Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»
(ВлГУ)**

УТВЕРЖДАЮ
Заведующий кафедрой ИСПИ
_____ И.Е. Жигалов
«20» марта 2025 г.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ УЧЕБНОЙ ДИСЦИПЛИНЫ ПРОФЕССИОНАЛЬНОЙ ПОДГОТОВКИ

«ОСНОВЫ ПРОЕКТИРОВАНИЯ БАЗ ДАННЫХ»

09.02.09 Веб-разработка
Разработчик веб приложений

Владимир, 2025

Методические указания к лабораторным работам учебной дисциплины профессиональной подготовки «Основы проектирования баз данных» разработал преподаватель КИТП Курьерова С.А.

Методические указания к лабораторным работам рассмотрены и одобрены на заседании УМК специальности 09.02.09 Веб-разработка
протокол № 1 от «10» марта 2025 г.

Председатель УМК специальности _____ И.Е. Жигалов

Методические указания к лабораторным работам рассмотрены и одобрены на заседании кафедры ИСПИ
протокол № 7а от «12» марта 2025 г.

СОДЕРЖАНИЕ

ЛАБОРАТОРНАЯ РАБОТА № 1.....3
ЛАБОРАТОРНАЯ РАБОТА № 2.....6
ЛАБОРАТОРНАЯ РАБОТА № 3.....16
ЛАБОРАТОРНАЯ РАБОТА № 4.....23
ЛАБОРАТОРНАЯ РАБОТА № 5.....33
ЛАБОРАТОРНАЯ РАБОТА № 6.....42
ЛАБОРАТОРНАЯ РАБОТА № 7.....48
ЛАБОРАТОРНАЯ РАБОТА № 8.....53
ЛАБОРАТОРНАЯ РАБОТА № 9.....61
ЛАБОРАТОРНАЯ РАБОТА № 10.....65
ЛАБОРАТОРНАЯ РАБОТА № 11.....66
ЛАБОРАТОРНАЯ РАБОТА № 12.....68
ЛАБОРАТОРНАЯ РАБОТА № 13.....70
ЛАБОРАТОРНАЯ РАБОТА № 14.....74

ЛАБОРАТОРНАЯ РАБОТА № 1

Знакомство с СУБД PostgreSQL и языком SQL.

Цель работы: познакомиться со СУБД PostgreSQL и языком SQL.

Теоретическая часть

Ребят! Поздравляю с началом учебного года! Начать первую лабораторную хочу с простой, а для кого-то может быть приятной работой. Хочу познакомиться . Поэтому для начала задам несколько вопросов (отвечать обязательно). Мы, так сказать просто поболтаем. Надеюсь, взаимный диалог мы будем поддерживать до конца курса, мне всегда важна обратная связь от студентов. А в конце этой работы мы ознакомься со средой, в которой будем работать в течения курса. Надеюсь, вам будет интересно провести этот курсе, так же, как и мне поработать с вами.

Вопросы:

1. Как твоё настроение? :)
2. Кофе или чай? :)
3. Темная тема IDE или светлая? :) Почему такая?
4. Какие качества преподавателя ты ценишь больше всего?
5. Почему ты пошёл на "программиста". Опиши качества хорошего программиста.
6. Какие у тебя любимые языки программирования?
7. Что ты ожидаешь от курса "Основы проектирования баз данных"? Как думаешь, будет ли полезен он тебе в будущем?

Пожалуйста, ответь на все вопросы письменно и вставь ответы в отчёт по лабораторной работе.

А теперь своими словами попытаюсь объяснить, что такое SQL и PostgreSQL. Начнём с коротких топ 10 фактов:

- SQL — это язык программирования;
- С помощью SQL можно делать выборки данных, записывать, изменять или удалять. Так же SQL позволяет описывать схему данных и многое другое;
- SQL существует с 1974 года! Но несмотря на это очень актуален в наши дни;
- SQL правильно произносить как [ˈɛs kjuː ˈɛl] (эс-кью-эль) или также как [ˈsiːkwəl] (сиквел), оба варианта встречаются; SQL поддерживают большинство распространенных реляционных систем управления баз данных (СУБД);
- PostgreSQL одна из наиболее популярных СУБД в мире;
- СУБД PostgreSQL полностью бесплатная и свободная;
- 775 000 строчек исходного кода у СУБД PostgreSQL;

- Первая версия PostgreSQL выпущена в 1989 году, а последняя 11 августа 2022 (данные на момент 2022-09-06);
- PostgreSQL не поддерживал SQL до 1994.

Первую работу мы будем вести в веб-сервисе https://www.db-fiddle.com/

Сервис позволяет:

1. писать запросы SQL
2. исполнять SQL запросы
3. сохранять запросы
4. выбирать различные СУБД и их версии

DB-fiddle поддерживает следующие СУБД:

1. PostgreSQL
2. MySQL
3. SQLite

Перейдём на https://www.db-fiddle.com/ и озаглавим работу и выберем СУБД:

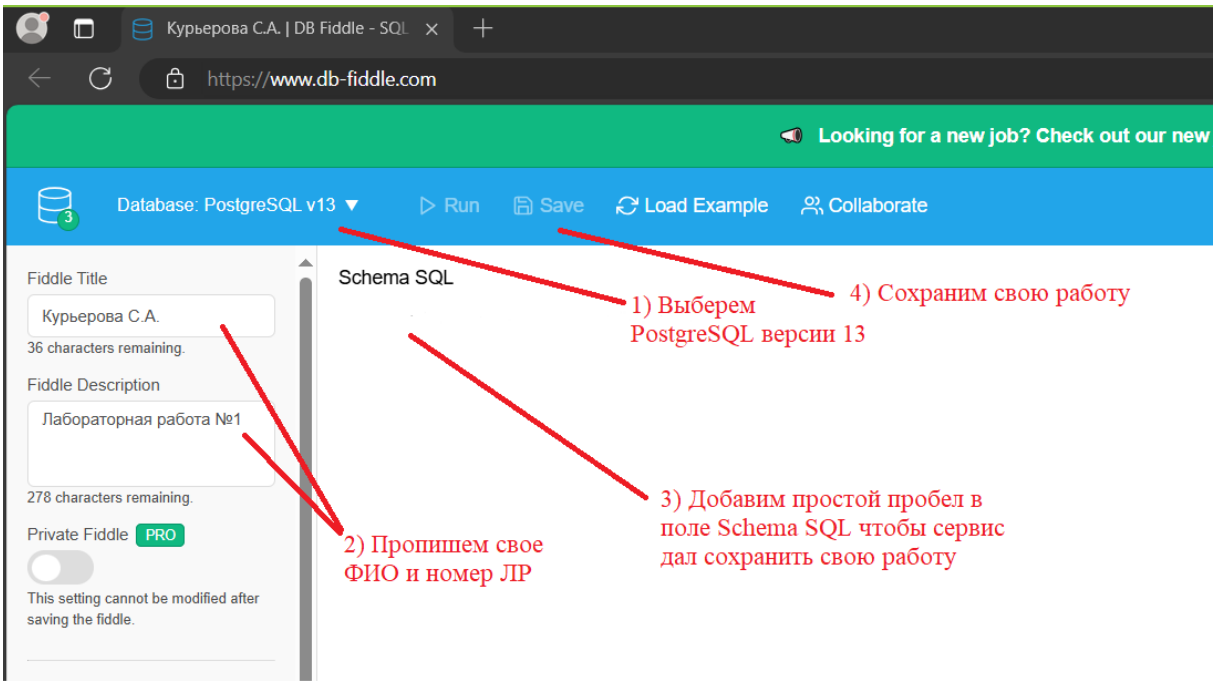


Рис. 1. Настройка db-fiddle

Сервис даст уникальную ссылку на эту работу:

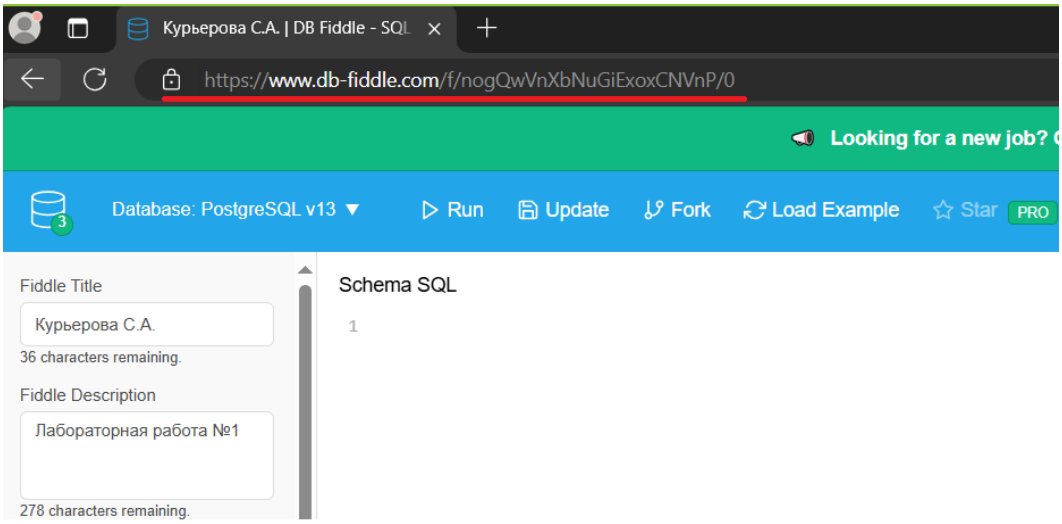


Рис. 2. Сохранение работы в db-fiddle

А теперь создадим таблицу:

```
CREATE TABLE student (
 id serial PRIMARY KEY, -- уникальный ID студента
name text NOT NULL, -- ФИО студента
birth_date date NOT NULL, -- дата рождения
group_name text NOT NULL, -- группа студента
sex boolean NOT NULL -- пол студента true - M, false - Ж
);
```

Вставим запрос в Schema SQL и обновим работу (нажмем кнопку Update) чтобы сохранить изменения.

Заполним таблицу данными:

```
INSERT INTO student (name, birth_date, group_name, sex)
VALUES ('Грекова Дана Евгеньевна', '2004-04-19', 'ИСПсп-120', false);
INSERT INTO student (name, birth_date, group_name, sex)
VALUES ('Нестеров Никита Русланович', '2004-03-18', 'ИСПсп-120', true);
INSERT INTO student (name, birth_date, group_name, sex)
VALUES ('Ершова Варвара Владимировна', '2005-01-02', 'ИСПсп-120', false);
INSERT INTO student (name, birth_date, group_name, sex)
VALUES ('Птицын Егор Игоревич', '2004-09-02', 'ИСПспк-220', true);
INSERT INTO student (name, birth_date, group_name, sex)
VALUES ('Зорина Полина Алексеевна', '2004-11-17', 'ИСПспк-220', false);
```

Задание

В https://www.db-fiddle.com/ создать проект с SQL запросами из текущей ЛР (CREATE TABLE, INSERT INTO). Добавьте по аналогии новую запись в таблицу со своими данными (ФИО, дата рождения, группа и т.д.)

В сети Интернет найдите SQL запрос, который выведет все данные из таблицы student. Введите этот запрос в поле Query SQL выполните это запрос (нажав на кнопку Run) и выведите всех студентов.

Не забудьте заполнить поля Fiddle Title своими инициалами, а поле Fiddle Description номером ЛР и вариантом работы (если имеется).

Требования к отчёту

1. Титульная страница

   Титульная страница должна содержать:

   1. Наименование учебного учреждения
   2. Наименование предмета
   3. Фамили и инициалы студента
   4. Фамилию и инициалы преподавателя, принимающего работу
   5. Номер индивидуального задания (если имеется)
2. Цель работы
3. Выполнение задания
4. Вывод по работе

ЛАБОРАТОРНАЯ РАБОТА № 2

Настройка рабочего пространства PgAdmin

Цель работы: познакомится с графическим клиентом PgAdmin.

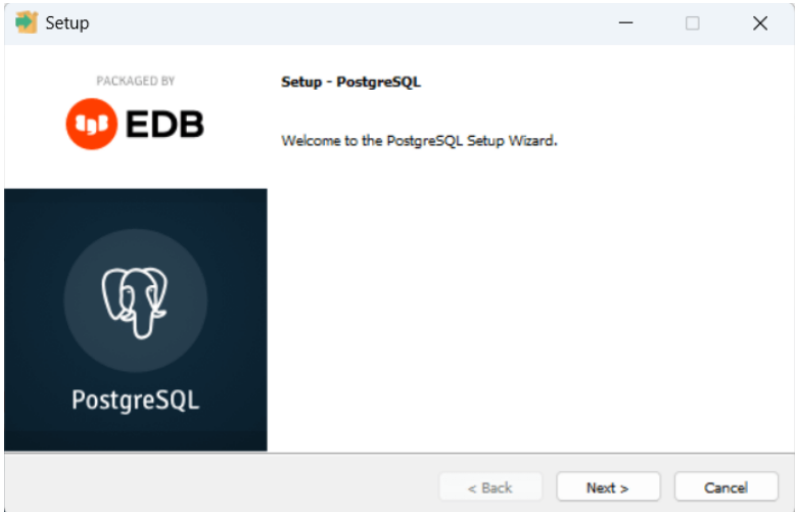
Теоретическая часть

PostgreSQL поддерживается для всех основных операционных систем - Windows, Linux, MacOS. Официальный сайт проекта: https://www.postgresql.org/.

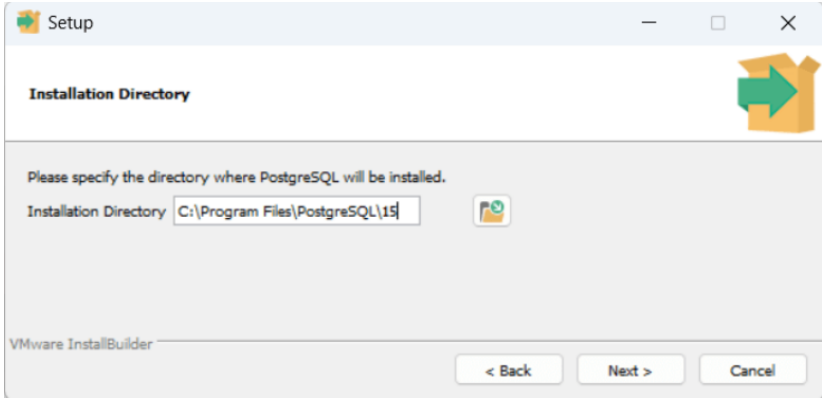
Установка

На странице https://www.postgresql.org/download/ можно найти ссылки на загрузку различных дистрибутивов для различных операционных систем.

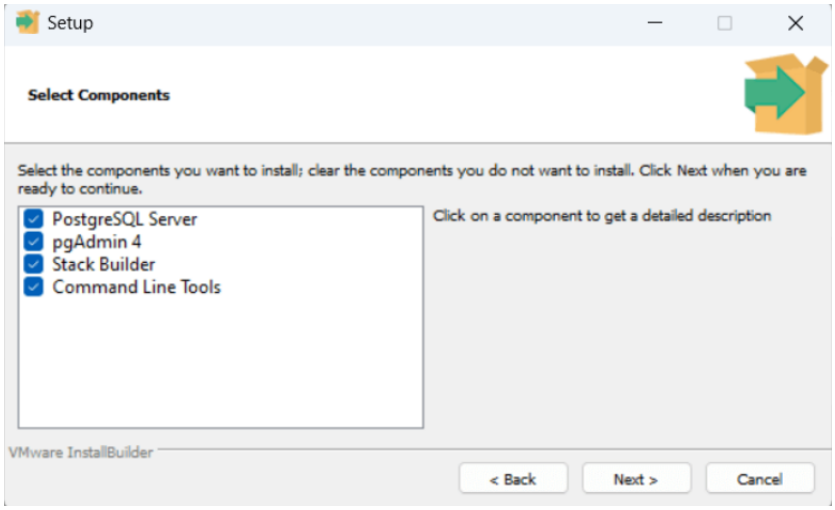
Запустим программу установки:



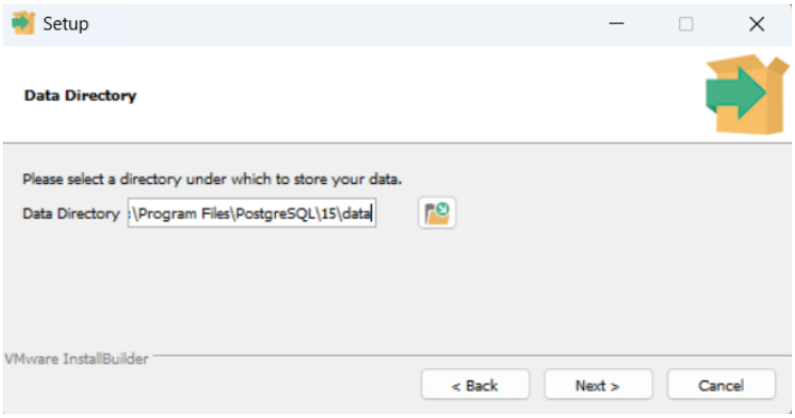
На следующем экране необходимо будет указать папку для установки. Оставим папку по умолчанию и перейдем к следующему шагу:



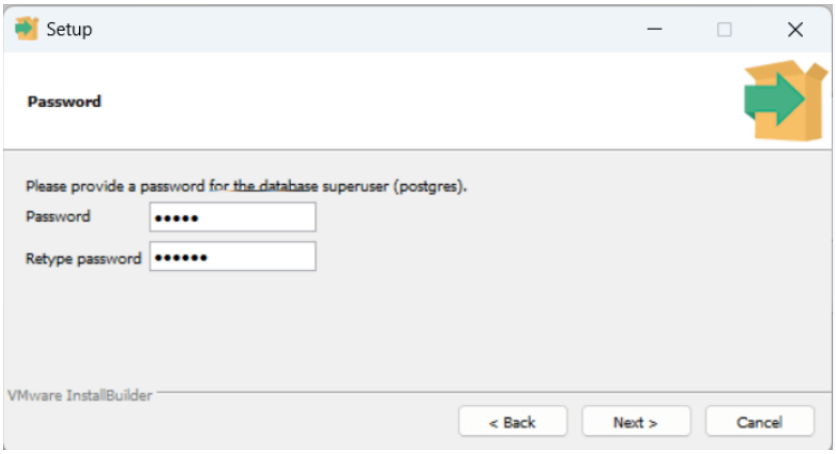
Далее будет предложено выбрать компоненты для установки:



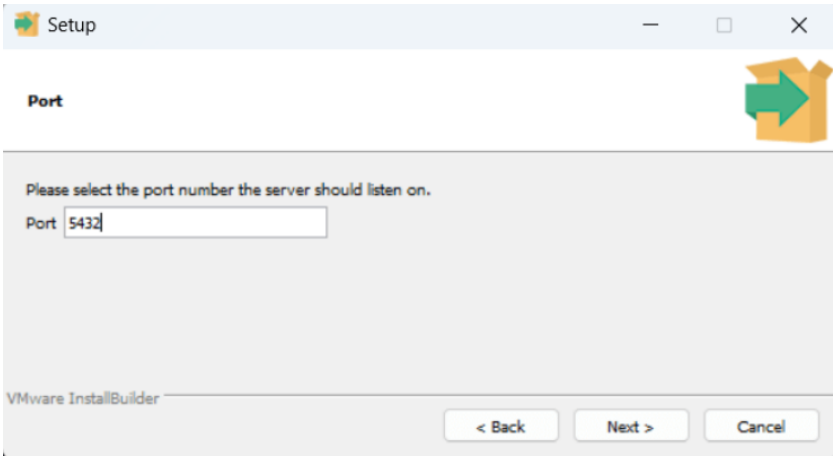
Оставим все компоненты по умолчанию и перейдем к следующему шагу. Далее будет предложено выбрать папку, где будут храниться базы данных:



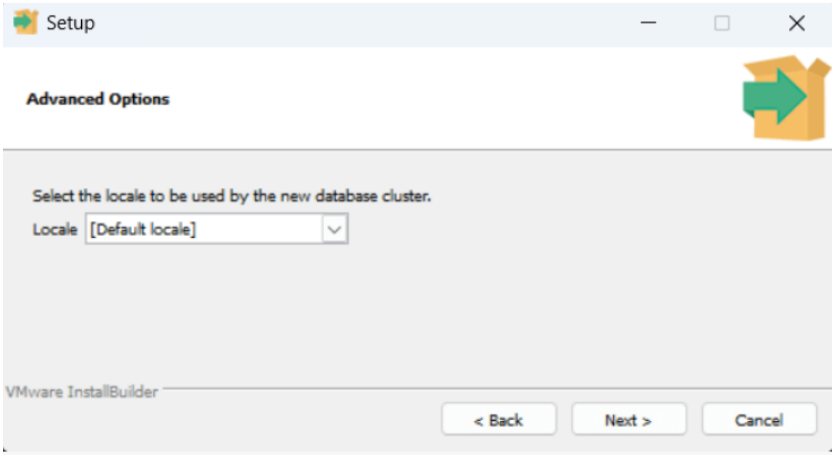
Оставим путь по умолчанию и перейдем к следующему шагу. Затем необходимо будет установить пароль для суперпользователя **postgres**:



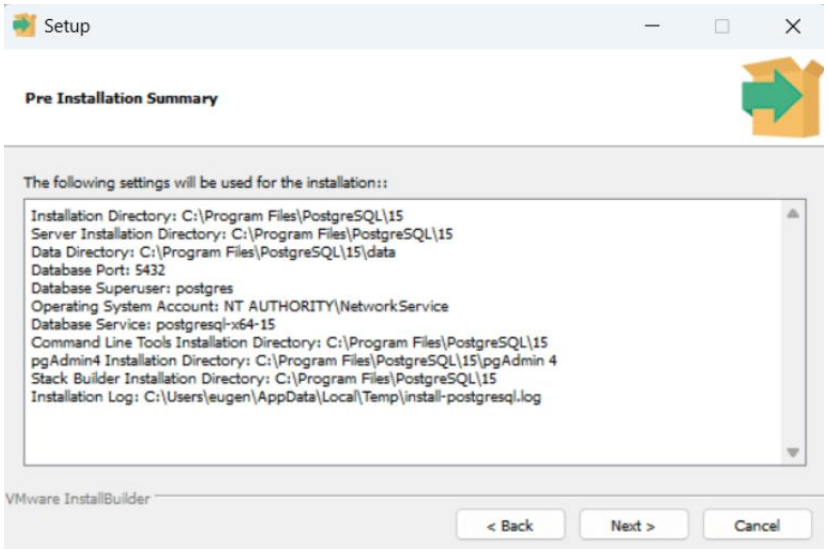
При установке запомним пароль, так как он потребуется для подключения к серверу. Затем нужно будет установить порт, по которому будет запускаться сервер. Можно оставить порт по умолчанию:



Далее можно будет установить локаль сервера. Оставим установку по умолчанию:

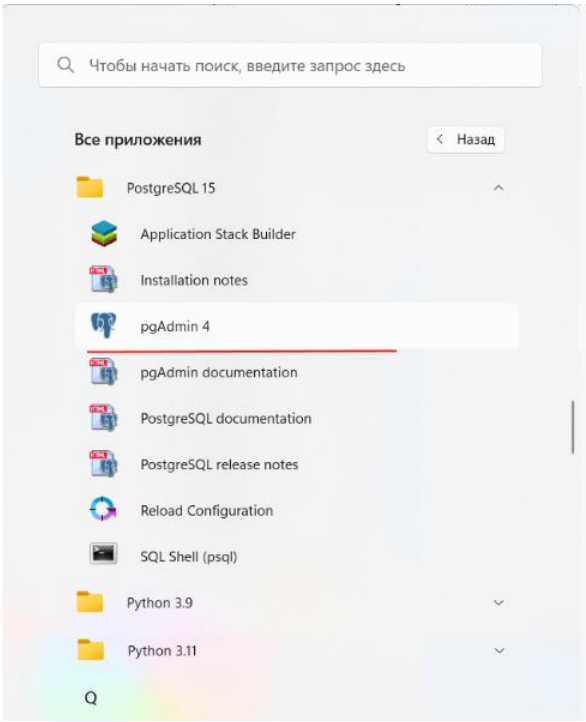


После этого мы увидим сводку по всем настройкам:

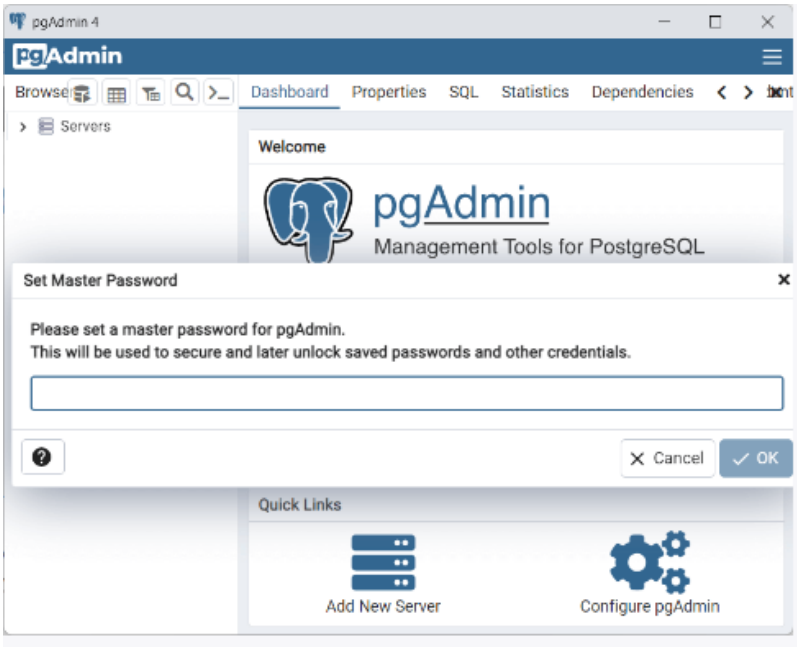


И если нас все устраивает, то можно нажать на кнопку Next, и начнется установка. И после завершения установки мы увидим следующее окно, и для выхода нажмем на кнопку Finish. Таким образом, сервер PostgreSQL установлен, и мы можем начинать с ним работать.

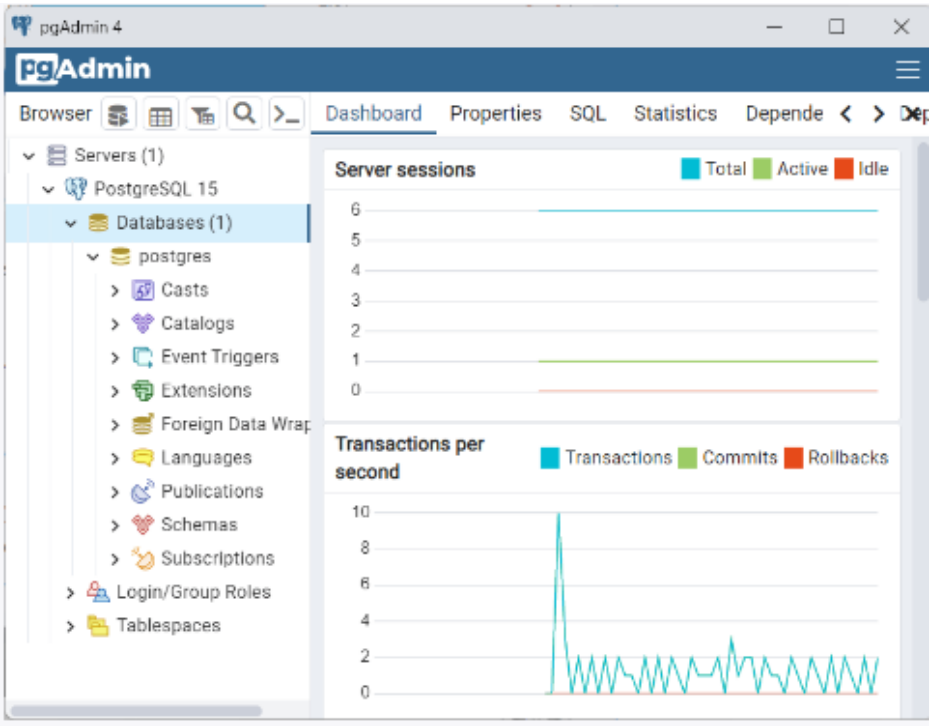
Для упрощения администрирования на сервере postgresql в базовый комплект установки входит такой инструмент как **pgAdmin**. Он представляет графический клиент для работы с сервером, через который мы в удобном виде можем создавать, удалять, изменять базы данных и управлять ими. Так, на Windows после установки мы можем найти значок pgAdmin в меню Пуск и запустить его:



После этого нам откроется следующая программа pgAdmin. При открытии также отобразится окно для ввода пароля для подключения к серверу Postgres:



Здесь необходимо ввести пароль для суперпользователя postgres, который был задан при установке PostgreSQL. После успешного логина нам откроется содержимое сервера:

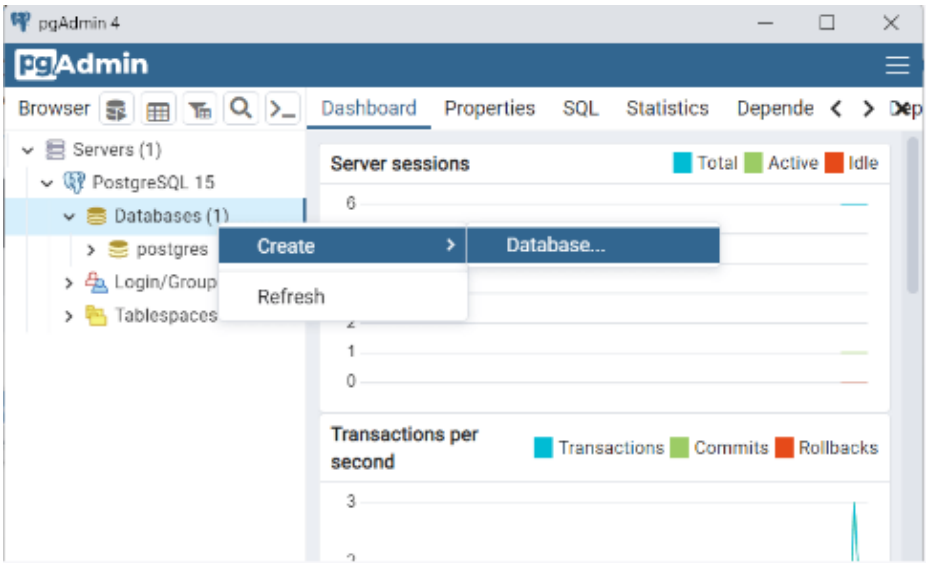


В частности, в узле **Databases** мы можем увидеть все имеющиеся базы данных. По умолчанию здесь есть только одна база данных - postgres.

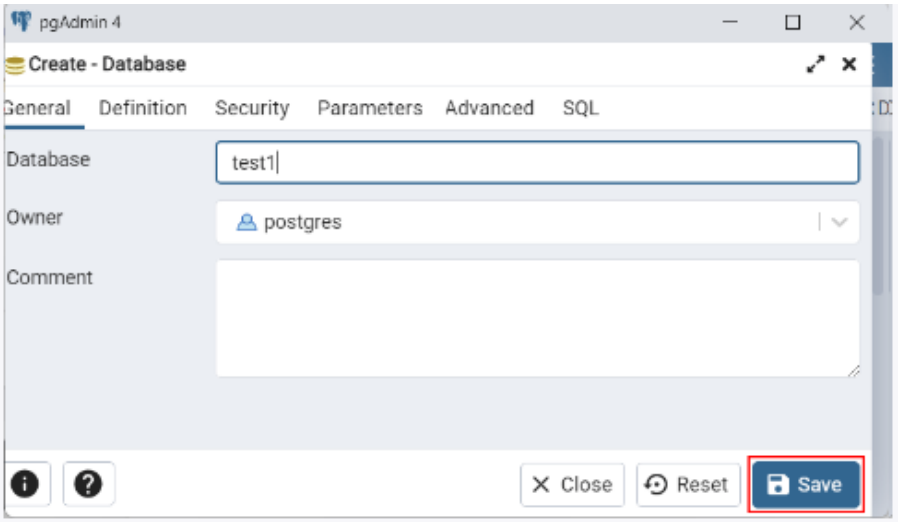
Также в правой части мы можем увидеть узел **Login/Group Roles**, который предназначен для управления пользователями и их ролями.

И третий узел - **Tablespaces** позволяет управлять местом хранения файлов баз данных.

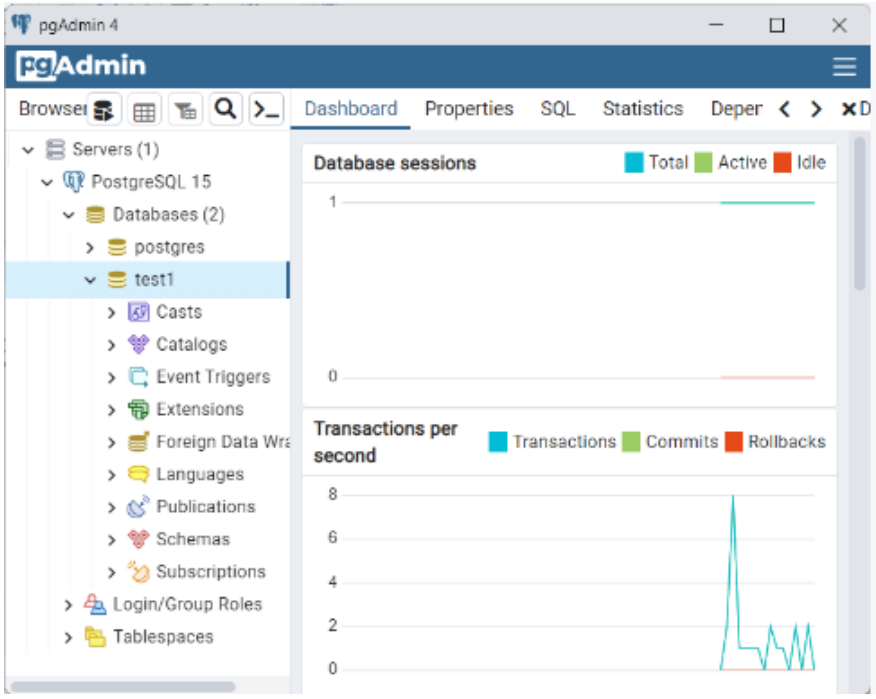
Теперь создадим свою базу данных. Для этого нажмем правой кнопкой мыши на узел **Databases**. И далее в контекстном меню выберем **Create->Database...**



После этого нам отобразится окно для создания базы данных. Введем название для БД, например, **test1** и нажмем на кнопку "Save":



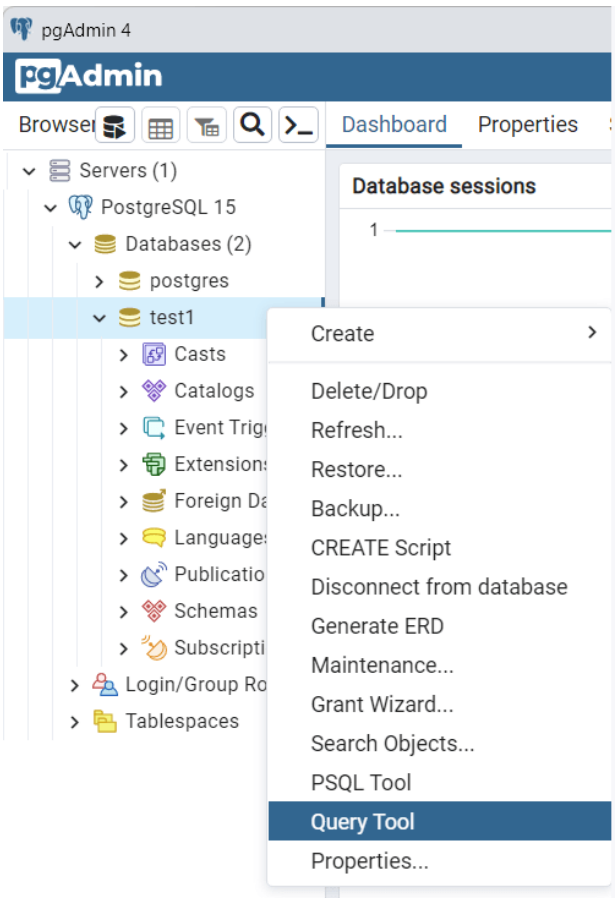
После этого в древовидном меню слева отобразится содержимое созданной базы данных test1:



Запросы SQL в pgAdmin

Как правило, работа с базой данных осуществляется с помощью специального языка запросов - SQL. Рассмотрим, как выполнять простейшие SQL-запросы к базе данных в pgAdmin.

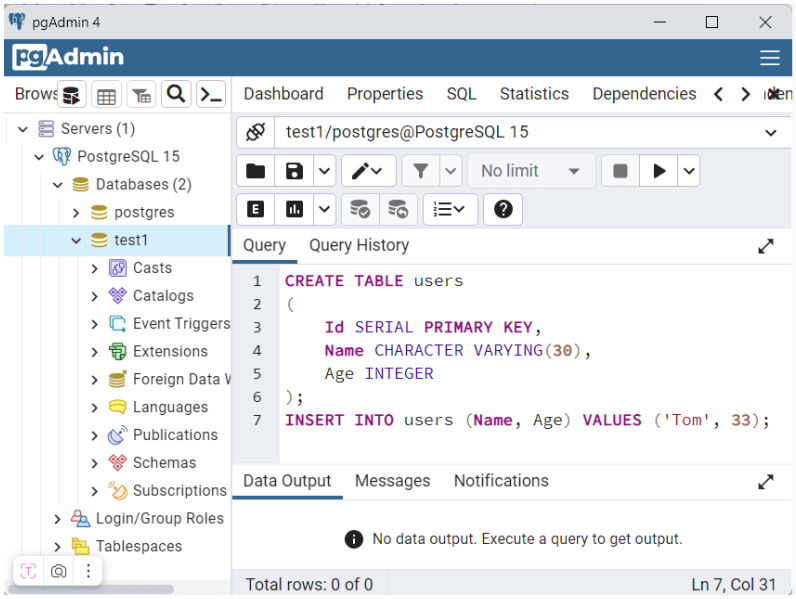
К примеру, возьмем базу данных test1, которая была создана в прошлой теме (или создадим новую) и добавим в нее таблицу и некоторые начальные данные. Для этого нажмем в правой части окна pgAdmin на базу данных правой кнопкой мыши и в появившемся контекстном меню выберем пункт **Query Tool**:



После этого в центральной части программы откроется поле для ввода кода SQL.

Введем следующий набор выражений:

```
CREATE TABLE users(
Id SERIAL PRIMARY KEY,
 Name CHARACTER VARYING(30),
Age INTEGER
);
INSERT INTO users (Name, Age) VALUES ('Tom', 33);
```

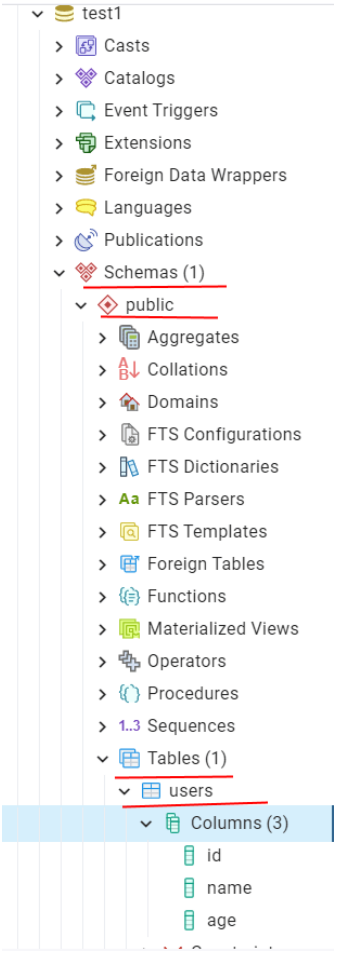


Фактически весь код разбивается на две части. Первая часть - инструкция CREATE TABLE, которая создает таблицу users с тремя столбцами Id, Name и Age. И вторая часть - инструкция INSERT, которая добавляет в таблицу одну строку.

Чтобы выполнить данные инструкции, нажмем над кодом в панели инструментов на стрелочку. И после этого в выбранную базу данных (test1) будет добавлена таблица users, в которую будет добавлена одна строка.

Впоследствии подобным образом будет выполняться любой другой код SQL к базе данных. Также выбирается нужная база данных, выбирается параметр Query Tool, и далее в поле ввода вводится код SQL, который затем выполняется.

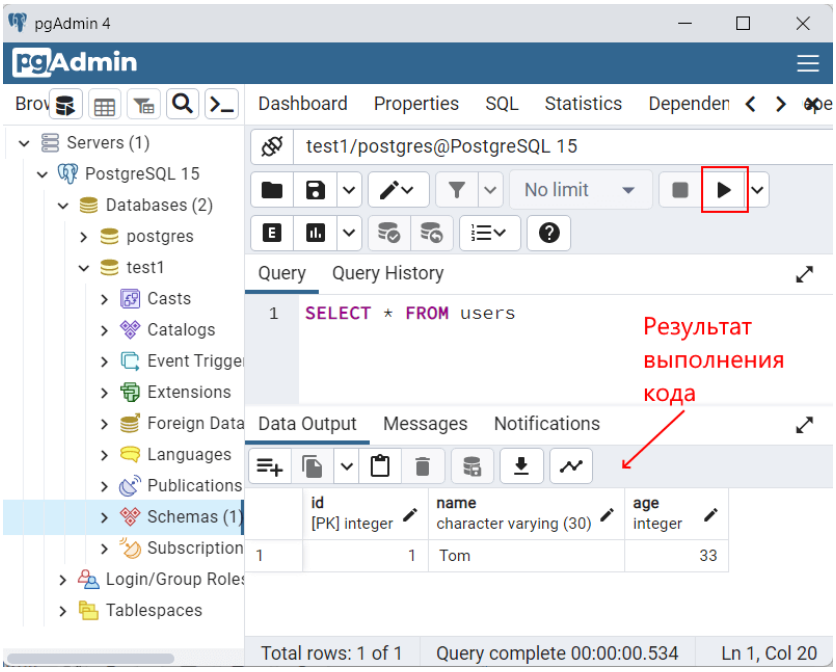
Следует отметить, что для каждой таблицы определяется схема. По умолчанию это схема "public". Поэтому чтобы найти таблицу, нам надо обратиться к узлу базы данных, раскрыть его, далее выбрать подузел **Schemas**, в нем подузел **public** (название схема), и далее в нем подузел **Tables**, который представляет все таблицы, ассоциированные со схемой public:



Теперь получим данные из таблицы, которые были добавлены при ее создании. Для этого выполним следующий код:

```
SELECT * FROM users
```

И внизу программы в поле Data Output мы увидим в табличном представлении те данные, которые ранее были добавлены.



Задание

Произвести установку PgAdmin, познакомиться с графическим клиентом, выполнить все действия, представленные в методичке.

Требования к отчёту

1. Титульная страница

   Титульная страница должна содержать:

   1. Наименование учебного учреждения
   2. Наименование предмета
   3. Фамили и инициалы студента
   4. Фамилию и инициалы преподавателя, принимающего работу
   5. Номер индивидуального задания (если имеется)
2. Цель работы
3. Выполнение задания
4. Вывод по работе

ЛАБОРАТОРНАЯ РАБОТА № 3

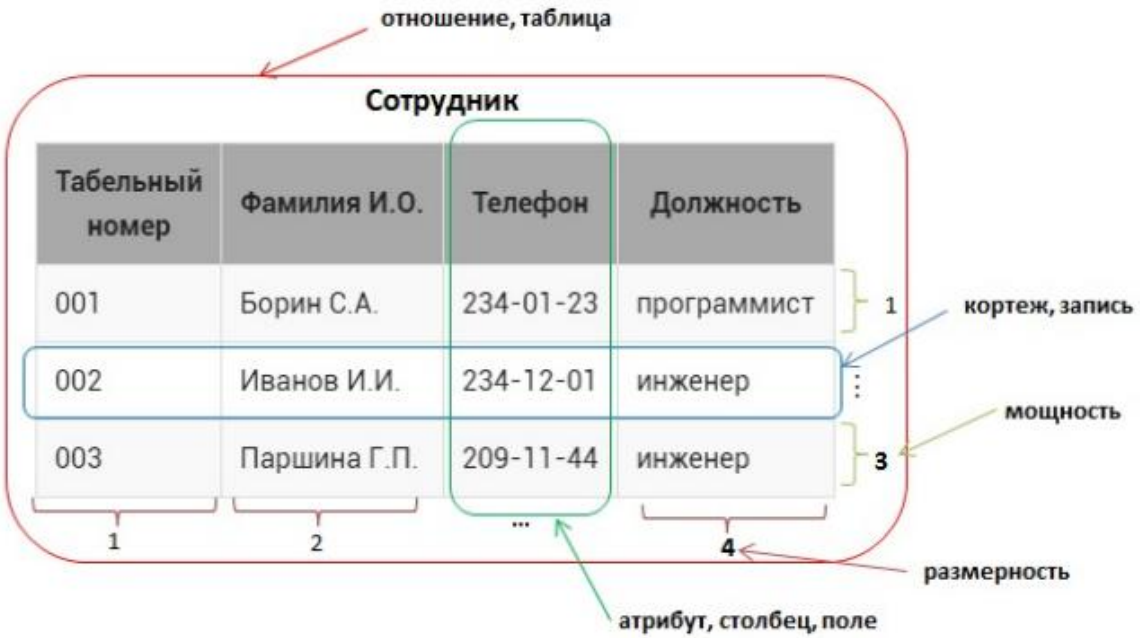
Правила создания таблиц в СУБД PostgreSQL

Цель работы: познакомится с правилами создания таблиц в СУБД PostgreSQL.

Теоретическая часть

Основные понятия реляционных баз данных

Реляционная модель была разработана в конце 1960-х годов Е.Ф. Коддом. Она определяет способ представления данных (структуру данных), методы защиты данных (целостность данных), и операции, которые можно выполнять с данными (манипулирование данными). Эта модель лежит в основе всех реляционных баз данных до настоящего времени. **Основные принципы реляционных баз данных:** все данные на концептуальном уровне представляются в виде объектов, заданных в виде строк и столбцов, называемых отношением, более распространенное название – таблица; в пересечение строки и столбца таблицы можно занести только одно значение; все операции выполняются над целыми отношениями и результатом этих операций является отношение. Пример отношения:



На примере таблицы *Сотрудник* рассмотрим **терминологию реляционных баз данных**:

- **отношение** – это структура данных целиком, набор записей (в обычном понимании – таблица), в примере –это Сотрудник;
- **кортеж** – это каждая строка, содержащая данные (более распространенный термин – запись), например, 001, Борин С. А., 234-01-23, программист, все кортежи в отношении должны быть различны;
- **мощность** – число кортежей в таблице (проще говоря, число записей\строк в таблице), в данном случае 3, мощность отношения может быть любой (от 0 до бесконечности), порядок следования кортежей(строк) - неважен;
- **атрибут** – это столбец в таблице (более распространенный термин – поле), в примере – Табельный номер, Фамилия И.О., Телефон, Должность);
- **размерность** – это число атрибутов в таблице, в данном случае – 4; размерность отношения должна быть больше 0, порядок следования атрибутов существенен;
- **домен атрибута** – это допустимые значения (неповторяющиеся), которые можно занести в поле, например для атрибута Должность домен – {инженер, программист}.

Отношение, реляционная модель

База данных, в том числе и реляционная, используется для формального описания некоторой предметной области реального мира, например, склада, учебного процесса и пр. **Обязательным этапом перед созданием базы данных является ее проектирование.**

Каждая такая таблица ассоциируется с неким информационным объектом или событием реального мира – человеком, документом, посещением и т.д.

Пример. Рассмотрим некоторый склад, на котором хранятся книги. Известно название книги, ее автор, количество экземпляров на складе и ее цена. Всю эту информацию можно представить в виде таблицы, состоящей из 4 столбцов (приведено только 4 записи, на самом деле их значительно больше)

Название	Автор	Цена, руб	Количество
Мастер и Маргарита	Булгаков М.А.	670.99	3
Белая гвардия	Булгаков М.А.	540.5	5

Название	Автор	Цена, руб	Количество
Идиот	Достоевский Ф.М.	460	10
Братья Карамазовы	Достоевский Ф.М.	799.01	2

Перед созданием таблицы в базе данных необходимо описать ее структуру. Для этого выполняется следующая последовательность шагов:

1. Дать таблице имя, пусть она будет называться book, вот некоторые **правила для выбора имен таблиц**:
    - может включать английские буквы, цифры и знак подчеркивания, должно начинаться с буквы;
    - имя должно быть уникальным в пределах базы данных.
    - Также **рекомендуется**:
        - чтобы имя было существительным в единственном числе;
        - имя должно быть понятным и соответствовать тому объекту, который оно описывает; имя должно быть как можно короче.
    - Важно. Имена таблиц по умолчанию, являются НЕ регистрозависимыми, то есть имя book и Book – одни и те же имена.
2. Определить структуру таблицы, из каких атрибутов (столбцов, полей) она будет состоять, в нашем случае это:
    1. title – поле для хранения названия книги;
    2. author – поле с фамилией автора книги;
    3. price – цена книги;

4. amount – количество книг.

5. **Правила по выбору имени поля**:

1. может включать английские буквы, цифры и знак подчеркивания, должно начинаться с буквы;

2. имя поля должно быть уникальным в пределах таблицы.

6. **Рекомендации по выбору имени поля информационного объекта:**

1. имя должно быть понятным и соответствовать тем данным, которые хранятся в поле;

2. имя может состоять из нескольких слов, тогда слова разделяются подчеркиванием, после подчеркивания слово пишется с маленькой буквы.

3. Включить ключевое поле book_id которое является ОБЯЗАТЕЛЬНЫМ ЭЛЕМЕНТОМ каждой реляционной таблицы. Ключевое поле является уникальным для каждой записи, однозначно определяет запись и в дальнейшем будет использоваться для связей с другими таблицами. **Рекомендации по именованию ключевых полей**: имя должно состоять из двух частей: начинаться с названия таблицы, которой поле принадлежит, затем через подчеркивание необходимо указать id Таким образом, наша таблица book будет выглядеть следующим образом:

book_id	title	author	price
1	Мастер и Маргарита	Булгаков М.А.	670.99

book_id	title	author	price
2	Белая гвардия	Булгаков М.А.	540.5
3	Идиот	Достоевский Ф.М.	460
4	Братья Карамазовы	Достоевский Ф.М.	799.01

Типы данных

Тип данных	Описание
`INTEGER`	Целое число.
`SERIAL`	Целое число, автоматически устанавливаемое в уникальную величину для каждой добавляемой строки. На рисунках это не отражено, но именно этот тип будет использоваться для столбцов «_id».
`CHAR`	Символьный массив фиксированного размера, размер указывается в скобках после названия типа. В столбцах данного типа PostgreSQL всегда будет сохранять ровно указанное количество символов. Если CHAR(256) служит для хранения одного символа, в базе данных будет занято как минимум 256 байт, которые и будут возвращены при извлечении данных.
`VARCHAR`	Это также символьный массив, но (как понятно из названия) переменной длины, и обычно место, занимаемое в базе данных, практически совпадает с фактическим размером сохраняемых данных. При обращении к полю `VARCHAR` возвращается то количество символов, которое и было сохранено. В скобках после названия типа указывается максимально возможная длина. Естественно возникает вопрос о том, зачем нужны оба эти типа, почему нельзя ограничиться одним `VARCHAR`? Дело в производительности. Записи фиксированной длины база данных может обрабатывать гораздо быстрее, чем записи переменной длины. Поэтому если, например, известно, что размер данных в столбце не больше четырех символов, то лучше всегда сохранять четыре символа, а не заставлять базу данных каждый раз определять размер, т. к. места выиграно немного, а вот производительность для переменной длины обычно уменьшается.
`DATE`	Позволяет хранить данные о годе, месяце и дне. Конечно же, есть и другие родственные типы, позволяющие хранить данные о времени (вместе с информацией о дате или без). Вернемся к этому позже.
`TIMESTAMP`	хранит дату и время. Занимает 8 байт. Для дат самое нижнее значение - 4713 г до н.э., самое верхнее значение - 294276 г н.э.
`TIME`	хранит время с точностью до 1 микросекунды без указания часового пояса. Принимает значения от 00:00:00 до 24:00:00. Занимает 8 байт.

Тип данных	Описание
`NUMERIC`	Возможность хранить числа с указанным количеством разрядов (первое число в скобках) и фиксированным количеством разрядов после запятой (второе число в скобках). Так, `NUMERIC(7,2)` сохранит ровно семь разрядов, два из них - после запятой.
`TEXT`	представляет текст произвольной длины.
`BOOLEAN`	может хранить одно из двух значений: true или false.

После описания структуры таблицы необходимо выбрать типы данных для каждого поля.

1. book_id - ключевой столбец, целое число, которое должно генерироваться автоматически - - SERIAL
2. 2. title - строка текста, ее длина выбирается в зависимости от данных, которые предполагается хранить в поле, предположим, что название книги не превышает 50 символов - VARCHAR (50);
3. 3. author - строка текста - VARCHAR (30);
4. 4. price - для описание денежного значения используется числовой тип данных с двумя знаками после запятой - DECIMAL (8,2);
5. 5. amount - целое число - INT.

Создание таблицы

Для создания таблицы используется SQL-запрос. В нем указывается какая таблица создается, из каких атрибутов(полей) она состоит и какой тип данных имеет каждое поле, при необходимости указывается описание полей (ключевое поле и т.д.).

Его структура:

- ключевые слова: CREATE TABLE
- имя создаваемой таблицы;
- открывающая круглая скобка «(»;
- название поля и его описание, которое включает тип поля и другие необязательные характеристики;
- запятая;
- название поля и его описание; ... закрывающая скобка «)».

Пример. Создадим таблицу genre следующей структуры:

Поле	Тип, описание
genre_id	SERIAL
name_genre	VARCHAR(30)

Запрос:

```
CREATE TABLE genre(
genre_id SERIAL,
name_genre VARCHAR (30)
);
```

Созданная таблица - пустая.

Рекомендации по записи SQL запроса

- Ключевые слова: SQL не является регистрозависимым языком (CREATE и create - одно и тоже ключевое слово).
- Ключевые слова SQL и типы данных рекомендуется записывать прописными (большими) буквами.
- Имена таблиц и полей - строчными (маленькими) буквами.
- SQL-запрос можно писать на нескольких строках.
- В конце SQL-запроса ставится точка с запятой (хотя если Вы пишете один запрос, это необязательно).

Для добавления данных применяется команда INSERT, которая имеет следующий формальный синтаксис:

```
INSERT INTO имя_таблицы (столбец1, столбец2, ... столбецN) VALUES (значение1, значение2, ... значениеN)
```

например:

```
INSERT INTO genre (name_genre) VALUES ('Фантастика');
INSERT INTO genre (name_genre) VALUES ('История');
INSERT INTO genre (name_genre) VALUES ('Роман');
INSERT INTO genre (name_genre) VALUES ('Стихи');
```

Задание

Сформулируйте SQL запрос для создания таблицы book:

Поле	Тип, описание
book_id	SERIAL

Поле	Тип, описание
title	VARCHAR(50)
author	VARCHAR(30)
price	DECIMAL(8, 2)
amount	INT

Создайте эту таблицу и добавьте в неё 4 самых любимых своих книг (у других список книг не списывать👀).

Требования к отчёту

1. Титульная страница

   Титульная страница должна содержать:

   1. Наименование учебного учреждения
   2. Наименование предмета
   3. Фамили и инициалы студента
   4. Фамилию и инициалы преподавателя, принимающего работу
   5. Номер индивидуального задания (если имеется)
2. Цель работы
3. Выполнение задания
4. Вывод по работе

ЛАБОРАТОРНАЯ РАБОТА № 4

Продвинутая выборка данных из таблиц

Цель работы: изучить следующие темы:

- Выборка данных из отдельных столбцов и присвоение им имён.
- Создание вычисляемых столбов.
- Математические функции.
  - CEILING;
  - ROUND;
  - FLOOR;
  - POWER;
  - SQRT;
  - DEGREES;
  - RADIANS;
  - ABS;

- PI.

- Оператор **IF**.
- Выборка данных по условию, оператор **WHERE**.
- Логические операции **AND**, **OR**, **NOT**.
- Операторы **BETWEEN**, **IN**.
- Сортировка данных **ORDER BY**.
- Оператор **LIKE**.

Теоретическая часть

Схема таблицы и данные

Схема таблицы

```
CREATE TABLE IF NOT EXISTS book (
book_id SERIAL PRIMARY KEY,
title VARCHAR (50),
author VARCHAR (30),
price DECIMAL (8, 2),
amount INT
);
```

запросы вставки

```
INSERT INTO book(title,author,price,amount) VALUES ('Бесы','Достоевский Ф.М.',800,3);
INSERT INTO book(title,author,price,amount) VALUES ('Игрок','Достоевский Ф.М.',700,8);
INSERT INTO book(title,author,price,amount) VALUES ('Война и мир','Толстой Л.Н.',1000,3);
INSERT INTO book(title,author,price,amount) VALUES ('Анна Каренина','Толстой Л.Н.',1200,7);
INSERT INTO book(title,author,price,amount) VALUES ('Отцы и дети','Тургенев И.С.',650,8);
INSERT INTO book(title,author,price,amount) VALUES ('Обломов','Гончаров И.А.',900,2);
INSERT INTO book(title,author,price,amount) VALUES ('Евгений Онегин','Пушкин А.С.',1000,3);
INSERT INTO book(title,author,price,amount) VALUES ('Собачье сердце','Булгаков М.А.',750,6);
INSERT INTO book(title,author,price,amount) VALUES ('Капитанская дочь','Пушкин А.С.',1500,1);
```

```
INSERT INTO book(title,author,price,amount) VALUES ('Мёртвые души','Гоголь Н.В.',900,2);
INSERT INTO book(title,author,price,amount) VALUES ('Ревизор','Гоголь Н.В.',500,4);
INSERT INTO book(title,author,price,amount) VALUES ('Шинель','Гоголь Н.В.',645,2);
INSERT INTO book(title,author,price,amount) VALUES ('Палата номер 6','Чехов А.П.',500,10);
INSERT INTO book(title,author,price,amount) VALUES ('Вишнёвый сад','Чехов А.П.',400,12);
INSERT INTO book(title,author,price,amount) VALUES ('Мастер и Маргарита','Булгаков М.А.',555,5);
INSERT INTO book(title,author,price,amount) VALUES ('Белая гвардия','Булгаков М.А.',666,6);
INSERT INTO book(title,author,price,amount) VALUES ('Идиот','Достоевский Ф.М.',777,7);
INSERT INTO book(title,author,price,amount) VALUES ('Братья Карамазовы','Достоевский Ф.М.',888,9);
INSERT INTO book(title,author,price,amount) VALUES ('Преступление и наказание','Достоевский Ф.М.',452,7);
INSERT INTO book(title,author,price,amount) VALUES ('Обломов','Гончаров И.А.',365,3);
INSERT INTO book(title,author,price,amount) VALUES ('Стихотворения и поэма','Есенин С.А.',578,6);
```

Выборка всех данных из таблицы

Для того чтобы отобрать все данные из таблицы используется SQL запрос следующей структуры:

- ключевое слово SELECT;
- символ «*»;
- ключевое слово FROM;
- имя таблицы.

Результатом является таблица, в которую включены все строки и столбцы указанной в запросе таблицы.

Пример

```
SELECT * FROM book
```

Выборка отдельных столбцов

Для того чтобы отобрать данные из определенных столбцов таблицы используется SQL запрос следующей структуры:

- ключевое слово SELECT;
- список столбцов таблицы через запятую;
- ключевое слово FROM;
- имя таблицы.

Результатом является таблица, в которую включены все данные из указанных после SELECT столбцов исходной таблицы.

Пример

```
SELECT title, author FROM book
```

Выборка отдельных столбцов и присвоение им имен (алиасов)

Для того чтобы отобрать данные из определенных столбцов таблицы и одновременно задать столбцам другие имена (псевдонимы) используется SQL запрос следующей структуры:

- ключевое слово SELECT;
- имя столбца; ключевое слово AS;
- название столбца - псевдоним (можно русскими буквами), но это должно быть одно слово, если название состоит из двух слов – соединяйте их подчеркиванием, например, Количество_книг; ...
- ключевое слово FROM;
- имя таблицы.

Пример

```
SELECT
title as 'Название книги',
author as Автор_книги
FROM book;
```

Выборка данных с созданием вычисляемого столбца

С помощью SQL запросов можно осуществлять вычисления по каждой строке таблицы с помощью вычисляемого столбца. Для него в списке полей после оператора SELECT указывается выражение и задается имя.

Выражение может включать имена столбцов, константы, знаки операций, встроенные функции.

Результатом является таблица, в которую включены все данные из указанных после SELECT столбцов, а также новый столбец, в каждой строке которого вычисляется заданное выражение.

Пример

```
SELECT
title,
author,
price,
amount,
price * amount AS total
FROM book;
```

Вычисляемое значение может быть и строкой:

Пример

```
SELECT
'Название книги скрыто' AS title,
author,
price,
amount,
price * amount AS total
FROM book;
```

Выборка данных, вычисляемые столбцы, математические функции

Общематематические функции

- log(B,X) логарифм
- ln(X) натуральый логарифм
- exp(X) экспонента в степени X
- abs(X) модуль числа X
- sqrt(x) квадратный корень из X
- power(X, Y) возведение X в степень Y
- mod(X,Y) возвращает остаток деления числа X на Y

Тригометрические функции

- PI() pi = 3.1415926...
- cos(X) косинус X. Результат в радианах.
- sin(X) синус X. Результат в радианах.
- tan(X) тангенс X. Результат в радианах.
- acos(X) арккосинус X. Результат в радианах.
- asin(X) арксинус X. Результат в радианах.
- atan(X) арктангенс X. Результат в радианах.
- degrees(X) конвертирует значение X из радиан в градусы

- radians(X) конвертирует значение X из градусов в радианы

В конце года цену всех книг на складе пересчитывают – снижают ее на 30%. Написать SQL запрос, который из таблицы book выбирает названия, авторов, количества и вычисляет новые цены книг. Столбец с новой ценой назвать new_price, цену округлить до 2-х знаков после запятой.

Пример

```
SELECT
title,
author,
amount,
price AS old_price,
round (price*0.7, 2) AS new_price
FROM book;
```

Выборка данных, вычисляемые столбцы, логические функции

В SQL реализована возможность заносить в поле значение в зависимости от условия. Для этого используется конструкция типа CASE WHEN THEN ELSE END:

```
CASE
WHEN логическое_выражение_1 THEN возвращаемое_значение_1
WHEN логическое_выражение_2 THEN возвращаемое_значение_2
ELSE возвращаемое_значение_если_все_логические_выражения_ложные
END
```

Пример

```
-- Для каждой книги из таблицы book установим скидку следующим образом:
-- если количество книг меньше или равно 4, то скидка будет составлять 50% от цены,
-- если меньше или равно 8 то скидка 30%, иначе скидки нет.
SELECT
title,
author,
amount,
price AS old_price,
CASE
WHEN amount<=4 THEN price*0.5
WHEN amount<=8 THEN price*0.7
ELSE price
```

```
END AS new_price
FROM book;
```

Выборка данных по условию

С помощью запросов можно включать в итоговую выборку не все строки исходной таблицы, а только те, которые отвечают некоторому условию. Для этого после указания таблицы, откуда выбираются данные, задается ключевое слово WHERE и логическое выражение, от результата которого зависит будет ли включена строка в выборку или нет. Если условие – истина, то строка(запись) включается в выборку, если ложь – нет. Логическое выражение может включать операторы сравнения (равно «=», не равно «<>», больше «>», меньше «=», меньше или равно «<=») и выражения, допустимые в SQL

Пример

```
-- Вывести название, автора и стоимость (цена умножить на количество) тех книг,
-- стоимость которых больше 4000 рублей
SELECT
title,
author,
price * amount AS total
FROM book
WHERE price * amount > 4000
```

В логическом выражении после WHERE нельзя использовать названия столбцов, присвоенные им с помощью AS , так как при выполнении запроса сначала вычисляется логическое выражение для каждой строки исходной таблицы, выбираются строки, для которых оно истинно. А только после этого формируется "шапка запроса" – столбцы, включаемые в запрос.

Выборка данных, логические операции

Логическое выражение после ключевого слова WHERE кроме операторов сравнения и выражений может включать логические операции (И AND , ИЛИ OR , НЕ NOT) и круглые скобки, изменяющие приоритеты выполнения операций.

Приоритеты операций:

1. круглые скобки
2. умножение (*), деление (/)
3. сложение (+), вычитание (-)
4. операторы сравнения (=, >, =, <=, <>)
5. NOT
6. AND

7. OR

Вывести название, автора, цену и количество всех книг, цена которых от 500 до 600 рублей

Пример

```
SELECT title, author, price, amount
FROM book
WHERE price >= 500 AND price <= 600;
```

Выборка данных, операторы BETWEEN, IN

Логическое выражение после ключевого слова WHERE может включать операторы BETWEEN и IN . Приоритет у этих операторов такой же как у операторов сравнения, то есть они выполняются раньше, чем NOT , AND , OR .

Оператор BETWEEN позволяет отобрать данные, относящиеся к некоторому интервалу, включая его границы.

Выбрать названия и количества тех книг, количество которых от 5 до 14 включительно.

Пример

```
SELECT title, amount
FROM book
WHERE amount BETWEEN 5 AND 14;
```

Пример

```
-- Выбрать названия и цены книг, написанных Булгаковым или Достоевским.
SELECT title, author, price
FROM book
WHERE author IN ('Булгаков М.А.', 'Достоевский Ф.М.');
```

Выборка данных с сортировкой

При выборке можно указывать столбец или несколько столбцов, по которым необходимо отсортировать отобранные строки. Для этого используются ключевое слово ORDER BY , после которого задаются имена столбцов. При этом строки сортируются по первому столбцу, если указан второй столбец, сортировка осуществляется только для тех строк, у которых значения первого столбца одинаковы. По умолчанию ORDER BY выполняет сортировку по возрастанию. Чтобы управлять направлением сортировки вручную, после имени столбца указывается ключевое слово ASC (по возрастанию, образовано от англ. in ascending order) или DESC (по убыванию, образовано от англ. in descending order).

Логический порядок операций для запроса SQL следующий:

1. FROM
2. WHERE
3. SELECT
4. ORDER BY

Поскольку сортировка выполняется позже SELECT, для указания столбцов, по которым выполняется сортировка, можно использовать имена, присвоенные им после SELECT, а также порядковый номер столбца в перечислении.

Пример

```
-- Вывести книги от самых дешёвых, до самых дорогих.
SELECT title, author, price
FROM book
ORDER BY price ASC
```

Пример

```
-- Вывести произведения отсортированные по длине названия, от самых коротких, до длинных
SELECT title, author
FROM book
ORDER BY length(title) ASC
```

Выборка данных, оператор LIKE

Оператор LIKE используется для сравнения строк. В отличие от операторов отношения равно (=) и не равно (<>), LIKE позволяет сравнивать строки не на полное совпадение (не совпадение), а в соответствии с шаблоном. Шаблон может включать обычные символы и символы-шаблоны. При сравнении с шаблоном, его обычные символы должны в точности совпадать с символами, указанными в строке. Символы-шаблоны могут совпадать с произвольными элементами символьной строки.

- % Любая строка, содержащая ноль или более символов
- _ (подчеркивание) Любой одиночный символ

Вывести названия книг, начинающихся с буквы «Б».

Пример

```
SELECT title
FROM book
WHERE title LIKE 'Б%'
```

```
-- Вывести название книг, состоящих ровно из 5 букв.
SELECT title
FROM book
WHERE title LIKE '_____'
```

или, что аналогично, так:

```
SELECT title
FROM book
WHERE length(title)==5;
```

Пример

```
-- Вывести названия книг, которые содержат букву "и" как отдельное слово,
-- если считать, что слова в названии отделяются друг от друга пробелами
-- и не содержат знаков препинания. Отсортировать по длине названия книги.
SELECT title
FROM book
WHERE title LIKE '_% и _%'
OR title LIKE 'и _%'
OR title LIKE '_% и'
OR title LIKE 'и'
ORDER BY length(title)
```

Варианты индивидуальных заданий

Вариант 1

Задание Магазин счёл, что классика уже не пользуется популярностью, поэтому необходимо в выборке: 1. Сменить всех авторов на "Джоан Роулинг". 2. К названию каждой книги в начале дописать "Гарри Поттер и". 3. Цену поднять на 42% для книг, количество которых меньше 5, иначе поднять цену на 64%. 4. В запросе вывести автора книги, название, старую цену, новую цену и количество книг на складе. 5. Отсортировать по названию книги и по убыванию цены (новой).

Вариант 2

Задание. На складе магазина затопило стеллаж с книгами авторов на букву "Т". Руководство решило продать их по сниженной цене со скидкой 80%. Необходимо вывести всех авторов на букву "Т", их книги, количество, старую цену, цену со скидкой и рассчитать суммарные потери от продажи с такой скидкой (с учетом количества на остатке). Вывести в порядке названия книг по алфавиту.

Вариант 3

Задание В магазине книг решили организовать акцию: при заказе одной книги любого из трех авторов покупатель получает подарок: если это Булгаков М.А. - плюшевую собаку. Если это Достоевский Ф.М. - топор. Если это Есенин С.А. - веревка и мыло. Если это Чехов А.П. - окуляры иначе подарка нет. При этом для каждого вышеперечисленного автора увеличилась цена продаваемой книги на 25%. Сделать SQL запрос для задания выше. Вывести название книги, автора, подарок, и новую цену книги. Результат отсортировать по названию книги.

Требования к отчёту

1. Титульная страница

   Титульная страница должна содержать:

   1. Наименование учебного учреждения
   2. Наименование предмета
   3. Фамили и инициалы студента
   4. Фамилию и инициалы преподавателя, принимающего работу
   5. Номер индивидуального задания (если имеется)
2. Цель работы
3. Выполнение задания
4. Вывод по работе

ЛАБОРАТОРНАЯ РАБОТА № 5

Связанные таблицы и соединение таблиц

Цель работы: изучить тему запросов связанных таблиц и соединение таблиц SQL. Научится применять их на практике.

Теоретическая часть

Связи между таблицами

Связь *«один ко многим»* имеет место, когда одной записи главной таблицы соответствует несколько записей связанной таблицы, а каждой записи связанной таблицы соответствует только одна запись главной таблицы. Обозначается это так:

Автор ---> Книга

Связь *«многие ко многим»* имеет место, когда каждой записи одной таблицы соответствует несколько записей во второй, и наоборот, каждой записи второй таблицы соответствует несколько записей в первой. Обозначается это так:

Автор <---> Книга

Разделим монолитную таблицу book на две

В этой таблице фамилии авторов повторяются для нескольких книг. А что, если придется вместо инициалов для каждого автора хранить его полное имя и отчество? Тогда, если в таблице содержится информация о 50 книгах Достоевского, придется 50 раз исправлять «Ф.М.» на «Федор Михайлович». При этом, если в некоторых записях использовать «Фёдор Михайлович» (с буквой ё), то мы вообще получим двух разных авторов...

Чтобы устранить эту проблему в реляционных базах данных создается новая таблица author, в которой перечисляются все различные авторы, а затем эта таблица связывается с таблицей book. При этом такая связь называется «один ко многим», таблица author называется главной, таблица book – связанной или подчиненной.

Создадим таблицу author

```
CREATE TABLE IF NOT EXISTS book_original (
book_id INTEGER SERIAL PRIMARY KEY,
title VARCHAR(50),
name_author VARCHAR(50),
price DECIMAL(8, 2),
amount INT
);
```

Вставим данные в таблицу

```
INSERT INTO book_original(title,name_author,price,amount) VALUES ('Бесы','Достоевский Ф.М.',800,3);
INSERT INTO book_original(title,name_author,price,amount) VALUES ('Игрок','Достоевский Ф.М.',700,8);
INSERT INTO book_original(title,name_author,price,amount) VALUES ('Война и мир','Толстой Л.Н.',1000,3);
INSERT INTO book_original(title,name_author,price,amount) VALUES ('Анна Каренина','Толстой Л.Н.',1200,7);
INSERT INTO book_original(title,name_author,price,amount) VALUES ('Отцы и дети','Тургенев И.С.',650,8);
INSERT INTO book_original(title,name_author,price,amount) VALUES ('Обломов','Гончаров И.А.',900,2);
INSERT INTO book_original(title,name_author,price,amount) VALUES ('Евгений Онегин','Пушкин А.С.',1000,3);
INSERT INTO book_original(title,name_author,price,amount) VALUES ('Собачье
```

```
сердце','Булгаков М.А.',750,6);
INSERT INTO book_original(title,name_author,price,amount) VALUES ('Капитанская дочь','Пушкин А.С.',1500,1);
INSERT INTO book_original(title,name_author,price,amount) VALUES ('Мёртвые души','Гоголь Н.В.',900,2);
INSERT INTO book_original(title,name_author,price,amount) VALUES ('Ревизор','Гоголь Н.В.',500,4);
INSERT INTO book_original(title,name_author,price,amount) VALUES ('Шинель','Гоголь Н.В.',645,2);
INSERT INTO book_original(title,name_author,price,amount) VALUES ('Палата номер 6','Чехов А.П.',500,10);
INSERT INTO book_original(title,name_author,price,amount) VALUES ('Вишнёвый сад','Чехов А.П.',400,12);
INSERT INTO book_original(title,name_author,price,amount) VALUES ('Мастер и Маргарита','Булгаков М.А.',555,5);
INSERT INTO book_original(title,name_author,price,amount) VALUES ('Белая гвардия','Булгаков М.А.',666,6);
INSERT INTO book_original(title,name_author,price,amount) VALUES ('Идиот','Достоевский Ф.М.',777,7);
INSERT INTO book_original(title,name_author,price,amount) VALUES ('Братья Карамазовы','Достоевский Ф.М.',888,9);
INSERT INTO book_original(title,name_author,price,amount) VALUES ('Преступление и наказание','Достоевский Ф.М.',452,7);
INSERT INTO book_original(title,name_author,price,amount) VALUES ('Обломов','Гончаров И.А.',365,3);
INSERT INTO book_original(title,name_author,price,amount) VALUES ('Стихотворения и поэма','Есенин С.А.',578,6);
```

Создадим таблицу авторов

```
CREATE TABLE author (
author_id SERIAL PRIMARY KEY,
name_author VARCHAR (50)
);
```

Вставим данные.

```
INSERT INTO author(name_author)
SELECT DISTINCT name_author
FROM book_original;
```

Сразу после создания таблицы author, мы вставили туда всех авторов из таблицы book, которую мы переименовали book_original, чтобы авторы не повторялись перед атрибутом name_author указано ключевое слово DISTINCT.

Создание таблицы с внешними ключами

При создании зависимой таблицы (таблицы, которая содержит внешние ключи) **необходимо учитывать**, что:

- каждый внешний ключ должен иметь такой же тип данных, как связанное поле главной таблицы (в наших примерах это INT);
- необходимо указать главную для нее таблицу и столбец, по которому осуществляется связь:

```
FOREIGN KEY (связанное_поле_зависимой_таблицы)
REFERENCES главная_таблица (связанное_поле_главной_таблицы)
```

По умолчанию любой столбец, кроме ключевого, может содержать значение NULL. При создании таблицы это можно переопределить, используя ограничение NOT NULL для этого столбца:

```
CREATE TABLE таблица (
столбец_1 INT NOT NULL,
столбец_2 VARCHAR (10)
);
```

В созданной таблице в столбец_1 не может содержать пустое значение, а столбец_2 - может. Для внешних ключей рекомендуется устанавливать ограничение NOT NULL

Создадим таблицу book

```
CREATE TABLE book (
book_id SERIAL PRIMARY KEY,
title VARCHAR (50),
price DECIMAL (8,2),
amount INT,
author_id INT NOT NULL,
FOREIGN KEY (author_id) REFERENCES author (author_id)
);
```

Вставим данные из оригинальной таблицы

```
INSERT INTO book(
title,
price,
amount,
author_id)
SELECT
title,
price,
amount,
(SELECT author_id FROM author WHERE author.name_author = book_original.name_author)
FROM book_original;
```

Так мы разделили монолитную таблицу book_original на book и author. В таблицу book добавлен внешний ключ на таблицу author. После создания таблицы book туда перенесены все значения из book_original, чтобы вставить внешний ключ мы использовали вложенный запрос, который искал первичный ключ автора в таблице author, которому соответствовало поле name_author из book_original.

Действия при удалении записи главной таблицы

С помощью выражения ON DELETE можно установить действия, которые выполняются для записей подчиненной таблицы при удалении связанной строки из главной таблицы. При удалении можно установить следующие опции:

- CASCADE: автоматически удаляет строки из зависимой таблицы при удалении связанных строк в главной таблице.
- SET NULL: при удалении связанной строки из главной таблицы устанавливает для столбца внешнего ключа значение NULL. (В этом случае столбец внешнего ключа должен поддерживать установку NULL).
- SET DEFAULT похоже на SET NULL за тем исключением, что значение внешнего ключа устанавливается не в NULL, а в значение по умолчанию для данного столбца.
- RESTRICT: отклоняет удаление строк в главной таблице при наличии связанных строк в зависимой таблице.

Пример

Будем считать, что при удалении автора из таблицы author, необходимо удалить все записи о книгах из таблицы book, написанные этим автором. Данное действие необходимо прописать при создании таблицы.

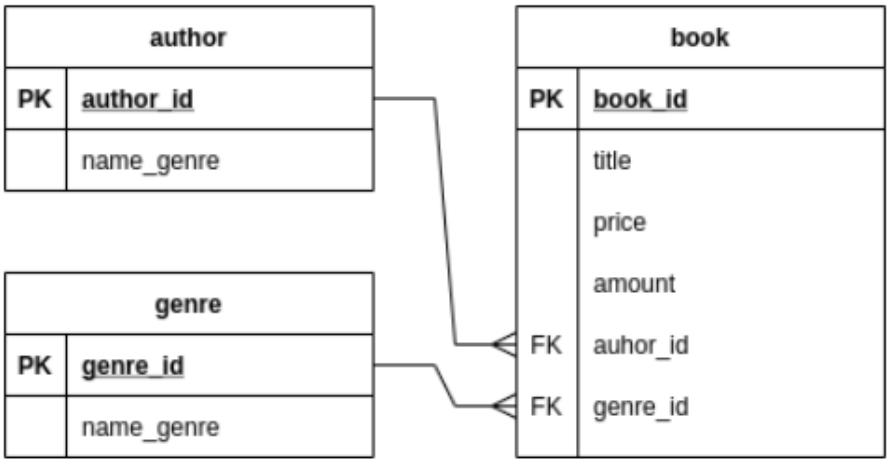
Запрос:

```
CREATE TABLE book (
book_id SERIAL PRIMARY KEY,
title VARCHAR (50),
author_id INT NOT NULL,
price DECIMAL (8,2),
amount INT, FOREIGN KEY (author_id) REFERENCES author (author_id) ON DELETE
CASCADE
);
```

Создадим таблицу genre

```
/*Создадим таблицу genre*/
CREATE TABLE genre (
genre_id SERIAL PRIMARY KEY,
name_genre VARCHAR(30)
);
/* Вставим данные в таблицу genre*/
INSERT INTO genre(name_genre) VALUES ('Роман');
INSERT INTO genre(name_genre) VALUES ('Приключение');
INSERT INTO genre(name_genre) VALUES ('Сказка');
INSERT INTO genre(name_genre) VALUES ('Документалка');
INSERT INTO genre(name_genre) VALUES ('Биография');
/* Вставим новый внешний ключ указывающий на таблицу genre для таблицы book */
ALTER TABLE book ADD genre_id INT;
ALTER TABLE book ADD CONSTRAINT fk_genre_id FOREIGN KEY (genre_id)
REFERENCES genre(genre_id);
/*Вставим случайные значения внешних ключей в таблицу genre*/
UPDATE book
SET genre_id = FLOOR( 1 + RAND( ) * (SELECT COUNT(*) FROM genre) );
```

В итоге мы получили вот такую схему БД:



Соединение INNER JOIN

Оператор внутреннего соединения INNER JOIN соединяет две таблицы. Порядок таблиц для оператора неважен, поскольку оператор является симметричным.

```
SELECT
...
FROM
таблица_1 INNER JOIN таблица_2
ON условие
...
```

Результат запроса формируется так:

- каждая строка одной таблицы сопоставляется с каждой строкой второй таблицы;
- для полученной «соединённой» строки проверяется условие соединения;
- если условие истинно, в таблицу результата добавляется соответствующая «соединённая» строка;

Вывести название книг и их авторов.

Запрос:

```
SELECT title, name_author
FROM
author INNER JOIN book
ON author.author_id = book.author_id;
```

Поскольку поля author_id в таблицах book и author называются одинаково, необходимо в запросах указывать полную ссылку на них (book.author_id и author.author_id).

Внешнее соединение LEFT и RIGHT OUTER JOIN

Оператор внешнего соединения LEFT OUTER JOIN (можно использовать LEFT JOIN) соединяет две таблицы. **Порядок таблиц для оператора важен, поскольку оператор не является симметричным.**

```
SELECT
 ...
FROM
таблица_1 LEFT JOIN таблица_2
ON условие
 ...
```

Результат запроса формируется так:

- в результат включается внутреннее соединение (INNER JOIN) первой и второй таблицы в соответствии с условием;
- затем в результат добавляются те записи первой таблицы (левой), которые не вошли во внутреннее соединение на шаге 1, для таких записей соответствующие поля второй таблицы заполняются значениями NULL.

Соединение RIGHT JOIN действует аналогично, только в пункте 2 первая таблица меняется на вторую и наоборот.

Пример

Вывести название всех книг каждого автора, если книг некоторых авторов в данный момент нет на складе – вместо названия книги указать Null.

Запрос:

```
SELECT name_author, title
FROM author LEFT JOIN book
ON author.author_id = book.author_id
ORDER BY name_author;
```

Так как в таблице book нет книг Лермонтова, напротив этой фамилии стоит Null.

Перекрестное соединение CROSS JOIN

Оператор перекрёстного соединения, или декартова произведения CROSS JOIN (в запросе вместо ключевых слов можно поставить запятую между таблицами) соединяет две таблицы. Порядок таблиц для оператора неважен, поскольку оператор является симметричным. Его структура:

```
SELECT
 ...
```

```
FROM
таблица_1 CROSS JOIN таблица_2
 ...
```

Или

```
SELECT
 ...
FROM таблица_1, таблица_2
 ...
```

Результат запроса формируется так: каждая строка одной таблицы соединяется с каждой строкой другой таблицы, формируя в результате все возможные сочетания строк двух таблиц.

Например, запрос:

```
SELECT name_author, name_genre
FROM author, genre;
```

каждому автору из таблицы author поставит в соответствие все возможные жанры из таблицы genre.

Варианты индивидуальных заданий

Вариант 1

Вывести название, жанр и цену тех книг, количество которых больше 8, в отсортированном по убыванию цены виде.

Вариант 2

Вывести, сколько книг написано в каждом жанре и сколько экземпляров каждого жанра на складе.

Требования к отчёту

1. Титульная страница

   Титульная страница должна содержать:

   1. Наименование учебного учреждения
   2. Наименование предмета
   3. Фамили и инициалы студента
   4. Фамилию и инициалы преподавателя, принимающего работу
   5. Номер индивидуального задания (если имеется)
2. Цель работы
3. Выполнение задания
4. Вывод по работе

ЛАБОРАТОРНАЯ РАБОТА № 6

Групповые операции

Цель работы: изучить тему вычислений для групп элементов и использование агрегатных функций в реляционных БД.

Теоретическая часть

GROUP BY

Оператор GROUP BY служит для распределения строк - результата запроса - по группам, в которых значения некоторого столбца, по которому происходит группировка, являются одинаковыми. Группировку можно производить как по одному столбцу, так и по нескольким.

Часто оператор GROUP BY применяется вместе с агрегатными функциями (COUNT, SUM, AVG, MAX, MIN). В этих случаях агрегатные функции служат для вычисления соответствующего агрегатного значения ко всему набору строк, для которых некоторый столбец - общий.

Выбор уникальных элементов столбца

Чтобы отобрать уникальные элементы некоторого столбца используется ключевое слово DISTINCT, которое размещается перед целевым столбцом.

```
SELECT DISTINCT author
FROM book
ORDER BY 1 -- сортируем по первому столбцу
```

Другой способ – использование оператора GROUP BY, который группирует данные при выборке, имеющие одинаковые значения в некотором столбце. Столбец, по которому осуществляется группировка, указывается после GROUP BY.

С помощью GROUP BY можно выбрать уникальные элементы столбца, по которому осуществляется группировка. Результат будет точно такой же как при использовании DISTINCT.

```
SELECT author
FROM book
GROUP BY author
ORDER BY 1 -- сортируем по первому столбцу
```

Выборка данных, групповые функции SUM и COUNT

При группировке над элементами столбца, входящими в группу, можно выполнить различные действия, например, просуммировать их или найти количество элементов в группе.

Функцию COUNT () можно применять к любому столбцу, в том числе можно использовать и *, если таблица не содержит пустых значений. Если же в столбцах есть значения Null, (для группы по автору Есенин в нашем примере), то

- COUNT (*) — подсчитывает все записи, относящиеся к группе, в том числе и со значением NULL;
- COUNT (имя_столбца) — возвращает количество записей конкретного столбца (только NOT NULL), относящихся к группе

ВАЖНО

Если столбец указан в SELECT БЕЗ применения групповой функции, то он обязательно должен быть указан и в GROUP BY. Иначе получим ошибку.

```
SELECT
author,
SUM (amount) as total_amount,
COUNT (title) as books
FROM book
GROUP BY author
ORDER BY total_amount DESC
```

В дополнение отмечу, если названия книг для одного и того же автора повторяются, а требуется посчитать количество **уникальных наименований**, можно использовать ключевое слово DISTINCT **в агрегатной функции** (и не только в COUNT).

```
SELECT
author,
COUNT (DISTINCT title)
FROM book
GROUP BY author;
```

Выборка данных, групповые функции MIN, MAX и AVG

К групповым функциям SQL относятся: MIN (), MAX () и AVG (), которые вычисляют минимальное, максимальное и среднее значение элементов столбца, относящихся к группе.

```
SELECT
author,
ROUND (AVG (price), 2) AS avg_price,
MAX (price) AS max_price,
MIN (price) AS min_price
```

```
FROM book
GROUP BY author
ORDER BY avg_price DESC
```

Выборка данных с вычислением, групповые функции

В качестве аргумента групповых функций SQL может использоваться не только столбец, но и любое допустимое в SQL арифметическое выражение.

Вывести среднюю стоимость книг каждого автора с учётом их количества.

```
SELECT
author,
ROUND (SUM (price * amount) / SUM (amount), 2) AS avg_price
FROM book
GROUP BY author
ORDER BY avg_price DESC
```

Вычисления по таблице целиком

Групповые функции позволяют вычислять итоговые значения по всей таблице. Например, можно посчитать общее количество книг на складе, вычислить суммарную стоимость и пр. Для этого после ключевого слова SELECT **указывается групповая функция для выражения или имени столбца, а ключевые слова GROUP BY опускаются.**

```
SELECT
SUM (amount) AS total_amount,
SUM (price * amount) AS total_price
FROM book;
```

Выборка данных по условию, групповые функции

В запросы с групповыми функциями можно включать условие отбора строк, которое в обычных запросах записывается после WHERE. В запросах с групповыми функциями вместо WHERE используется ключевое слово HAVING, которое размещается после оператора GROUP BY.

Найти минимальную и максимальную цену книг всех авторов, общая стоимость книг которых больше 5000.

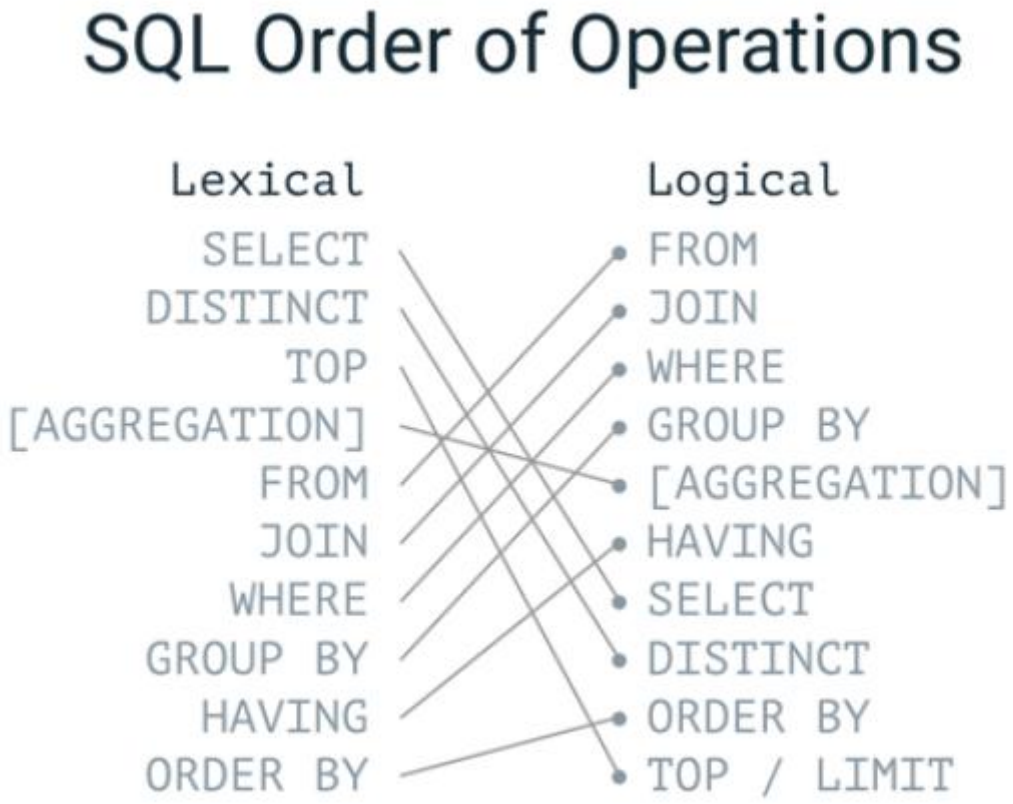
```
SELECT
author,
MIN(price) AS min_price,
MAX(price) AS max_price,
```

```
SUM(price * amount) AS total_price
FROM
book
GROUP BY
author
HAVING
SUM(price * amount) > 5000
ORDER BY
total_price DESC
```

Выборка данных по условию, групповые функции, WHERE и HAVING

WHERE и HAVING могут использоваться в одном запросе. При этом необходимо учитывать порядок выполнения SQL запроса на выборку на СЕРВЕРЕ:

Порядок выполнения операторов в SQL



Сначала определяется таблица, из которой выбираются данные FROM, затем из этой таблицы отбираются записи в соответствии с условием WHERE, выбранные данные агрегируются GROUP BY, из агрегированных записей выбираются те, которые удовлетворяют условию после HAVING. Потом формируются данные результирующей

выборки, как это указано после SELECT (вычисляются выражения, присваиваются имена и пр.). Результирующая выборка сортируется, как указано после ORDER BY.

Важно! Порядок ВЫПОЛНЕНИЯ запросов — это не порядок ЗАПИСИ ключевых слов в запросе на выборку. Порядок записи (синтаксис запроса) остается таким же, как рассматривался ранее в курсе. Порядок ВЫПОЛНЕНИЯ нужен для того, чтобы понять, почему, например, в WHERE нельзя использовать имена выражений из SELECT и в HAVING тоже (см. картинку выше). Просто SELECT выполняется компилятором позже, чем WHERE, поэтому ему неизвестно, какое там выражение написано.

Вывести максимальную и минимальную цену книг каждого автора, кроме Есенина, количество экземпляров книг которого больше 10.

```
SELECT author,
MIN (price) AS Минимальная_цена,
MAX (price) AS Максимальная_цена
FROM book
WHERE author <> 'Есенин С.А.'
GROUP BY author
HAVING SUM (amount) > 10;
```

Этот запрос будет работать, если его переписать следующим образом:

```
SELECT author,
MIN (price) AS Минимальная_цена,
MAX (price) AS Максимальная_цена
FROM book
GROUP BY author
HAVING SUM (amount) > 10 AND author <> 'Есенин С.А.';
```

Несмотря на то, что результат будет одинаковым, так делать **не рекомендуется.** «Потому что как написано - запрос сначала выбирает всех авторов, потом выводит данные, рассчитывая минимальное и максимальное значение цены для каждого, и только после всего убирает Есенина. Можно убрать Есенина в данном случае раньше и не использовать ресурсы базы для расчета его минимального и максимального значения, как это сделано в первом варианте. На небольшой базе быстродействия не ощутить, но если выполнять такое на продуктивной, то второй вариант значительно проигрывает...».

Варианты индивидуальных заданий

Вариант 1

Узнать сколько авторов, у которых есть книги со стоимостью более 500 и количеством более 1 шт на складе, при количестве различных названий произведений не менее 2-х. Вывести автора, количество различных произведений автора, минимальную цену и количество книг.

Вариант 2

Вывести авторов и суммарную стоимость их книг, если есть хотя бы одна книга их авторства по цене выше 750Р.

Суммарная стоимость — это не сумма всех цен книг автора, а то, чтобы мы получили, продав все книги автора, которые есть у нас в наличие.

Вариант 3

Сколько денег понадобиться покупателю, чтобы приобрести все виды книг по одному экземпляру каждой? Сколько юы потратил покупатель, не заплатив бы НДС.

В России НДС равен 20%. Однако некоторые виды книг могут рассчитывать на льготную ставку. Льготной ставкой НДС 10% могут пользоваться только издатели периодических печатных изданий и книг, связанных с образованием, наукой и культурой. Однако мы будем считать, что все книги в нашем магазине облагаются НДС 20%.

Имена результирующих столбцов:

- Стоимость_всех_книг_по_одному_экз
- Стоимость_всех_книг_по_одному_экз_без_НДС (с округлением до сотых)
- Количество_купленных_книг

Требования к отчёту

1. Титульная страница

   Титульная страница должна содержать:

   1. Наименование учебного учреждения
   2. Наименование предмета
   3. Фамили и инициалы студента
   4. Фамилию и инициалы преподавателя, принимающего работу
   5. Номер индивидуального задания (если имеется)
2. Цель работы
3. Выполнение задания
4. Вывод по работе

ЛАБОРАТОРНАЯ РАБОТА № 7

Агрегатные функции

Цель работы: изучить тему агрегатных функций в SQL. Научится применять их на практике.

Теоретическая часть

Агрегатные функции вычисляют одно значение над некоторым набором строк. В PostgreSQL имеются следующие агрегатные функции:

- AVG: находит среднее значение. Входной параметр должен представлять один из следующих типов: smallint, int, bigint, real, double precision, numeric, interval. Для целочисленных параметров результатом будет значение типа numeric, для параметров, которые представляют число с плавающей точкой, - значение типа double precision.
- BIT_AND: выполняет операцию побитового умножения (операции логического И) для чисел следующих типов: smallint, int, bigint, bit. Если параметр содержит значение NULL, то возвращается NULL.
- BIT_OR: выполняет операцию побитового сложения (операции логического ИЛИ) для чисел следующих типов: smallint, int, bigint, bit. Если параметр содержит значение NULL, то возвращается NULL.
- BOOL_AND: выполняет операцию логического умножения для значений типа bool. Если входные все значения равны true, то возвращается true, иначе возвращается false.
- BOOL_OR: выполняет операцию логического сложения для значений типа bool. Если входные хотя бы одно из значений равно true, то возвращается true, иначе возвращается false.
- COUNT (*): находит количество строк в запросе
- COUNT (expression): находит количество строк в запросе, для которых expression не содержит значение NULL.
- SUM: находит сумму значений
- MIN: находит наименьшее значение
- MAX: находит наибольшее значение
- STRING_AGG (expression, delimiter): соединяет с помощью delimiter все текстовые значения из expression в одну строку.

В качестве параметра все агрегатные функции принимают выражение, которое представляет критерий для определения значений. Зачастую, в качестве выражения выступает название столбца, над значениями которого надо проводить вычисления.

Если в наборе нет строк, то все агрегатные функции за исключением COUNT (*) возвращают значение NULL.

Avg

Функция Avg возвращает среднее значение на диапазоне значений столбца таблицы.

Пусть в базе данных у нас есть таблица товаров Products, которая описывается следующими выражениями:

```
CREATE TABLE Products(
Id SERIAL PRIMARY KEY,
ProductName VARCHAR(30) NOT NULL,
Company VARCHAR(20) NOT NULL,
ProductCount INT DEFAULT 0,
Price NUMERIC NOT NULL,
IsDiscounted BOOL
);

INSERT INTO Products (ProductName, Company, ProductCount, Price, IsDiscounted)
VALUES
('iPhone X', 'Apple', 3, 76000, false),
('iPhone 8', 'Apple', 2, 71000, true),
('iPhone 7', 'Apple', 5, 42000, true),
('Galaxy S9', 'Samsung', 2, 46000, false),
('Galaxy S8 Plus', 'Samsung', 1, 56000, true),
('Desire 12', 'HTC', 5, 28000, true),
('Nokia 9', 'HMD Global', 6, 38000, true);
```

Найдем среднюю цену товаров из базы данных:

```
SELECT AVG(Price) AS Average_Price FROM Products;
```

Для поиска среднего значения в качестве выражения в функцию передается столбец Price. Для получаемого значения устанавливается псевдоним Average_Price, хотя можно его и не устанавливать.

Также мы можем применить фильтрацию. Например, найти среднюю цену для товаров какого-то определенного производителя:

```
SELECT AVG(Price) FROM Products
WHERE Company='Apple';
```

И, кроме того, мы можем находить среднее значение для более сложных выражений. Например, найдем среднюю сумму всех товаров, учитывая их количество:

```
SELECT AVG (Price * ProductCount) FROM Products
```

Count

Функция Count вычисляет количество строк в выборке. Есть две формы этой функции. Первая форма COUNT (*) подсчитывает число строк в выборке:

```
SELECT COUNT (*) FROM Products;
```

Вторая форма функции вычисляет количество строк по определенному столбцу, при этом строки со значениями NULL игнорируются:

```
SELECT COUNT (DISTINCT Company) FROM Products;
```

Оператор DISTINCT указывает, что надо взять именно уникальные значения из столбца Company.

Min и Max

Функции Min и Max возвращают соответственно минимальное и максимальное значение по столбцу. Например, найдем минимальную цену среди товаров:

```
SELECT MIN(Price) FROM Products;
```

Поиск максимальной цены:

```
SELECT MAX(Price) FROM Products;
```

Данные функции также игнорируют значения NULL и не учитывают их при подсчете.

Sum

Функция Sum вычисляет сумму значений столбца. Например, подсчитаем общее количество товаров:

```
SELECT SUM(ProductCount) FROM Products;
```

Также вместо имени столбца может передаваться вычисляемое выражение. Например, найдем общую стоимость всех имеющихся товаров:

```
SELECT SUM(ProductCount * Price) FROM Products;
```

BOOL_AND и BOOL_OR

Допустим, нам надо узнать, есть ли в таблице товары, которые подлежать скидке, то есть у которых IsDiscounted = true. В этом случае можно выполнить функцию BOOL_OR, которая возвращает true, если хотя бы одно значение в наборе равно true:

```
SELECT BOOL_OR(IsDiscounted) FROM Products;
```

Если нам надо узнать, все ли товары подлежат скидке, то можно применить функцию BOOL_AND, которая возвращает true, если все значения в наборе равны true:

```
SELECT BOOL_AND(IsDiscounted) FROM Products;
```

STRING_AGG

Функция STRING_AGG() объединяет набор строковых значений или значений bytea. Например, выберем названия всех товаров:

```
SELECT STRING_AGG(ProductName, ', ') FROM Products;
```

Или выберем всех производителей:

```
SELECT STRING_AGG(DISTINCT Company, ', ') FROM Products;
-- результат операции
-- Apple, HMD Global, HTC, Samsung
```

Чтобы выбрать уникальных производителей, здесь также применяется оператор DISTINCT, иначе у нас бы повторялись значения.

Комбинирование функций

Объединим применение нескольких функций:

```
SELECT COUNT(*) AS ProdCount,
SUM(ProductCount) AS TotalCount,
MIN(Price) AS MinPrice,
MAX(Price) AS MaxPrice,
AVG(Price) AS AvgPrice
FROM Products;
```

Варианты

Задание 1: Анализ продаж в интернет-магазине

Описание: Вам предоставлена база данных интернет-магазина, содержащая таблицы orders (заказы), customers (покупатели), products (товары) и order_items (элементы заказа). Ваша задача — использовать агрегатные функции для анализа данных о продажах.

Таблицы:

- orders (order_id, customer_id, order_date)
- customers (customer_id, customer_name, customer_email)
- products (product_id, product_name, product_price)
- order_items (order_item_id, order_id, product_id, quantity)

Задачи:

1. Найти общую сумму продаж за каждый месяц.
2. Определить среднюю сумму заказа для каждого покупателя.
3. Найти топ-3 товаров по количеству проданных единиц.

4. Определить общую сумму продаж для каждого товара.
5. Найти количество заказов, сделанных каждым покупателем.

Задание 2: Анализ данных о студентах и их успеваемости

Описание: Вам предоставлена база данных университета, содержащая таблицы students (студенты), courses (курсы), enrollments (зачисления) и grades (оценки). Ваша задача — использовать агрегатные функции для анализа данных о студентах и их успеваемости.

Таблицы:

- students (student_id, student_name, student_email)
- courses (course_id, course_name)
- enrollments (enrollment_id, student_id, course_id)
- grades (grade_id, enrollment_id, grade)

Задачи:

1. Найти средний балл по каждому курсу.
2. Определить максимальный и минимальный баллы для каждого студента.
3. Найти количество студентов, зачисленных на каждый курс.
4. Определить средний балл для каждого студента.
5. Найти топ-3 студентов по среднему баллу.

Задание 3: Анализ данных о сотрудниках и их зарплатах

Описание: Вам предоставлена база данных компании, содержащая таблицы employees (сотрудники), departments (отделы) и salaries (зарплаты). Ваша задача — использовать агрегатные функции для анализа данных о сотрудниках и их зарплатах.

Таблицы:

- employees (employee_id, employee_name, department_id)
- departments (department_id, department_name)
- salaries (salary_id, employee_id, salary)

Задачи:

1. Найти среднюю зарплату по каждому отделу.
2. Определить максимальную и минимальную зарплаты для каждого сотрудника.
3. Найти количество сотрудников в каждом отделе.
4. Определить общую сумму зарплат для каждого отдела.
5. Найти топ-3 сотрудников по зарплате.

Требования к отчёту

1. Титульная страница

   Титульная страница должна содержать:

   1. Наименование учебного учреждения

2. Наименование предмета
3. Фамили и инициалы студента
4. Фамилию и инициалы преподавателя, принимающего работу
5. Номер индивидуального задания (если имеется)

2. Цель работы
3. Выполнение задания
4. Вывод по работе

ЛАБОРАТОРНАЯ РАБОТА № 8

Вложенные запросы

Цель работы: изучить тему вложенных запросов в SQL. Научится применять их на практике.

Теоретическая часть

Схема таблицы и данные

Схема таблицы

```
CREATE TABLE IF NOT EXISTS book (
book_id serial PRIMARY KEY,
title VARCHAR (50),
author VARCHAR (30),
price DECIMAL (8, 2),
amount INT
);
```

запросы вставки

```
INSERT INTO book(title,author,price,amount) VALUES ('Бесы','Достоевский Ф.М.',800,3);
INSERT INTO book(title,author,price,amount) VALUES ('Игрок','Достоевский Ф.М.',700,8);
INSERT INTO book(title,author,price,amount) VALUES ('Война и мир','Толстой
Л.Н.',1000,3);
INSERT INTO book(title,author,price,amount) VALUES ('Анна Каренина','Толстой
Л.Н.',1200,7);
INSERT INTO book(title,author,price,amount) VALUES ('Отцы и дети','Тургенев
И.С.',650,8);
INSERT INTO book(title,author,price,amount) VALUES ('Обломов','Гончаров И.А.',900,2);
INSERT INTO book(title,author,price,amount) VALUES ('Евгений Онегин','Пушкин
```

```
A.C.',1000,3);
INSERT INTO book(title,author,price,amount) VALUES ('Собачье сердце','Булгаков
M.A.',750,6);
INSERT INTO book(title,author,price,amount) VALUES ('Капитанская дочь','Пушкин
A.C.',1500,1);
INSERT INTO book(title,author,price,amount) VALUES ('Мёртвые души','Гоголь
H.B.',900,2);
INSERT INTO book(title,author,price,amount) VALUES ('Ревизор','Гоголь H.B.',500,4);
INSERT INTO book(title,author,price,amount) VALUES ('Шинель','Гоголь H.B.',645,2);
INSERT INTO book(title,author,price,amount) VALUES ('Палата номер 6','Чехов
А.П.',500,10);
INSERT INTO book(title,author,price,amount) VALUES ('Вишнёвый сад','Чехов
А.П.',400,12);
INSERT INTO book(title,author,price,amount) VALUES ('Мастер и Маргарита','Булгаков
M.A.',555,5);
INSERT INTO book(title,author,price,amount) VALUES ('Белая гвардия','Булгаков
M.A.',666,6);
INSERT INTO book(title,author,price,amount) VALUES ('Идиот','Достоевский Ф.М.',777,7);
INSERT INTO book(title,author,price,amount) VALUES ('Братья Карамазовы','Достоевский
Ф.М.',888,9);
INSERT INTO book(title,author,price,amount) VALUES ('Преступление и
наказание','Достоевский Ф.М.',452,7);
INSERT INTO book(title,author,price,amount) VALUES ('Обломов','Гончаров И.А.',365,3);
INSERT INTO book(title,author,price,amount) VALUES ('Стихотворения и поэма','Есенин
С.А.',578,6);
```

Вложенные запросы

SQL позволяет создавать вложенные запросы. Вложенный запрос (подзапрос, внутренний запрос) – это запрос внутри другого запроса SQL.

Вложенный запрос используется для выборки данных, которые будут использоваться в условии отбора записей основного запроса. Его применяют для:

- сравнения выражения с результатом вложенного запроса;
- определения того, включено ли выражение в результаты вложенного запроса;
- проверки того, выбирает ли запрос определенные строки.

Вложенный запрос имеет следующие компоненты:

- ключевое слово SELECT после которого указываются имена столбцов или выражения (чаще всего список содержит один элемент) ;
- ключевое слово FROM и имя таблицы, из которой выбираются данные;
- необязательное предложение WHERE;
- необязательное предложение GROUP BY;
- необязательное предложение HAVING.

Вложенные запросы могут включаться в WHERE или HAVING так (в квадратных скобках указаны необязательные элементы, через | – один из элементов):

- WHERE | HAVING выражение оператор_сравнения (вложенный запрос);
- WHERE | HAVING выражение, включающее вложенный запрос;
- WHERE | HAVING выражение [NOT] IN (вложенный запрос);
- WHERE | HAVING выражение оператор_сравнения ANY | ALL (вложенный запрос).

Также вложенные запросы могут вставляться в основной запрос после ключевого слова SELECT.

Вложенный запрос, возвращающий одно значение

Вложенный запрос, возвращающий одно значение, может использоваться в условии отбора записей WHERE как обычное значение совместно с операциями =, <>,> =, <=,>,

Вывести информацию о самых дешевых книгах, хранящихся на складе.

Для реализации этого запроса нам необходимо получить минимальную цену из столбца price таблицы book, а затем вывести информацию о тех книгах, цена которых равна минимальной. Первая часть – поиск минимума – реализуется вложенным запросом.

```
SELECT title, author, price, amount
FROM book
WHERE price = (
SELECT MIN (price)
FROM book
);
```

Вложенный запрос определяет минимальную цену книг во всей таблице, а затем в основном запросе для каждой записи проверяется, равна ли цена минимальному значению, если равна, информация о книге включается в результирующую таблицу запроса.

Рекомендация. При использовании вложенного запроса рекомендуется сначала проверить, правильно ли он работает, если выдается верный результат – использовать код в качестве вложенного запроса.

```
SELECT title, author, price, amount
FROM book
WHERE price = (
SELECT MIN (price)
FROM book
);
```

ожидаемый результат

title	author	price	amount
Обломов	Гончаров И.А.	365.00	3

Использование вложенного запроса в выражении

Вложенный запрос, возвращающий одно значение, может использоваться в выражениях как обычный операнд, например, к нему можно что-то прибавить, отнять и пр.

Пример

Вывести информацию о книгах, количество экземпляров которых отличается от среднего количества экземпляров книг на складе более чем на 1. То есть нужно вывести и те книги, количество экземпляров которых меньше среднего на 1, и больше среднего на 1.

```
SELECT title, author, amount
FROM book
WHERE ABS(amount - (SELECT AVG(amount) FROM book)) > 1
```

Вложенный запрос, оператор IN

Вложенный запрос может возвращать несколько значений одного столбца. Оператор IN определяет, совпадает ли указанное в логическом выражении значение с одним из значений, содержащихся во вложенном запросе, при этом логическое выражение получает значение истина. Оператор NOT IN выполняет обратное действие – выражение истинно, если значение не содержится во вложенном запросе.

Вывести информацию о книгах тех авторов, общее количество экземпляров книг которых не менее 30.

```
SELECT title, author, amount, price
FROM book
WHERE author IN (
```

```
SELECT author
FROM book
GROUP BY author
HAVING SUM(amount) >= 30
)
```

Вложенный запрос, операторы ANY и ALL

Вложенный запрос, возвращающий **несколько значений одного столбца**, можно использовать для отбора записей с помощью операторов ANY и ALL совместно с операциями отношения (=, <>, <=,> =,<, >).

Операторы ANY и ALL используются в SQL для сравнения некоторого значения с результирующим набором вложенного запроса, состоящим из одного столбца. При этом тип данных столбца, возвращаемого вложенным запросом, должен совпадать с типом данных столбца (или выражения), с которым происходит сравнение.

При использовании оператора ANY в результирующую таблицу будут включены все записи, для которых выражение со знаком отношения верно **хотя бы для одного элемента результирующего запроса**. Как работает оператор ANY ():

- amount > ANY (10, 12) эквивалентно amount > 10
- amount < ANY (10, 12) эквивалентно amount < 12
- amount = ANY (10, 12) эквивалентно (amount = 10) OR (amount = 12) , а также amount IN (10,12)
- amount <> ANY (10, 12) вернет все записи с любым значением amount, включая 10 и 12

При использовании оператора ALL в результирующую таблицу будут включены все записи, для которых выражение со знаком отношения верно для всех элементов результирующего запроса. Как работает оператор ALL :

- amount > ALL (10, 12) эквивалентно amount > 12
- amount < ALL (10, 12) эквивалентно
- amount < 10 amount = ALL (10, 12) не вернет ни одной записи, так как эквивалентно (amount = 10) AND (amount = 12)
- amount <> ALL (10, 12) вернет все записи кроме тех, в которых amount равно 10 или 12

Важно! Операторы ALL и ANY можно использовать только с вложенными запросами. В примерах выше (10, 12) приводится как результат вложенного запроса просто для того, чтобы показать, как эти операторы работают. В запросах так записывать нельзя.

Вывести информацию о тех книгах, количество которых меньше САМОГО МАЛЕНЬКОГО среднего количества книг каждого автора.

```
SELECT title, author, amount, price
FROM book
WHERE amount < ALL (
SELECT AVG(amount)
FROM book
GROUP BY author
);
```

Вывести информацию о тех книгах, количество которых меньше САМОГО БОЛЬШОГО среднего количества книг каждого автора.

```
SELECT title, author, amount, price
FROM book
WHERE amount < ANY (
SELECT AVG(amount)
FROM book
GROUP BY author
);
```

Вложенный запрос после SELECT

Вложенный запрос может располагаться после ключевого слова SELECT. В этом случае результат выполнения запроса выводится в отдельном столбце результирующей таблицы. При этом результатом запроса может быть **либо одно значение, тогда оно будет повторяться во всех строках, либо несколько значений, количество которых равно количеству отобранных записей в основном запросе.**

Вывести информацию о книгах, количество экземпляров которых отличается от среднего количества экземпляров книг на складе более чем на 3, а также указать среднее значение количества экземпляров книг.

```
SELECT
title,
author,
amount,
(SELECT AVG (amount) FROM book) AS average_amount
FROM book
WHERE abs (amount - (SELECT AVG (amount) FROM book)) > 3;
```

Варианты заданий

Вариант 1

Посчитать сколько и каких экземпляров книг нужно заказать поставщикам, чтобы на складе стало одинаковое количество экземпляров каждой книги, равное значению самого большего количества экземпляров одной книги на складе. Вывести название книги, ее автора, текущее количество экземпляров на складе и количество заказываемых экземпляров книг. Последнему столбцу присвоить имя Заказ. Отсортировать запрос по количеству заказу в обратном порядке (от большего к меньшему).

Вариант 2

Вывести информацию (автора, книгу и количество) о тех книгах, количество экземпляров которых в таблице book не дублируется. Отсортировать результат полю amount (Подсказка, должен использоваться оператор GROUP BY)

Вариант 3

Вывести информацию (автора, название, цену и РАЗНИЦУ ЦЕНЫ С МИНИМАЛЬНОЙ ЦЕНОЙ) о тех книгах, цены которых превышают минимальную цену книги на складе не более чем на 150 рублей в отсортированном по возрастанию цены виде.

Вариант 4

При продаже всех книг, какая книга принесет больше всего выручки, в процентах. Вам нужно учитывать не только цену, но и количество книг. Подсказки Выручка от книги = цена книги * количество Сделайте запрос, который вычисляет сколько бы принесла продажа всех экземпляров для каждой книги Найдите книги с максимальной выручкой и выведите её название и автора Постройте запрос, вычисляющий выручку от продажи ВСЕХ книги, и поделите выручку найденной книги на это значение, используя округление до двух знаков после запятой.

Вариант 5

Питерский БОМЖ, как житель культурной столицы, решил закупиться книгами. Чтобы накопить, он сдаёт алюминиевые банки, которые принимают по цене 70 рублей/кг. В день он собирает 3 кг алюминия. Он хочет прочитать книги всех авторов. Однако его бюджет ограничен. Он решил купить самую дешёвую книгу каждого автора по одному экземпляру. Посчитайте с помощью SQL запроса, сколько дней БОМЖу надо собирать алюминиевые банки, чтобы выполнить своё желание. Округлите до ближайшего наибольшего целого.

Требования к отчёту

3. Титульная страница

   Титульная страница должна содержать:

   1. Наименование учебного учреждения
   2. Наименование предмета
   3. Фамили и инициалы студента
   4. Фамилию и инициалы преподавателя, принимающего работу
   5. Номер индивидуального задания (если имеется)

4. Цель работы
5. Выполнение задания
6. Вывод по работе

ЛАБОРАТОРНАЯ РАБОТА № 9

Запросы корректировки данных

Цель работы: изучить тему запросов корректировки данных SQL. Научится применять их на практике

Теоретическая часть

Добавление записей из другой таблицы

С помощью запроса на добавление можно не только добавить в таблицу конкретные значения (список VALUES), но и записи из другой таблицы, отобранные с помощью запроса на выборку. В этом случае вместо раздела VALUES записывается запрос на выборку, начинающийся с SELECT. В нем можно использовать WHERE , GROUP BY , ORDER BY .

Правила соответствия между полями таблицы и вставляемыми значениями из запроса:

- количество полей в таблице и количество полей в запросе должны совпадать;
- должно существовать прямое соответствие между позицией одного и того же элемента в обоих списках, поэтому первый столбец запроса должен относиться к первому столбцу в списке столбцов таблицы, второй – ко второму столбцу и т.д.
- типы столбцов запроса должны быть совместимы с типами данных соответствующих столбцов таблицы (целое число можно занести в поле типа DECIMAL, обратная операция – недопустима).

Создадим таблицу supply:

```
CREATE TABLE IF NOT EXISTS supply (
supply_id SERIAL PRIMARY KEY,
title VARCHAR (50),
author VARCHAR (30),
```

```
price DECIMAL (8, 2),
amount INTEGER
);
```

И занесём значения из таблицы book в таблицу supply. Количество заносимых книг должно быть на складе меньше среднего количества. В amount таблицы supply записать среднее количество на складе, в виде целого числа с округлением вниз.

```
INSERT INTO supply (title, author, price, amount)
SELECT
title,
author,
price,
FLOOR((SELECT AVG(amount) FROM book))
FROM book
WHERE
amount < (SELECT AVG(amount) FROM book)
```

Запросы на обновление

Под обновлением данных подразумевается изменение значений в существующих записях таблицы. При этом возможно как изменение значений полей в группе строк (даже всех строк таблицы), так и правка значения поля отдельной строки.

Изменение записей в таблице реализуется с помощью запроса UPDATE. Простейший запрос на обновление выглядит так:

```
UPDATE таблица SET поле = выражение
```

где

- таблица – имя таблицы, в которой будут проводиться изменения;
- поле – поле таблицы, в которое будет внесено изменение;
- выражение – выражение, значение которого будет занесено в поле.

В связи с кризисом книги подорожали. Увеличим их цену на 15%

```
UPDATE book SET price = ROUND(price * 1.15, 2)
```

С помощью запросов на обновление можно изменять не все записи в таблице (как в предыдущем запросе), а только часть из них. Для этого в запрос включается ключевое слово WHERE, после которого указывается условие отбора строк для изменения.

Уменьшить на 30% цену тех книг в таблице book, количество которых меньше 5.

```
UPDATE book
SET price = ROUND(price*0.7, 2)
WHERE amount < 5
```

Запросы на обновление нескольких столбцов

Запросом UPDATE можно обновлять значения нескольких столбцов одновременно. В этом случае простейший запрос будет выглядеть так:

```
UPDATE таблица SET поле1 = выражение1, поле2 = выражение2
```

На складе, кроме хранения и получения книг, выполняется их оптовая продажа. Для реализации этого действия включим дополнительный столбец buy в таблицу book:

```
ALTER TABLE
book
ADD COLUMN
buy INTEGER NOT NULL DEFAULT 0
```

заполним новосозданный столбец случайными числами от 1 до 4 включительно

```
UPDATE book
SET buy = FLOOR (1 + RAND () *5);
```

В столбце buy покупатель указывает количество книг, которые он хочет приобрести. Для каждой книги, выбранной покупателем, необходимо уменьшить ее количество на складе на указанное в столбцеbuy количество, а в столбец buy занести 0. Если покупатель хочеть купить больше, чем есть на складе, то ничего не обновляем.

```
UPDATE book
SET amount = amount - buy, buy = 0 WHERE amount >= buy
```

Запросы на удаление

Запросы корректировки данных позволяют удалить одну или несколько записей из таблицы. Простейший запрос на удаление имеет вид:

```
DELETE FROM таблица;
```

Этот запрос удаляет все записи из указанной после FROM таблицы. Запрос на удаления позволяет удалить не все записи таблицы, а только те, которые удовлетворяют условию, указанному после ключевого слова WHERE:

```
DELETE FROM таблица
WHERE условие;
```

Пример

Удалить из таблицы supply все книги, названия которых есть в таблице book.

Запрос:

```
DELETE FROM supply
WHERE title IN (
SELECT title
FROM book
);
```

Запросы на создание таблицы

Новая таблица может быть создана на основе данных из другой таблицы. Для этого используется запрос SELECT, результирующая таблица которого и будет новой таблицей базы данных. При этом имена столбцов запроса становятся именами столбцов новой таблицы. Запрос на создание новой таблицы имеет вид:

```
CREATE TABLE имя_таблицы AS
SELECT ...
```

Пример

Создать таблицу заказ (ordering), куда включить авторов и названия тех книг, количество экземпляров которых в таблице book меньше 4. Для всех книг указать одинаковое количество экземпляров 5.

```
CREATE TABLE ordering AS
SELECT author, title, 5 AS amount
FROM book
WHERE amount < 4;
```

Варианты индивидуальных заданий

Вариант 1

Создать таблицу заказ (ordering), куда включить авторов и названия тех книг, количество которых в таблице book меньше максимального. Для всех книг в таблице ordering указать такое значение, которое позволит выровнять количество книг до максимального в таблице book.

Вариант 2

Занести из таблицы supply в таблицу book только те книги, названия которых отсутствуют в таблице book, при этом количество этих книг в таблице supply должно обнулиться. Т.е. все эти книги со склада передали в магазин.

Вариант 3

Создать таблицу заказ (ordering), куда включить авторов и названия тех книг, количество экземпляров которых в таблице book меньше среднего количества экземпляров книг в таблице book. В таблицу включить столбец amount, в котором для всех книг указать одинаковое значение - среднее количество экземпляров книг в таблице book.

Требования к отчёту

1. Титульная страница

   Титульная страница должна содержать:

   1. Наименование учебного учреждения
   2. Наименование предмета
   3. Фамили и инициалы студента
   4. Фамилию и инициалы преподавателя, принимающего работу
   5. Номер индивидуального задания (если имеется)
2. Цель работы
3. Выполнение задания
4. Вывод по работе

ЛАБОРАТОРНАЯ РАБОТА № 10

Перечисления в PostgreSQL

Цель работы: ознакомиться с перечислениями в PostgreSQL. Научиться применять перечисления на практике.

Теоретическая часть

Перечисления (enum) – это такие типы данных, которые состоят из статических, упорядоченных списков значений. Они эквивалентны типам enum в некоторых языках программирования. В качестве примера типа enum можно привести дни недели или список состояний для каких-либо данных.

Объявление перечислений

Тип перечислений создаются с помощью команды CREATE TYPE, например так:

```
CREATE TYPE mood AS ENUM ('sad', 'ok', 'happy');
```

Созданные типы enum можно использовать в определениях таблиц и функций, как и любые другие:

```
CREATE TYPE mood AS ENUM ('sad', 'ok', 'happy');
CREATE TABLE person (name text, current_mood mood);
INSERT INTO person VALUES ('Moe', 'happy');
SELECT * FROM person WHERE current_mood = 'happy';
```

name	current_mood
Moe	happy

Задание

Создать два перечисления: students и movies. В качестве значений перечисление students представить своё ФИО и ФИО 4-х своих одногруппников. В качестве значений в перечисление movies представить пять своих самых любимых фильмов.

Требования к отчёту

1. Титульная страница

   Титульная страница должна содержать:

   1. Наименование учебного учреждения
   2. Наименование предмета
   3. Фамили и инициалы студента
   4. Фамилию и инициалы преподавателя, принимающего работу
   5. Номер индивидуального задания (если имеется)
2. Цель работы
3. Выполнение задания
4. Вывод по работе

ЛАБОРАТОРНАЯ РАБОТА № 11

Изучение объектов представления

Цель работы: Реализация базы данных выполняется с помощью СУБД PostgreSQL. Минимальная реализация системы подразумевает создаение базы данных и запросов на SQL.

Теоретическая часть

Представление (VIEW) — объект базы данных, являющийся результатом выполнения запроса к базе данных, определенного с помощью оператора SELECT, в момент обращения к представлению.

Представления иногда называют «виртуальными таблицами». Такое название связано с тем, что представление доступно для пользователя как таблица, но само оно не содержит данных, а извлекает их из таблиц в момент обращения к нему. Если данные изменены в базовой таблице, то пользователь получит актуальные данные при обращении к представлению, использующему данную таблицу; кэширования результатов выборки из таблицы при работе представлений не производится. При этом, механизм кэширования

запросов (query cache) работает на уровне запросов пользователя безотносительно к тому, обращается ли пользователь к таблицам или представлениям.

Представления могут основываться как на таблицах, так и на других представлениях, т.е. могут быть вложенными (до 32 уровней вложенности).

Создание представления:

```
CREATE VIEW <имя_представления> (<имя столбца 1>, <имя столбца 2>, …, <имя столбца n>)
AS SELECT <имя столбца 1>, <имя столбца 2>, …, <имя столбца n>
FROM <Таблица >
WHERE ...
ORDER BY ...
…;
```

Зачем нужны представления:

- для разграничения прав доступа (могут дать доступ к части таблице);
- Для упрощений комплексных операций выборок.

Примеры представлений:

Представление с вложенным запросом

```
CREATE VIEW "BookWithAuthorT" (title) AS
SELECT book.title
FROM book
WHERE book.author_id IN
(SELECT author_id FROM author WHERE name_author LIKE "T%");
SELECT * FROM "BookWithAuthorT";
```

Задание

Сделать любое представление (кроме тех, которые полностью повторяют структуру таблиц или тех, которые похожи на примерах выше). При создании представления должен быть употреблён:

1 вариант

вложенный запрос

2 вариант запрос соединения (JOIN).

Требования к отчёту

1. Титульная страница

   Титульная страница должна содержать:

   1. Наименование учебного учреждения

2. Наименование предмета
3. Фамили и инициалы студента
4. Фамилию и инициалы преподавателя, принимающего работу
5. Номер индивидуального задания (если имеется)

2. Цель работы
3. Выполнение задания
4. Вывод по работе

ЛАБОРАТОРНАЯ РАБОТА № 12

Изучение механизмов индексов

Цель работы: изучить механизма индексов в PostgreSQL

Теоретическая часть

FAQ о индексах

Что это?

Индекс представляет из себя структуру, в которой хранятся значения одного или нескольких столбца[ов] таблицы и ссылок на строки, где эти значения расположены. Для хранения индексов чаще всего используются "бинарные деревья". Индексы в PostgreSQL — специальные объекты базы данных, предназначенные в основном для ускорения доступа к данным. Это вспомогательные структуры: любой индекс можно удалить и восстановить заново по информации в таблице.

Зачем?

Для ускорения выполнения выборок по определённым столбцам. Использование индексов значительно сокращает время выполнения запроса, что позволяет быстрее получать данные.

Как создать индекс?

```
CREATE INDEX ON <название индекса>
ON <название таблицы> (<столбец 1>, <столбец 2>, …, <столбец n>);
```

Когда есть смысл создавать индекс?

Индексы создаются для повышения производительности поиска данных. Таблицы могут иметь огромное количество строк, которые хранятся в произвольном порядке. Без индекса поиск нужных строк идёт по порядку (последовательно), что на больших объемах данных отнимает много времени.

Индекс - обычно один или несколько столбцов таблицы и указателей на соответствующие строки таблицы, позволяет искать строки, удовлетворяющие критерию

поиска. Ускорение работы с использованием индексов достигается в первую очередь за счёт того, что индекс имеет структуру, оптимизированную под поиск — например, в PostgreSQL b-дерева. Индекс лучше использовать на тех столбцах таблицы, на которые вы чаще всего будете накладывать условия через where column_name = ...

Пример

Пусть у нас есть таблица:

```
CREATE TABLE book (
book_id SERIAL PRIMARY KEY,
title VARCHAR (50),
price DECIMAL (8,2),
amount INT,
author_id INT NOT NULL,
FOREIGN KEY (author_id) REFERENCES author (author_id)
);
```

К которой часто выполняется такой запрос:

```
SELECT *
FROM book
WHERE title='название книги';
```

А в самой таблице хранится много записей книг. Создадим индекс в соответствие с SELECT запросом, который часто применяется к таблице book

```
CREATE INDEX book_title_index ON book(title);
```

Задание

1. Создать произвольный SELECT запрос с обязательным присутствием WHERE
2. Создать индекс и включить в него те столбцы, которые вошли в условие отбора в запросе выборке. Или в контексте своей практической работы создать произвольный индекс в структуре своей БД.

Требования к отчёту

1. Титульная страница

   Титульная страница должна содержать:

   1. Наименование учебного учреждения
   2. Наименование предмета
   3. Фамили и инициалы студента
   4. Фамилию и инициалы преподавателя, принимающего работу
   5. Номер индивидуального задания (если имеется)

2. Цель работы
3. Выполнение задания
4. Вывод по работе

ЛАБОРАТОРНАЯ РАБОТА № 13

Знакомство с ограничениями-проверками

Цель работы: ознакомиться с ограничениями-проверками в СУБД PostgreSQL.

Теоретическая часть

Ограничения-проверки

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

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

Выполнение третье строки в скрипте ниже покажет ошибку Schema Error: error: new row for relation "product" violates check constraint "product_price_check"

```
INSERT INTO product(name, price) VALUES ('Компьютер Игровой', 40000.00);
INSERT INTO product(name, price) VALUES ('Компьютер Офисный', 120000.00);
-- Выполнение вставки ниже вызовет ошибку
INSERT INTO product(name, price) VALUES ('Калькулятор Игровой', -120000.00);
```

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 столбец, для которого задаётся ограничение, иначе оно не имеет большого смысла.

Ограничения-проверка с именем

Вы можете также присвоить ограничению отдельное имя. Это улучшит сообщения об ошибках и позволит вам ссылаться на это ограничение, когда вам понадобится изменить его. Сделать это можно так:

```
CREATE TABLE product (
product_no serial PRIMARY KEY,
name text,
price numeric CONSTRAINT positive_price CHECK (price > 0)
);
```

То есть, чтобы создать именованное ограничение, напишите ключевое слово CONSTRAINT, а за ним идентификатор и собственно определение ограничения. (Если вы не определите имя ограничения таким образом, система выберет для него имя за вас.)

Ограничение-проверка на несколько столбцов

Ограничение-проверка может также ссылаться на несколько столбцов. Например, если вы храните обычную цену и цену со скидкой, так вы можете гарантировать, что цена со скидкой будет всегда меньше обычной:

```
CREATE TABLE product (
product_no serial PRIMARY KEY,
name text,
price numeric CHECK (price > 0),
discounted_price numeric CHECK (discounted_price > 0),
CHECK (price > discounted_price)
);
```

Первые два ограничения определяются похожим образом, но для третьего используется новый синтаксис. Оно не связано с определённым столбцом, а представлено отдельным элементом в списке. Определения столбцов и такие определения ограничений можно переставлять в произвольном порядке.

Про первые два ограничения можно сказать, что это ограничения столбцов, тогда как третье является ограничением таблицы, так как оно написано отдельно от определений столбцов. Ограничения столбцов также можно записать в виде ограничений таблицы, тогда как обратное не всегда возможно, так как подразумевается, что ограничение столбца ссылается только на связанный столбец. (Хотя PostgreSQL этого не требует, но для совместимости с другими СУБД лучше следовать это правилу.) Ранее приведённый пример можно переписать и так:

```
CREATE TABLE product (
product_no serial PRIMARY KEY,
name text,
price numeric,
```

```
CHECK (price > 0),
discounted_price numeric, CHECK (discounted_price > 0),
CHECK (price > discounted_price)
);
```

Или даже так:

```
CREATE TABLE product (
product_no serial PRIMARY KEY,
name text,
price numeric CHECK (price > 0),
discounted_price numeric,
CHECK (discounted_price > 0 AND price > discounted_price)
);
```

Это дело вкуса.

Ограничениям таблицы можно присваивать имена так же, как и ограничениям столбцов:

```
CREATE TABLE product (
product_no serial PRIMARY KEY,
name text,
price numeric,
CHECK (price > 0),
discounted_price numeric,
CHECK (discounted_price > 0),
CONSTRAINT valid_discount CHECK (price > discounted_price)
);
```

Задание

Вариант 0

Создать таблицу processor

Со следующими полями:

- model - модель процессора
- threeds - максимальное количество потоков, который может обработать процессор. Целое положительное число не равное нулю.
- clock_rate - максимальная тактовая частота процессора в гигагерцах

При этом поля должны иметь следующие характеристики:

- model - не пустое значение до 20 символов
- threeds - целое положительное число не равное нулю.
- clock_rate - положительное число формата xx.xx не равное нулю.

Необходимо выбирать наименьший подходящий тип, данный для каждого поля. Задать все возможные ограничения проверки для всех полей, согласно их характеристикам. Заполнить таблицу 4-мя записями содержащие случайно выбранные модели 4-х моделей процессоров.

Вариант 1

Создать таблицу student

Со следующими полями:

- full_name - ФИО студента
- age - возраст
- rating - рейтинг

При этом поля должны иметь следующие характеристики:

- full_name - не пустое значение до 80 символов
- age - целое число от 0 до 150 не равное NULL
- rating - число от 0 до 100, с возможными двумя знаками после запятой, с повышенными требованиями по точности при вычислениях, не равное NULL

Необходимо выбирать наименьший подходящий тип, данный для каждого поля. Задать все возможные ограничения проверки для всех полей, согласно их характеристикам. Заполнить таблицу 4-мя записями содержащие ФИО 4 случайных студента из вашей группы

Вариант 2

Создать таблицу airplane

Со следующими полями:

- model - модель самолёта
- max_speed - максимальная скорость
- max_passenger_count - максимальная загрузка самолёта пассажирами

При этом поля должны иметь следующие характеристики:

- model - не пустое значение до 40 символов
- max_speed - число от 0 до 5000, с возможными двумя знаками после запятой, с повышенными требованиями по точности при вычислениях, не равное NULL
- max_passenger_count - целое число от 0 до 1200 не равное NULL

Необходимо выбирать наименьший подходящий тип, данный для каждого поля. Задать все возможные ограничения проверки для всех полей, согласно их характеристикам. Заполнить таблицу 4-мя записями содержащие случайно выбранные модели 4-х пассажирских самолётов.

Требования к отчёту

1. Титульная страница

   Титульная страница должна содержать:

   1. Наименование учебного учреждения
   2. Наименование предмета
   3. Фамили и инициалы студента
   4. Фамилию и инициалы преподавателя, принимающего работу
   5. Номер индивидуального задания (если имеется)
2. Цель работы
3. Выполнение задания
4. Вывод по работе

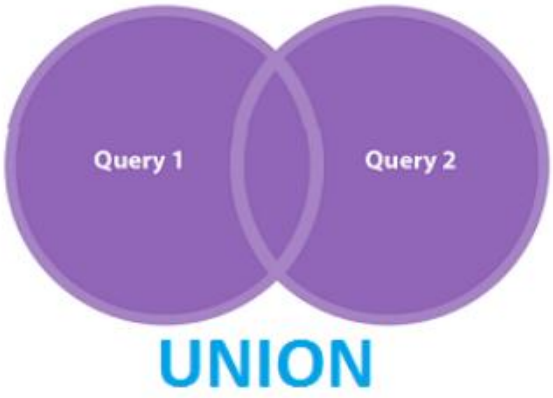
ЛАБОРАТОРНАЯ РАБОТА № 14

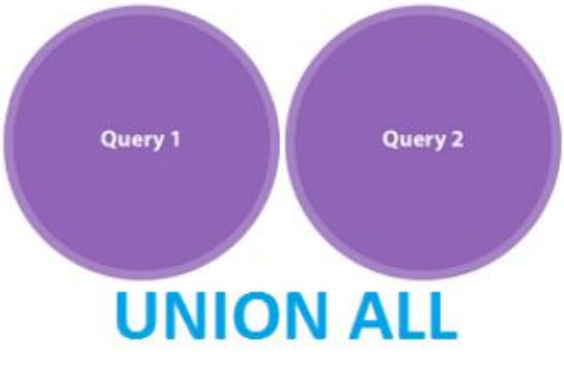
Сочетание запросов. Union, Intersect, Except

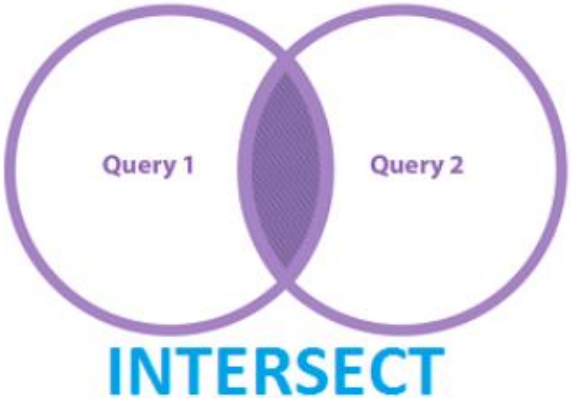
Цель работы: изучить механизм сочетания запросов. Вспомнить операции над множествами: объединение, пересечение и вычитание. Применить на практике операции Union, Intersect, Except.

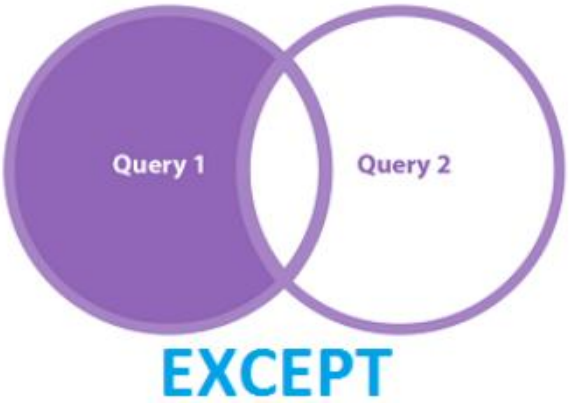
Теоретическая часть

UNION по сути добавляет результаты второго запроса к результатам первого (хотя никакой порядок возвращаемых строк при этом не гарантируется). Более того, эта операция убирает дублирующиеся строки из результата так же, как это делает DISTINCT, если только не указано UNION ALL.









Оператор INTERSECT возвращает только те строки, которые есть одновременно в обоих запросах.

EXCEPT возвращает все строки, которые есть в результате первого запроса, но отсутствуют в результате второго. (Иногда это называют разницей между двух запросов). И здесь дублирующиеся строки отфильтровываются, если не указано ALL.

Чтобы использовать оператор INTERSECT, UNION, EXCEPT, столбцы (select_list) следующим правилам:

1. Количество столбцов и их порядок должны совпадать.
2. Типы данных столбцов должны быть совместимы.

Примеры INTERSECT, UNION, UNION ALL, EXCEPT

Исходные данные

```
SELECT title, release_year FROM most_popular_films;
```

title	release_year
Матрица	1999
Гарфилд	2004
Рататуй	1999
Бойцовский клуб	1999
Шрэк	2001

```
SELECT title, release_year FROM top_rated_films;
```

title	release_year
Матрица	1999
Брат 2	2000
Гарфилд	2004
Стюарт Литтл	1999
Рататуй	1999

Примеры INTERSECT

```
SELECT title, release_year FROM top_rated_films
INTERSECT
SELECT title, release_year FROM most_popular_films;
```

title	release_year
Матрица	1999
Рататуй	1999
Гарфилд	2004

Пример UNION

```
SELECT title, release_year FROM top_rated_films
UNION
SELECT title, release_year FROM most_popular_films;
```

title	release_year
Брат 2	2000
Стюарт Литтл	1999
Матрица	1999
Шрэк	2001
Бойцовский клуб	1999
Гарфилд	2004
Рататуй	1999

Пример UNION ALL

```
SELECT title, release_year FROM top_rated_films
UNION ALL
SELECT title, release_year FROM most_popular_films;
```

title	release_year
Матрица	1999
Брат 2	2000
Гарфилд	2004
Стюарт Литтл	1999
Рататуй	1999
Матрица	1999
Гарфилд	2004
Рататуй	1999
Бойцовский клуб	1999
Шрэк	2001

Пример EXCEPT

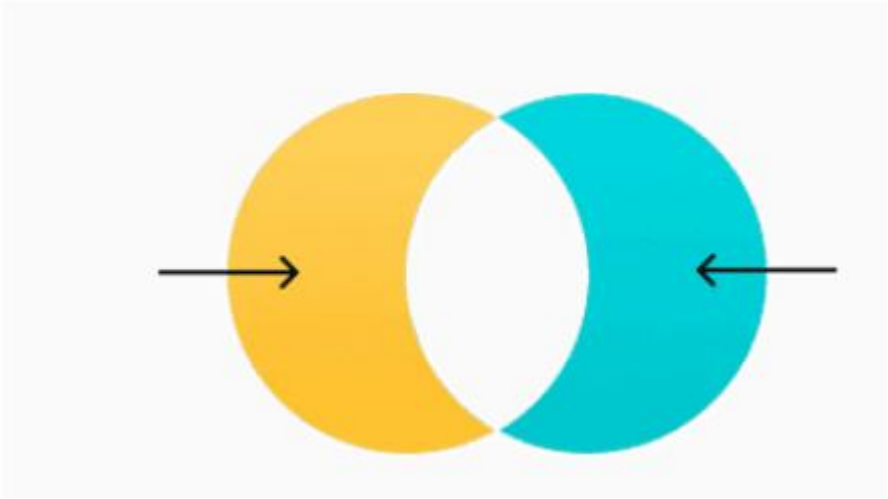
```
SELECT title, release_year FROM top_rated_films
EXCEPT
SELECT title, release_year FROM most_popular_films;
```

title	release_year
Стюарт Литтл	1999
Брат 2	2000

Задание

Вариант 1

Вывести все фильм из таблиц top_rated_films, most_popular_films, которые одновременно только в одной таблице.



(если кружки — это таблицы, то записи из жёлтой и голубой области нужно вывести)

Содержание таблиц top_rated_films , most_popular_films : Пример результата работы:

```
SELECT title, release_year FROM most_popular_films;
```

title	release_year
Матрица	1999
Гарфилд	2004
Рататуй	1999
Бойцовский клуб	1999
Шрэк	2001

```
SELECT title, release_year FROM top_rated_films;
```

title	release_year
Матрица	1999
Брат 2	2000
Гарфилд	2004
Стюарт Литтл	1999
Рататуй	1999

Требования к отчёту

1. Титульная страница

   Титульная страница должна содержать:

   1. Наименование учебного учреждения
   2. Наименование предмета
   3. Фамили и инициалы студента
   4. Фамилию и инициалы преподавателя, принимающего работу
   5. Номер индивидуального задания (если имеется)
2. Цель работы
3. Выполнение задания
4. Вывод по работе

СПИСОК ЛИТЕРАТУРЫ

1. Турманов, В. SQL для хранения, обработки и анализа данных / В. Турманов, Б. Гайфуллин. – М.: СОЛОН-ПРЕСС. URL : https://www.studentlibrary.ru/book/ISBN9785913594631.html (дата обращения: 10.03.2025)
2. Новиков, Б. А. Основы технологий баз данных : учебное пособие / Новиков Б. А. , Горшкова Е. А. , Графеева Н. Г. ; под ред. Е. В. Рогова. - 2-е изд. – М. : ДМК Пресс. URL : https://www.studentlibrary.ru/book/ISBN9785970608418.html (дата обращения: 10.03.2025)
3. Рогов, Е. В. PostgreSQL изнутри / Е. В. Рогов. – М. : ДМК Пресс. URL: https://www.studentlibrary.ru/book/ISBN9785937001221.html (дата обращения: 10.03.2025)
4. Мартишин, С. А. Базы данных. Практическое применение СУБД SQL- и NoSQL-типа для проектирования информационных систем : учебное пособие / С.А. Мартишин, В.Л. Симонов, М.В. Храпченко. — М. : ФОРУМ : ИНФРА-М. URL : https://znanium.ru/catalog/product/1912454 (дата обращения: 10.03.2025)