неполной, т.к. набор атрибутов {Фамилия, Группа, Факультет}, в соответствии с  $F_1$ , функционально зависит от атрибута №\_зачетки, входящего в состав атрибутов левой части функциональной зависимости  $F_3$ .

Проекции в результате декомпозиции:

$R_1 (\text{PK}(\text{№\_зачетки}), \text{Фамилия, Группа, Факультет})$

$R_2 (\text{PK}(\text{№\_зачетки, Семестр, Предмет}), \text{Преподаватель, Вид\_Работы, Оценка}).$

Отношение R1

<b>№ зачетки (PK)</b>	<b>Фамилия</b>	<b>Группа</b>	<b>Факультет</b>
01	Панов	Г1	Ф1
02	Туров	Г2	Ф1

Отношение R2

№ зачетки	Семестр	Предмет	Преподаватель	Вид работы	Оценка
РК					
01	1	Химия	Сомов	Экз	Отл
01	1	Физика	Петров	Экз	Отл

01	1	История	Львов	Экз	Отл
02	1	Химия	Сомов	Экз	Хор
02	1	Физика	Петров	Экз	Отл
02	1	История	Львов	Экз	Хор

Отношение R2 не находится во 2НФ:

1. неполная функциональной зависимости непервичного атрибута *Преподаватель* от ключа отношения
2. неполная функциональной зависимости атрибута Вид\_Работы от ключа отношения

Проекции в результате декомпозиции отношения R2:

R3 (PK(№\_зачетки, Семестр, Предмет), Оценка)

R4 (PK(Предмет), Преподаватель)

R5 (PK(Семестр, Предмет), Вид\_Работы)

### Третья нормальная форма (3НФ).

Отношение находится в 3НФ тогда и только тогда, когда выполняются следующие условия (Кодд): отношение находится во второй нормальной форме и ни один неключевой атрибут не находится в транзитивной функциональной зависимости от потенциального ключа.

Определение 3НФ запрещает наличие транзитивных зависимостей между непервичными атрибутами, поэтому анализу подлежит только отношение

R1 (PK(№\_зачетки), Фамилия, Группа, Факультет), в котором присутствует несколько непервичных атрибутов.

В результате получатся две проекции:

R6 (PK(№\_зачетки), Фамилия, Группа)

R7 (PK(Группа), Факультет).

Результат приведения к 3НФ отношения «Студент»:

1. R7 (№\_зачетки, Фамилия, *Группа(FK)*)

2. R6 (Группа, Факультет)
3. R3 (№ зачетки, Семестр, Предмет, Оценка)
4. R4 (Предмет, Преподаватель)
5. R5 (Семестр, Предмет, Вид\_Работы)

#### Первая нормальная форма

1. Каждая строка содержит данные, относящиеся к одному объекту или его части
2. Каждый столбец должен иметь уникальное имя
3. Каждый столбец содержит данные одного атрибута объекта. Все элементы столбца должны быть одного типа
4. Две строки таблицы не могут быть идентичны
5. Порядок строк и столбцов в отношении не имеет значения

#### Вторая нормальная форма

1. Таблица должна быть в первой нормальной форме
2. Все неключевые атрибуты должны зависеть от всех ключевых атрибутов

#### Третья нормальная форма

1. Таблица должна быть во второй нормальной форме
2. Таблица не должна иметь неключевые атрибуты, находящиеся в транзитивной зависимости от первичного ключа

### **1.4. СУБД PostgreSQL. Язык SQL**

**PostgreSQL** — это *реляционная система управления базами данных* (РСУБД). Это означает, что это система управления данными, представленными в виде *отношений* (relation). Отношение — это математически точное обозначение *таблицы*. Хранение данных в таблицах так распространено сегодня, что это кажется самым очевидным вариантом, хотя есть множество других способов организации баз данных.

**PostgreSQL** — СУБД с открытым исходным кодом, основой которого был код, написанный в Беркли. Она поддерживает большую часть стандарта SQL и предлагает множество современных функций:

- сложные запросы
- внешние ключи

- триггеры
- изменяемые представления
- транзакционная целостность
- многоверсионность

Кроме того, пользователи могут всячески расширять возможности PostgreSQL, например создавая свои:

- типы данных
- функции
- операторы
- агрегатные функции
- методы индексирования
- процедурные языки

А благодаря свободной лицензии, PostgreSQL разрешается бесплатно использовать, изменять и распространять всем и для любых целей — личных, коммерческих или учебных.

PostgreSQL поддерживает работу со следующими объектами баз данных:

- Таблицы
- Индексы
- Пользователи и группы (роли)
- Языки (для создания функций)
- Функции (FUNCTION)
- Триггеры (TRIGGER)
- Правила (RULE)
- Представления (VIEW)
- Правила преобразования типов (CAST)
- Типы данных (TYPE)
- Последовательности (SEQUENCE)

Для сохранения данных, их поиска, обновления, извлечения из базы и удаления в PostgreSQL используется язык **SQL**. **SQL** — язык программирования структурированных запросов (**SQL, Structured Query Language**). С 1974 года, когда язык структурированных запросов только появился, он обеспечивает взаимодействие с системами управления базами данных (СУБД) во всём мире.

Язык SQL включает в себя:

1) **SQL DDL** - языка определения данных (DDL) даёт возможность независимо создавать базу данных, определять её структуру, использовать, а затем сбрасывать по завершению манипуляций;

2) **SQL DML** - языка управления данными (DML) для поддержки уже существующих баз данных на эффективном с точки зрения трудозатрат и производительности языке ввода, изменения и извлечения данных в отношении базы данных;

3) **SQL DCL** - язык контроля данных (DCL), когда нужно защитить свою базу данных от повреждения и неправильного использования.

### 1.4.1. Определение данных

Для создания новой таблицы в базе данных, необходимо использовать команду **CREATE TABLE**, при выполнении которой необходимо указать имя таблицы и перечислить все имена столбцов и их типы.

```
CREATE TABLE Товары (
    Название товара    varchar(80),
    Категория          varchar(80),
    Цена за единицу     real,
    Дата поставки      date
);
```

Основные категории типов данных: числовые типы (см. рисунок 1.19), символьные типы (см. рисунок 1.20), типы дата/время (см. рисунок 1.21).

Имя	Размер	Описание	Диапазон
smallint	2 байта	целое в небольшом диапазоне	-32768 .. +32767
integer	4 байта	типичный выбор для целых чисел	-2147483648 .. +2147483647
bigint	8 байт	целое в большом диапазоне	-9223372036854775808 .. 9223372036854775807
decimal	переменный	вещественное число с указанной точностью	до 131072 цифр до десятичной точки и до 16383 — после
numeric	переменный	вещественное число с указанной точностью	до 131072 цифр до десятичной точки и до 16383 — после
real	4 байта	вещественное число с переменной точностью	точность в пределах 6 десятичных цифр
double precision	8 байт	вещественное число с переменной точностью	точность в пределах 15 десятичных цифр
smallserial	2 байта	небольшое целое с автоувеличением	1 .. 32767
serial	4 байта	целое с автоувеличением	1 .. 2147483647
bigserial	8 байт	большое целое с автоувеличением	1 .. 9223372036854775807

Рисунок 1.19 – Числовые типы



Имя	Описание
character varying( <i>n</i> ), varchar( <i>n</i> )	строка ограниченной переменной длины
character( <i>n</i> ), char( <i>n</i> )	строка фиксированной длины, дополненная пробелами
text	строка неограниченной переменной длины

Рисунок 1.20 – Символьные типы

Имя	Размер	Описание	Наименьшее значение	Наибольшее значение	Точность
timestamp [ ( <i>p</i> ) ] [ without time zone ]	8 байт	дата и время (без часового пояса)	4713 до н. э.	294276 н. э.	1 микросекунда
timestamp [ ( <i>p</i> ) ] with time zone	8 байт	дата и время (с часовым поясом)	4713 до н. э.	294276 н. э.	1 микросекунда
date	4 байта	дата (без времени суток)	4713 до н. э.	5874897 н. э.	1 день
time [ ( <i>p</i> ) ] [ without time zone ]	8 байт	время суток (без даты)	00:00:00	24:00:00	1 микросекунда
time [ ( <i>p</i> ) ] with time zone	12 байт	время дня (без даты), с часовым поясом	00:00:00+1559	24:00:00-1559	1 микросекунда
interval [ <i>поля</i> ] [ ( <i>p</i> ) ]	16 байт	временной интервал	-178000000 лет	178000000 лет	1 микросекунда

Рисунок 1.21 – Типы дата/время

Более подробное описание типов можно найти на сетевом ресурсе: <https://postgrespro.ru/docs/postgresql/13/datatype> (PostgreSQL : Документация: 13: Глава 8. Типы данных : Компания Postgres Professional).

Любому столбцу можно назначить значение по умолчанию. Когда добавляется новая строка и каким-то её столбцам не присваиваются значения, эти столбцы принимают значения по умолчанию.

```
CREATE TABLE Товары (
    Название товара    varchar(80),
    Категория          varchar(80),
    Цена за единицу    real DEFAULT 100,
    Дата поставки      date
);
```

## Ограничения

1) **Ограничение-проверка** — наиболее общий тип ограничений. В его определении вы можете указать, что значение данного столбца должно удовлетворять логическому выражению (проверке истинности).

```
CREATE TABLE Товары (
    Название товара    varchar(80),
    Категория          varchar(80),
    Цена за единицу    real CHECK (price > 0),
    Дата поставки      date
);
```

```
);
```

2) **Ограничения уникальности** гарантируют, что данные в определённом столбце или группе столбцов уникальны среди всех строк таблицы. Ограничение записывается так:

```
CREATE TABLE Товары (  
    Название товара    varchar(80) UNIQUE,  
    Категория          varchar(80),  
    Цена за единицу    real CHECK (price > 0),  
    Дата поставки      date  
);
```

3) **Ограничение первичного ключа** означает, что образующий его столбец или группа столбцов может быть уникальным идентификатором строк в таблице.

```
CREATE TABLE Товары (  
    Код товара          integer PRIMARY KEY,  
    Название товара    varchar(80) UNIQUE,  
    Категория          varchar(80),  
    Цена за единицу    real CHECK (price > 0),  
    Дата поставки      date  
);
```

4) **Ограничение внешнего ключа** указывает, что значения столбца (или группы столбцов) должны соответствовать значениям в некоторой строке другой таблицы. Это называется *ссылочной целостностью* двух связанных таблиц.

```
CREATE TABLE Категории (  
    Код категории      integer PRIMARY KEY,  
    Название категории varchar(80) UNIQUE  
);  
  
CREATE TABLE Товары (  
    Код товара          integer PRIMARY KEY,  
    Название товара    varchar(80) UNIQUE,  
    Категория          integer REFERENCES Категории (Код категории),  
    Цена за единицу    real CHECK (price > 0),  
    Дата поставки      date  
);
```

### Добавление нового столбца

```
ALTER TABLE Товары ADD COLUMN Описание text;
```

### Удаление столбца

```
ALTER TABLE Товары DROP COLUMN Описание;
```

### Изменение значения по умолчанию

```
ALTER TABLE Товары ALTER COLUMN Цена за единицу SET DEFAULT 1000;
```

## Изменение типа данных столбца

```
ALTER TABLE Товары ALTER COLUMN Цена за единицу TYPE numeric(10,2);
```

## Переименование столбца

```
ALTER TABLE Товары RENAME COLUMN Код товара TO Код;
```

## Добавление ограничений

```
ALTER TABLE Товары ADD CHECK (Название товара <> '');
```

```
ALTER TABLE Категории  
ADD CONSTRAINT Ограничение1 UNIQUE (Название категории);
```

### 1.4.2. Модификация данных

#### Добавление новых данных

Добавление данных в таблицу «Категории»

```
INSERT INTO Категория (Код категории, Название категории)  
VALUES (1, 'Мясные продукты');
```

или сокращенно (без указания имен столбцов):

```
INSERT INTO Категория VALUES (1, 'Мясные продукты');
```

Добавление данных в таблицу «Товары»

```
INSERT INTO Товары VALUES (1, 'Колбаса Краковская', 1, 1500, '22.11.2000');
```

Код категории	Название категории	Код товара	Название товара	Категория	Цена за единицу	Дата
1	Мясные продукты	1	Колбаса Краковская	1	1500	22.11.2000

#### Изменение данных

```
UPDATE Категория  
SET Название категории = 'Полуфабрикаты'  
WHERE Код категории = 1;
```

```
UPDATE Товары  
SET Цена за единицу = 1000, Дата='23.11.2020'  
WHERE Код товара = 1;
```

#### Удаление данных

```
DELETE FROM Товары WHERE Код товара = 1;
```

## Возврат данных из измененных строк

Иногда бывает полезно получать данные из модифицируемых строк в процессе их обработки. Это возможно с использованием предложения **RETURNING**, которое можно задать для команд **INSERT**, **UPDATE** и **DELETE**. Применение **RETURNING** позволяет обойтись без дополнительного запроса к базе для сбора данных и это особенно ценно, когда как-то иначе трудно получить изменённые строки надёжным образом.

```
INSERT INTO Категории (Название категории)
VALUES ('Молочные продукты') RETURNING Код категории;
```

### 1.4.3. Запросы

#### Предложение FROM

Предложение FROM образует таблицу из одной или нескольких ссылок на таблицы, разделённых запятыми.

```
FROM табличная_ссылка [, табличная_ссылка [, ...]]
```

Здесь табличной ссылкой может быть имя таблицы (возможно, с именем схемы), производная таблица, например подзапрос, соединение таблиц или сложная комбинация этих вариантов. Если в предложении FROM перечисляются несколько ссылок, для них применяется перекрёстное соединение (то есть декартово произведение их строк).

**Соединённая таблица** — это таблица, полученная из двух других (реальных или производных от них) таблиц в соответствии с правилами соединения конкретного типа.

Приведем примеры соединений.

Исходные таблицы:

Код категории	Название категории
1	Мясные продукты
2	Молочные продукты

Код товара	Название товара	Код категории
1	Колбаса Краковская	1
2	Колбаса Доктор Зло	1

```
SELECT * FROM Категории CROSS JOIN Товары;
```

Результат содержит все возможные сочетания строк из двух таблиц (т. е. их декартово произведение), а набор столбцов объединяет в себе столбцы первой таблицы со следующими за ними столбцами второй таблицы.

Код категории	Название категории	Код товара	Название товара	Код категории
---------------	--------------------	------------	-----------------	---------------

1	Мясные продукты	1	Колбаса Краковская	1
1	Мясные продукты	2	Колбаса Доктор Зло	1
2	Молочные продукты	1	Колбаса Краковская	1
2	Молочные продукты	2	Колбаса Доктор Зло	1

*SELECT \**  
*FROM Категории INNER JOIN Товары*  
*ON Категории.Код категории = Товары.Код категории;*

Для каждой строки из первой таблицы в результирующей таблице содержится строка из второй таблице, удовлетворяющая условию соединения.

Код категории	Название категории	Код товара	Название товара	Код категории
1	Мясные продукты	1	Колбаса Краковская	1
1	Мясные продукты	2	Колбаса Доктор Зло	1

*SELECT \* FROM Категории INNER JOIN Товары USING (Код категории);*

USING — это сокращённая запись условия, полезная в ситуации, когда с обеих сторон соединения столбцы имеют одинаковые имена. Она принимает список общих имён столбцов через запятую и формирует условие соединения с равенством этих столбцов, исключая избыточные столбцы, так как они содержат одинаковые значения.

Код категории	Название категории	Код товара	Название товара
1	Мясные продукты	1	Колбаса Краковская
1	Мясные продукты	2	Колбаса Доктор Зло

*SELECT \* FROM Категории NATURAL INNER JOIN Товары;*

NATURAL — сокращённая форма USING: она образует список USING из всех имён столбцов, существующих в обеих входных таблицах. Как и с USING, эти столбцы оказываются в выходной таблице в единственном экземпляре.

*SELECT \**  
*FROM Категории LEFT JOIN Товары*  
*ON Категории.Код категории = Товары.Код категории;*

Сначала выполняется внутреннее соединение (INNER JOIN). Затем в результат добавляются все строки из первой таблицы, которым не соответствуют никакие строки во второй таблицы, а вместо значений столбцов второй таблицы вставляются NULL.

Код категории	Название категории	Код товара	Название товара	Код категории
1	Мясные продукты	1	Колбаса Краковская	1
1	Мясные продукты	2	Колбаса Доктор Зло	1
2	Молочные продукты			

```
SELECT *
FROM Категории RIGHT JOIN Товары
ON Категории.Код категории = Товары.Код категории;
```

Сначала выполняется внутреннее соединение (INNER JOIN). Затем в результат добавляются все строки из второй таблицы, которым не соответствуют никакие строки в первой таблице, а вместо значений столбцов первой таблицы вставляются NULL.

#### Псевдонимы таблиц

Таблицам и ссылкам на сложные таблицы в запросе можно дать временное имя, по которому к ним можно будет обращаться в рамках запроса. Такое имя называется псевдонимом таблицы.

```
FROM имя_таблицы AS псевдоним
```

```
SELECT *
FROM "очень_длинное_имя_таблицы" s JOIN "другое_длинное_имя" a
ON s.id = a.num;
```

#### Предложение WHERE

Предложение WHERE записывается так:

```
WHERE условие_ограничения
```

где *условие\_ограничения* — любое выражение значения, выдающее результат типа boolean.

Несколько примеров запросов с **WHERE**:

```
SELECT ... FROM table1 WHERE id > 5

SELECT ... FROM table1 WHERE price IN (10, 20, 30)

SELECT ... FROM table1 WHERE id IN (SELECT id FROM table2)
```


```
SELECT ...  
FROM table1  
WHERE p1 IN (SELECT p3 FROM table2 WHERE p2 = table1.p1 + 10)
```

```
SELECT ...  
FROM table1  
WHERE p1 BETWEEN  
    (SELECT p3 FROM table2 WHERE p2 = table1.p1 + 10) AND 100
```

```
SELECT ...  
FROM table1  
WHERE EXISTS (SELECT p1 FROM table2 WHERE p2 > table1.p1)
```

## 2. Практические задания

Для выполнения практических заданий необходимо изучить основы языка SQL, используя данное пособие или другие электронные ресурсы.

Каждая работа включает в себя текст задания и образец его выполнения. Задания, которые студент должен выполнить обозначены символом . Все составленные SQL-запросы необходимо сохранять в текстовом файле для последующей демонстрации.

Для предотвращения потерь данных, необходимо выполнять резервное копирование базы данных. Резервное копирование можно выполнять с помощью специальных утилит pg\_dump и psql, которые запускаются через командную строку.

**Создании копии:** pg\_dump -U postgres hdg >d:\1.sql

**Извлечение из копии:** psql -U postgres hdg <d:\1.sql

Желаем удачи.

### 2.1. Создание базы данных в СУБД PostgreSQL

Для знакомства с СУБД PostgreSQL создадим новую базу данных.

**Предметная область:** компания, оказывающая риэлтерские услуги. ER-модель предметной области представлена на рисунке 2.2.

Для создания таблиц базы данных, необходимо запустить **pgAdmin**. Далее произвести подключение к локальному или удаленному серверу.

Далее необходимо раскрыть пункт дерева объектов с именем созданного подключения, выбрать пункт **Базы данных**, щелкнуть ПКМ и выбрать пункт **Создать - База данных** (см. рисунок 2.1).

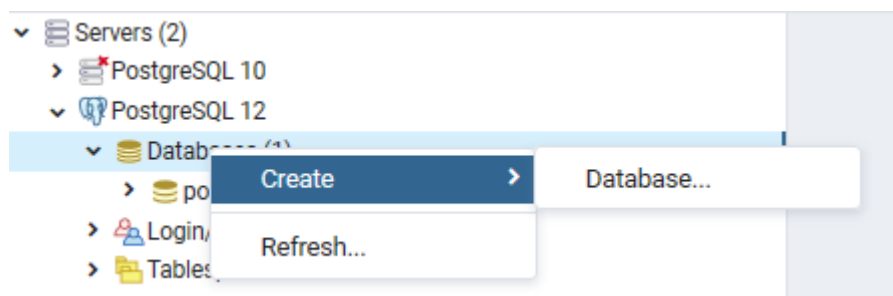


Рисунок 2.1 – Создание базы данных



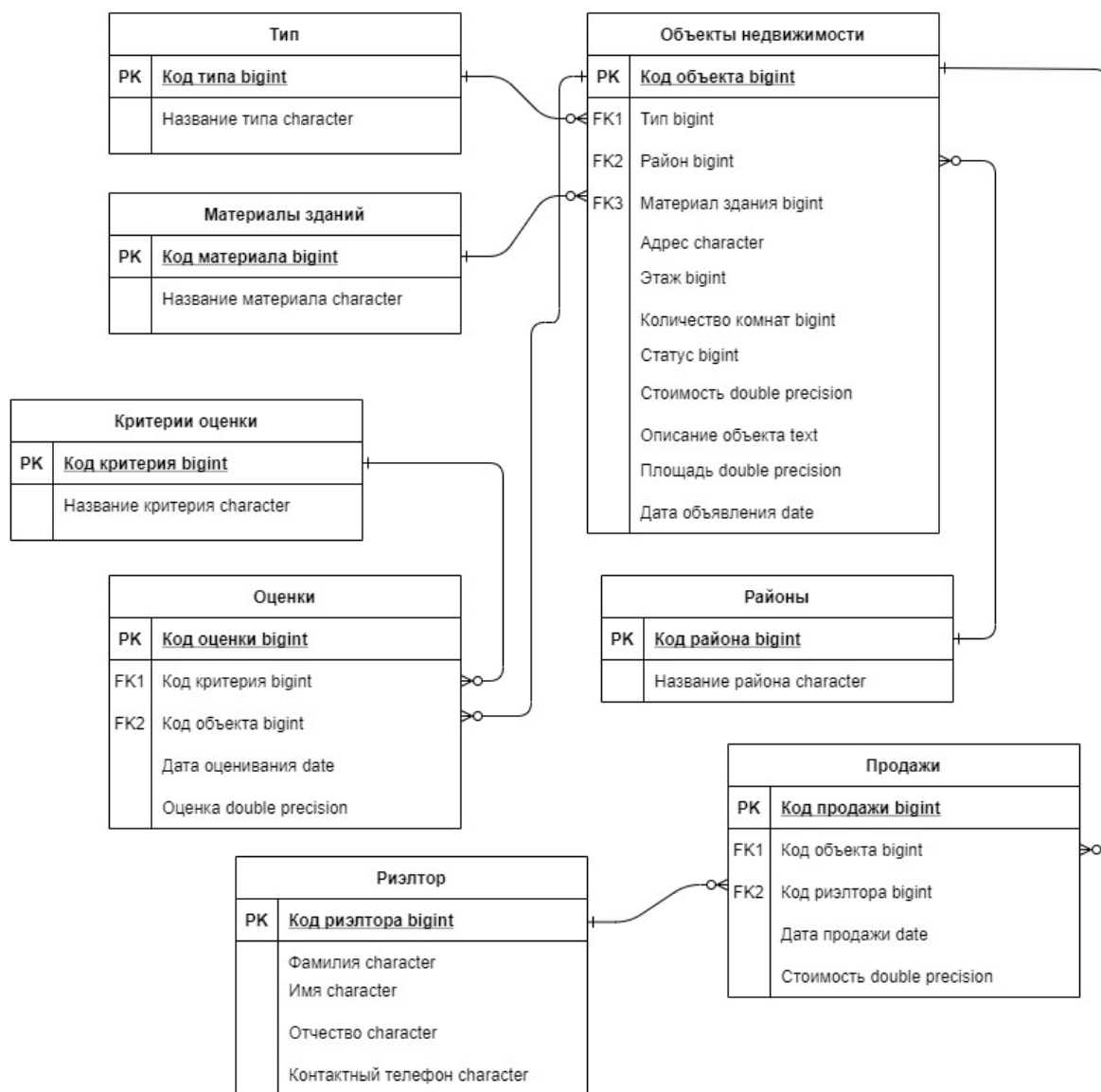


Рисунок 2.2 – ER-диаграмма

Укажите имя базы данных – realtor-Фамилия.

Теперь необходимо создать таблицы. Для этого раскроем дерево объектов: **Схемы – public**. Для создания таблицы, щелкнем ПКМ на объекте **Таблицы** и выберем пункт: **Создать – Таблица** (см. рисунок 2.3).

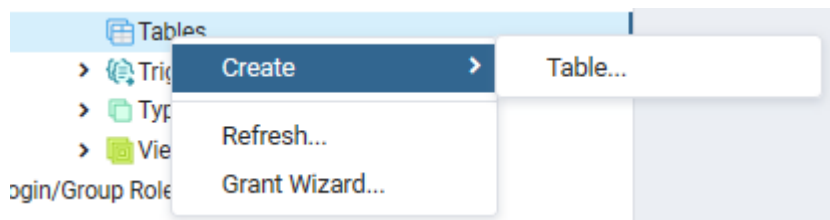


Рисунок 2.3 – Создание новой таблицы



Создайте 8 таблиц.

**Внимание! Имена таблиц и полям необходимо задавать на английском языке, без пробелов, маленькими буквами.**

#### Таблица «Тип»

Код типа	bigint, <b>primary key</b>
Название типа	character

Для добавления первичного ключа необходимо установить переключатель **Primary Key?** в положение «Yes» (см. рисунок 2.4).

Columns							+
	Name	Data type	Length	Precision	Not NULL?	Primary key?	
		id	bigint		No	Yes	

Рисунок 2.4 – Установка первичного ключа

#### Таблица «Районы»

Код района	bigint, <b>primary key</b>
Название района	character


#### Таблица «Материалы здания»

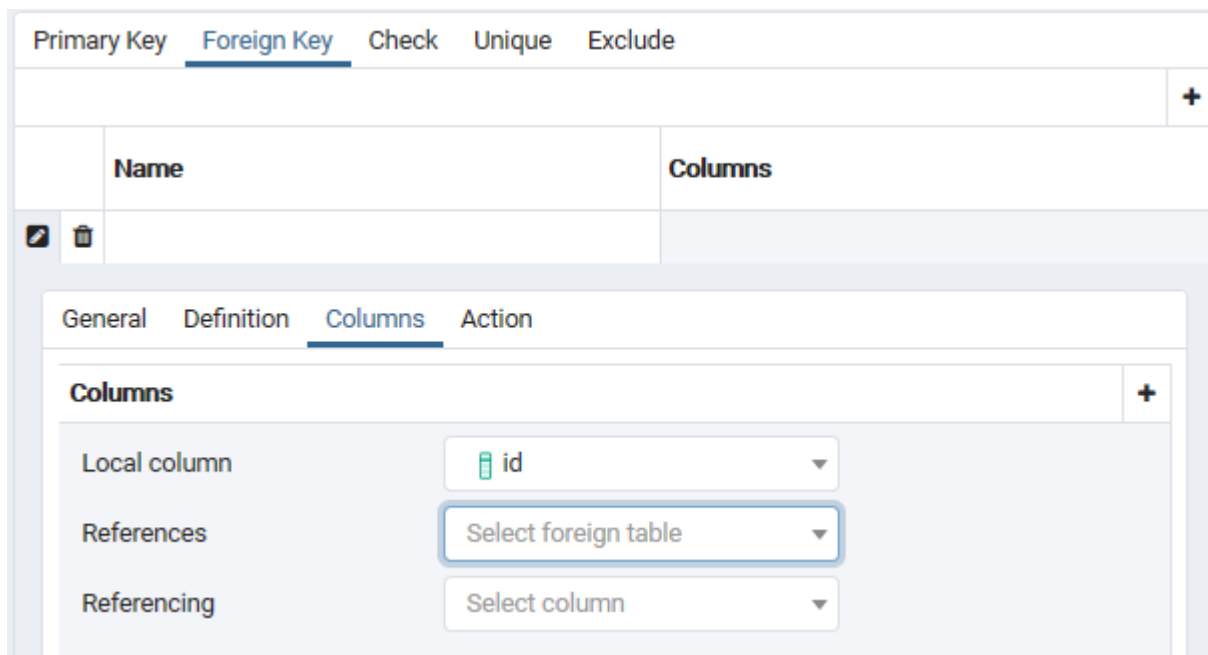
Код материала	bigint, <b>primary key</b>
Название материала	character

#### Таблица «Объекты недвижимости»

Код объекта	bigint, <b>primary key</b>
Район	bigint, внешний ключ (Районы)
Адрес	character
Этаж	bigint
Количество комнат	bigint
Тип	bigint, внешний ключ (Тип)
Статус (1 – в продаже, 0 – продана)	bigint
Стоимость	double precision
Описание объекта	text
Материал здания	bigint, внешний ключ (Материалы зданий)

Площадь	double precision
Дата объявления	date

Для добавления внешнего ключа необходимо перейти на вкладку **Ограничения (Constraints)**, выбрать вкладку **Внешний ключ (Foreign Key)**, нажать кнопку «+», далее  для редактирования строки. На вкладке **Колонки**, выбираем локальную колонку (Local column), таблицу (References) и колонку (Referencing), с которой необходимо создать связь (см. рисунок 2.5).



The screenshot shows a database management tool interface. At the top, there are tabs for 'Primary Key', 'Foreign Key' (which is selected), 'Check', 'Unique', and 'Exclude'. Below these tabs is a table with columns 'Name' and 'Columns'. A '+' button is visible in the top right corner of this table. Below the table, there is a sub-interface for the 'Foreign Key' configuration. It has tabs for 'General', 'Definition', 'Columns' (which is selected), and 'Action'. Under the 'Columns' tab, there is a section titled 'Columns' with a '+' button. This section contains three dropdown menus: 'Local column' with 'id' selected, 'References' with 'Select foreign table' selected, and 'Referencing' with 'Select column' selected.

Рисунок 2.5 – Установка внешнего ключа

Таблица «Критерии оценки»

Код критерия	bigint, <b>primary key</b>
Название критерия	character

Таблица «Оценки»

Код оценки	bigint, <b>primary key</b>
Код объекта	bigint, внешний ключ (Объекты недвижимости)
Дата оценивания	date
Код критерия	bigint, внешний ключ (Критерии оценки)
Оценка	double precision

### Таблица «Риэлтор»

Код риэлтора	bigint, <b>primary key</b>
Фамилия	character
Имя	character
Отчество	character
Контактный телефон	character

### Таблица «Продажа»

Код продажи	bigint, <b>primary key</b>
Код объекта	bigint, внешний ключ (Объекты недвижимости)
Дата продажи	date
Код риэлтора	bigint, внешний ключ (Риэлтор)
Стоимость	double precision

В результате необходимо получить следующий окончательный набор таблиц (см. рисунок 2.6):

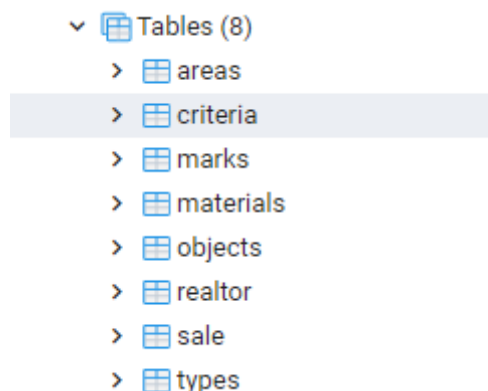


Рисунок 2.6 – Список таблиц базы данных

Для добавления новой записи используется SQL-команда **INSERT**:

**INSERT INTO** имя\_таблицы (имя\_поля)  
**VALUES** (значение)

Если полей несколько, то их имена и значения необходимо перечислить через запятую в соответствующих скобках:

**INSERT INTO** имя\_таблицы (имя\_поля1, имя\_поля2, имя\_поля3)  
**VALUES** (значение1, значение2, значение3)

Текст команды можно сократить, если значения полей указываются в том же порядке

**INSERT INTO имя\_таблицы  
VALUES (значение1, значение2, значение3)**

Например, чтобы добавить еще одну запись в таблицу **Группы**, необходимо написать:

**INSERT INTO ГРУППЫ (Название) VALUES ('КБ-11')**

Значение текстовых полей необходимо оформлять в апострофах (').

*Примеры добавления данных в созданные таблицы (название полей и таблиц приведены условно):*

**INSERT INTO t1 (p1,p2) VALUES (1,'Квартира') // таблица «Тип»**

**INSERT INTO t2 (p1,p2) VALUES (1,'Ленинский') // таблица «Районы»**

**INSERT INTO t3 (p1,p2) VALUES (1,'кирпич') // таблица «Материалы зданий»**

**INSERT INTO t4 VALUES (1,1,'ул. Савушкина 3 кв. 25',2,2,1,1,1000000,'Превосходная квартира',1,30.5,'12.03.2017') // таблица «Объект недвижимости»**

**INSERT INTO t5 VALUES (1,'экология') // таблица «Критерии оценки»**

**INSERT INTO t6 VALUES (1,'28.03.2017',1,3.5) // таблица «Оценки»**

**INSERT INTO t7 VALUES**

**(1,'Иванов','Георгий','Петрович','89608521245') // таблица «Риэлтор»**

**INSERT INTO t8 VALUES (1,'28.03.2017',1,80000.10,1) // таблица «Продажи»**



Добавьте новые данные в таблицы. Количество записей указано в таблице ниже (см. таблицу 2.1).

Таблица 2.1 – Количество записей в таблицах

Тип	3 (квартира, дом, апартаменты)
Районы	4
Материалы зданий	2 (панель, кирпич)
Объекты недвижимости	30
Критерии оценки	6 (экология, чистота, соседи, условия для детей, магазины, безопасность)
Оценки	50
Риэлтор	5
Продажи	20

## 2.2. Выборка данных из базы данных посредством SQL-запросов

Для выборки данных используется SQL-команда **SELECT**.

*Синтаксис команды:*

**SELECT** имя\_поля **FROM** имя\_таблицы **WHERE** условие\_отбора

Все поля, которые в итоге должны быть отражены в результате запроса перечисляются через запятую после ключевого слова **SELECT**. Если необходимо выбрать все поля таблицы ставится \*. После ключевого слова **FROM** перечисляются таблицы, из которых необходимо выбрать данные, если их несколько, то их тоже необходимо перечислить через запятую. После ключевого слова **WHERE** указывается условие отбора. Если выборку необходимо осуществить с использованием нескольких таблиц, то после **WHERE** необходимо указать связь между таблицами. Для объединения нескольких условий используются логические операторы **AND (И)** или **OR (ИЛИ)**.

### Примеры SQL-запросов:

1. *Выбрать адреса всех объектов недвижимости:*

**SELECT** Адрес  
**FROM** Объект недвижимости

2. *Выбрать все данные об объектах недвижимости:*

**SELECT** \*  
**FROM** Объект недвижимости (\* означает, что выбираем все поля таблицы)

3. *Выберем все объекты недвижимости с двумя комнатами:*

**SELECT** \*  
**FROM** Объект недвижимости  
**WHERE** Количество комнат = 2

4. *Выбрать адреса всех квартир*

В данном запросе используются данные из двух таблиц: *Тип* и *Объект недвижимости*.

**SELECT** Адрес  
**FROM** Тип, Объект недвижимости  
**WHERE** Название типа = 'квартира'  
**AND** Тип.Код типа= Объект недвижимости.Тип

Подчеркнутым выделено часть запроса, в которой указывается по каким полям (первичный и внешний ключи) существует связь между таблицами. Логический оператор **AND** используется для связки двух условий.

Результат выполнения запроса **/SELECT Адрес, Тип FROM Объект недвижимости/** - будет примерно следующим:

Адрес	Тип
ул. Савушкина 30 кв.15	1
ул. Победы 15 кв. 5	2
ул. Свердлова 100 кв.5	1

Цифры в поле **Тип** не говорят ни о чем (хотелось бы увидеть название типа), поэтому в данном запросе необходимо использовать две таблицы: из одной выбрать адрес объекта недвижимости (табл. **Объект недвижимости**), а из другой – название типа (табл. **Тип**). При этом в разделе **WHERE** необходимо указать по каким полям связаны таблицы:

**SELECT** Адрес, Название типа  
**FROM** Тип, Объект недвижимости  
**WHERE** Тип.Код типа= Объект недвижимости.Тип

Результат:

Адрес	Тип
ул. Савушкина 30 кв.15	квартира
ул. Победы 15 кв. 5	дом
ул. Свердлова 100 кв.5	квартира



**Создайте SQL-запросы:**

1) Выбрать все объекты недвижимости, расположенные на 2 этаже

Вх. данные	Вых. данные
этаж <u>Пример:</u> <i>этаж=2</i>	адрес, стоимость

2) Выбрать двухкомнатные объекты недвижимости с площадью более указанного значения

Вх. данные	Вых. данные
количество комнат, площадь <u>Пример:</u> <i>количество комнат=2</i> <i>площадь&gt;30</i>	адрес, этаж, название типа

3) Вывести объекты недвижимости, добавленные после указанной даты со стоимостью ниже указанного значения

Вх. данные	Вых. данные
дата добавления, стоимость <u>Пример:</u> дата добавления > 20.10.2017 стоимость < 1000000	адрес, этаж, количество комнат, название района

4) Вывести все квартиры с площадью равной указанному значению

Вх. данные	Вых. данные
площадь, тип объекта <u>Пример:</u> площадь = 30 тип объекта = квартира	адрес, стоимость, название материала здания

5) Вывести объекты недвижимости, расположенные в указанном районе  
стоимостью «ОТ» и «ДО»

Вх. данные	Вых. данные
стоимость, название района <u>Пример:</u> стоимость > 1000000 И < 2000000 название района = Кировский	адрес, площадь, этаж, название типа объекта

6) Вывести все панельные объекты недвижимости, расположенные на указанном  
этаже и статусом «в продаже»

Вх. данные	Вых. данные
этаж, статус, материал здания <u>Пример:</u> этаж = 2 статус = 1 материал здания = панель	адрес, описание, дата объявления, название района

7) Вывести квартиры с площадью более указанного значения, расположенные в  
указанном районе

Вх. данные	Вых. данные
тип объекта, площадь, название района <u>Пример:</u> тип объекта = квартира площадь > 30 название района = Кировский	адрес, описание объекта

8) Вывести дома, имеющие более 2 комнат, расположенные в указанном районе

Вх. данные	Вых. данные
тип объекта, количество комнат, название района <u>Пример:</u> тип объекта = дом количество комнат > 2 название района = Ленинский	адрес, этаж, количество комнат



9) Вывести дома, проданные после указанной даты

Вх. данные	Вых. данные
тип объекта, дата продажи <u>Пример:</u> тип объекта=дом дата продажи > 20.10.2017	адрес, описание, площадь, название материала здания

10) Вывести квартиры, расположенные в панельном доме на 2 этаже стоимостью менее указанного значения

Вх. данные	Вых. данные
тип объекта, этаж, стоимость, материал здания <u>Пример:</u> тип объекта=квартира материал здания = панель этаж=2 стоимость<2000000	адрес, описание, количество комнат

11) Вывести фамилии риэлтор, которые продали двухкомнатные объекты недвижимости

Вх. данные	Вых. данные
количество комнат <u>Пример:</u> количество комнат=2	фамилия, имя, отчество риэлтора

12) Выбрать список квартир, проданных риэлтором выше указанной продажной стоимости

Вх. данные	Вых. данные
тип объекта, фамилия риэлтора, продажная стоимость <u>Пример:</u> тип объекта=квартира фамилия риэлтора=Иванов продажная стоимость>2000000	адрес, этаж, количество комнат

13) Выбрать все оценки объекта недвижимости по указанному адресу

Вх. данные	Вых. данные
адрес <u>Пример:</u> адрес = «ул. Пушкина»	название критерия оценки, оценка

14) Выбрать двухкомнатные объекты недвижимости, у которых имеется оценка по критерию «Условие для детей»

Вх. данные	Вых. данные
количество комнат, критерий <u>Пример:</u> количество комнат = 2	адрес, название района, название типа объекта недвижимости

критерий = «Условие для детей»	
--------------------------------	--

15) Вывести разницу между заявленной и продажной стоимостью объектов недвижимости, расположенных на 2 этаже

Вх. данные	Вых. данные
стоимость, продажная стоимость, этаж <i>Пример:</i> <i>этаж = 2</i>	адрес, разница, ФИО риэлтора

Порядок сортировки определяет предложение **ORDER BY**:

```
SELECT список_выборки
FROM табличное_выражение
ORDER BY выражение_сортировки1 [ASC | DESC] [NULLS { FIRST | LAST }]
[, выражение_сортировки2 [ASC | DESC] [NULLS { FIRST | LAST }] ...]
```

Когда указывается несколько выражений, последующие значения позволяют отсортировать строки, в которых совпали все предыдущие значения. Каждое выражение можно дополнить ключевыми словами ASC или DESC, которые выбирают сортировку соответственно по возрастанию или убыванию. По умолчанию принят порядок по возрастанию (ASC). Для определения места значений NULL можно использовать указания NULLS FIRST и NULLS LAST, которые помещают значения NULL соответственно до или после значений не NULL.

### 2.3. Использование агрегатных функций в SQL-запросах

Иногда возникает необходимость рассчитать количество строк результата, определить среднее, максимально или минимальное значение для поля. Для этих целей используются агрегатные функции (см. таблица 2.2).

Таблица 2.2 – Агрегатные функции SQL

Агрегатная функция	Назначение
COUNT(*)	количество строк, возвращаемых запросом
MAX(имя_поля)	максимальное значение для поля
MIN(имя_поля)	минимальное значение для поля
AVG (имя_поля)	среднее значение для поля
SUM (имя_поля)	сумма значений всех строк поля

#### Примеры использования агрегатных функций:

Подсчитывает количество объектов недвижимости, расположенных на 2 этаже.

```
SELECT COUNT(*)
FROM Объекты недвижимости
WHERE Этаж=2
```

Определяет максимальную стоимость объекта недвижимости.  
**SELECT** MAX(Стоимость)  
**FROM** Объекты недвижимости

Для функции **COUNT(\*)** в качестве параметра можно указать \* или названия поля.  
Для других функций указывается только название поля, для которого происходит расчет.

### Создайте SQL-запросы:

1) Определите количество продаваемых объектов недвижимости, расположенных на 2 этаже

Вх. данные	Вых. данные
этаж, статус <u>Пример:</u> <i>этаж=2</i> <i>статус=1</i>	количество объектов недвижимости

2) Определить количество квартир, имеющих стоимость ниже указанного значения

Вх. данные	Вых. данные
тип объекта, стоимость <u>Пример:</u> <i>тип объекта=квартира</i> <i>стоимость&lt;2000000</i>	количество квартир

3) Определить общую стоимость всех двухкомнатных объектов недвижимости, расположенных в указанном районе

Вх. данные	Вых. данные
количество комнат, название района <u>Пример:</u> <i>количество комнат=2</i> <i>название района = Ленинский</i>	общая стоимость объектов недвижимости

4) Определить среднюю стоимость дома с указанным количеством комнат и площадью

Вх. данные	Вых. данные
тип объекта, количество комнат, площадь <u>Пример:</u> <i>тип объекта=дом</i> <i>количество комнат = 2</i> <i>площадь = 30</i>	средняя стоимость дома

5) Определить максимальную продажную стоимость объекта недвижимости, проданного указанным риэлтором

Вх. данные	Вых. данные
риэлтор <u>Пример:</u>	максимальная продажная стоимость объектов недвижимости

риэлтор=Иванов	
----------------	--

6) Определить минимальную продажную стоимость квартиры, проданной в диапазоне дат «ОТ» и «ДО»

Вх. данные	Вых. данные
тип объекта, дата продажи <u>Пример:</u> тип объекта=квартира дата продажи>20.10.2017 И <25.10.2017	минимальная продажная стоимость квартиры

7) Определить среднюю оценку объектов недвижимости, расположенных в указанном районе

Вх. данные	Вых. данные
название района <u>Пример:</u> название района=Кировский	средняя оценка объектов недвижимости

8) Определить среднюю оценку квартир панельного дома с указанным количеством комнат

Вх. данные	Вых. данные
тип объекта, количество комнат, материал здания <u>Пример:</u> тип объекта=квартира материал здания = панель количество комнат=2	средняя оценка квартир

9) Определить среднюю оценку дома по критерию «Условия для детей»

Вх. данные	Вых. данные
тип объекта, критерий оценки <u>Пример:</u> тип объекта=дом критерий оценки=Условия для детей	средняя оценка дома

10) Определить среднюю оценку апартаментов по критерию «Безопасность», проданных указанным риэлтором

Вх. данные	Вых. данные
тип объекта, критерий оценки, риэлтор <u>Пример:</u> тип объекта=апартаменты риэлтор= Иванов критерий оценки = Безопасность	средняя оценка апартаментов

11) Определить среднюю продажную стоимость 1м<sup>2</sup> для квартир, которые были проданы в указанную дату «ОТ» и «ДО»

Вх. данные	Вых. данные
тип объекта, дата продажи <u>Пример:</u> тип объекта=квартира дата продажи > 20.10.2017 И < 25.10.2017	средняя продажная стоимость квартиры

12) Определить максимальную стоимость 1м<sup>2</sup> для квартир, расположенных в указанном районе.

Вх. данные	Вых. данные
тип объекта, название района <u>Пример:</u> тип объекта=квартира название района = Советский	адрес, максимальная стоимость 1м <sup>2</sup> квартиры

13) Определите количество объектов недвижимости, проданных указанным риэлтором

Вх. данные	Вых. данные
риэлтор <u>Пример:</u> риэлтор=Иванов	количество объектов недвижимости

14) Определить максимальную площадь объекта недвижимости, продаваемого по указанной стоимости

Вх. данные	Вых. данные
стоимость <u>Пример:</u> стоимость=1500000	максимальная площадь объекта недвижимости

15) Определить квартиры, у которых разница между заявленной и продажной стоимостью является максимальной.

Вх. данные	Вых. данные
тип объекта <u>Пример:</u> тип объекта=квартира	максимальная разница между заявленной и продажной стоимостью

## 2.4. Группировка данных посредством SQL-запросов

Предложение **GROUP BY** используется для определения групп выходных строк, к которым могут применяться агрегатные функции (**COUNT**, **MIN**, **MAX**, **AVG** и **SUM**). Если это предложение отсутствует, и используются агрегатные функции, то все столбцы с именами, упомянутыми в **SELECT**, должны быть включены в агрегатные функции, и эти функции будут применяться ко всему набору строк, которые удовлетворяют предикату запроса.

### Синтаксис SQL-команды:

```
SELECT список_выборки  
FROM ...  
[WHERE ...]  
GROUP BY группирующий_столбец [, группирующий_столбец] ...
```

В группированной таблице столбцы, не включённые в список GROUP BY, можно использовать только в агрегатных выражениях.

В следующем примере вычисляется общая сумма продаж по каждому направлению. Исходные таблицы:

Код товара	Название товара	Цена
1	Колбаса	300
2	Кефир	500

Код продажи	Код товара	Количество
1	1	5
2	1	10
3	2	12

```
SELECT р.Название товара, (sum(s.Количество) * р.Цена) AS Стоимость  
FROM Товары р LEFT JOIN Продажи s USING (Код товара)  
GROUP BY р.Название товара, р.Цена;
```

В этом примере столбцы «р.Название товара» и «р.Цена» должны присутствовать в списке GROUP BY, так как они используются в списке выборки. Столбец «s.Количество» может отсутствовать в списке GROUP BY, так как он используется только в агрегатном выражении (sum(...)), вычисляющем сумму продаж. Для каждого продукта этот запрос возвращает строку с итоговой суммой по всем продажам данного продукта.

Результат запроса:

Название товара	Стоимость
Колбаса	4500
Кефир	6000



Создайте SQL-запросы:

1) Вывести информацию о количестве объектов недвижимости по каждому этажу

Вх. данные	Вых. данные
	этаж — количество объектов недвижимости

В данном запросе необходимо использовать группировку данных. Для этого используется ключевое слово **GROUP BY**, которое указывает по какому полю группировать данные.

Подсчитать сколько однокомнатных и двухкомнатных объектов недвижимости находится в продаже

### Пример

**SELECT** Количество комнат, COUNT(\*)

**FROM** Объекты недвижимости

**GROUP BY** Количество комнат

2) Вывести информацию о количестве объектов недвижимости по каждому району

Вх. данные	Вых. данные
	название района – количество объектов недвижимости

3) Вывести информацию о количестве двухкомнатных объектах недвижимости по каждому типу

Вх. данные	Вых. данные
количество комнат <i>Пример:</i> <i>количество комнат = 2</i>	тип объекта – количество объектов недвижимости

4) Вывести информацию о средней стоимости объектов недвижимости, расположенных на 2 этаже по каждому материалу здания

Вх. данные	Вых. данные
этаж <i>Пример:</i> <i>этаж=2</i>	материал здания – средняя стоимость объекта недвижимости

5) Вывести информацию о максимальной стоимости квартир, расположенных в каждом районе

Вх. данные	Вых. данные
тип объекта <i>Пример:</i> <i>тип объекта = квартира</i>	название района – максимальная стоимость квартиры

6) Вывести информацию о количестве квартир, проданных каждым риэлтором

Вх. данные	Вых. данные
тип объекта	ФИО риэлтора – количество квартир

<u>Пример:</u> тип объекта = квартира	
--	--

7) Вывести информацию об общей стоимости апартаментов, расположенных в каждом районе

Вх. данные	Вых. данные
тип объекта <u>Пример:</u> тип объекта = апартаменты	название район – общая стоимость апартаментов

8) Вывести информацию о средней стоимости объектов недвижимости с площадью «ОТ» и «ДО» по каждому типу объекта

Вх. данные	Вых. данные
площадь <u>Пример:</u> площадь >30 И <50	тип объекта– средняя стоимость объектов недвижимости

9) Вывести информацию о средней оценке объектов недвижимости по каждому району

Вх. данные	Вых. данные
	название района – средняя оценка

10) Вывести информацию об общей продажной стоимости апартаментов, проданных в диапазоне дат «ОТ» и «ДО» по каждому риэлтору

Вх. данные	Вых. данные
тип объекта, дата продажи <u>Пример:</u> тип объекта = апартаменты дата продажи = >20.09.2017 И <20.09.2018	ФИО риэлтора – общая продажная стоимость апартаментов

11) Вывести информацию о средней оценке по каждому критерию для объекта недвижимости

Вх. данные	Вых. данные
адрес <u>Пример:</u> адрес = Победы 10 кв. 15	название критерия – средняя оценка

12) Вывести информацию о средней площади квартир по каждому району.

Вх. данные	Вых. данные
тип объекта <u>Пример:</u> тип объекта = квартира	название район – средняя площадь

13) Вывести информацию о максимальной и минимальной оценке по каждому критерию для объекта недвижимости



Вх. данные	Вых. данные
адрес <i>Пример:</i> <i>адрес = Победы 10 кв. 15</i>	название критерия – максимальная оценка, минимальная оценка

14) Вывести информацию о количестве объектах недвижимости по количеству комнат, у которых разница между продажной и заявленной стоимостью больше 10000.

Вх. данные	Вых. данные
	количество комнат – количество объектов недвижимости

15) Вывести информацию о средней стоимости квартир по каждому району, в описании которых встречается слово «с ремонтом».

Вх. данные	Вых. данные
тип объекта <i>Пример:</i> <i>тип объекта = квартира</i>	название район – средняя стоимость

Предикат **LIKE** сравнивает строку, указанную в первом выражении, для вычисления значения строки, называемого проверяемым значением, с образцом, который определен во втором выражении для вычисления значения строки. В образце разрешается использовать два трафаретных символа:

- символ подчеркивания (  ), который можно применять вместо любого единичного символа в проверяемом значении;
- символ процента (%) заменяет последовательность любых символов (число символов в последовательности может быть от 0 и более) в проверяемом значении.

Если проверяемое значение соответствует образцу с учетом трафаретных символов, то значение предиката равно **TRUE**. Ниже приводится несколько примеров написания образцов (см. таблица 2.3).

Таблица 2.3 – Примеры написания образцов

Образец	Описание
'abc%'	Любые строки, которые начинаются с букв «abc»
'abc_'	Строки длиной строго 4 символа, причем первыми символами строки должны быть «abc»
'%z'	Любая последовательность символов, которая обязательно заканчивается символом «z»
'%Rostov%'	Любая последовательность символов, содержащая слово «Rostov» в любой позиции строки
'% % %'	Текст, содержащий не менее 2-х пробелов, например, "World Wide Web"

**Пример**

Исходные данные:

Адрес	Дата объявления
ул. Савушкина, 10 кв. 10	19.10.2017
ул. Победы 5 кв.5	20.10.2017
ул. Лавушкина 10	11.11.2017

**SQL-команда:**

**SELECT** Адрес, Дата объявления

**FROM** Объект недвижимости

**WHERE** Адрес **LIKE** '%кина%'

Результат:

Адрес	Дата объявления
ул. Савушкина, 10 кв. 10	19.10.2017
ул. Лавушкина 10	11.11.2017

## 2.5. Создание представлений

**Представление (VIEW)** – это таблица, чье содержание выбирается из других таблиц с помощью выполнения запроса. Представления работают в запросах и операторах DML точно также как и основные таблицы, но не содержат никаких собственных данных. Представление создается командой CREATE VIEW. Она состоит из слов CREATE VIEW (СОЗДАТЬ ПРЕДСТАВЛЕНИЕ), имени представления, которое нужно создать, слова AS (КАК), и далее запроса, как в следующих примерах:

1. CREATE VIEW Student\_1 AS

SELECT \* FROM Students WHERE group = 'PP-11';

2. CREATE VIEW Student\_2 (group, rating) AS

SELECT group, AVG (ball) FROM Students GROUP BY group;

Представление можно использовать точно так же как и любую другую таблицу. Она может быть запрошена, модифицирована, вставлена в, удалена из, и соединена с, другими таблицами и представлениями.

Сделаем запросы таких представлений:

1. SELECT \* FROM Student\_1;

2. SELECT \* FROM Student\_1 WHERE gender='M';

3. SELECT \* FROM Student\_2;



### Создайте представления:

1) Вывести информацию о максимальной оценке двухкомнатных объектах недвижимости по каждому району

Вх. данные	Вых. данные
количество комнат <u>Пример:</u> <i>количество комнат = 2</i>	название района – максимальная оценка

2) Вывести информацию об оценках, сделанных после указанной даты по указанному объекту недвижимости

Вх. данные	Вых. данные
адрес, дата оценивания <u>Пример:</u> <i>адрес = «ул. Пушкина 32»</i> <i>дата оценивания &gt; 20.09.2018</i>	название критерия, оценка, дата оценивания

3) Вывести информацию о средней оценке по каждому критерию по объекту недвижимости, в описании которого встречается слово «рядом с центром»

Вх. данные	Вых. данные
описание <u>Пример:</u> <i>описание = «...рядом с центром...»</i>	название критерия – средняя оценка

4) Вывести информацию об объектах недвижимости, у которых разница между заявленной и продажной стоимостью составляет больше 100000 рублей и проданную указанным риэлтором

Вх. данные	Вых. данные
риэлтор, стоимость, продажная стоимость <u>Пример:</u> <i>риэлтор=Иванов</i>	адрес, название района

5) Вывести информацию об объектах недвижимости, у которых разница между заявленной и продажной стоимостью составляет не более 20 % и расположенных в указанном районе

Вх. данные	Вых. данные
риэлтор, стоимость, продажная стоимость <u>Пример:</u> <i>район=Ленинский</i>	адрес, название района

6) Вывести информацию о двухкомнатных домах, которые расположены в переулках указанного района и стоимость 1м<sup>2</sup> меньше 30000 рублей

Вх. данные	Вых. данные
------------	-------------

район, тип объекта, количество комнат, описание <u>Пример:</u> район=Кировский тип объекта = дом количество комнат=2 описание = «переулок.....»	адрес, название материала здания
--	----------------------------------

7) Вывести информацию об объектах недвижимости, проданных в указанном году.

Вх. данные	Вых. данные
год <u>Пример:</u> год=2017	адрес, название района

Для выполнения запроса используйте *Extract(YEAR from "Date" )*.

8) Вывести информацию о количестве квартир, расположенных в каждом районе, объявление о продаже которых были размещены в указанном году.

Вх. данные	Вых. данные
год, тип объекта <u>Пример:</u> год=2017 тип объекта=квартира	название района – количество квартир

9) Вывести разницу в % между заявленной и продажной стоимостью для объектов недвижимости, проданных указанным риэлтором в текущем году.

Вх. данные	Вых. данные
риэлтор, год <u>Пример:</u> год=2020 риэлтор=Иванов	адрес, разница в %

10) Определить среднюю стоимость 1 м<sup>2</sup> для каждого района.

Вх. данные	Вых. данные
	название района – средняя стоимость 1м <sup>2</sup>

## 2.6. Создание функций

**Функция** - объект базы данных, используемый для выполнения анализа и различных вычислений.

**CREATE [ OR REPLACE ] FUNCTION**

*имя\_функции* ([ [ *метод\_аргумента* ] [*имя\_аргумента* ] *тип\_аргумента* [,...] ] )

**RETURNS** *тип\_возвращаемого\_значения*  
**AS** '*определение*'  
**LANGUAGE** '*язык*'

**CREATE FUNCTION** *имя\_функции* ([[ *метод\_аргумента* ] [ *имя\_аргумента* ] *тип\_аргумента* [,...] ]) –

после ключевых слов **CREATE FUNCTION** указывается *имя создаваемой функции*, после чего в круглых скобках перечисляются аргументы, разделенные запятыми.

Для каждого аргумента достаточно указать только тип, но при желании можно задать метод (in, out; по умолчанию in) и имя.

Если список в круглых скобках пуст, функция вызывается без аргументов (хотя сами круглые скобки обязательно должны присутствовать как в определении функции, так и при ее использовании).

Ключевые слова **OR REPLACE** используются для изменения уже существующей функции.

**RETURNS** *тип\_возвращаемого\_значения* - тип данных, возвращаемый функцией.

**AS** '*определение*' - программное определение функции.

В процедурных языках (таких, как PL/pgSQL) оно состоит из кода функции. Для откомпилированных функций *C* указывается абсолютный системный путь к файлу, содержащему объектный код.

**LANGUAGE** '*язык*'. Название языка, на котором написана функция. В аргументе может передаваться имя любого процедурного языка (такого, как plpgsql или plperl, если соответствующая поддержка была установлена при компиляции), *C* или SQL.

Примеры функций:

Данный пример производит подсчет количества проданных риэлтором (код риэлтора - 42) объектов недвижимости:

```
CREATE FUNCTION dup(in a integer, out f bigint)
RETURNS bigint
    AS 'SELECT COUNT(*) FROM "Продажи" WHERE "Продажи"."Код риэлтора"=$1'
    LANGUAGE SQL;

SELECT * FROM dup(42);
```

Данная функция производит подсчет суммы двух чисел:

```
CREATE FUNCTION add(integer, integer) RETURNS integer
```

```
AS 'select $1 + $2;'  
LANGUAGE SQL
```

Данная функция возвращает список проданных риэлтором объектов недвижимости:

```
CREATE FUNCTION dup(in integer)  
RETURNS TABLE (f1 double precision, f2 date)  
AS 'SELECT "Стоимость", "Дата продажи" FROM "Продажи" WHERE  
"Продажи". "Код риэлтора"=$1 '  
LANGUAGE SQL;  
  
SELECT * FROM dup(42);
```

Создание функции, возвращающей столбец текстовых значений:

```
CREATE FUNCTION list(integer) RETURNS SETOF character AS '  
-- Функция возвращает имена риэлторов с кодом больше $1  
SELECT Имя FROM Риэлторы WHERE kod>$1;  
' LANGUAGE sql;
```

Использование функции, возвращающей столбец текстовых значений:

```
SELECT list(10) AS Результат;
```

Результат:

Результат
Иванов
Петров
Николаев

Создание функции, возвращающей столбец записей:

```
CREATE FUNCTION list(integer) RETURNS SETOF record AS '  
-- Функция возвращает сведения о поставщиках с рейтингом больше $1  
SELECT Имя, Телефон FROM Риэлторы WHERE kod>$1;  
' LANGUAGE sql;
```

*Примечание.* Функция, возвращающая столбец записей, может быть использована в операторе **SELECT**, если она определена как функция SQL.

Использование функции, возвращающей столбец записей:

```
SELECT list(10) AS Результат;
```

Результат:

Результат
("Иванов", "8888")

(“Петров”, “9999”)

(“Николаев”, “7777”)



**Создайте функции:**

1) Конвертирует стоимость объекта недвижимости в евро и долларах.

Вх. параметр	Вых. параметр
курс, код объекта недвижимости	стоимость в евро

2) Рассчитывает заработную плату риэлтора по формуле:  $N \cdot S + R$ , где N – количество проданных объектов недвижимости в месяц (подсчет осуществляется автоматически по данным таблицы «Продажи» с использованием агрегатной функции), S – ставка, R – премия.

Вх. параметр	Вых. параметр
S, R, начальная и конечная дата, фамилия риэлтора	размер заработной платы

3) Рассчитывает процент изменения продажной стоимости объекта недвижимости от первоначально заявленной.

Вх. параметр	Вых. параметр
код объекта недвижимости	процент изменения

4) Формирует список средних оценок по каждому критерию для объекта недвижимости.

Вх. параметр	Вых. параметр
код объекта недвижимости	название критерия – средняя оценка

5) Формирует список объектов недвижимости, стоимость  $1\text{м}^2$  у которых находится в заданном диапазоне.

Вх. параметр	Вых. параметр
начальная цена $1\text{м}^2$ , конечная цена $1\text{м}^2$ , тип объекта недвижимости	адрес, название района, количество комнат

6) Рассчитывает процент изменения общей продажной стоимости квартир между годом №1 и годом №2.

7) Рассчитывает какой процент составляет площадь каждого типа комнаты объекта недвижимости от общей площади.

## 2.7. Подзапросы. Предложение HAVING, CASE в SQL-запросах



1) Вывести адреса объектов недвижимости, у которых стоимость 1 м<sup>2</sup> меньше средней стоимости по району.

2) Вывести название районов, в которых количество проданных квартир больше 5.

Зачастую мы используем условие **HAVING** совместно с условием **GROUP BY**, чтобы отфильтровать строки, которые не удовлетворяют указанному условию.

Следующий запрос отображает синтаксис вызова условия HAVING:

```
SELECT
    column_1,
    aggregate_function (column_2)
FROM
    tbl_name
GROUP BY
    column_1
HAVING
    condition;
```

Предложение **HAVING** устанавливает условие для групп, состоящих из строк, созданных предложением **GROUP BY** после его [предложения] применения, тогда как предложение **WHERE** устанавливает условие для отдельных строк перед применением условия **GROUP BY**. В этом и состоит главное отличие условия **HAVING** от условия **WHERE**.

В PostgreSQL вы можете использовать условие **HAVING** без условия **GROUP BY**. В таком случае, условие **HAVING** обратит запрос в одну группу. К тому же, список **SELECT** и условие **HAVING** могут ссылаться только на столбцы из агрегатных функций. Данный тип запроса возвращает одну строку, если условие в предложении **HAVING** принимает значение true, или нулевую строку, если оно равно false.

3) Вывести адреса квартир и название района, средняя оценка которых выше 3,5 баллов.

4) Вывести ФИО риэлторов, которые продали меньше 5 объектов недвижимости

5) Определить годы, в которых было размещено от 2 до 3 объектов недвижимости

6) Определить адреса квартир, стоимость 1м<sup>2</sup> которых меньше средней по району.

7) Определить ФИО риэлторов, которые ничего не продали в текущем году.

8) Вывести названия районов, в которых средняя площадь продаваемых квартир больше 30м<sup>2</sup>.

9) Вывести для указанного риэлтора (ФИО) года, в которых он продал больше 2 объектов недвижимости.



10) Вывести ФИО риэлторов, которые заработали премию в текущем месяце больше 40000 рублей. Премия рассчитывается по формуле:

$$\text{общая стоимость всех проданных квартир} * 15\%$$

11) Вывести количество однокомнатных и двухкомнатных квартир в указанном районе. Формат вывода:

Вид квартиры	Количество объектов недвижимости
Однокомнатных квартир	2
Двухкомнатных квартир	5

Выражение CASE в SQL представляет собой общее условное выражение, напоминающее операторы if/else в других языках программирования:

*CASE WHEN условие THEN результат*

*[WHEN ...]*

*[ELSE результат]*

*END*

Предложения CASE можно использовать везде, где допускаются выражения. Каждое *условие* в нём представляет собой выражение, возвращающее результат типа boolean. Если результатом выражения оказывается true, значением выражения CASE становится *результат*, следующий за условием, а остальная часть выражения CASE не вычисляется. Если же условие не выполняется, за ним таким же образом проверяются все последующие предложения WHEN. Если не выполняется ни одно из *условий* WHEN, значением CASE становится *результат*, записанный в предложении ELSE. Если при этом предложение ELSE отсутствует, результатом выражения будет NULL.

12) Определить индекс средней оценки по каждому критерию для указанного объекта недвижимости. Вывести среднюю оценку и эквивалентный текст согласно таблице:

Диапазон	Эквивалентный текст
от 90% до 100%	превосходно
от 80% до 90%	очень хорошо
от 70% до 80%	хорошо
от 60% до 70%	удовлетворительно
до 60 %	неудовлетворительно

Образец вывода результатов:

Критерий	Средняя оценка	Текст
Экология	5 из 5	превосходно
Чистота	4 из 5	очень хорошо
Соседи	3,5 из 5	хорошо

13) Добавить новую таблицу «Структура объекта недвижимости» с колонками: Объект недвижимости, Тип комнаты, Площадь. Установите ограничение-проверку

**площади**, которая должна быть больше нуля и **типа комнаты** (1, 2, 3, 4), где 1 – кухня, 2 – зал, 3 – спальня, 4 – санузел.

**Ограничение-проверка** — наиболее общий тип ограничений. В его определении вы можете указать, что значение данной колонки должно удовлетворять логическому выражению (проверке истинности). Например, цену товара можно ограничить положительными значениями так:

```
CREATE TABLE products (  
    product_no integer,  
    name text,  
    price numeric CHECK (price > 0)  
);
```

Как вы видите, ограничение определяется после типа данных, как и значение по умолчанию. Значения по умолчанию и ограничения могут указываться в любом порядке. Ограничение-проверка состоит из ключевого слова CHECK, за которым идёт выражение в скобках. Это выражение должно включать колонку, для которой задаётся ограничение, иначе оно не имеет большого смысла.

14) Вывести информацию о комнатах для объекта недвижимости.

Тип комнаты	Площадь
Зал	20
Кухня	5

15) Вывести количество объектов недвижимости по каждому району, общая площадь которых больше 40 м<sup>2</sup>. Использовать таблицу «Структура объекта недвижимости».

16) Используя функции для работы с датой: extract(field from timestamp) и age(timestamp, timestamp), вывести квартиры, которые были проданы не позже 4 месяцев после размещения объявления о их продаже.

17) Вывести адреса объектов недвижимости, стоимость 1м<sup>2</sup> которых меньше средней всех объектов недвижимости по району, объявления о которых были размещены не более 4 месяцев назад. Формат вывода:

Адрес	Статус
ул. Победы 40	продано
Ул. Савушкина 20	в продаже

18) Вывести информацию о количестве продаж в предыдущем и текущем годах по каждому району, а также процент изменения.

Название района	2018	2019	Разница в %
Кировский	20	40	100
Ленинский	30	20	-33,33

## Список литературы

1. Королева О.Н. Базы данных [Электронный ресурс]: курс лекций/ Королева О.Н., Мажукин А.В., Королева Т.В.— Электрон. текстовые данные.— Москва: Московский гуманитарный университет, 2012.— 66 с.— Режим доступа: <http://www.iprbookshop.ru/14515.html>.— ЭБС «IPRbooks»
2. Ревунков Г.И. Проектирование баз данных [Электронный ресурс]: учебное пособие по курсу «Банки данных»/ Ревунков Г.И.— Электрон. текстовые данные.— Москва: Московский государственный технический университет имени Н.Э. Баумана, 2009.— 20 с.— Режим доступа: <http://www.iprbookshop.ru/31513.html>.— ЭБС «IPRbooks»
3. Дьяков И.А. Базы данных. Язык SQL [Электронный ресурс]: учебное пособие/ Дьяков И.А.— Электрон. текстовые данные.— Тамбов: Тамбовский государственный технический университет, ЭБС АСВ, 2012.— 81 с.— Режим доступа: <http://www.iprbookshop.ru/64070.html>.— ЭБС «IPRbooks»
4. Баженова И.Ю. SQL и процедурно-ориентированные языки [Электронный ресурс]/ Баженова И.Ю.— Электрон. текстовые данные.— Москва: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016.— 166 с.— Режим доступа: <http://www.iprbookshop.ru/57532.html>.— ЭБС «IPRbooks»
5. Кара-Ушанов В.Ю. SQL - язык реляционных баз данных [Электронный ресурс]: учебное пособие/ Кара-Ушанов В.Ю.— Электрон. текстовые данные.— Екатеринбург: Уральский федеральный университет, ЭБС АСВ, 2016.— 156 с.— Режим доступа: <http://www.iprbookshop.ru/68419.html>.— ЭБС «IPRbooks»
6. Кузнецов С.Д. Введение в модель данных SQL [Электронный ресурс]/ Кузнецов С.Д.— Электрон. текстовые данные.— Москва: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016.— 350 с.— Режим доступа: <http://www.iprbookshop.ru/73664.html>.— ЭБС «IPRbooks»
7. Крис Фиайли SQL [Электронный ресурс]/ Крис Фиайли— Электрон. текстовые данные.— Саратов: Профобразование, 2019.— 452 с.— Режим доступа: <http://www.iprbookshop.ru/87984.html>.— ЭБС «IPRbooks»
8. PostgreSQL : Документация : Компания Postgres Professional. [Электронный ресурс]. URL: <https://postgrespro.ru/docs/postgresql>
9. Что такое SQL и для чего нужен: простыми словами, где используется SQL. [Электронный ресурс]. URL: <https://www.zeluslugi.ru/info-czentr/it-glossary/term-sql>