

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания
по выполнению лабораторных работ
по дисциплине «Проектирование систем искусственного
интеллекта»

для студентов направления подготовки
09.03.02 «Информационные системы и технологии»
Профиль «Прикладное программирование в интеллектуальных
информационных системах»

Ставрополь, 2026

Лабораторная работа № 1. Начало работы с PostgreSQL в инжиниринге данных

Цель работы: познакомиться с особенностями применения SQL в инжиниринге данных

Теоретическое обоснование

Системные требования

PostgreSQL 13 можно установить не на все версии Windows, в частности официально поддерживаются следующие версии и только 64 битные:

- Windows Server 2012 R2;
- Windows Server 2016;
- Windows Server 2019.

Как видим, в официальном перечне нет Windows 10, однако установка на данную систему проходит без проблем, как и последующее функционирование PostgreSQL.

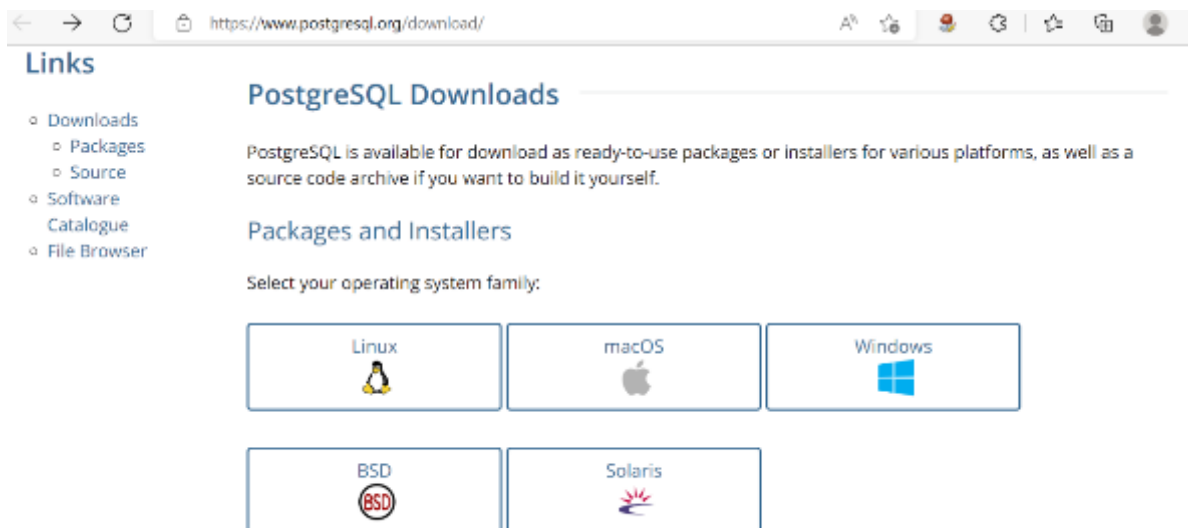
Кроме этого есть и другие требования:

- Процессор как минимум с частотой 1 гигагерц;
- 2 гигабайта оперативной памяти;
- Как минимум 512 мегабайт свободного места на диске (рекомендуется больше для установки дополнительных компонентов);
- Также рекомендовано, чтобы все обновления операционной системы Windows были установлены.

Ссылка для скачивания

[PostgreSQL: Downloads](#)

Выберите вам нужную операционную систему, на которую будете устанавливать PostgreSQL, например, Windows,



На следующей странице выберите ссылку загрузки для инсталляции



На следующей странице вам следует выбрать версию PostgreSQL, загрузить инсталляцию на свой диск и установить программу, следуя ее диалоговым окнам.

Перед тем как двигаться дальше, давайте разберемся в основах архитектуры системы PostgreSQL. Составив картину взаимодействия частей PostgreSQL, будет проще понять материал этой части. Говоря техническим языком, PostgreSQL реализован в архитектуре клиент-сервер. Рабочий сеанс PostgreSQL включает следующие взаимодействующие процессы (программы):

- Главный серверный процесс, управляющий файлами баз данных, принимающий подключения клиентских приложений и выполняющий различные запросы клиентов к базам данных. Эта программа сервера БД называется postgres.

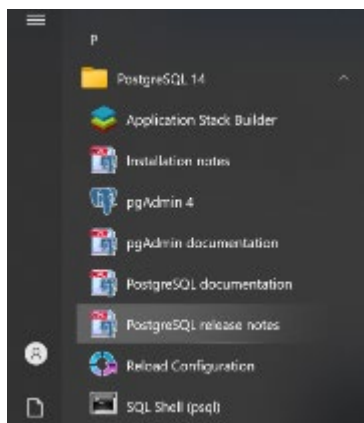
- Клиентское приложение пользователя, желающее выполнять операции в базе данных. Клиентские приложения могут быть очень разнообразными: это может быть текстовая утилита, графическое приложение, веб-сервер, использующий базу данных для отображения веб-страниц, или специализированный инструмент для обслуживания БД. Некоторые клиентские приложения поставляются в составе дистрибутива PostgreSQL, однако большинство создают сторонние разработчики.

Как и в других типичных клиент-серверных приложениях, клиент и сервер могут располагаться на разных компьютерах. В этом случае они взаимодействуют по сети TCP/IP. Важно не забывать это и понимать, что файлы, доступные на клиентском компьютере, могут быть недоступны (или доступны только под другим именем) на компьютере-сервере.

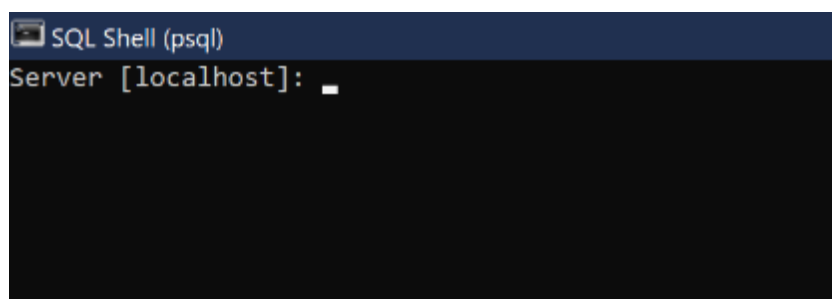
Сервер PostgreSQL может обслуживать одновременно несколько подключений клиентов. Для этого он запускает («порождает») отдельный процесс для каждого подключения. Можно сказать, что клиент и серверный процесс общаются, не затрагивая главный процесс postgres. Таким образом, главный серверный процесс всегда работает и ожидает подключения клиентов, принимая которые, он организует взаимодействие клиента и отдельного серверного процесса. (Конечно всё это происходит незаметно для пользователя, а эта схема рассматривается здесь только для понимания).

Методика выполнения

Далее мы будем использовать командную строку в качестве клиентского приложения для взаимодействия с сервером.

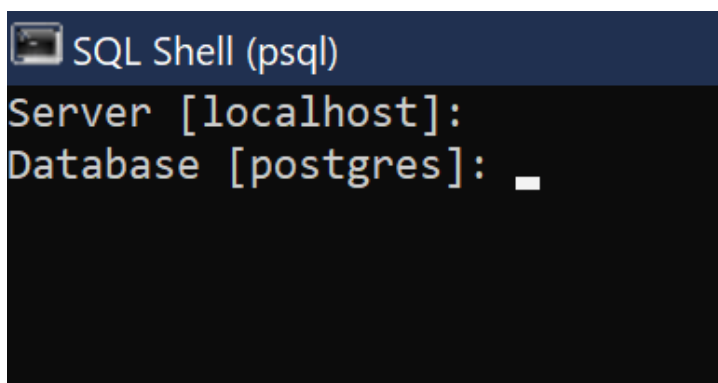


После запуска SQL Shell нас встречает надпись Server [local|host]



В квадратных скобках введено стандартное значение. Вводим Enter.

Появляется надпись Database [postgres]:



Нажимаем Enter.

Следующая строка Port [5432]: Порт по умолчанию - 5432. Если PostgreSQL устанавливается впервые, то он скорее всего свободен. Если окажется, что этот порт уже занят другим экземпляром PostgreSQL, то можно указать другое значение, например 5433.

```
SQL Shell (psql)
Server [localhost]:
Database [postgres]:
Port [5432]: _
```

Нажимает Enter.

Появляется строка ввода логина и потом строка пароля. Пароль вводится тот, который был введен при установке.

```
SQL Shell (psql)
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]: _
```

В строке Username [postgres]: нажимаете Enter. В строке вводите Пароль вводится тот, который был введен при установке. При вводе пароля вы не увидите символов. Это так задумано для защиты пароля от посторонних глаз.

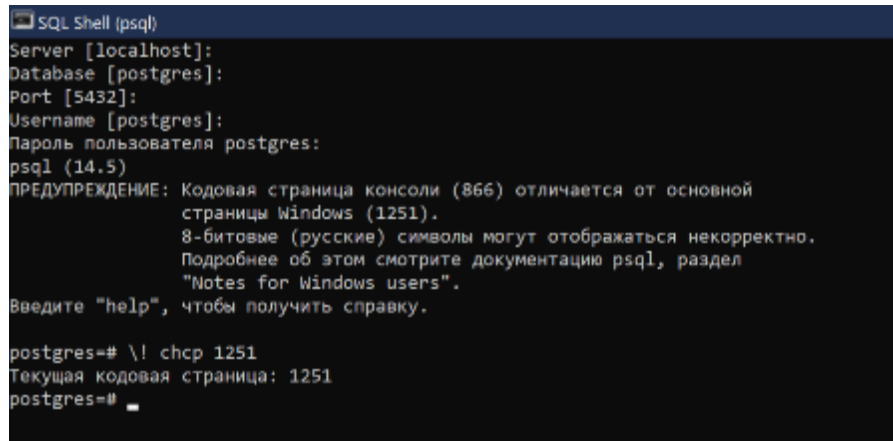
Нажимает Enter.

При успешном входе будет написана версия postgres. Обратите внимание на сообщение, которое говорит о том, что некоторые символы кириллицы могут отображаться некорректно, если мы не введем нужную команду перед началом работы с базой.

```
SQL Shell (psql)
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Пароль пользователя postgres:
psql (14.5)
ПРЕДУПРЕЖДЕНИЕ: Кодовая страница консоли (866) отличается от основной
страницы Windows (1251).
8-битовые (русские) символы могут отображаться некорректно.
Подробнее об этом смотрите документацию psql, раздел
"Notes for Windows users".
Введите "help", чтобы получить справку.
postgres=# _
```

В строке консоли наберите следующую команду:

`\! chcp 1251`



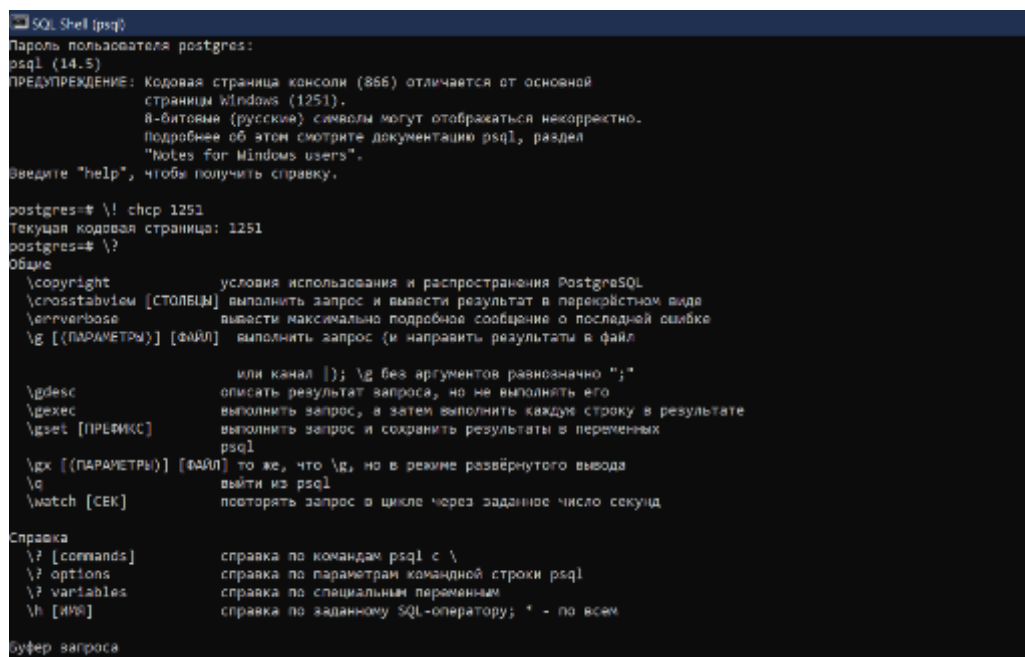
```
SQL Shell (psql)
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Пароль пользователя postgres:
psql (14.5)
ПРЕДУПРЕЖДЕНИЕ: Кодовая страница консоли (866) отличается от основной
                  страницы Windows (1251).
                  8-битовые (русские) символы могут отображаться некорректно.
                  Подробнее об этом смотрите документацию psql, раздел
                  "Notes for Windows users".
Введите "help", чтобы получить справку.

postgres=# \! chcp 1251
Текущая кодовая страница: 1251
postgres=#
```

Таким образом мы изменили кодовую страницу на 1251.

Данную команду рекомендуется вводить в начале каждой сессии.

Заметим, что команды, начинающиеся с обратного слеша (`\`) обращаются не к базе данных, а к окну консоли. Чтобы узнать список всех доступных команд, нужно ввести команду `\?`.



```
SQL Shell (psql)
Пароль пользователя postgres:
psql (14.5)
ПРЕДУПРЕЖДЕНИЕ: Кодовая страница консоли (866) отличается от основной
                  страниц Windows (1251).
                  8-битовые (русские) символы могут отображаться некорректно.
                  Подробнее об этом смотрите документацию psql, раздел
                  "Notes for Windows users".
Введите "help", чтобы получить справку.

postgres=# \! chcp 1251
Текущая кодовая страница: 1251
postgres=# \?
Общие
 \copyright      условия использования и распространения PostgreSQL
 \crosstabview [СТОЛБЦЫ] выполнить запрос и вывести результат в перекрёстном виде
 \errverbose      вывести максимально подробное сообщение о последней ошибке
 \g [(ПАРАМЕТРЫ)] [ФАЙЛ] выполнить запрос (и направить результаты в файл
                       или канал ); \g без аргументов равнозначно ";"
 \gdesc          описать результат запроса, но не выполнять его
 \gexec          выполнить запрос, а затем выполнить каждую строку в результате
 \gset [ПЕРЕМЕННЫЕ] выполнить запрос и сохранить результаты в переменных
                  psql
 \gx [(ПАРАМЕТРЫ)] [ФАЙЛ] то же, что \g, но в режиме развёрнутого вывода
 \q             выйти из psql
 \watch [СЕК]    повторять запрос в цикле через заданное число секунд

Справка
 \? [commands]   справка по командам psql с \
 \? options       справка по параметрам командной строки psql
 \? variables     справка по специальным переменным
 \h [ИМЯ]         справка по заданному SQL-оператору; * - по всем

буфер запроса
```

Появится список всех команд, разделенных на разделы. Просмотрите эти команды.

Введите команду `\l` – это команда. Эта команда позволит вам увидеть все доступные базы данных.

```
postgres=# \l
```

Список баз данных					
Имя	Владелец	Кодировка	LC_COLLATE	LC_CTYPE	Права доступа
postgres	postgres	UTF8	Russian_Russia.1251	Russian_Russia.1251	
template0	postgres	UTF8	Russian_Russia.1251	Russian_Russia.1251	=c/postgres +
template1	postgres	UTF8	Russian_Russia.1251	Russian_Russia.1251	=c/postgres +
(3 строки)					
postgres=#					

По умолчанию их будет уже три, но для обучения давайте создадим свою собственную базу.

Для создания базы данных необходимо ввести команду CREATE DATABASE.

CREATE DATABASE starter;

starter – название новой базы данных. В конце команды необходимо поставить точку с запятой, поскольку уже мы обращаемся к базе данных.

После успешного выполнения команды консоль выведет сообщение

```
postgres=# \l
```

Список баз данных					
Имя	Владелец	Кодировка	LC_COLLATE	LC_CTYPE	Права доступа
postgres	postgres	UTF8	Russian_Russia.1251	Russian_Russia.1251	
template0	postgres	UTF8	Russian_Russia.1251	Russian_Russia.1251	=c/postgres +
template1	postgres	UTF8	Russian_Russia.1251	Russian_Russia.1251	=c/postgres +
(3 строки)					
postgres=# CREATE DATABASE starter;					
CREATE DATABASE					
postgres=#					

Повторим команду \l и увидим в списке уже новую созданную нами базу данных.

```
postgres=# \l
```

Список баз данных					
Имя	Владелец	Кодировка	LC_COLLATE	LC_CTYPE	Права доступа
postgres	postgres	UTF8	Russian_Russia.1251	Russian_Russia.1251	
starter	postgres	UTF8	Russian_Russia.1251	Russian_Russia.1251	
template0	postgres	UTF8	Russian_Russia.1251	Russian_Russia.1251	=c/postgres +
template1	postgres	UTF8	Russian_Russia.1251	Russian_Russia.1251	=c/postgres +
(4 строки)					
postgres=#					

Для работы с этой базой данных нам необходимо сначала присоединиться к ней.

В начале строки консоли мы сейчас видим postgres=# - это название базы к которой мы сейчас подключены. Чтобы подключиться к другой базе данных используем команду

\c starter

```
postgres=# \c starter
Вы подключены к базе данных "starter" как пользователь "postgres".
postgres=#
```

При верно введенной команде видим сообщение об успешном подключении к базе данных. Здесь мы видим название текущей базы и текущего пользователя. И как можно заметить начало строки в консоле заменилась на starter=#

Чтобы вернуться обратно нужно ввести \c postgres

Если необходимо удалить какую-нибудь базу данных, то используется команда DROP DATABASE. Она полностью удаляет выбранную базу данных и использовать ее нужно с осторожностью.

Не пытайтесь удалять одну из трех первоначальных баз данных, так как это может вызвать необратимые последствия и придется переустанавливать postgres.

Основные команды

\! chcp 1251	Команда CHCP используется для просмотра или изменения текущей кодовой страницы в окне командной строки Windows. Кодовая страница (Code Page или сокращенно CP) определяет соответствие между двоичным кодом и соответствующим ему символом, отображаемом на экране. Для кодирования текстов на русском языке (то есть букв кириллицы) наиболее широко применяется следующая кодовая страница : Windows-1251, она же Microsoft code page 1251 (CP1251) в операционных системах семейства Windows. Данная команда необходима для корректного отображения кириллицы в окне консоли.
--------------	--

	Данное решение временное – только до следующего запуска консоли. Чтобы навсегда поменять кодовую страницу, нужно найти файл: <i>C:\Program Files\PostgreSQL\13\scripts\runpsql.bat</i> Скопируйте его, например в папку документы. Нажмите на него правой клавишей мыши и выбрать пункт изменить его содержание добавлением одной строки перед запуском psql.exe. В результате будет примерно такой код: cmd.exe /c chcp 1251 REM Run psql <i>"C:\Program Files\PostgreSQL\13\bin\psql.exe" -h</i> <i>%server% -U %username% -d %database% -p</i> <i>%port%</i> Сохраните файл и после этого замените исходный файл на полученный в папке: <i>C:\Program Files\PostgreSQL\13\scripts\</i> В результате подобных изменений ошибок возникать не должно и не придется делать манипуляции каждый раз в консоли.
\?	Команда \? выводит список всех доступных команд по разделам. Аналог команды "help".
\l	Команда \l выводит список всех существующих баз данных.
CREATE DATABASE	Команда CREATE DATABASE [имя_базы_данных] создаёт новую базу данных с названием [имя_базы_данных]. Пример:

	<code>CREATE DATABASE starter;</code>
DROP DATABASE	Команда DROP DATABASE [имя_базы_данных] удаляет существующую базу данных с названием [имя_базы_данных]. Пример: <code>DROP DATABASE starter;</code>
<code>\c</code>	Команда <code>\c [имя_бд]</code> подключит вас к базе данных с названием [имя_бд]. Пример: <code>\c starter</code>

Лабораторная работа № 2. Начало работы с pgAdmin4

Цель работы: освоить интерфейс pgAdmin4 и использовать при работе с данными

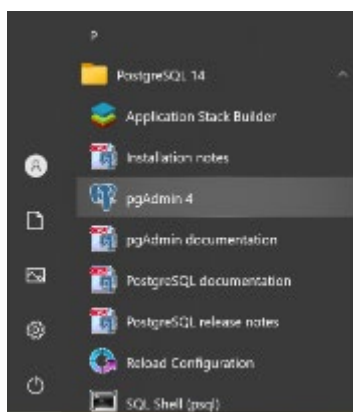
Теоретическое обоснование

pgAdmin 4 это новая реализация бесплатного инструмента управления СУБД PostgreSQL. **pgAdmin 4** используется для написания SQL запросов, разработки процедур, функций, а также для администрирования PostgreSQL.

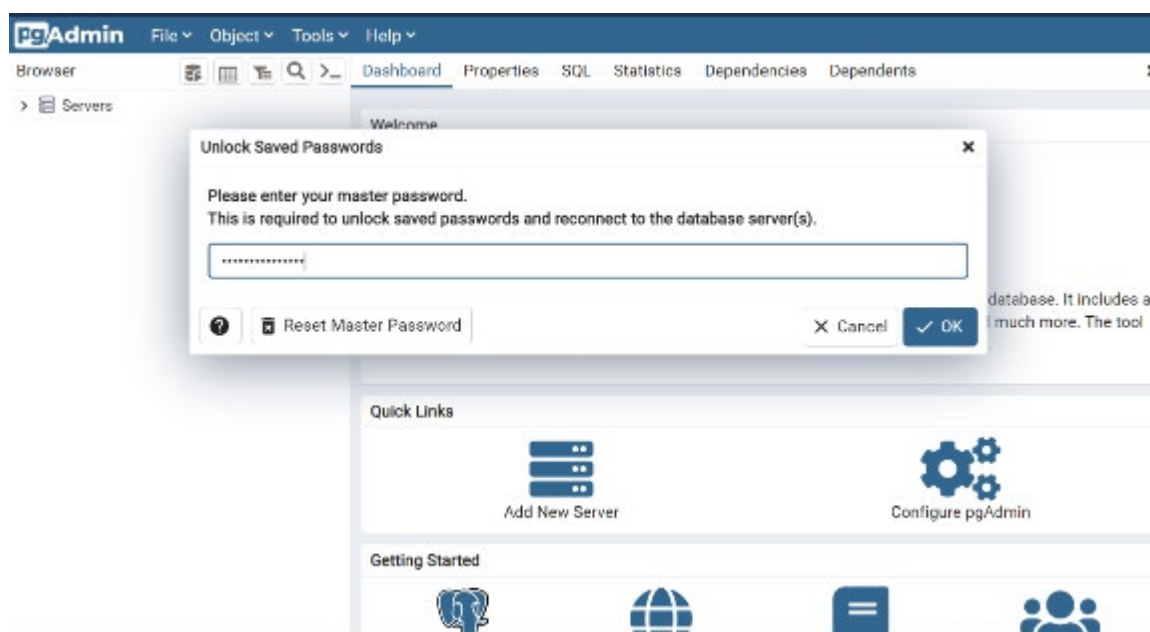
Этот инструмент был установлен в комплекте инструментов postgres. Давайте посмотрим на то, как мы начать работать с ним.

Методика выполнения

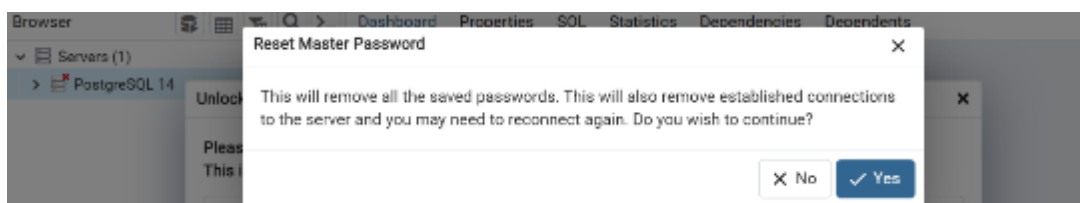
1. Запустить pgAdmin4 можно через меню пуск. Он находится на вкладке Пуск-PostgreSQL14-pgAdmin4.



2. После запуска, вас встретит окно ввода пароля. Сюда необходимо ввести тот пароль, что вы указали при установке : “*****”.

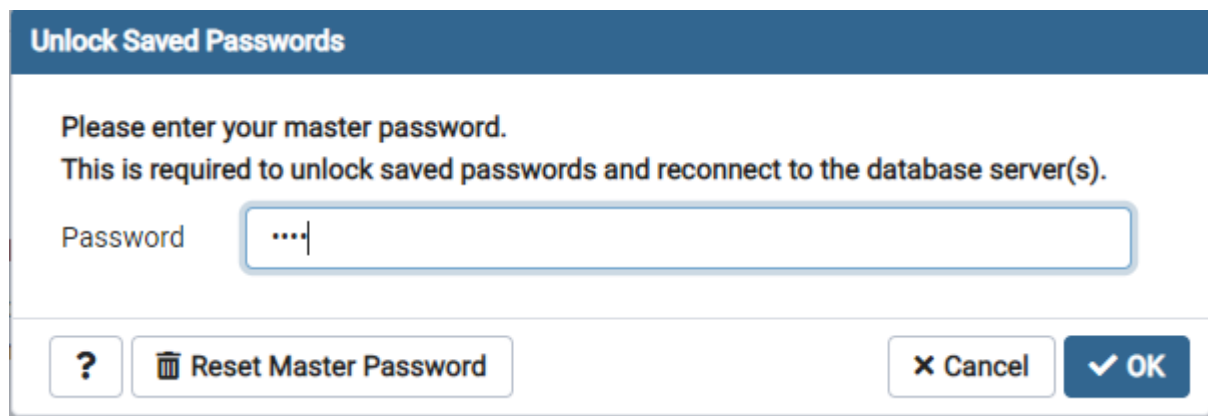


3. Если вы забыли пароль, то его можно поменять, нажав на кнопку Reset Master Password. В окне Reset Master Password вводим новый пароль и применяем изменения.

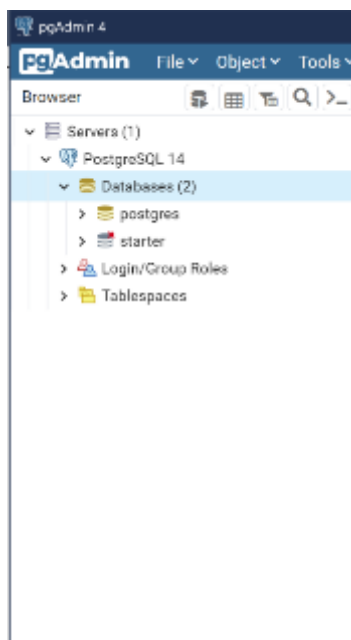


4. После входа, вас встретит пустой экран, без какой-либо информации. Мы не видим её так как не подключены ни к одному серверу. Нажмите на вкладку Servers в левой части экрана, а затем сделайте двойной щелчок по вкладке PostgreSQL14.

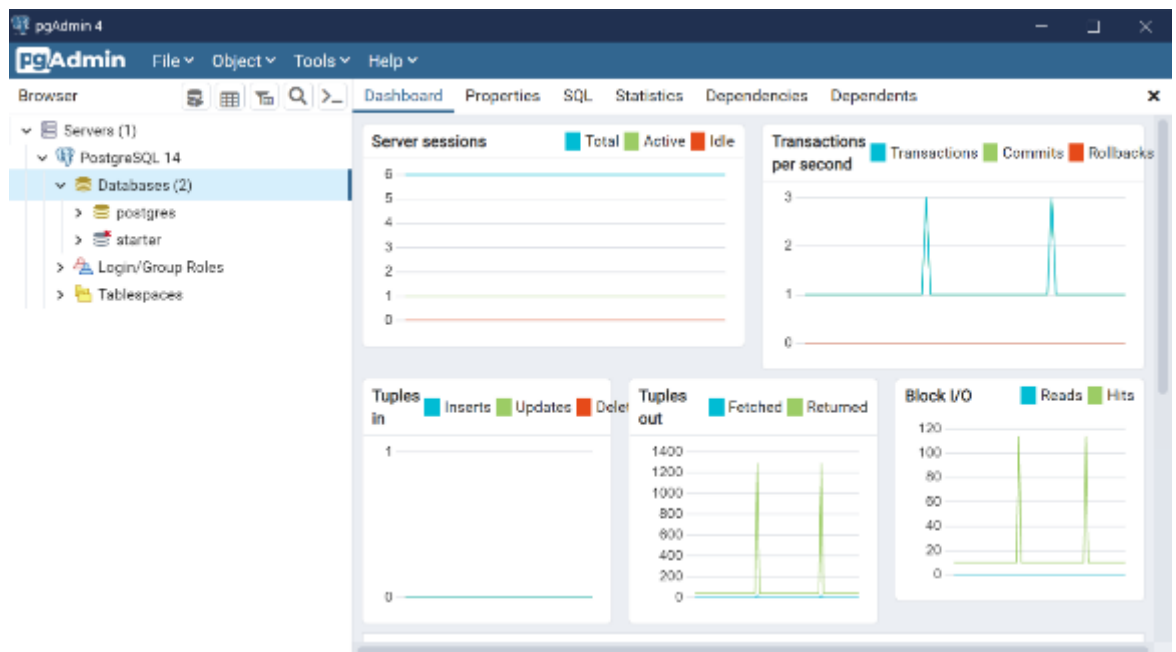
5. Нам предлагают повторно ввести пароль. Сюда мы тоже вводим то, что придумали при установке : “***”.



6. После ввода пароля мы сможем увидеть все созданные нами базы данных. Для отображения метрик по нужной вам базе данных, нажмите на неё. Они находятся под вкладкой "Databases".



7. Для каждой из этих баз можно просматривать текущие метрики по запросам во вкладке Dashboard. Эта вкладка выбрана у вас автоматически.



Лабораторная работа № 3. Работа с таблицами в PostgreSQL

Цель работы: освоить работу с таблицами в PostgreSQL

Откроем консоль SQL Shell и повторим все процедуры подключения. Подключимся к ранее созданной базе starter с помощью уже знакомой команды \c starter.

```
postgres=# \c starter
Вы подключены к базе данных "starter" как пользователь "postgres".
starter=#
```

Создадим таблицу в базе с использованием команды CREATE TABLE.

Синтаксис команды следующий:

Команда CREATE TABLE [имя_таблицы] ([имя_столбца] [тип данных столбца],...) создает таблицу с названием [имя_таблицы]. Внутри круглых скобок заполняется нужное количество столбцов, их имена и соответствующие имени типы.

Создадим таблицу student. В круглых скобках указываем имена столбцов с их типами:

```
CREATE TABLE student (
    id INT,
    name VARCHAR(50),
```

```
);  
surname VARCHAR(50),  
gender VARCHAR(50),  
mobile VARCHAR(50),  
birthday DATE  
);  
  
SQL Shell (psql)  
Server [localhost]:  
Database [postgres]:  
Port [5432]:  
Username [postgres]:  
Пароль пользователя postgres:  
psql (14.5)  
ПРЕДУПРЕЖДЕНИЕ: Кодовая страница консоли (866) отличается от основной  
                  страницы Windows (1251).  
                  8-битовые (русские) символы могут отображаться некорректно.  
                  Подробнее об этом смотрите документацию psql, раздел  
                  "Notes for Windows users".  
Введите "help", чтобы получить справку.  
  
postgres=# \i chcr 1251  
Текущая кодовая страница: 1251  
postgres=# \c starter  
Вы подключены к базе данных "starter" как пользователь "postgres".  
starter=# CREATE TABLE student(  
starter=# id INT, name VARCHAR(50), surname VARCHAR(50), gender VARCHAR(50),  
starter=# mobile VARCHAR(50), birthday DATE);  
CREATE TABLE  
starter=#
```

При успешном создании таблицы после нажатия Enter будет выведено сообщение CREATE TABLE.

Чтобы посмотреть какие таблицы и какие отношения между ними существуют используется команда `\d` выводит список отношений между схемами.

Введите эту команду

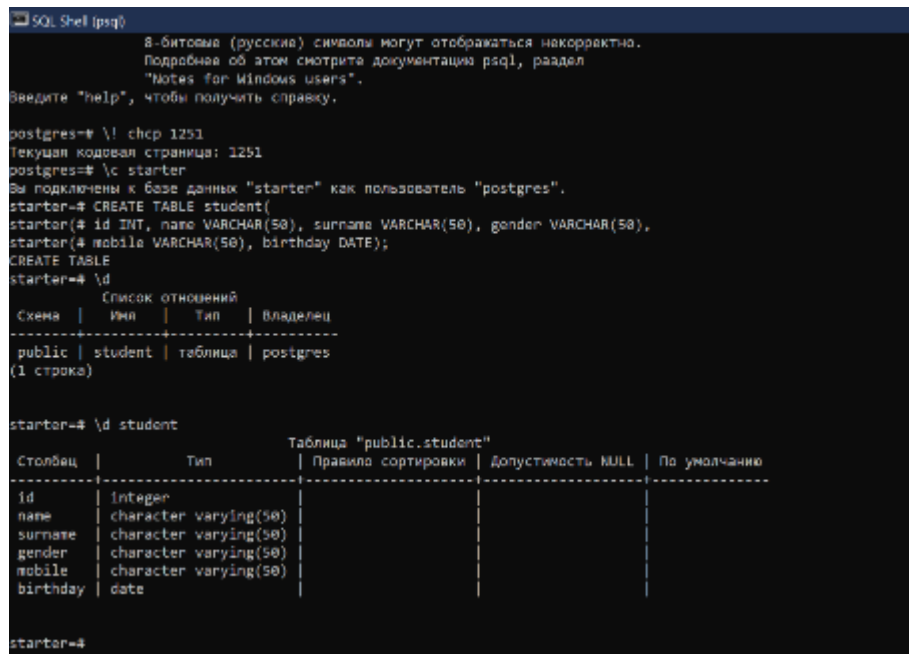
```
SQL Shell (psql)  
Server [localhost]:  
Database [postgres]:  
Port [5432]:  
Username [postgres]:  
Пароль пользователя postgres:  
psql (14.5)  
ПРЕДУПРЕЖДЕНИЕ: Кодовая страница консоли (866) отличается от основной  
                  страницы Windows (1251).  
                  8-битовые (русские) символы могут отображаться некорректно.  
                  Подробнее об этом смотрите документацию psql, раздел  
                  "Notes for Windows users".  
Введите "help", чтобы получить справку.  
  
postgres=# \i chcr 1251  
Текущая кодовая страница: 1251  
postgres=# \c starter  
Вы подключены к базе данных "starter" как пользователь "postgres".  
starter=# CREATE TABLE student(  
starter=# id INT, name VARCHAR(50), surname VARCHAR(50), gender VARCHAR(50),  
starter=# mobile VARCHAR(50), birthday DATE);  
CREATE TABLE  
starter=# \d  
                  Список отношений  
Схема | Имя | Тип | Владелец  
-----  
public | student | таблица | postgres  
(1 строка)  
  
starter=#
```

Появится таблица со всеми нами созданными элементами таблицы и последовательностью.

Если нам необходимо узнать информацию о таблице student, то используется команда `\d [имя_таблицы]`, которая показывает характеристики таблицы с данным именем.

Пример :

`\d student;`



```
SQL Shell (psql)
8-битовые (русские) символы могут отображаться некорректно.
Подробнее об этом смотрите документацию psql, раздел
"Notes for Windows users".
Введите "help", чтобы получить справку.

postgres=# \! chcp 1251
Текущая кодовая страница: 1251
postgres=# \c starter
Вы подключены к базе данных "starter" как пользователь "postgres".
starter=# CREATE TABLE student(
starter(# id INT, name VARCHAR(50), surname VARCHAR(50), gender VARCHAR(50),
starter(# mobile VARCHAR(50), birthday DATE);
CREATE TABLE
starter=# \d
      Список отношений
+-----+-----+-----+-----+
| Схема | Имя  | Тип   | Владелец |
+-----+-----+-----+-----+
| public | student | таблица | postgres |
+-----+-----+-----+-----+
(1 строка)

starter=# \d student
      Таблица "public.student"
+-----+-----+-----+-----+-----+
| Столбец | Тип          | Правило сортировки | Допустимость NULL | По умолчанию |
+-----+-----+-----+-----+-----+
| id       | integer      |                      |                     |               |
| name     | character varying(50) |                      |                     |               |
| surname  | character varying(50) |                      |                     |               |
| gender   | character varying(50) |                      |                     |               |
| mobile   | character varying(50) |                      |                     |               |
| birthday | date         |                      |                     |               |
+-----+-----+-----+-----+-----+
starter=#
```

Для удаления таблицы используется команда

DROP TABLE

Синтаксис этой команды следующий:

Команда **DROP TABLE [имя_таблицы]** удаляет таблицу с названием **[имя_таблицы]**.

Пример :

`DROP TABLE student;`

Выполните эту команду и удалите таблицу student.

В этом задании мы научимся управлять данными внутри таблицы. В предыдущем задании мы удалили таблицу student. Давайте пересоздадим эту таблицу и заполним ее данными.

Но для начала давайте отобразим ее содержимое с использованием команды **SELECT * FROM [имя_таблицы]**. Эта команда выводит в консоль всё содержимое таблицы с данным именем.

Пример :

```
SELECT * FROM student;
```

Знак * означает выбор всех столбцов

```
starter=# SELECT * FROM student;
 id | name | surname | gender | mobile | birthday
-----+-----+-----+-----+-----+-----
(0 строк)

starter=#
```

Если команда совершится успешно, то мы увидим в консоле графическую таблицу с содержимым. Так как мы таблицу только создали, то внутри нее конечно пусто.

Давайте заполним ее какими-нибудь данными. Вставляем данные построчно, т.е. по одному набору за раз. Это можно сделать с помощью команды

INSERT INTO [имя_таблицы] ([перечисление имён столбцов]) VALUES ([наполнение для перечисленных столбцов]) помещает данные в таблицу.

[перечисление имён столбцов] в данном случае уточняет, в какие столбцы данные будут записаны, а **[наполнение для перечисленных столбцов]** помещает данные в нужном порядке.

Пример :

```
INSERT INTO student(name, surname, gender, mobile, birthday) VALUES ('Michael', 'Scott', 'male', '+79884561237', '2000-04-12');
```

Обратите внимание на формат ввода даты дня рождения.

```
INSERT 0 1
INSERT 0 1
starter=#
```

После ввода Enter при удачном вводе данных появляется надпись
INSERT 0 1

Давайте теперь проверим содержимое таблицы с помощью уже знакомой команды `SELECT * FROM student;`

И убедимся, что таблица содержит только что введенные данные.

```
SQL Shell (psql)
starter=# CREATE TABLE student (
starter=# id INT,
starter=# name VARCHAR(50),
starter=# surname VARCHAR(50),
starter=# gender VARCHAR(50),
starter=# mobile VARCHAR(50),
starter=# birthday DATE
starter=# );
CREATE TABLE
starter=#
starter=# INSERT INTO student(name, surname, gender, mobile, birthday) VALUES ('Michael', 'Scott', 'male', '+79884561237', '2008-04-12');
INSERT 0 1
starter=# SELECT * FROM student;
 id | name  | surname | gender | mobile  | birthday
-----+-----+-----+-----+-----+-----
  1 | Michael | Scott  | male   | +79884561237 | 2008-04-12
(1 строка)

starter=#
```

Обратите внимание, что поле id пустое. Это произошло из-за того, что мы не указали столбец id для заполнения его какими-либо данными.

Мы заполнили одну строку, ну что делать, если нам надо заполнить большее количество строк. «Ручное» заполнение довольно трудоемкое. Для автоматической генерации данных можно воспользоваться специальными сайтами для генерации таблиц данными mockaroo.com, databasetestdata.com, generatedata.com или воспользоваться уже сгенерированным файлом, прикрепленный к этому заданию.

При генерации укажите число рядов 500 и задайте новой таблице имя student_auto.

Для загрузки файла в базу в консоле введите команду \i

Синтаксис этой команды \i '[путь к файлу]' позволит импортировать готовые функции из файла на устройстве.

Пример :

```
\i 'D:/ИнжинирингДанных/Postgresql/students_auto.sql'
```

Обратите внимание на формат ввода пути к файлу.

```

starter=# SELECT * FROM student;
 id | name | surname | gender | mobile | birthday
-----+-----+-----+-----+-----+-----
  1 | Michael | Scott | male | +79884561237 | 2000-04-12
(1 строка)

starter=# \i 'D:/Инжиниринг/Данных/Postgresql/students_auto.sql'
CREATE TABLE
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1
INSERT 0 1

```

При спешном вводе команды мы видим длинную вереницу сообщений об успешной вставке данных в базу.

Когда консоль забивается информацией, то необходимо ее очистить. Для этого используется команда `\! cls`

Выберем все строки новой таблицы `student_auto`
`SELECT * FROM student_auto:`

SQL Shell (psql)

```

starter=# SELECT * FROM student_auto;

```

id	name	surname	gender	mobile	birthday
1	Corey	Humberstone	Bigender	601-336-8268	2021-05-06
2	Rina	Leaves	Non-binary	440-842-7249	2020-12-18
3	Leyla	Culpen	Non-binary	798-150-7796	2020-12-01
4	Kenna	Nosby	Genderqueer	398-767-3358	2021-02-07
5	Berkie	Vule	Non-binary	779-336-6742	2021-05-30
6	Avrit	Dahl	Genderfluid	639-484-1428	2021-08-07
7	Arndrei	Cockney	Genderfluid	953-996-3483	2021-06-10
8	Gatas	Dobbie	Agender	161-421-5781	2021-03-19
9	Ulrich	Bridge	Female	975-401-0123	2021-01-15
10	Georgiana	Bahnke	Bigender	568-165-8199	2021-02-26
11	Fina	Cokly	Polygender	200-747-6136	2021-07-12
12	Chicky	Schanfelder	Genderfluid	903-625-6677	2021-08-07
13	Rodina	Denmes	Male	586-183-6597	2021-00-19
14	Jacqueline	Micah	Female	765-924-7203	2021-05-03
15	Othelia	Kernley	Genderfluid	250-350-3500	2021-03-28
16	Hedwiga	Leile	Bigender	742-197-7722	2020-12-15
17	Miguelito	Dawid	Female	580-295-1034	2021-06-04
18	Aetha	Tidgewell	Genderfluid	447-135-5218	2020-10-16
19	Baldwin	Cordall	Polygender	314-694-9253	2021-05-22
20	Lia	Kerrott	Genderfluid	220-178-8717	2021-04-12
21	Aurilia	Dansunna	Polygender	378-864-6864	2021-05-21
22	Balcina	Panderad	Female	988-699-8718	2021-04-20
23	Lewiss	Yanyshev	Female	356-598-3839	2021-05-18
24	tollen	Strang	Agender	777-620-6954	2021-07-14
25	Xynemes	Olifard	Agender	742-610-0426	2021-03-17
26	Orelia	Baunert	Non-binary	256-837-1008	2021-00-03
27	Victoir	Saile	Agender	965-743-7963	2020-09-07
28	Mort	Marchinov	Female	822-216-7261	2021-03-28
29	Rosabelle	Fenich	Agender	829-323-5779	2020-10-31
30	Anelina	Lardinar	Polygender	598-751-2283	2021-05-19
31	Enlyn	Ciobutaru	Male	801-774-4694	2021-08-13

Мы можем уточнить, какие именно столбцы мы выбираем, заменив `*` на перечисление имен столбцов.

Пример :

```
SELECT name, surname FROM student;
```

Выборка данных. Скрипты

В данном задании разберем процесс выборки данных. Для начала сделаем выборку всех данных из нашей таблицы student.

Выбираем все данные командой SELECT *FROM, но в конце добавляем новые критерии

SELECT *FROM student_auto OFFSET 10 LIMIT 50:

Командой OFFSET мы пропускаем 10 строк с начала таблицы. Количество выводимых строк ограничим командой LIMIT.

SQL Shell (psql)

```
starter=# SELECT * FROM student_auto OFFSET 10 LIMIT 50;
```

id	name	surname	gender	mobile	birthday
11	Enna	Cokly	Polygender	289-747-5136	2021-07-12
12	Chicky	Schonfelder	Genderfluid	903-625-6677	2021-08-07
13	Rodina	Dennes	Male	586-183-6597	2021-08-19
14	Jacqueline	Micah	Female	765-924-7283	2021-05-03
15	Othelia	Kennley	Genderfluid	250-350-3598	2021-03-28
16	Hedwiga	Leile	Bigender	742-197-7722	2020-12-15
17	Miguelita	Dawid	Female	580-295-1034	2021-06-04
18	Retha	Tidgewell	Genderfluid	447-135-5218	2020-10-16
19	Baldwin	Cordall	Polygender	314-694-9253	2021-05-22
20	Lia	Kernott	Genderfluid	220-178-8717	2021-04-12
21	Aurilia	Densunba	Polygender	378-864-9864	2021-05-21
22	Delcine	Pendered	Female	988-699-8710	2021-04-20
23	Lewiss	Vanyshiev	Female	356-598-3839	2021-05-18
24	Collen	Strang	Agender	777-620-6934	2021-07-14
25	Xymenes	Olifard	Agender	742-610-0426	2021-03-17
26	Orelia	Baumert	non-binary	256-837-1000	2021-08-03
27	Victoir	Saile	Agender	965-743-7963	2020-09-07
28	Mort	Nenichinov	Female	822-216-7261	2021-03-28
29	Rosabelle	Fenich	Agender	829-323-5779	2020-10-31
30	Ancline	Landiner	Polygender	598-751-2283	2021-05-19
31	Enlyn	Clobutaru	Male	801-774-4694	2021-08-13
32	Wally	Nadga	Agender	740-168-8291	2021-02-19
33	Cayla	Spata	Agender	148-609-6202	2021-08-03
34	Dacey	Lovel	Polygender	164-975-2195	2021-03-28
35	Liane	Lonansey	Bigender	132-737-9648	2020-09-02
36	Janessy	Abbots	Female	573-635-9892	2021-06-07
37	Cy	Wagnerin	Female	988-387-2252	2021-03-16
38	Pet	Bloxham	Genderqueer	103-828-3158	2021-07-23
39	Charlton	Janous	Genderqueer	248-974-1770	2021-02-28
40	Monti	Fenlon	Agender	923-167-7875	2021-02-24
41	Othelia	Deaton	Genderfluid	268-123-4960	2020-10-08

SQL Shell (psql)

```
(50 строк)
```

29	Rosabelle	Fenich	Agender	829-323-5779	2020-10-31
30	Ancline	Landiner	Polygender	598-751-2283	2021-05-19
31	Enlyn	Clobutaru	Male	801-774-4694	2021-08-13
32	Wally	Nadga	Agender	740-168-8291	2021-02-19
33	Cayla	Spata	Agender	148-609-6202	2021-08-03
34	Dacey	Lovel	Polygender	164-975-2195	2021-03-28
35	Liane	Lonansey	Bigender	132-737-9648	2020-09-02
36	Janessy	Abbots	Female	573-635-9892	2021-06-07
37	Cy	Wagnerin	Female	988-387-2252	2021-03-16
38	Pet	Bloxham	Genderqueer	103-828-3158	2021-07-23
39	Charlton	Janous	Genderqueer	248-974-1770	2021-02-28
40	Monti	Fenlon	Agender	923-167-7875	2021-02-24
41	Othelia	Deaton	Genderfluid	268-123-4960	2020-10-08
42	Rodrigo	Dunbrail	Bigender	201-571-6159	2021-05-21
43	Dillane	Lampard	Bigender	544-231-9712	2021-01-07
44	Ansaie	Escalante	Polygender	200-109-6404	2021-02-05
45	Myrvyn	Presale	non-binary	777-652-2605	2021-03-16
46	Bonie	Prebble	Agender	554-970-2127	2021-07-25
47	Girny	Branney	Polygender	114-048-7360	2021-06-13
48	Ulberio	Helmes	Female	277-658-0021	2021-02-20
49	Sander	Keay	non-binary	107-482-6661	2021-03-10
50	Shaylynn	Chapelhow	Polygender	349-221-2231	2021-07-29
51	Jimmy	Kleinhaut	Male	123-393-7442	2020-09-10
52	Cullen	Wittering	Polygender	705-230-3984	2021-08-04
53	Hirsch	Hulanco	Polygender	565-506-8634	2021-03-26
54	Bartholomaeus	Klinnrok	non-binary	340-746-2611	2021-01-18
55	Del	Louth	Bigender	762-779-1521	2021-03-13
56	Leigha	Dansay	Bigender	971-228-2875	2021-05-11
57	Mona	Paolacci	Genderfluid	209-693-8554	2021-02-24
58	Delaney	MacCague	Agender	700-988-8161	2021-07-18
59	Pren	Posen	Polygender	451-905-2126	2021-04-29
60	Keroly	Kumy	non-binary	908-705-3984	2021-06-25

Как мы видим id начинается с 11 позиции и выведены только 50 строк.

Итак,

LIMIT

Команда **LIMIT** [целое_число] ограничивает количество выводимых строк до значения числа.

Пример :

```
SELECT * FROM travelers LIMIT 25;
```

OFFSET

Команда **OFFSET** [целое_число] пропускает количество записей в таблице с начала, равное значению числа.

Пример :

```
SELECT surname FROM travelers OFFSET 15;
```

Существует и другая команда для ограничения вывода строк.

FETCH FIRST [] ROW ONLY

Команда **FETCH FIRST** [целое_число] **ROW ONLY** показывает только количество первых строк, равное значению числа

Пример :

```
SELECT name, email FROM travelers FETCH FIRST 20 ROW ONLY;
```

Выполним запрос с этой командой к таблице student_auto.

```
SELECT * FROM student_auto OFFSET 100 FETCH FIRST 10 ROW ONLY;
```

```
SQL Shell (psql)
starter=# SELECT * FROM student_auto OFFSET 100 FETCH FIRST 10 ROW ONLY;
 id | name      | surname | gender | mobile | birthday
-----+-----+-----+-----+-----+-----
101 | Anjanette | Worster | Genderfluid | 460-296-7177 | 2021-08-04
102 | Cammy    | Cugin  | Female | 372-739-5886 | 2020-12-26
103 | Felike   | Catanheira | Genderqueer | 478-812-6605 | 2020-10-27
104 | Rosy     | Challen | Bigender | 877-212-6652 | 2020-10-24
105 | Gabriel  | Sokell  | Genderqueer | 482-147-0643 | 2021-01-17
106 | Lita     | MacLise | Non-binary | 869-737-3336 | 2020-10-07
107 | Ab       | Treadwell | Genderqueer | 183-613-1424 | 2020-11-21
108 | Livvyy   | Troup   | Male | 520-592-5512 | 2020-11-21
109 | Almeta   | Haistwell | Agender | 334-967-9255 | 2021-06-28
110 | Giulietta | Agett   | Agender | 782-465-0262 | 2021-01-20
(10 строк)

starter=#
```

Как мы видим, полученный результат идентичен результату команды LIMIT.

Не однократно мы видели результаты команды SELECT со «*» в виде копии таблицы, которая хранится на сервере базы данных. Но что, если мы хотим вывести не все столбцы, а некоторые из них. В таком случае «*» нужно заменить на уточняющее условие. Давайте попробуем вывести столбцы только имени и фамилии. Для этого замени «*» на name, surname, разделенных запятой.

```
SELECT name, surname FROM student_auto;
```

```
SQL Shell (psql)
starter=# SELECT name, surname FROM student_auto;
 name | surname
-----+-----
Correy | Humberstone
Rina  | Leaves
Leyla | Culpen
Karna | Hosby
Berkie | Yule
Avrit | Dahl
Andrei | Cockney
Gates | Dobbie
Ulrich | Bridge
Georgiana | Behnke
Lena  | Cokly
Chicky | Schonfelder
Rodina | Denmes
Jacqueline | Micah
Othelia | Kernley
Hedwiga | Leile
Miguelita | Dawid
Ratha  | Tidgewell
Balduin | Cordall
Lia    | Kermott
Aurilia | Densunbe
Delcine | Pendered
Lewiss  | Vanyshev
Collen  | Strang
Xymenes | Olifard
Orelia  | Baumert
Victoir  | Saile
Mort    | Nemchinov
Rosabella | Fenich
```

В результате, мы видим только те столбцы, которые указали.

Сделаем еще один запрос по полю birthday

```
SELECT birthday FROM student_auto;
```

```
SQL Shell (psql)
starter=# SELECT birthday FROM student_auto;
 birthday
-----
2021-05-06
2020-12-18
2020-12-01
2021-02-07
2021-05-30
2021-08-07
2021-06-10
2021-03-19
2021-01-15
2021-02-26
2021-07-12
2021-08-07
2021-08-19
2021-05-08
2021-03-28
2020-12-15
2021-06-04
2020-10-16
2021-05-22
2021-04-12
2021-05-21
2021-04-20
2021-05-18
2021-07-14
2021-03-17
2021-08-08
2020-09-07
2021-03-28
2020-10-31
2021-05-19
2021-08-13
```

Чтобы задать другой порядок вывода данных используется команда
ORDER BY [] ASC/DESC

Команда **ORDER BY [имя_столбца]** показывает выборку, отсортированную по выбранному столбцу.

С помощью приписки **ASC** или **DESC** можно расположить выборку в возрастающем или убывающем порядке соответственно.

Пример :

```
SELECT name, surname FROM travelers ORDER BY name ASC;
```

Выполним запрос по полю birthday с упорядочением в убывающем порядке:

```
SELECT * FROM student_auto ORDER BY birthday DESC;
```

SQL Shell (psql)

```
starter=# SELECT * FROM student_auto ORDER BY birthday DESC;
```

id	name	surname	gender	mobile	birthday
418	Blondie	Barltrop	Genderfluid	369-255-9129	2021-08-29
418	Blondie	Barltrop	Genderfluid	369-255-9129	2021-08-29
418	Blondie	Barltrop	Genderfluid	369-255-9129	2021-08-29
399	Seka	Harly	Male	853-826-0288	2021-08-28
399	Seka	Harly	Male	853-826-0288	2021-08-28
480	Elihu	Stent	Female	458-777-2977	2021-08-28
480	Elihu	Stent	Female	458-777-2977	2021-08-28
399	Seka	Harly	Male	853-826-0288	2021-08-28
480	Elihu	Stent	Female	458-777-2977	2021-08-28
296	Morse	Albery	Female	621-496-6036	2021-08-27
296	Morse	Albery	Female	621-496-6036	2021-08-27
74	Llywellyn	Fynor	Non-binary	687-135-4156	2021-08-27
296	Morse	Albery	Female	621-496-6036	2021-08-27
74	Llywellyn	Fynor	Non-binary	687-135-4156	2021-08-27
74	Llywellyn	Fynor	Non-binary	687-135-4156	2021-08-27
313	Andras	Cutill	Genderqueer	304-653-9778	2021-08-25
313	Andras	Cutill	Genderqueer	304-653-9778	2021-08-25
313	Andras	Cutill	Genderqueer	304-653-9778	2021-08-25
412	Shea	Lapworth	Male	914-307-1809	2021-08-24
177	Chad	Raat	Genderfluid	146-531-5558	2021-08-24
177	Chad	Raat	Genderfluid	146-531-5558	2021-08-24
177	Chad	Raat	Genderfluid	146-531-5558	2021-08-24
412	Shea	Lapworth	Male	914-307-1809	2021-08-24
412	Shea	Lapworth	Male	914-307-1809	2021-08-24
240	Rochella	Sly	Bigender	569-865-7651	2021-08-22
474	Thorpe	Zannotti	Bigender	746-724-1834	2021-08-22
474	Thorpe	Zannotti	Bigender	746-724-1834	2021-08-22
240	Rochella	Sly	Bigender	569-865-7651	2021-08-22
240	Rochella	Sly	Bigender	569-865-7651	2021-08-22
474	Thorpe	Zannotti	Bigender	746-724-1834	2021-08-22
350	Buffy	Willis	Bigender	957-734-3893	2021-08-28

Выполним еще один запрос по полю name, но расположим данные по возрастаную.

```
SELECT name FROM student_auto ORDER BY birthday ASC;
```

SQL Shell (psql)

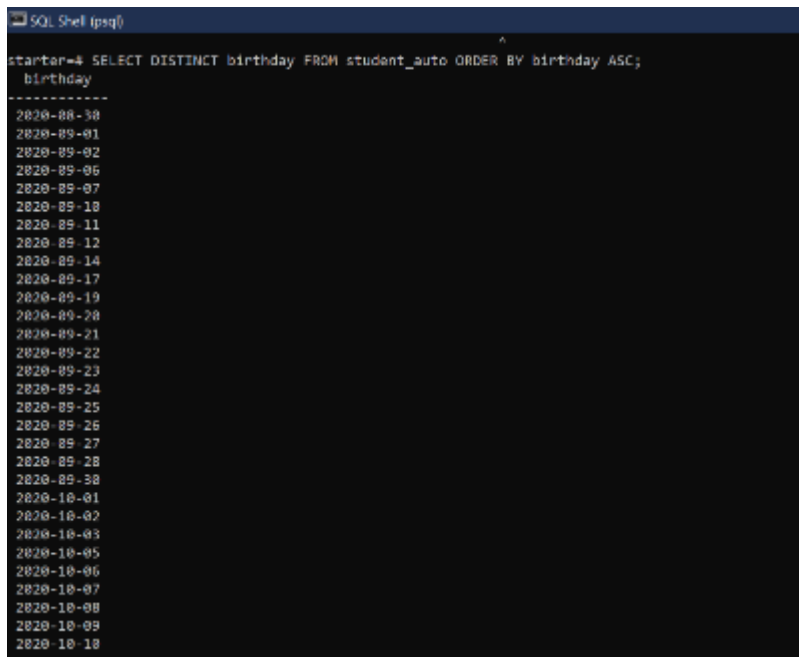
```
starter=# SELECT name FROM student_auto ORDER BY birthday ASC;
```

name
Babette
Giraldo
Babette
Giraldo
Babette
Giraldo
Gypsy
Gypsy
Gypsy
Liane
Liane
Bethany
Bethany
Liane
Bethany
Aluino
Trever
Trever
Trever
Aluino
Aluino
Lynnell
Victoir
Lynnell
Loutitia
Victoir
Victoir
Loutitia
Lynnell
Loutitia
Frazer

В результате мы получим список фамилий в алфавитном порядке. Но в этом списке, видим дубликаты. Выполним запрос, который уберет дубликаты с помощью команды `DISTINCT`.

В следующем запросе введем команду DISTINCT, которая уберет дублирующие значения.

```
SELECT DISTINCT birthday FROM student_auto ORDER BY birthday ASC;
```



```
SQL Shell (psql)
starter=# SELECT DISTINCT birthday FROM student_auto ORDER BY birthday ASC;
 birthday
-----
 2020-08-30
 2020-09-01
 2020-09-02
 2020-09-06
 2020-09-07
 2020-09-10
 2020-09-11
 2020-09-12
 2020-09-14
 2020-09-17
 2020-09-19
 2020-09-20
 2020-09-21
 2020-09-22
 2020-09-23
 2020-09-24
 2020-09-25
 2020-09-26
 2020-09-27
 2020-09-28
 2020-09-30
 2020-10-01
 2020-10-02
 2020-10-03
 2020-10-05
 2020-10-06
 2020-10-07
 2020-10-08
 2020-10-09
 2020-10-10
```

Можно уточнить запрос и например, вывести, данные только о девушках. Для этого используется команда

WHERE

Команда **WHERE** **логическое_выражение** сравнивает значения в строках таблицы с выбранной логикой.

В зависимости от верности или не верности логического выражения, команда включит строку в выборку.

Пример запроса:

```
SELECT * FROM student_auto WHERE gender = 'Female';
```

SQL Shell ipq6

```

starter=# SELECT * FROM student_auto WHERE gender = 'Female';

```

id	name	surname	gender	mobile	birthday
9	Ulrich	Bridge	Female	975-481-8123	2021-01-15
14	Jacqueline	Wich	Female	765-924-7283	2021-05-03
17	Miguelita	David	Female	588-295-1834	2021-06-04
22	Delcine	Pendered	Female	988-699-8718	2021-04-28
23	Lewis	Yanyushev	Female	356-590-1839	2021-05-10
28	Mort	Nemchinov	Female	822-216-7261	2021-03-28
36	Janey	Abbott	Female	573-635-9082	2021-06-07
37	Cy	Wagnerin	Female	988-387-2252	2021-03-16
48	Ulberto	Melres	Female	277-658-8011	2021-02-28
62	Laincy	Swaffield	Female	555-558-4213	2020-10-28
70	Alisander	Burdin	Female	518-741-9227	2021-07-14
81	Frod	Pinn	Female	752-803-0513	2021-03-24
87	Luise	Colashill	Female	588-191-4154	2020-11-28
93	Katrinka	Casay	Female	584-649-8258	2021-04-22
98	Beryle	Padrick	Female	528-348-0838	2020-09-14
102	Carry	Cogin	Female	372-739-5886	2020-12-26
140	Grace	Sedgwick	Female	356-888-0361	2021-03-02
153	Cosmo	Coulshaw	Female	334-581-1646	2021-01-23
156	Hunley	Inksler	Female	979-954-7883	2020-09-20
158	Viloria	Heel	Female	633-657-1558	2021-04-03
160	Chico	Seviour	Female	873-489-9343	2021-02-08
169	Dill	Jendrys	Female	922-351-8984	2021-01-15
173	Bendicty	Mizan	Female	484-849-6153	2021-04-01
179	Janus	Mozillius	Female	182-141-9569	2021-01-31
183	Justin	Rycroft	Female	218-252-5928	2021-04-24
188	Kinny	Lo Gallo	Female	478-596-8113	2020-09-28
193	Salaich	English	Female	385-951-9427	2021-06-28

Как видим из последнего рисунка, мы получили данные только о девушках.

Для составления логических выражений используются, следующие знаки:

=	Равно
<>	Не равно
<	Меньше
<=	Меньше или равно
>	Больше
>=	Больше или равно
BETWEEN	Между двумя значениями
IS NULL	Отсутствует запись

Логические утверждения WHERE могут быть записаны в виде составных условий выбора данных. Например, запрос

позволяет сузить диапазон запроса.

Далее, создадим еще одну таблицу методом генерации, как это было сделано на предыдущих уроках или используйте готовую таблицу из файла, прикрепленного к работе `travelers.sql`. Проведем импорт таблицы из данного файла в нашу БД, используя знакомую нам уже команду `\i`

Вывод INSERT 0 1 говорит об успешном импорте данных.

Давайте проверим успешности импорта таблицы с помощью выборки всех данных таблицы `travelers.sql`

id	name	surname	email	gender	country
1	Cornelius	Dutch	cdutch@cornelia.org	Female	Russia
2	Yvette	Gruby	ygruby1@tina.com	Male	Russia
3	Konstantine	Yakovl	kyakovl2@tripadvisor.com	Female	Indonesia
4	Darrel	Siddleley	dsiddleley@businessweek.com	Female	China
5	Olva	Leopard	oleopard4@skyrock.com	Male	China
6	Dalli	Chapelhow	dchapelhow@godaddy.com	Female	China
7	Kerwin	Lannas	klannas6@istats.com	Female	Thailand
8	Marie-Jeanne	MacDougall	rmacdougall7@istockphoto.com	Female	Germany
9	Hyacintha	McCritchie	hmccritchie8@fc2.com	Female	Morocco
10	Jedidiah	Cayser	jtcayser9@wordpress.com	Female	China
11	Clary	Roa	croa@godaddy.com	Male	China
12	Ladanna	Confort	lconfortb@olala.or.jp	Male	Uganda
13	Ranica	Kulvar	rkulvarc@weebly.com	Male	Brazil
14	Hallory	Warhan	rwarhand@ebay.co.uk	Female	Vietnam
15	Goddin	Gratrac	ggratrac@networksolutions.com	Male	China
16	Flory	Mayhead	fmayheadf@gnm.org	Female	Panama
17	Hunter	Annluell	hannluellg@lower-blog.com	Female	Russia
18	Brannon	Cottisford	bcottisfordh@yellowpages.com	Female	China
19	Gale	Di Nisco	gdiniscoi@unich.edu	Male	United Kingdom
20	Salomo	Dyka	sdyka.jk@java.com	Male	Finland
21	Yasmeen	Tanser	ytanser@shareasale.com	Male	Finland
22	Rodie	Garnity	rgarnityl@powerworld.com	Male	Greece
23	Maggie	Wiggott	mwiggott@ix.com	Male	Peru
24	Jody	Rallin	jrallin@coxlog.jp	Female	Brazil
25	Joey	Madson	jmadson@bellsouth.net	Male	China

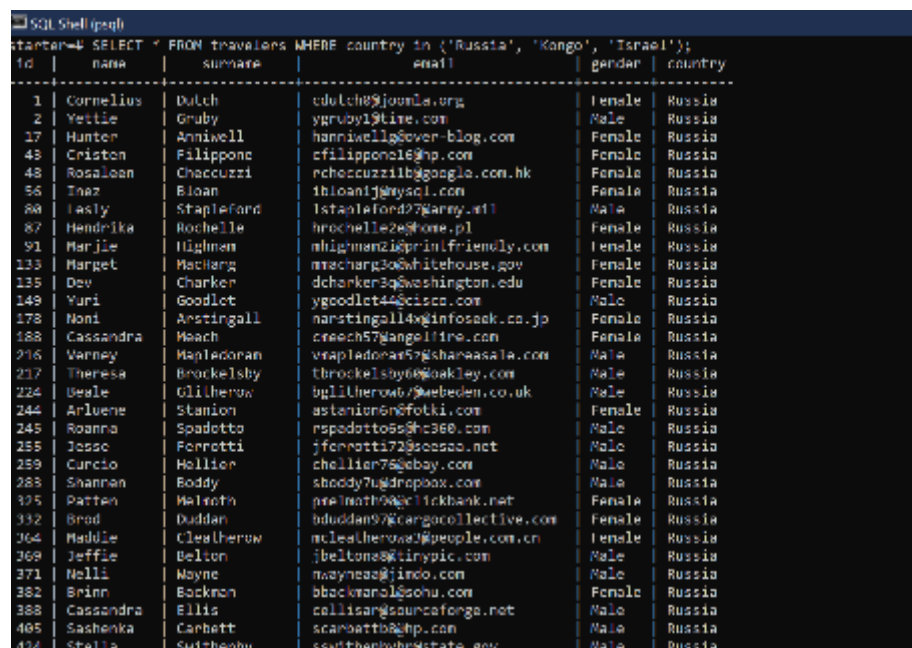
Как результат, мы видим точную копию таблицы, которая хранится на сервере.

Допустим нам нужна информация из таблицы travelers по путешественникам из разных стран. Для этого после слова WHERE после слова in мы в скобках через запятую добавляем список стран, которые хотим вывести в запросе. Ключевое слово in сравнивает значения в скобках со значениями с поле country.

Пример запроса

```
SELECT * FROM travelers WHERE country in ('Russia', 'Kongo', 'Israel');
```

Консоль выведет значения с теми данными, которые были перечислены в скобках.



id	name	surname	email	gender	country
1	Cornelius	Dutch	cdutch89@icloud.org	Female	Russia
2	Vettie	Gruby	ygruby19@live.com	Male	Russia
17	Hunter	Anniwell	hanniwellg@over-blog.com	Female	Russia
43	Cristen	Filippone	cfilippone16@hp.com	Female	Russia
48	Rosalean	Chaccuzzi	rchaccuzzi18@google.com.hk	Female	Russia
56	Inez	Bloan	ibloan1@mysq1.com	Female	Russia
80	Lesly	Stapleford	lstapleford27@arry.atl	Male	Russia
87	Hendrika	Rochelle	hrochelle26@stone.pl	Female	Russia
91	Marjile	Higmen	mhigmen216@printfriendly.com	Female	Russia
133	Margot	MacLarg	mrecharg306@whitehouse.gov	Female	Russia
135	Dev	Charker	dcharker39@washington.edu	Female	Russia
149	Yuri	Goodlet	ygoodlet44@cisco.com	Male	Russia
178	Moni	Arstingall	marstingall4w@infosack.co.jp	Female	Russia
188	Cassandra	Meach	cmeach57@gangalline.com	Female	Russia
216	Verney	Mapledoran	vmapledoran57@sharaasala.com	Male	Russia
217	Theresa	Brockelsby	tbrockelsby66@oakley.com	Male	Russia
224	Deale	Gillherow	bgillherow67@webden.co.uk	Male	Russia
244	Anluene	Stanion	astanion6@fotki.com	Female	Russia
245	Roanna	Spadotto	rspadotto66@ic360.com	Male	Russia
255	Jesse	Ferretti	jferretti712@coocaso.net	Male	Russia
259	Curcio	Hellier	chellier76@ebay.com	Male	Russia
283	Shannon	Boddy	sboddy7@dropbox.com	Male	Russia
305	Patten	Walroth	pwalroth90@kickbank.net	Female	Russia
332	Brad	Duddan	bduddan97@caracollective.com	Female	Russia
364	Maddie	Cleatherow	mcleatherow67@people.com.cn	Female	Russia
369	Jeffie	Belton	jbelton88@tinypic.com	Male	Russia
371	Nelli	Wayne	nwayne8@jindo.com	Male	Russia
382	Brinn	Backman	bbackman1@sohu.com	Female	Russia
388	Cassandra	Ellis	cellisar@saucsaforge.net	Male	Russia
405	Sashenka	Carbett	scarbett88@hp.com	Male	Russia
424	Stella	Sutthorpe	ssutthorpe@hrstate.gov	Male	Russia

Попробуем сделать запрос по буквам страны в ее названии.

Для этого воспользуемся ключевым словом BETWEEN.

```
SELECT * FROM travelers WHERE country BETWEEN 'A' and 'D';
```

SQL Shell (psql)

```
starter=# SELECT * FROM travelers WHERE country BETWEEN 'A' and 'D';
```

id	name	surname	email	gender	country
4	Dannel	Siddaley	dsiddaley@businessweek.com	Female	China
5	Olivia	Leopard	oleopard@skyrock.com	Male	China
6	Delli	Chapelhow	dchapelhow@godaddy.com	Female	China
10	Jedidiah	Cayser	jcayser9@wordpress.com	Female	China
11	Clary	Rea	creece@godaddy.com	Male	China
13	Renique	Kulver	rkulverc@weebly.com	Male	Brazil
15	Godwin	Gratrex	ggratrex@networksolutions.com	Male	China
18	Brannon	Cottisford	bcottisford@yahoo.com	Female	China
24	Jody	Ballin	jballin@gexhlog.jp	Female	Brazil
25	Berti	Najana	bnajana@uix.com	Male	China
32	Trudie	Westwater	twestwater@rambler.ru	Male	China
42	Margaretha	Beuckham	mbeuckham1@comcast.net	Female	Colombia
45	Marc	Sant	msant118@devhub.com	Male	China
52	Leonard	Shefton	lshefton17@1688.com	Female	China
58	Enigrit	Swatten	bswatten11@unicef.org	Male	Brazil
60	Hedris	Chippis	hchippis@nydailynews.com	Male	China
62	Court	Klarbt	cklarbt@pki.org	Female	China
66	Shawdn	Yvaktn	siyaktn17@linklist.com	Male	Canada
69	Quannie	Brodest	qbrodest1@comcast.net	Female	Czech Republic
71	Horatius	Fargher	hfargher17@vina.com	Male	Colombia
78	Ioni	Exall	texall125@instagram.com	Male	China
79	Sheffy	Redclyffe	sredclyffe26@usa.gov	Male	China
81	Heather	Colbourne	hcolbourne28@shad.gov	Male	China
89	Ardelle	McGurgan	amcgurgan2@marriott.com	Female	China
90	Ellie	Broske	ebroske2@argun.com	Female	Colombia
95	Sarae	Torrett	storrett2@google.co.jp	Female	China
96	Lorianne	Ciccitti	lciccitti2@zinearticles.com	Female	China
98	Gtraud	Kiddle	gkiddle2@tribia.com	Female	China
101	Sarey	Devonish	sdevonish2@nbcnews.com	Male	Brazil
105	Florie	Puckingham	fpuckingham2@blog.com	Female	Brazil
111	Clovie	Bilan	cbilan32@google.ru	Male	China

В результате в консоль выведутся значения, которые начинаются с A, B, C, D.

Можно отсортировать даже по последовательности символов. Например, нам нужны путешественники у которых почта заканчивается только на .edu.

Используя команду LIKE мы оставим только те строки, в которых почта имеет символы внутри одинарных кавычек. Знак % - это специальный символ. Он заменяет весь текст, который следует до искомой нами комбинации.

SELECT * FROM travelers WHERE email LIKE '%.edu';

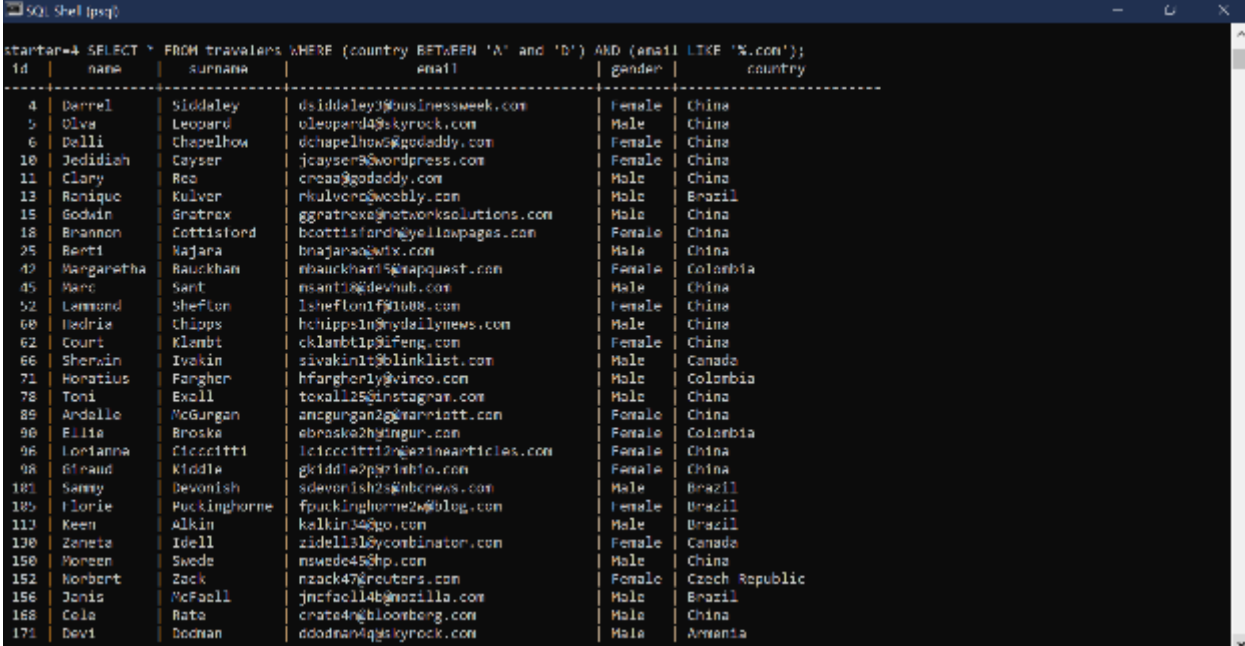
SQL Shell (psql)

```
starter=# SELECT * FROM travelers WHERE email LIKE '%.edu';
```

id	name	surname	email	gender	country
19	Gale	Di Nisco	gdinisco@unich.edu	Male	United Kingdom
26	Sebastien	Vines	svines@columbia.edu	Male	Philippines
39	Sargent	Lipman	slipman12@mit.edu	Female	Uruguay
46	Rasia	Arnold	rarnold19@msu.edu	Male	Egypt
116	Spenser	Pollicott	spollicott37@illinois.edu	Male	Lithuania
135	Dev	Charker	dcharker3@washington.edu	Female	Russia
151	Diandra	Aleixo	daleixo4@cornell.edu	Female	Palau
158	Welsh	Brackharst	wbrackharst4@stanu.edu	Male	Azerbaijan
172	Jana	Cuttriss	jcuttriss4@berkeley.edu	Female	Bulgaria
191	Rosana	Ratnega	rratnega5@princeton.edu	Male	Czech Republic
214	Neda	Litchmore	nlitchmore5@stanu.edu	Female	China
215	Charlita	Shadfourth	cshadfourth5@upenn.edu	Female	Vietnam
233	Ferdinande	Stockhill	fstockhill6@washington.edu	Female	Thailand
241	Larry	Cliburn	lcliburn6@si.edu	Female	Hungary
252	Britte	Barents	bbarents6@washington.edu	Male	China
295	Sigfrid	Banbridge	sbanbridge8@ucla.edu	Male	China
315	Iorgos	Boschmann	iboschmann8@washington.edu	Male	Democratic Republic of the Congo
330	Kathi	Baptie	kbaptie9@unc.edu	Female	Nicaragua
345	Jexinah	Chanson	jchanson9@utexas.edu	Male	Brazil
367	Joby	Inglesant	jinglesant@harvard.edu	Male	Malaysia
374	Fern	Firnisad	ffirnisad@ucsd.edu	Female	Greece
378	Harriett	Zecchetti	hzecchetti@washington.edu	Female	Canada
410	Cortney	Daudray	cdaudray@cmu.edu	Female	Croatia
452	Philippe	Madgewick	pmadgewick@ucsd.edu	Female	Philippines
462	Hurbert	Parell	hparell@mit.edu	Female	Canada
463	Govan	Halladey	ghalladey@utexas.edu	Male	Vietnam

Можно сделать комбинированные запросы. Объединим запрос по определенным странам и с выполненным, только что запросов с почтой, только выберем теперь строки с почтой .com

SELECT * FROM travelers WHERE (country BETWEEN 'A' and 'D') AND (email LIKE '%.com');



id	name	surname	email	gender	country
4	Darrel	Siddaley	dsiddaley1@businessweek.com	Female	China
5	Olva	Leopard	oleopard49@skyrock.com	Male	China
6	Delli	Chapelhow	dchapelhow6@godaddy.com	Female	China
10	Jedidiah	Cayser	jcayser9@wordpress.com	Female	China
11	Clary	Ree	creea9@godaddy.com	Male	China
13	Ronique	Kulver	rkulver@webly.com	Male	Brazil
15	Godwin	Grotrex	ggrotrex@networksolutions.com	Male	China
18	Brannon	Cottisford	bcottisford@yellowpages.com	Female	China
25	Berti	Najana	bnajana@vix.com	Male	China
42	Margaretha	Bauckhan	mbauckhan15@mapquest.com	Female	Colombia
45	Marc	Sant	msant18@devhub.com	Male	China
52	Lamond	Sheflon	lsheflon19@1608.com	Female	China
60	Nedrie	Chlops	hchlops1@nydailynews.com	Male	China
62	Court	Klanbi	cklanbt1@163.com	Female	China
66	Sherwin	Ivakin	sivakin1@olinklist.com	Male	Canada
71	Horatius	Fargher	hfargher1@vimeo.com	Male	Colombia
78	Toni	Exall	texall125@instagram.com	Male	China
89	Ardalle	McGurgan	amcgurgan2@varexpress.com	Female	China
90	Ellie	Broska	ebroska2@yimgur.com	Female	Colombia
96	Lorianna	Ciccitt1	lciccitt12@artinearticles.com	Female	China
98	Gleaud	Kiodla	gkiodla2@pinterest.com	Female	China
101	Sammy	Devonish	sdevonish2@nbcnews.com	Male	Brazil
105	Florie	Puckingham	fpuckingham2@blog.com	Female	Brazil
113	Keen	Alkin	kalkin14@go.com	Male	Brazil
130	Zeneta	Idell	zidell13@combinator.com	Female	Canada
150	Woreen	Swede	nswede45@hp.com	Male	China
152	Norbert	Zack	nzack47@reuters.com	Female	Czech Republic
156	Jonis	McFaell	jmcfaell48@mozilla.com	Male	Brazil
168	Cola	Rata	crata4@bloomberg.com	Male	China
171	Devi	Bochman	dbochman49@skyrock.com	Male	Armenia

Давайте теперь попробуем посчитать на сколько уникальны имена наших путешественников.

Для этого используем команду COUNT.

Команда **COUNT(*)** сосчитает количество одинаковых значений в столбце выборки до неё.

Пример :

```
SELECT count(*) FROM travelers;
```

```
SQL Shell (psql)
starter=# SELECT count(*) FROM travelers;
count
-----
1000
(1 строка)

starter=#
```

Как мы видим, функция COUNT (*) возвращает количество строк, возвращаемых оператором SELECT, т.е. количество строк таблицы.

Выполним другой запрос с группировкой по полю name

SELECT name, COUNT(*) FROM travelers GROUP BY name;

```
SQL Shell (psql)
starter=# SELECT name, COUNT(*) FROM travelers GROUP BY name;
name | count
-----+-----
Carly | 2
Ewan | 1
Kev | 1
Bebe | 1
Romain | 1
Peggie | 1
Eten | 1
Brnaba | 1
Launce | 2
Connado | 1
Say | 1
Aurea | 1
Suzanna | 1
Tuck | 1
Anstice | 1
Karla | 1
Hirsch | 1
Sylvester | 1
Washington | 1
Wayne | 1
Annaliese | 1
Barbabas | 1
Dalt | 1
Clary | 1
Panchu | 1
Keene | 1
Sheryl | 1
```

Давайте выполним запрос и узнаем откуда путешественников больше всего. И воспользуемся командой HAVING

Команда **HAVING** очень похожа на команду **WHERE**. Её основное отличие в том, что она применяется сразу к строкам, сгруппированным командой **GROUP BY()**.

Пример :

```
SELECT name, surname FROM travelers GROUP BY count(name) > 1;
```

Выполним следующий запрос:

```
SELECT country, COUNT(*) FROM travelers GROUP BY country
HAVING COUNT(*)>25;
```

```
SQL Shell (psql)
starter=# SELECT country, COUNT(*) FROM travelers GROUP BY country HAVING COUNT(*)>25;
 country | count
-----+-----
Indonesia | 104
Portugal  | 31
France    | 48
Philippines | 63
China     | 164
Russia    | 65
Brazil    | 38
Poland    | 30
(8 строк)
```

Как мы видим, больше всего путешественников из Индонезии и Китая.

Но для больших таблиц, такой просмотр очень неудобный для поиска наибольших значений. Давайте отсортируем по убыванию, используя запрос:

```
SELECT country, COUNT(*) FROM travelers GROUP BY country
HAVING COUNT(*)>25 ORDER BY COUNT(*) DESC;
```

```
starter=# SELECT country, COUNT(*) FROM travelers GROUP BY country HAVING COUNT(*)>25 ORDER BY COUNT(*) DESC;
 country | count
-----+-----
China    | 164
Indonesia | 104
Russia   | 65
Philippines | 63
France   | 48
Brazil   | 38
Portugal | 31
Poland   | 30
(8 строк)

starter=#
```

Теперь давайте поучимся переименовывать столбцы таблицы.

```
starter=# \d travelers
          Таблица "public.travelers"
-----+-----+-----+-----+-----
Столбец | Тип      | Правило сортировки | Допустимость NULL | По умолчанию
-----+-----+-----+-----+-----
id       | integer  |                     |                     |
name     | character varying(50) |                     |                     |
surname  | character varying(50) |                     |                     |
email    | character varying(50) |                     |                     |
gender   | character varying(50) |                     |                     |
country  | character varying(50) |                     |                     |
(8 строк)

starter=#
```

Допустим, мы хотим переименовать столбец id на n, surname на s. Для этого используется команда AS

```
SELECT id As n, surname As s FROM travelers;
```

Лабораторная работа № 4. Базовая арифметика. Время и дата

Цель работы: освоить базовую арифметику при работе с данными

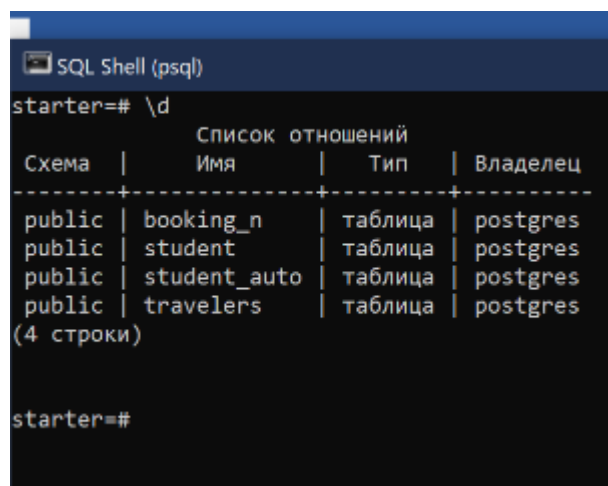
Для этой работе нужно создать новую таблицу бронирования путешествия. Или воспользоваться готовой таблицей, которая находится в разделе занятия.

Зайдите в консоль SQL Shell (psql) и загрузите таблицу в ранее созданную базу данных starter.

После импорта таблицы сделаем запрос и посмотрим содержание таблицы (рисунок 1)

```
starter=# SELECT * FROM booking_n;
```

Выполним команду \d и убедимся, что таблица импортирована.



```
SQL Shell (psql)
starter=# \d
          Список отношений
Схема |      Имя      | Тип   | Владелец
-----+-----+-----+-----
public | booking_n     | таблица | postgres
public | student       | таблица | postgres
public | student_auto  | таблица | postgres
public | travelers     | таблица | postgres
(4 строки)

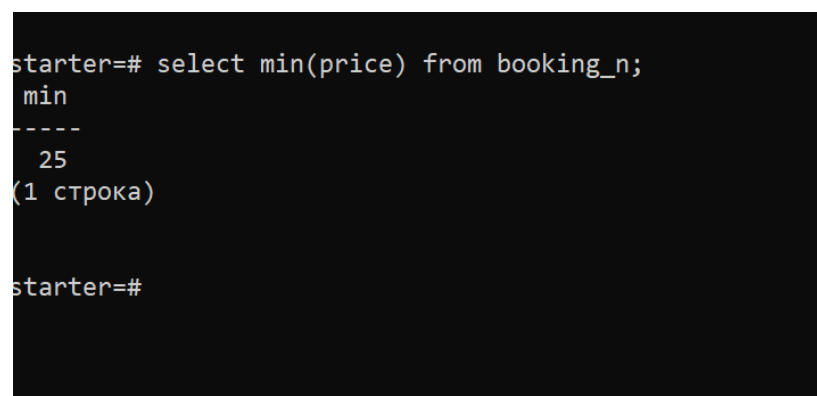
starter=#
```

Рисунок 2

Давайте узнаем самую высокую цену на бронирование. Для этого реализуем следующий запрос:

```
starter=# select max(price) from booking_n;
```

```
starter=# select min(price) from booking_n;
```



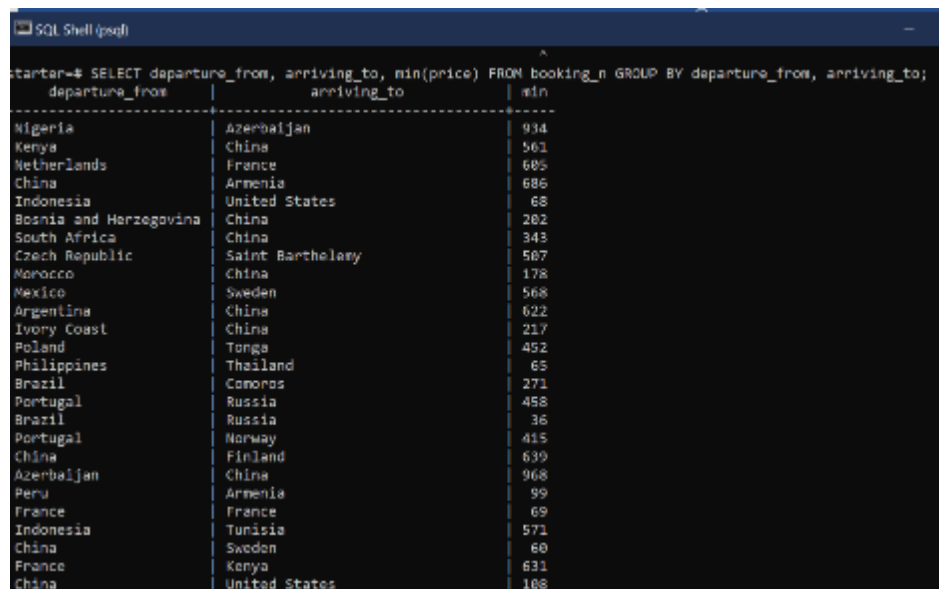
```
starter=# select min(price) from booking_n;
 min
-----
  25
(1 строка)

starter=#
```

Рисунок 4

Теперь найдем самое дешевое бронирование для всех типов, которые есть в таблице

```
SELECT departure_from, arriving_to, min(price) FROM booking_n GROUP BY departure_from, arriving_to;
```



departure_from	arriving_to	min
Nigeria	Azerbaijan	934
Kenya	China	561
Netherlands	France	605
China	Armenia	686
Indonesia	United States	68
Bosnia and Herzegovina	China	282
South Africa	China	343
Czech Republic	Saint Barthelemy	507
Morocco	China	178
Mexico	Sweden	568
Argentina	China	622
Ivory Coast	China	217
Poland	Tonga	452
Philippines	Thailand	65
Brazil	Comoros	271
Portugal	Russia	458
Brazil	Russia	36
Portugal	Norway	415
China	Finland	639
Azerbaijan	China	968
Peru	Armenia	99
France	France	69
Indonesia	Tunisia	571
China	Sweden	60
France	Kenya	631
China	United States	108

Рисунок 5

Сделаем следующий запрос, в котором узнаем сколько в среднем туристы тратят на бронирование.

```
starter=# SELECT AVG(price) FROM booking_n;
```

Узнать стоимость бронирования перед перелетом можно сложить все значения с помощью функции SUM:

```
starter=# SELECT SUM(price) FROM booking_n;
```

```
starter=# SELECT departure_from, max(price) FROM booking_n GROUP BY departure_from ORDER BY max(price) DESC;
```

Итак, мы с вами познакомились со следующими функциями

MIN()

Команда MIN(имя_столбца) покажет минимальное значение среди всех значений внутри столбца.

Пример:

:

```
SELECT min(price) FROM booking_n WHERE name LIKE '%sia';
```

MAX()

Команда **MAX(имя_столбца)** покажет минимальное значение среди всех значений внутри столбца.

Пример:

```
SELECT max(price) FROM booking_n WHERE surname BETWEEN 'S' AND 'X';
```

AVG()

Команда **AVG(имя_столбца)** покажет среднее арифметическое значение среди всех значений внутри столбца.

Пример:

```
SELECT avg(price) from booking_n;
```

SUM()

Команда **SUM(имя_столбца)** покажет сумму всех значений внутри столбца.

```
SELECT sum(price) FROM booking_n;
```

На этом этапе изучим работу с датой и временем. Если нам надо точно узнать о текущей дате вызов функции `now()` выдаст информацию эту информацию (рисунок 1)

```
SQL Shell (psql)
starter=# select now();
          now
-----
2022-11-01 12:54:42.362683+03
(1 строка)

starter=#
```

Рисунок 1

Если необходимо узнать только дату без уточнения времени, то добавляют к функции now() параметр DATE и тогда запрос имеет вид:

```
starter=# select now()::DATE;
```

```
SQL Shell (psql)
          ^
starter=# select now()::DATE;
          now
-----
2022-11-01
(1 строка)

starter=#
```

Рисунок 2

Узнаем текущую дату, которая будет ровно через год. Запрос имеет следующий вид

```
starter=# select now()- INTERVAL '1 year';
```

```
starter=# select now()- INTERVAL '1 year';
           ?column?
-----
2021-11-01 18:35:07.507705+03
(1 строка)
```

Рисунок 3

Узнаем дату 10 лет назад.

```
starter=# select now()- INTERVAL '10 year';
```

```
starter=# select now()- INTERVAL '10 year';  
          ?column?  
-----  
2012-11-01 18:46:36.699395+04  
(1 строка)
```

Рисунок 4

Если год большой промежуток времени, то можно использовать месяцы, дни, часы. Например, делая следующий запрос, мы узнаем дату ровно 25 дней назад

```
select now()- interval '25 days';
```

```
SQL Shell (psql)  
starter=# select now()- interval '25 days';  
          ?column?  
-----  
2022-10-13 18:45:04.021873+03  
(1 строка)
```

Рисунок 5

Численные значения могут быть любыми

```
select now()- interval '125 months';
```

```
starter=# select now()- interval '125 months';  
          ?column?  
-----  
2012-06-07 18:47:37.099797+04  
(1 строка)
```

Рисунок 6

Иногда необходимо вытащить из даты, что-то одно с помощью функции extract. Например, текущий год:

```
select extract(year from now());
```

```

starter=# select extract(year from now());
extract
-----
      2022
(1 строка)

```

Рисунок 7

Или текущий месяц:

```
select extract(months from now());
```

```

starter=# select extract(months from now());
extract
-----
      11
(1 строка)

```

Рисунок 8

Давайте из таблицы student базы данных starter сделаем запрос и посчитаем сколько прожито дней студентом с момента рождения до текущего числа

```
select name, surname, age(now(), birthday) as age from student_auto;
```

```

starter=# select name, surname, age(now(), birthday) as age from student_auto;
 name | surname | age
-----|-----|-----
Correy | Nunberstone | 1 year 6 mons 1 day 19:05:12.206245
Rina | Leaves | 1 year 10 mons 20 days 19:05:12.206245
Leyla | Culpen | 1 year 11 mons 6 days 19:05:12.206245
Karna | Hosby | 1 year 9 mons 19:05:12.206245
Berkie | Yule | 1 year 5 mons 8 days 19:05:12.206245
Avrit | Dahl | 1 year 3 mons 19:05:12.206245
Andrei | Cockney | 1 year 4 mons 27 days 19:05:12.206245
Gates | Debbie | 1 year 7 mons 19 days 19:05:12.206245
Ulrich | Bridge | 1 year 9 mons 23 days 19:05:12.206245
Georgiana | Behnke | 1 year 8 mons 9 days 19:05:12.206245
Erna | Cokly | 1 year 3 mons 26 days 19:05:12.206245
Chicky | Schonfelder | 1 year 3 mons 19:05:12.206245
Rodina | Dennes | 1 year 2 mons 19 days 19:05:12.206245
Jacqueline | Micah | 1 year 6 mons 4 days 19:05:12.206245
Othelia | Kernley | 1 year 7 mons 10 days 19:05:12.206245
Hedwiga | Leile | 1 year 10 mons 23 days 19:05:12.206245
Miguelita | Dawid | 1 year 5 mons 3 days 19:05:12.206245
Retha | Tidgewell | 2 years 22 days 19:05:12.206245
Balduin | Cordall | 1 year 5 mons 16 days 19:05:12.206245
Lia | Kermott | 1 year 6 mons 25 days 19:05:12.206245
Aurilia | Densunbe | 1 year 5 mons 17 days 19:05:12.206245
Dalcine | Pendered | 1 year 6 mons 17 days 19:05:12.206245
Leafse | Vancubau | 1 year 5 mons 20 days 19:05:12.206245

```

Рисунок 9

Итак, для работы с датой и временем служат следующие команды:

1. Команда **NOW()** выведет текущую дату вплоть до миллисекунд.

Пример :

```
SELECT * FROM NOW();
```

2. Приписка **::DATE** к команде **NOW** сузит ответ только до нужного типа времени.

Вместо **DATE** можно использовать любое значение из таблицы типов.

Пример :

```
SELECT FROM NOW()::DATE;
```

3. Команда **INTERVAL**, прибавленная или отнятая от **NOW()**, покажет точную дату ровно через выбранный отрезок времени в будущем или прошлом.

Не забывайте множественные числа!

Пример :

```
SELECT FROM NOW() + INTERVAL('12 MINUTES');
```

4. Команда **EXTRACT** вытащит нужное значение из ответа функции.

Пример :

```
SELECT EXTRACT (SECOND FROM NOW());
```

И используются следующие типы времени

YEAR	Год
MONTH	Месяц

DAY	День
HOUR	Час
MINUTE	Минута
SECOND	Секунда
MILLISECOND	Миллисекунда

Лабораторная работа № 5. Первичные ключи. Ограничения.

Цель работы: освоить понятие первичного ключа на практике и научиться работать с ограничениями

В каждой таблице базы данных может существовать первичный ключ. Под первичным ключом понимают поле и набор полей однозначно идентифицирующих запись. Это означает, что первичный ключ может быть только один.

Добавим таблицу booking_n в базу данных starter

```
starter=# \d
          Список отношений
  Схема |      Имя      | Тип   | Владелец
-----+-----+-----+-----
 public | booking_n     | таблица | postgres
 public | student       | таблица | postgres
 public | student_auto  | таблица | postgres
 public | travelers     | таблица | postgres
(4 строки)
```

Рисунок 1

```
starter=# \d booking_n
          таблица "public.booking_n"
  Столбец |      Тип      | Правило сортировки | Допустимость NULL | По умолчанию
-----+-----+-----+-----+-----
 id       | integer       |                     | not null          |
 name     | character varying(50) |                     | not null          |
 surname  | character varying(50) |                     | not null          |
 departure_from | character varying(50) |                     | not null          |
 arriving_to   | character varying(50) |                     | not null          |
 price      | integer       |                     | not null          |
```

Рисунок 2

Если вы видите значение not null, то это значит, что поле должно быть заполненным, т.е. не допустимо отсутствие значений.

Первичным ключом является поле id.

Для работы с первичным ключом существуют следующие команды:

ALTER TABLE

Команда ALTER TABLE [имя_таблицы]

ADD/DROP [имя_столбца] позволит внести изменения в таблицу.

ADD и DROP добавляют и удаляют столбец соответственно.

Примеры :

ALTER TABLE travelers ADD login VARCHAR(25);

ALTER TABLE travelers DROP login;

DROP CONSTRAINT

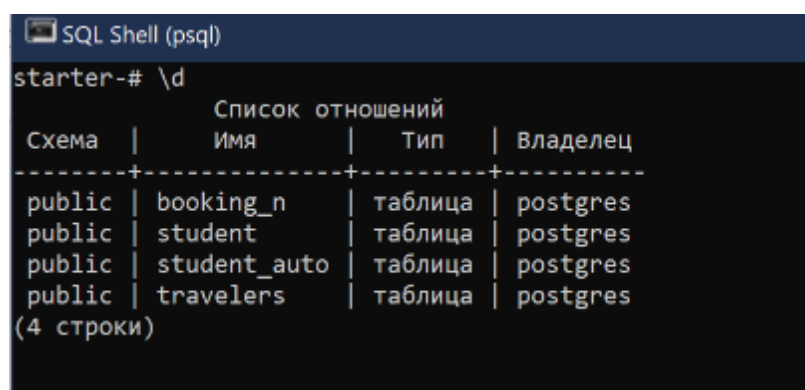
Команда ALTER TABLE имя_таблицы DROP

CONSTRAINT (имя_ограничителя) поможет избавиться от ограничений, наложенных на таблицу.

Пример:

ALTER TABLE booking_n DROP CONSTRAINT booking_n_pkey;

Вернемся к таблице путешественников travelers



SQL Shell (psql)

```
starter-# \d
```

Схема	Имя	Тип	Владелец
public	booking_n	таблица	postgres
public	student	таблица	postgres
public	student_auto	таблица	postgres
public	travelers	таблица	postgres

(4 строки)

Рисунок 1

Сделаем запрос, в котором посчитаем частоту появления разных имен.

select name, count(*) from travelers group by name;

```
starter=# select name, count(*) from travelers group by name;
```

name	count
Carly	2
Exan	1
Kev	1
Bebe	1
Romain	1
Peggie	1
Etan	1
Brnaba	1
Launce	2
Conrado	1
Say	1
Aurea	1
Suzanna	1
Tuck	1
Anstice	1
Karla	1
Hirsch	1
Sylvester	1
Washington	1
Wayne	1
Annaliese	1
Barbabas	1
Dalt	1
Clary	1
Pancho	1
Keene	1
Sheryl	1

Рисунок 2

Как видим из запроса, повторяются имена не часто.

Пусть нас интересуют имена, которые повторяются, хотя бы более одного раза.

`select name, count(*) from travelers group by name having count(*)>1;`

```
starter=# select name, count(*) from travelers group by name having count(*)>1;
```

name	count
Carly	2
Launce	2
Pattan	3
Shelly	2
Darya	3
Ellynn	2
Merry	2
Salomo	2
Spenser	2
Rafaelita	2
Corrie	2
Suzann	2
Anata	2
Heriberto	2
Theresa	2
Marc	2
Fina	2
Nile	2
Christabel	2
Marthe	2
Cesare	2
Gino	2
Delcine	2
Cassandra	2
Kylie	2
Berti	2
Kamoko	2
Ferdinanda	2
Charlot	2
Hugh	2
Velvet	2

Рисунок 3

Создадим ограничения.

Откройте таблицу `travelers` в Notepad++. И скопируем, например, первую строчку (рисунок 4)

`insert into travelers (id, name, surname, email, gender, country) values (1, 'Cornelius', 'Dutch', 'cdutch0@joomla.org', 'Female', 'Russia');`

и вставим эту строку в консоль postgresql

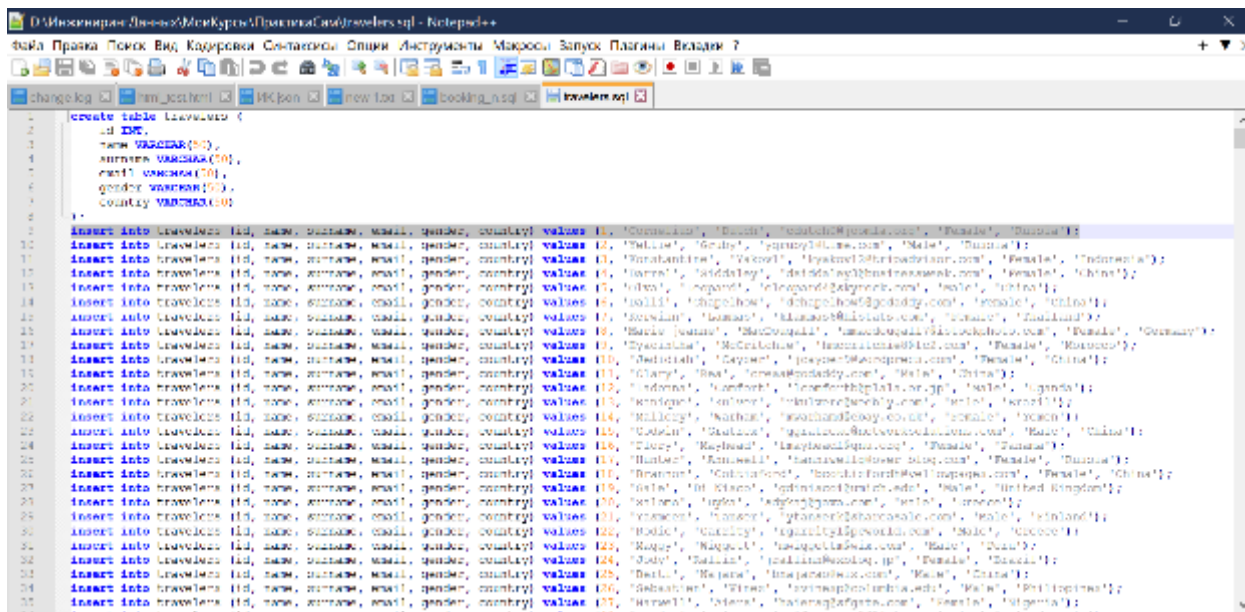


Рисунок 4

Как результат, успешная вставка в таблицу (рисунок 5)

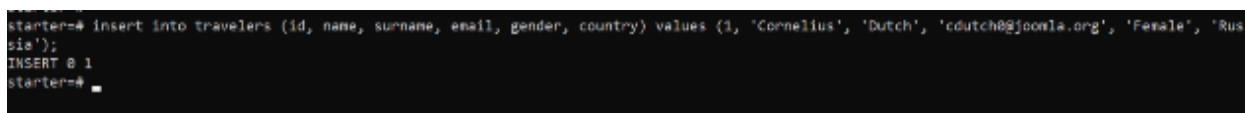


Рисунок 5

Сделаем запрос на вывод всех записей с данным id. Увидим 2 записи.
select * from travelers where id=1;



Рисунок 6

Создадим первичный ключ, который ограничит дублирование записи
alter table travelers add constraint unique_id UNIQUE(id);

```

starter=# alter table travelers add constraint unique_id UNIQUE(id);
ОШИБКА: создать уникальный индекс "unique_id" не удалось
ПОДРОБНОСТИ: Ключ (id)=(1) дублируется.
starter=#

```

Рисунок 7

Но получаем ошибку, так как, если в таблице, существуют дубликаты, то это правило не будет применено.

Данную ситуацию можно исправить, например, удалив, добавленную выше запись.

```
delete from travelers where name='Cornelius';
```

```
starter=# delete from travelers where name='Cornelius';
DELETE 2
starter=#
```

Рисунок 8

Теперь повторим команду

```
alter table travelers add constraint unique_id UNIQUE(id);
```

И ошибки не появились.

```
starter=# alter table travelers add constraint unique_id UNIQUE(id);
ALTER TABLE
starter=#
```

Рисунок 9

Посмотрим структуру таблицы travelers:

```
starter=# \d travelers
```

		Таблица "public.travelers"		
Столбец	Тип	Правила сортировки	Допустимость NULL	По умолчанию
id	integer			
name	character varying(50)			
surname	character varying(50)			
email	character varying(50)			
gender	character varying(50)			
country	character varying(50)			

```
Индексы:
"unique_id" UNIQUE CONSTRAINT, btree (id)
```

Рисунок 10

Как видим на рисунке 10, внизу появилась надпись о ключевом поле и теперь не сможем внести не уникальные значения.

Повторим запрос с внесением дублирующей записи (рисунок 11)

```
starter=# insert into travelers (id, name, surname, email, gender, country) values (1, 'Cornelius', 'Butch', 'cdutch@jocnls.org', 'Female', 'Russia');
ОШИБКА: повторяющееся значение ключа нарушает ограничение уникальности "unique_id"
ПОДРОБНОСТИ: Ключ "{id}=1" уже существует.
starter=#
```

Рисунок 11

Как видим, появляется сообщение об ошибке о внесении повторяющихся значений.

Ограничения можно накладывать не только на уникальные значения в столбцах, но и на сам формат написания этих значений в принципе.

```
select * from travelers;
```

```
starter=# select * from travelers;
```

	id	name	surname	email	gender	country
2	Yvette	Gruby	ygruby1@time.com		Male	Russia
3	Konstantine	Yakovl	kyakov1@tripadvisor.com		Female	Indonesia
4	Darrel	Siddalay	dsiddalay1@businessweek.com		Female	China
5	Olva	Leopard	oleopard4@skyrock.com		Male	China
6	Dalli	Chapelhow	dchapelhow5@godaddy.com		Female	China
7	Kerwinn	Lamms	klamms@histats.com		Female	Thailand
8	Marie-Jeanne	MacDougall	mmacdougall7@istockphoto.com		Female	Germany
9	Hyacinthe	McCritchie	hmcritchie@fc2.com		Female	Morocco
10	Jedidiah	Cayser	jcayser9@wordpress.com		Female	China
11	Clary	Rea	creea@godaddy.com		Male	China
12	Ladonna	Confort	lconfort@plala.or.jp		Male	Uganda
13	Sanique	Kulver	rkulver@weebly.com		Male	Brazil
14	Mallory	Warham	mwarham@ebay.co.uk		Female	Yemen
15	Godwin	Stotrex	gstotrex@networksolutions.com		Male	China
16	Flory	Nayhead	fnayhead@gnu.org		Female	Panama
17	Hunter	Anniwell	hanniwell@gover-blog.com		Female	Russia
18	Brannon	Cottisford	bcottisford@yellowpages.com		Female	China
19	Gale	Di Nisco	gdinisco@urich.edu		Male	United Kingdom
20	Salosa	Dyke	sdylke@java.com		Male	Greece
21	Yasmeen	Tanser	ytanser@shareasale.com		Male	Finland

Рисунок 12

Давайте с имитируем ошибку введение данных в таблицу. Например, в поле Пол, в какой-либо записи, вместо Female напишем F

```

4      gender VARCHAR(50),
7      country VARCHAR(50)
8  );
9  insert into travelers (id, name, surname, email, gender, country) values (1, 'Cornelius', 'Butch', 'cbutch@joomla.org', 'F', 'Bosnia');
10 insert into travelers (id, name, surname, email, gender, country) values (2, 'Yvette', 'Gruby', 'ygruby1@time.com', 'Male', 'Russia');
```

Данное изменение без помех внесется в таблицу, поскольку нет правил ограничивающий синтаксис для столбца gender.

Добавим такую команду, но чтобы получилось наверняка, выполним команду

```
select distinct gender from travelers;
```

```
starter=# select distinct gender from travelers;
```

```

SQL Shell (psql)
starter=# select distinct gender from travelers;
gender
-----
Female
F
Male
(3 строки)
```

Рисунок 14

И увидим все вариации написания Пола на текущий момент.

Напишем следующий запрос

```
alter table travelers add constraint gender_constraint check(gender='Female'
or gender='Male');
```

```
sql> alter table travelers add constraint gender_constraint check(gender='Female' or gender='Male');
ОШИБКА: ограничение-проверку "gender_constraint" отношения "travelers" нарушает некоторая строка
sql>
```

Снова ошибка. Потому что, таблица содержит нарушителей этого правила. Поэтому такие данные нам необходимо вначале удалить.

```
delete from travelers where gender='F';
```

Теперь снова применим запрос

```
alter table travelers add constraint gender_constraint check(gender='Female'
or gender='Male');
```

```
sql> alter table travelers add constraint gender_constraint check(gender='Female' or gender='Male');
sql> alter table travelers add constraint gender_constraint check(gender='Female' or gender='Male');
sql>
```

Таким образом, рассмотрены команды

Команда **ALTER TABLE имя_таблицы ADD CONSTRAINT имя_ограничения UNIQUE(имя_столбца)** создаст ограничение, согласно которому в таблицу смогут быть добавлены только записи с уникальными значениями внутри указанного столбца.

Пример:

```
ALTER TABLE travelers ADD CONSTRAINT id_unique UNIQUE(id);
```

Команда **ALTER TABLE имя_таблицы ADD CONSTRAINT имя_ограничения CHECK(логическое_выражение)** создаст ограничение, согласно которому внутри столбцов смогут быть только определенные значения.

Пример:

```
ALTER TABLE travelers ADD CONSTRAINT gender_check
CHECK(gender = 'Female' OR gender = 'Male');
```

Лабораторная работа № 6. UPSERT. Внешние ключи

Цель работы: освоить команду UPSERT и работу с внешними ключами

В этой работе мы будем использовать таблицу booking_n. Давайте вспомним как она выглядит с помощью выборки всей таблицы

```
select * from booking_n;
```

Пусть нам известно, что пассажир под номером 5, был занесен в таблицу с ошибкой с не правильным именем. При подробном осмотре записи, видим, что указано имя Rurik

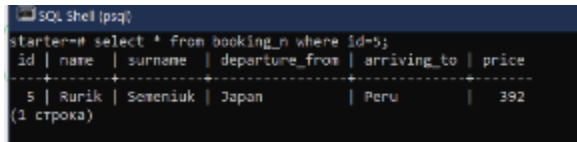


4	Tanny	Jobbins	Indonesia		
5	Rurik	Semeniuk	Japan		

Необходимо изменить это имя на настоящее.

Сделать мы это можем при помощи команды UPSERT. Во время выполнения этой команды мы можем уточнить где конкретно происходит изменение с помощью той же логики, что мы использовали для вывода всей строки под номером 5.

```
select * from booking_n where id=5;
```

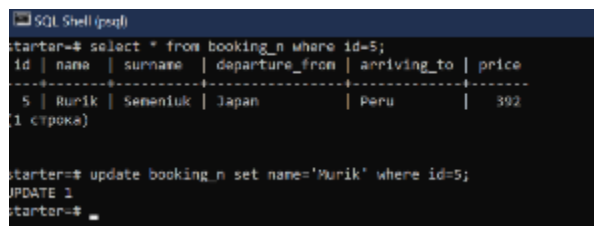


```
SQL Shell (psql)
starter=# select * from booking_n where id=5;
 id | name | surname | departure_from | arriving_to | price
----+-----+-----+-----+-----+-----
  5 | Rurik | Semeniuk | Japan          | Peru        | 392
(1 строка)
```

Изменяем именно Rurik.

Введем запрос на изменение имени:

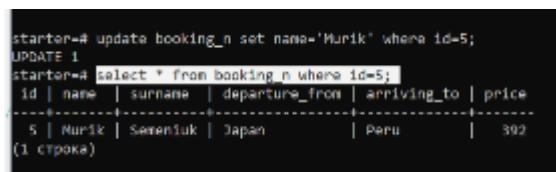
```
update booking_n set name='Murik' where id=5;
```



```
SQL Shell (psql)
starter=# select * from booking_n where id=5;
 id | name | surname | departure_from | arriving_to | price
----+-----+-----+-----+-----+-----
  5 | Rurik | Semeniuk | Japan          | Peru        | 392
(1 строка)

starter=# update booking_n set name='Murik' where id=5;
UPDATE 1
starter=#
```

и проверяем изменения повторением предыдущей команды.



```
SQL Shell (psql)
starter=# update booking_n set name='Murik' where id=5;
UPDATE 1
starter=# select * from booking_n where id=5;
 id | name | surname | departure_from | arriving_to | price
----+-----+-----+-----+-----+-----
  5 | Murik | Semeniuk | Japan          | Peru        | 392
(1 строка)
```

Подобным образом можно изменить сразу несколько полей в таблице.

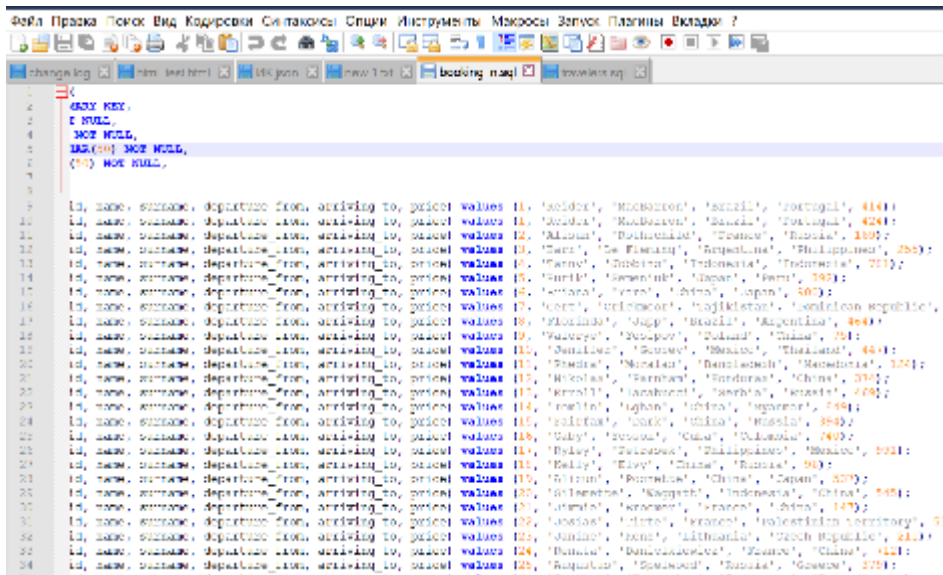
Допустим были перепутаны места вылета и прибытия

Update booking_n set departure_from='Peru', arriving_to='Japan' where id=5;

```
starter=# Update booking_n set departure_from='Peru', arriving_to='Japan' where id=5;
UPDATE 1
starter=# select * from booking_n where id=5;
 id | name | surname | departure_from | arriving_to | price 
-----+-----+-----+-----+-----+-----
  5 | Murik | Semenjuk | Peru           | Japan       | 392
(1 строка)
```

Изменять существующие поля в базе данных можно даже на уровне импорта готовой таблицы с помощью скрипта.

Давайте перейдем в наш скрипт для создания таблицы booking_n.



```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

И скопируем запись знакомого нам уже пассажира под id=5.

Однако при попытке вставить этой строки мы получим сообщение о попытке вставки записи с существующим ID.

```
starter=# insert into booking_n (id, name, surname, departure_from, arriving_to, price) values (5, 'Rurik', 'Semeniuk', 'Japan', 'Peru', 392);
ОШИБКА: повторяющееся значение ключа нарушает ограничение уникальности "booking_n_pkey"
ПОДРОБНОСТИ: Ключ "(id)=(5)" уже существует.
starter=#
```

Как известно, это связано с правилом настройки уникального ключа нашей таблицы. Это поведение можно скорректировать, дописав для консоли ее действия при возникновении конфликта с текущими правилами таблицы. Это осуществляется командой **on conflict**

```
insert into booking_n (id, name, surname, departure_from, arriving_to, price)
values (5, 'Rurik', 'Semeniuk', 'Japan', 'Peru', 392) on conflict (id) do nothing;
```

В нашем случае при конфликтующих таблицах мы не делаем ничего - **do nothing**;

Выполняем команду и консоль в данный момент не выдала ошибку.

```
starter=# insert into booking_n (id, name, surname, departure_from, arriving_to, price) values (5, 'Rurik', 'Semeniuk',  
'Japan', 'Peru', 392) on conflict (id) do nothing;  
INSERT 0 0  
starter=#
```

Команду **on conflict** нельзя применить к столбцам не связанным ни к каким правилами. На данный момент в таблице booking_n правило применяется только к столбцу id, так как он является столбцом уникального ключа.

```
starter=# \d booking_n;
```

Столбец	Тип	Правило сортировки	Допустимость NULL	По умолчанию
id	integer		not null	
name	character varying(50)		not null	
surname	character varying(50)		not null	
departure_from	character varying(50)		not null	
arriving_to	character varying(50)		not null	
price	integer		not null	

Индексы:
"booking_n_pkey" PRIMARY KEY, btree (id)

Если мы вернемся в текстовый редактор, то мы быстро создадим пассажира с id=0.

В качестве примера привяжем три адреса таблицы к случайным людям из таблицы travelers.

```
(3 строки)  
starter=# UPDATE travelers SET address_id = 1 WHERE id = 2;  
UPDATE 1  
starter=# SELECT * FROM travelers LIMIT (3);
```

id	name	surname	email	gender	country	address_id
3	Konstantine	Yakovl	kyakovl2@tripadvisor.com	Female	Indonesia	
4	Darrel	Siddaley	dsiddaley3@businessweek.com	Female	China	
5	Olva	Leopard	oleopard4@skyrock.com	Male	China	

```
(3 строки)
```

Столбцы address_id и id в таблице travelers не связаны ни какими правилами. Поэтому они могут быть не одинаковыми.

Но для столбца таблицы address уже создано правило, по которому они обязаны быть одинаковыми.

Лабораторная работа № 7. Соединение таблиц. Экспорт в CSV

Цель работы: освоить технологию соединения таблиц и проводить экспорт данных

При соединении таблиц мы всегда ориентируемся на те столбцы, которые были объявлены внешними ключами.

В данном случае `address_id` в таблице путешественников `travelers` является внешним ключом для таблицы `address`.

Соединить таблицы можно с помощью команды `JOIN ON/`

Ко всей выборке таблицы `travelers` мы присоединяем информацию из таблицы `address` только в том месте где у нас есть совпадение по внешним ключам. Информация будет присоединена только в тех строках, где `address_id` равен `id` из таблицы `address`.

То есть на данный момент, это должно произойти ровно для одной записи.

```
SQL Shell (psql)
starter=# SELECT * FROM address;
 id | street | house
-----+-----+-----
  1 | Groove Street | Central House
  2 | Davis Avenue | Left House
  3 | Capital Boulevard | Right House
(3 строки)

starter=# SELECT * FROM travelers JOIN address ON travelers.address_id = address.id;
 id | name | surname | email | gender | country | address_id | id | street | house
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
  2 | Yettie | Gruby | ygruby1@time.com | Male | Russia | 1 | 1 | Groove Street | Central House
(1 строка)

starter=#
```

Соединенная таблица, автоматически достроила таблицу адресов справа, так как она является присоединяемой.

```
starter=# \d address
          Таблица "public.address"
+-----+-----+-----+-----+-----+
| Столбец | Тип | Правило сортировки | Допустимость NULL | По умолчанию |
+-----+-----+-----+-----+-----+
| id | bigint | | not null | nextval('address_id_seq'::regclass) |
| street | character varying(50) | | | |
| house | character varying(50) | | | |
+-----+-----+-----+-----+-----+
Индексы:
"address_pkey" PRIMARY KEY, btree (id)
Ссылки:
TABLE "travelers" CONSTRAINT "travelers_address_id_fkey" FOREIGN KEY (address_id) REFERENCES address(id)

starter=# SELECT travelers.name, travelers.surname, address.street, address.house FROM travelers
starter=#
```

Присоединять таблицу можно не только полностью, но и по частям.

Давайте возьмем столбцы только с именем и фамилией с таблицы путешественников travelers и столбцы адреса и дома с таблицы address.

Это присоединение работает по той же самой логике, что и предыдущее соединение и показывает, только нам необходимую информацию.

```
starter=# SELECT travelers.name, travelers.surname, address.street, address.house FROM travelers
starter=# JOIN address ON travelers.address_id = address.id;
 name | surname | street | house
-----+-----+-----+-----
 Yettie | Gruby   | Groove Street | Central House
(1 строка)

starter=# _
```

В ответе видим таблицу, созданную из четырех столбцов.

Существует еще несколько способов соединения таблиц. Left join полностью покажет ту таблицу которую присоединяют и только некоторую информацию из присоединяемой таблицы. Эта таблица находится слева

```
starter=# SELECT * FROM travelers LEFT JOIN address ON address.id = travelers.address_id;
```

id	name	surname	email	gender	country	address_id	id	street	house
3	Konstantine	Yakovl	kyakovl2@tripadvisor.com	Female	Indonesia				
4	Darrel	Siddeley	csiddeley@businessweek.com	Female	China				
5	Olivia	Leopards	oleopards4@skyrock.com	Male	China				
6	Dalli	Chapelhow	dchapelhow@godaddy.com	Female	China				
7	Kervinn	Lamano	klamano6@bittats.com	Female	Thailand				
8	Marie-Jeanne	Kochbuegall	mkoebuegall@stockphoto.com	Female	Germany				
9	Hyacintha	McCritchie	hmcritchie@icf2.com	Female	Morocco				
10	Jedidiah	Cayser	jcayser9@wordpress.com	Female	China				
11	Clary	Kca	crca2@godaddy.com	Male	China				
12	Ladonna	Confort	lconfort@bellsouth.net	Male	Uganda				
13	Harique	Silver	rkilver@bellsouth.com	Male	Brazil				
14	Mallory	Warham	mwarham@btu.ac.uk	Female	Yemen				
15	Gedwin	Gracres	ggracres@networksolutions.com	Male	China				
16	Flury	Redhead	fredhead@bnu.org	Female	Panama				
17	Hunter	Antiswell	hamiswell@over-blog.com	Female	Russia				
18	Brannon	Cottisford	bcottisford@yellowpages.com	Female	China				
19	Gale	Di Hisco	gdhisco@unich.edu	Male	United Kingdom				
20	Salome	Dyke	sdyk@jaya.com	Male	Greece				
21	Vasmeen	Tanser	vtanser@sharcstate.com	Male	Finland				
22	Rodrie	Garrity	rgarrity@pcworld.com	Male	Greece				
23	Maggy	Vipgett	mvipgett@bix.com	Male	Peru				
24	Jode	Kallim	jcallim@bix.com	Female	Brazil				
25	Berti	Majara	bmajara@bix.com	Male	China				
26	Valerian	Wynn	vwynn@unich.edu	Male	United Kingdom				

RIGHT JOIN

Команда [имя_таблицы_1] RIGHT JOIN [имя_таблицы_2] ON [логическое_утверждение] осуществляет формирование таблицы из двух или нескольких таблиц.

В команде RIGHT JOIN, как и в команде LEFT JOIN, важен порядок следования таблиц. Будет полностью добавлена имя_таблицы_1, и лишь некоторые строки из имя_таблицы_2.

```
SELECT * FROM travelers RIGHT JOIN booking_n ON travelers.id =
booking_n.id
```

LEFT JOIN

Команда `[имя_таблицы_1] LEFT JOIN [имя_таблицы_2] ON [логическое_утверждение]` осуществляет формирование таблицы из двух или нескольких таблиц.

В команде **LEFT JOIN**, как и в команде **RIGHT JOIN**, важен порядок следования таблиц. Будет полностью добавлена *имя_таблицы_1*, и лишь некоторые строки из *имя_таблицы_2*.

```
SELECT * FROM travelers LEFT JOIN booking_n ON travelers.id = booking_n.id
```

FULL JOIN

Команда `[имя_таблицы_1] FULL JOIN [имя_таблицы_2] ON [логическое_утверждение]` осуществляет формирование таблицы из двух или нескольких таблиц. В получившуюся выборку войдут все строки без исключения.

Пример :

```
SELECT * FROM travelers FULL JOIN booking_n ON travelers.id = booking_n.id
```

Когда мы производим выборку из таблицы базы данных Postgres в консоли, результат тоже отображается в этом окне. Чтобы сохранить его в каком-либо другом файле, нам нужно вручную выделить всю выборку, нажать **Ctrl+C**, а затем **Ctrl+V** в текстовом файле.

Но что если выборка слишком большая, и не умещается в окне консоли? Мы можем сделать экспорт выборки в файл формата **.csv** или **.sql** в таком же виде, в каком она представлена в консоли.

В данном примере сделаем выборку из таблицы путешественников **travelers** и сохраним её в отдельный документ с расширением **.csv**.

1. Сделаем выборку из таблицы **travelers**, например, первые 50 строк, с помощью команды **SELECT * FROM travelers WHERE id < 50**.

Посмотрим на её общий внешний вид и запомним его.

```

starter=# SELECT * FROM travelers WHERE id < 50;

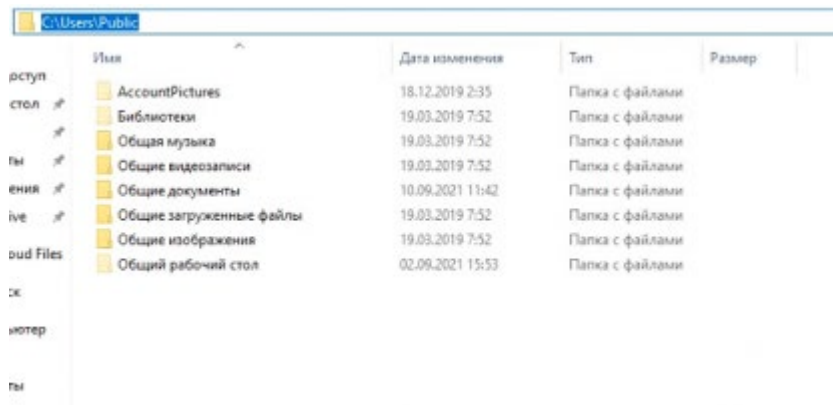
```

id	name	surname	email	gender	country	address_id
1	Cornelius	Dutch	cdutch08@gmaila.org	Female	Russia	
2	Yettie	Gruby	ygruby1@tine.com	Male	Russia	1
3	Konstantine	Yakovl	kyakov12@tripadvisor.com	Female	Indonesia	
4	Darrel	Siddaley	dsiddaley1@businessweek.com	Female	China	
5	Olva	Leopard	oleopard4@skyrock.com	Male	China	
6	Dalli	Chapelhow	dchapelhow5@godaddy.com	Female	China	
7	Kerwinn	Lannas	klannas6@hstats.com	Female	Thailand	
8	Marie-jeanne	MacDougall	mmacdougall7@istockphoto.com	Female	Germany	
9	Hyacintha	McCritchie	hmccritchie8@fc2.com	Female	Morocco	
10	Jedidiah	Cayser	jcayser9@wordpress.com	Female	China	
11	Clary	Rea	creaa@godaddy.com	Male	China	
12	Ladonna	Confort	lcomfortb@piala.or.jp	Male	Uganda	
13	Ranique	Kulver	rkulverc@weebly.com	Male	Brazil	
14	Mallory	Warham	mwarhamd@ebay.co.uk	Female	Yemen	
15	Godwin	Gratrac	ggratracex@networksolutions.com	Male	China	
16	Flory	Mayhead	fmayheadf@gnu.org	Female	Panama	
17	Hunter	Anniwell	hanniwellg@over-blog.com	Female	Russia	
18	Brannon	Cottisford	bcottisfordh@yellowpages.com	Female	China	
19	Gale	Di Nisco	gdiniscoi@umich.edu	Male	United Kingdom	
20	Salomo	Dyka	sdyka@java.com	Male	Greece	
21	Yasmeen	Tanser	ytanserk@shareasale.com	Male	Finland	
22	Rodie	Garrity	rgarrityl@pcworld.com	Male	Greece	
23	Maggy	Wiggett	mwiggett@wix.com	Male	Peru	
24	Jody	Rallin	jrallinn@exblog.jp	Female	Brazil	
25	Berti	Najara	bnajarao@vix.com	Male	China	
26	Sebastien	Vines	svinesp@columbia.edu	Male	Philippines	
27	Harwell	Aiers	haiersq@sfgate.com	Female	Nigeria	
28	Frederigo	Moorey	fmooreyr@flickr.com	Male	Indonesia	
29	Phyllis	Leyborne	pleybornes@foxnews.com	Male	Peru	
30	Sharia	Pantin	spantint@flavors.me	Female	France	
31	Rhonda	Ever	reveru@ycombinator.com	Male	Sierra Leone	
32	Trudie	Westwater	twestwater@rambler.ru	Male	China	
33	Wallis	Featherston	wfeatherstonw@facebook.com	Male	Philippines	
34	Cyb	Hollon	cmollonx@discovery.com	Male	Poland	
35	Sara	Philcox	sphilcox@chc.ca	Male	Portugal	
36	Eolande	Cadwell	ecadwell18@dailynews.com	Male	United States	
37	Darya	Crackett	dcrackett1@netvibes.com	Male	Indonesia	
38	Jessee	Griffen	jgriffenl1@auda.org.au	Male	Philippines	
39	Sargent	Lipman	slipman17@nit.edu	Female	Uruguay	
40	Karin	Milroy	kmilroy13@volasite.com	Female	Indonesia	
41	Bartholomey	William	bwilliam14@merriam-webster.com	Female	Indonesia	
42	Margaretha	Bauckham	mbauckham15@apquest.com	Female	Colombia	
43	Cristen	Fillippone	cfillippone16@hp.com	Female	Russia	
44	Britta	Alfonso	balfonso17@paypal.com	Male	Indonesia	
45	Marc	Sant	msant18@devhub.com	Male	China	
46	Rasia	Arnold	rarnold19@osu.edu	Male	Egypt	
47	Christian	Stonham	cstonham1a@mozilla.com	Female	Philippines	
48	Rosaleen	Checuzzi	rchecuzzi1b@google.com.hk	Female	Russia	
49	Lib	Bilson	lbilson1c@de.vu	Female	Greece	
(49 строк)						

SELECT * FROM travelers WHERE id < 50;

2. Для экспорта нам необходимо знать путь к папке, куда мы будем делать этот экспорт. На разных устройствах для пользователей права доступа к тем или иным папкам не одинаковы, поэтому выберем папку, доступную для всех пользователей. Это папка **Users/Public** на том диске, где установлена ваша ОС Windows.

Скопируем полный путь к папке.



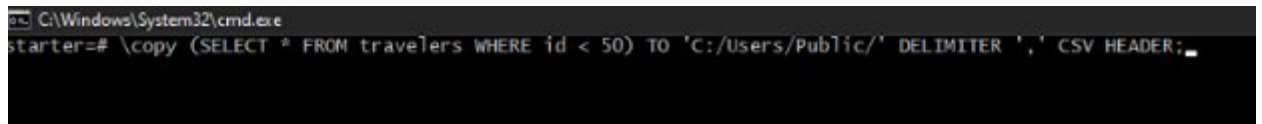
	Имя	Дата изменения	Тип	Размер
доступ	AccountPictures	18.12.2019 2:35	Папка с файлами	
стол	Библиотеки	19.03.2019 7:52	Папка с файлами	
ты	Общая музыка	19.03.2019 7:52	Папка с файлами	
ения	Общие видеозаписи	19.03.2019 7:52	Папка с файлами	
иве	Общие документы	10.09.2021 11:42	Папка с файлами	
oud Files	Общие загруженные файлы	19.03.2019 7:52	Папка с файлами	
ж	Общие изображения	19.03.2019 7:52	Папка с файлами	
нотер	Общий рабочий стол	02.09.2021 15:53	Папка с файлами	
ты				

3. Вспомним, что для импорта скрипта в postgres нам необходимо провести некоторые манипуляции с путем к папке: развернуть слэши и поставить одинарные кавычки с каждой стороны. Для этого лучше воспользоваться текстовым редактором Notepad++, и скопировать эту информацию уже оттуда.



4. Для экспорта мы используем функцию `\copy`, с командой выборки, заключенной в круглые скобки. Команда `TO` указывает на папку, куда будет скопирована выборка. После неё мы вставляем видоизмененный нами полный путь к файлу.

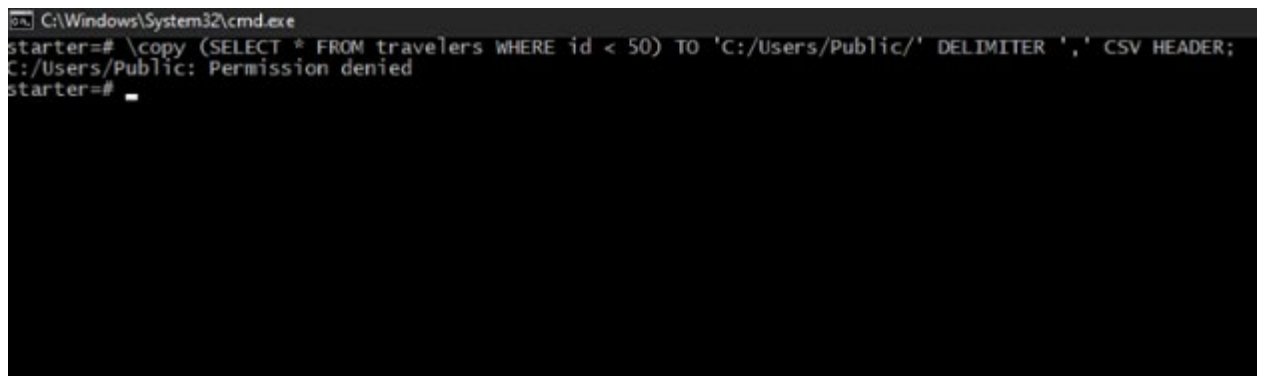
5. Если вспомнить, что у каждого файла есть своё расширение после названия файла, то и следующие две команды объяснить довольно просто: **DELIMITER ','** устанавливает разделитель между данными, которые нужно сохранить для будущего экспорта, а **CSV HEADER** задает файлу расширение `csv`.



```
C:\Windows\System32\cmd.exe
starter=# \copy (SELECT * FROM travelers WHERE id < 50) TO 'C:/Users/Public/' DELIMITER ',' CSV HEADER;
```

```
\copy (SELECT * FROM travelers WHERE id < 50) TO 'C:/Users/Public'
DELIMITER ',' CSV HEADER;
```

6. Если запустим команду в таком виде, то получим ошибку: **Permission Denied**. Она вызвана тем, что указали путь для экспорта, но не указали имя нового файла. Вы можете сделать это в текстовом редакторе, а можете прямо в окне консоли, перемещаясь с помощью стрелок. Назовем файл “sample”.



```
C:\Windows\System32\cmd.exe
starter=# \copy (SELECT * FROM travelers WHERE id < 50) TO 'C:/Users/Public/' DELIMITER ',' CSV HEADER;
C:/Users/Public: Permission denied
starter=#
```

7. Если команда завершилась успешно, консоль выдаст нам сообщение **COPY** с количеством строк в выборке. В нашем случае это 49. Давайте зайдём в файл экспорта. Он находится в той папке, путь к которой мы указали.

```
C:\Windows\System32\cmd.exe
starter-# \copy sample (SELECT * FROM travelers WHERE id < 50) TO 'C:/Users/Public/' DELIMITER ',' CSV HEADER;
C:/Users/Public: Permission denied
starter-# \copy (SELECT * FROM travelers WHERE id < 50) TO 'C:/Users/Public/sample.csv' DELIMITER ',' CSV HEADER;
COPY 49
starter-# _
```

```
\copy (SELECT * FROM travelers WHERE id < 50) TO
'C:/Users/Public/sample.csv' DELIMITER ',' CSV HEADER;
```

8. Внутри файла можем увидеть полную копию нашей выборки.

temp.bd	sample.csv
1	id,name,surname,email,gender,country,address_id
2	1,Cornelius,Dutch,cdutch0@joomla.org,Female,Russia,
3	2,Yettie,Gruby,ygruby1@time.com,Male,Russia,1
4	3,Konstantine,Yakovl,kyakovl2@tripadvisor.com,Female,Indonesia,
5	4,Darrel,Siddaley,dsiddaley3@businessweek.com,Female,China,
6	5,Olva,Leopard,oleopard4@skyrock.com,Male,China,
7	6,Dalli,Chapelhow,dchapelhow5@godaddy.com,Female,China,
8	7,Kerwinn,Lammas,klammas6@histats.com,Female,Thailand,
9	8,Marie-jeanne,MacDougall,mmacdougall7@istockphoto.com,Female,Germany,
10	9,Hyacinttha,McCritchie,hmccritchie8@fc2.com,Female,Morocco,
11	10,Jedidiah,Cayser,jcayser9@wordpress.com,Female,China,
12	11,Clary,Rea,crea@godaddy.com,Male,China,
13	12,Ladonna,Comfort,lcomfortb@plala.or.jp,Male,Uganda,
14	13,Ranique,Kulver,rkulverc@weebly.com,Male,Brazil,
15	14,Mallory,Warham,mwarhamd@ebay.co.uk,Female,Yemen,
16	15,Godwin,Gratrex,ggratrexe@networksolutions.com,Male,China,
17	16,Flory,Mayhead,fmayheadf@gnu.org,Female,Panama,
18	17,Hunter,Anniwell,hanniwellg@over-blog.com,Female,Russia,
19	18,Brannon,Cottisford,bcottisfordh@yellowpages.com,Female,China,
20	19,Gale,Di Nisco,gdiniscoi@umich.edu,Male,United Kingdom,
21	20,Salomo,Dyka,sdykaj@java.com,Male,Greece,
22	21,Yasmeen,Tanser,ytanserk@shareasale.com,Male,Finland,
23	22,Rodie,Garrity,rgarrityl@pcworld.com,Male,Greece,
24	23,Maggy,Wiggett,mwiggettm@wix.com,Male,Peru,
25	24,Jody,Rallin,jrallinn@exblog.jp,Female,Brazil,
26	25,Berti,Najara,bnajarao@wix.com,Male,China,
27	26,Sebastien,Vines,svinesp@columbia.edu,Male,Philippines,
28	27,Harwell,Aiers,haiersq@sfgate.com,Female,Nigeria,
29	28,Fredrigo,Moorey,fmooreyr@flickr.com,Male,Indonesia,
30	29,Phyllis,Leyborne,pleybornes@foxnews.com,Male,Peru,
31	30,Sharia,Pantin,spantint@flavors.me,Female,France,
32	31,Rhonda,Ever,reveru@ycombinator.com,Male,Sierra Leone,
33	32,Trudie,Westwater,twestwaterv@rambler.ru,Male,China,
34	33,Wallis,Featherston,wfeatherstonw@facebook.com,Male,Philippines,
35	34,Cyb,Mollon,cmollonx@discovery.com,Male,Poland,
36	35,Sara,Phillcox,sphillcoxy@cbc.ca,Male,Portugal,
37	36,Eolande,Cadwell,ecadwellz@nydailynews.com,Male,United States,
38	37,Darya,Crackett,dcrackett10@netvibes.com,Male,Indonesia,
39	38,Jessee,Griffen,jgriffen11@auda.org.au,Male,Philippines,
40	39,Sargent,Lipman,slipman12@mit.edu,Female,Uruguay,
41	40,Karin,Milroy,kmilroy13@yolasite.com,Female,Indonesia,
42	41,Bertholemy,Milliam,bhiliaml4@merriam-webster.com,Female,Indonesia,
43	42,Margaretha,Bauckham,mbauckham15@mapquest.com,Female,Colombia,
44	43,Cristen,Filippone,cfilippone16@hp.com,Female,Russia,
45	44,Britta,Alfonso,balfonsol7@paypal.com,Male,Indonesia,
46	45,Marc,Sant,msant18@devhub.com,Male,China,
47	46,Rasia,Arnold,rarnold19@msu.edu,Male,Egypt,
48	47,Christian,Stonham,cstonham1a@mozilla.com,Female,Philippines,
49	48,Rosaleen,Checcuzzi,rcheccuzzi1b@google.com.hk,Female,Russia,
50	49,Lib,Bilson,lbilson1c@de.vu,Female,Greece,
51	

Лабораторная работа № 8-9.

Знакомство с базой данных «Авиаперевозки»

Простые запросы

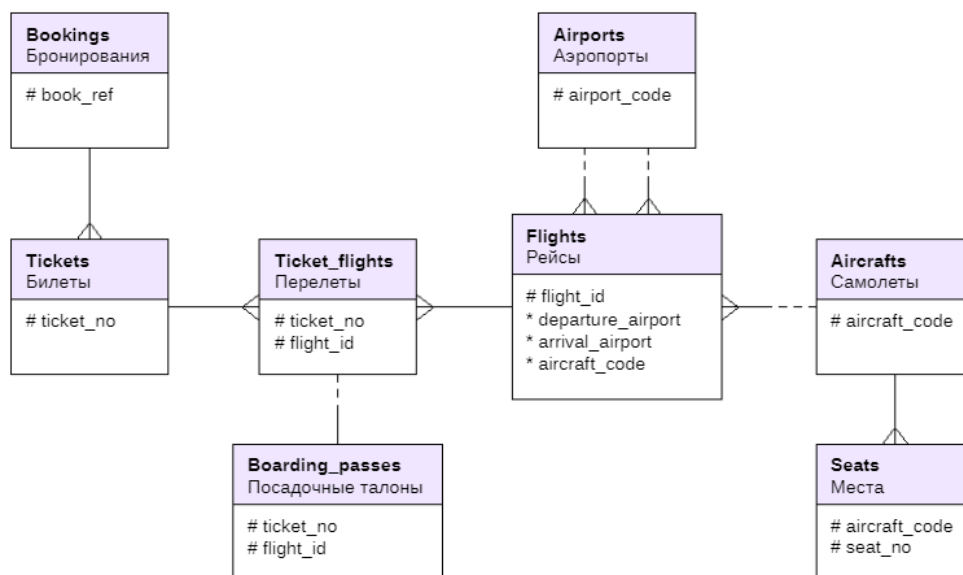
Цель работы: изучить схему данных БД «Авиаперевозки». Познакомится с платформой DBeaver.

Теоретическое обоснование

DBeaver - это кроссплатформенное приложение, поскольку оно поддерживает платформы MacOS, Windows и Linux.

Мультиплатформенный универсальный (работает с большим числом СУБД, как SQL, так и NoSQL) менеджер баз данных DBeaver как раз и является инструментом, который в значительной степени облегчает проектирование, реализацию, модификацию и работу с различными СУБД.

В своей работе мы будем использовать базу данных «Авиаперевозки». Файл базы данных прикреплен на вашей платформе.



Методика выполнения

1. В консоле SQL Shell (psql) загрузите базу данных «demo», используя команду \i 'C:/Users/Елена/Desktop/demo_small.sql', в апострофах указывается путь нахождения файла базы данных.

```

C:\SQL>Shell (psql)
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Пароль пользователя postgres:
psql (14.5)
ПРЕДУПРЕЖДЕНИЕ: Кодовая страница консоли (866) отличается от основной
страницы Windows (1251).
8-битовые (русские) символы могут отображаться некорректно.
Подробнее об этом смотрите документацию psql, раздел
"Notes for Windows users".
Введите "help", чтобы получить справку.

postgres=# \! chcp 1251
Текущая кодовая страница: 1251
postgres=# \i 'C:/Users/Елена/Desktop/demo_small.sql'
SET
SET
SET
SET
SET
SET
SET
DROP DATABASE
CREATE DATABASE
Вы подключены к базе данных "demo" как пользователь "postgres".
SET
SET
SET

```

2. При успешном подключении базы в строке консоли будет выведено название, подключенной базы

```
SQL Shell (psql)
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
ALTER TABLE
REFRESH MATERIALIZED VIEW
demo=# \d
```

Список отношений			
Schema	Имя\тип	Тип	Владелец
bookings	aircrafts	таблица	postgres
bookings	airports	таблица	postgres
bookings	boarding_passes	таблица	postgres
bookings	bookings	таблица	postgres
bookings	flights	таблица	postgres
bookings	flights_flight_id_seq	последовательность	postgres
bookings	flights_v	представления	postgres
bookings	routes	материализованное представление	postgres
bookings	seats	таблица	postgres
bookings	ticket_flights	таблица	postgres
bookings	tickets	таблица	postgres

```
(11 строк)
```

```
demo=#
```

3. Посмотрим, какие сущности и таблицы содержатся в БД. Для этого выполните команду \d

4. Но для изучения структуры БД и выполнения запросов воспользуемся кроссплатформенной системой DBeaver.

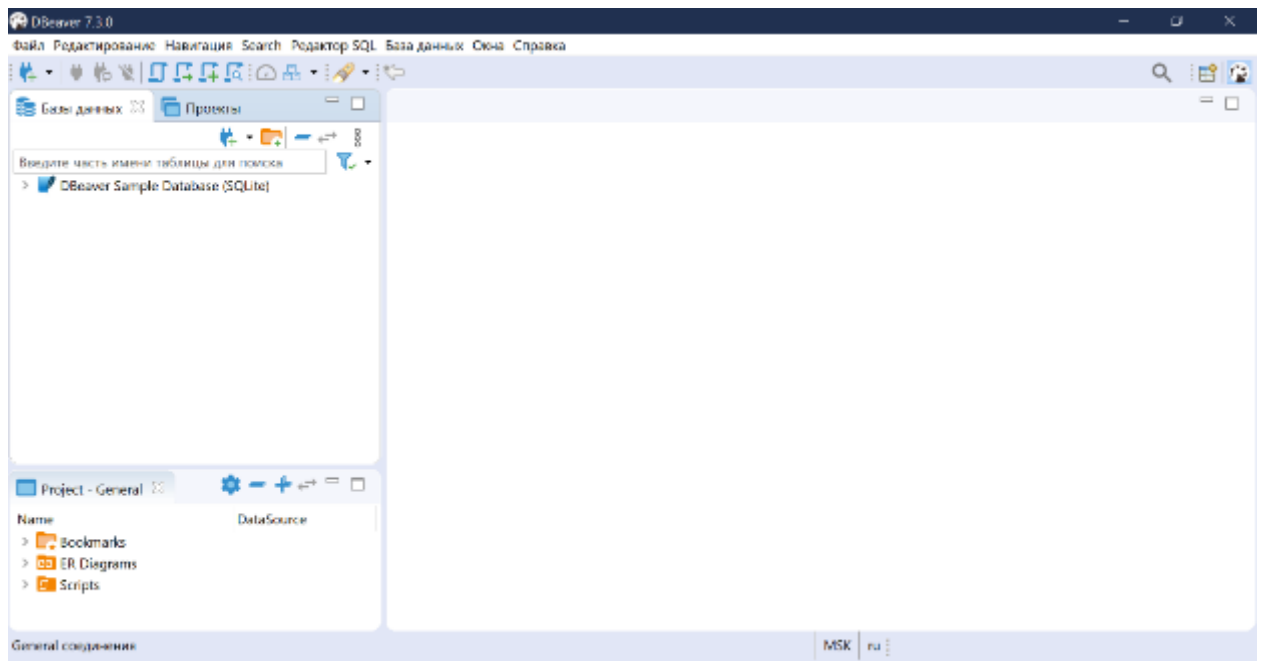


Рисунок 1 - Окно интерфейса DBeaver

5. Для подключения БД нажмите кнопку «Новое соединение» (рисунок 2)

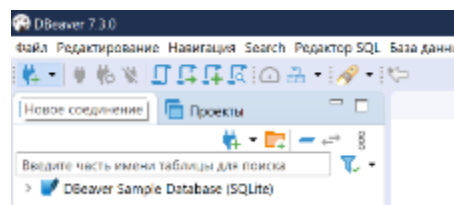


Рисунок 2 – Подключение БД

6. Откроется окно для выбора СУБД. Выберите PostgreSQL (рисунок 3)

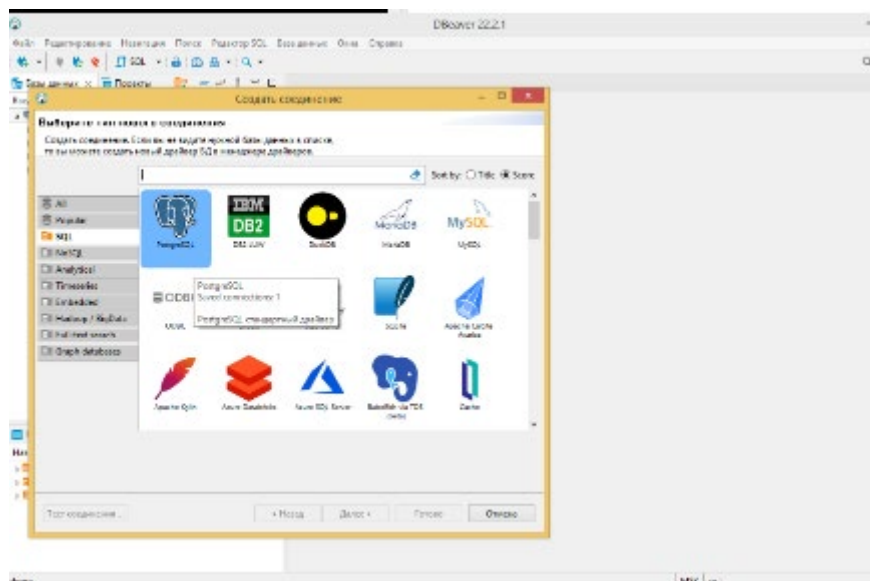


Рисунок 3 – Окно выбора нового соединения

7. В следующем окне (рисунок 4) необходимо провести настройку соединения. В поле База данных введите название БД демо как на

рисунке 4. Далее нажмите кнопку «Настройка драйвера». И выберите драйвер как на рисунке 5

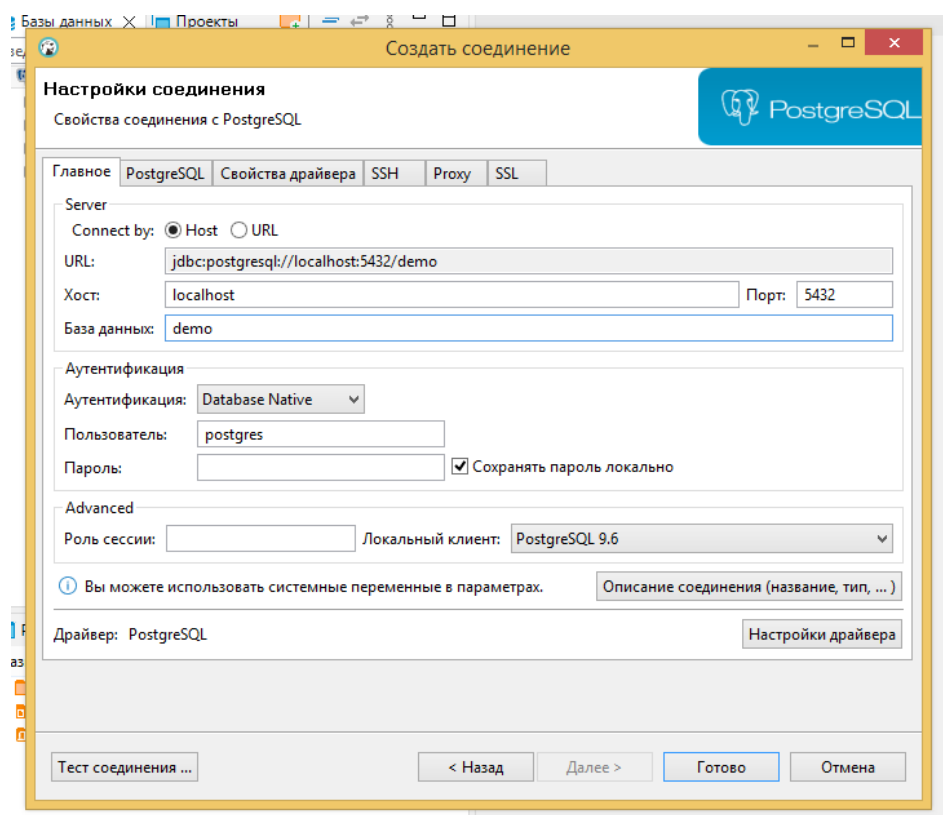


Рисунок 4 – Настройка соединения

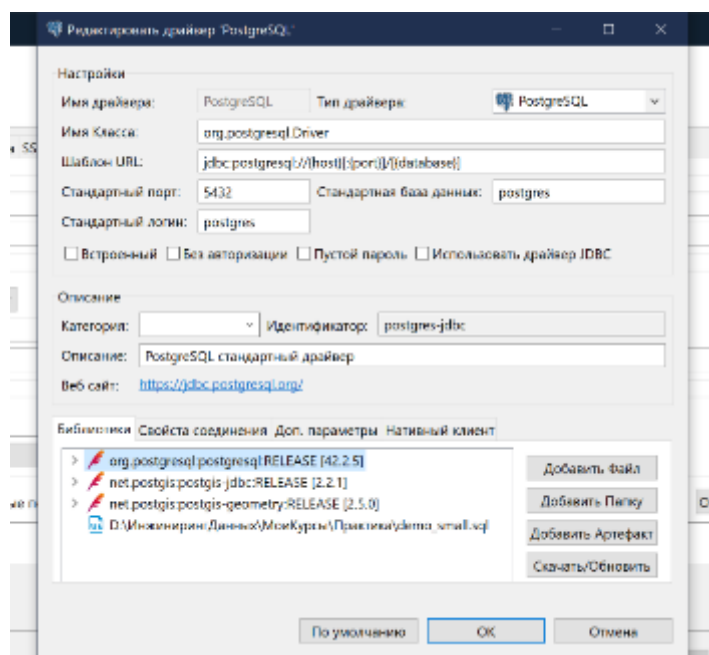


Рисунок 5 – Выбор драйвера

8. В поле Пароль введите пароль, который вы задали при установке PostgreSQL. Если настройки были сделаны верно, то в левой части окна у области списка БД появится БД demo (рисунок 6).

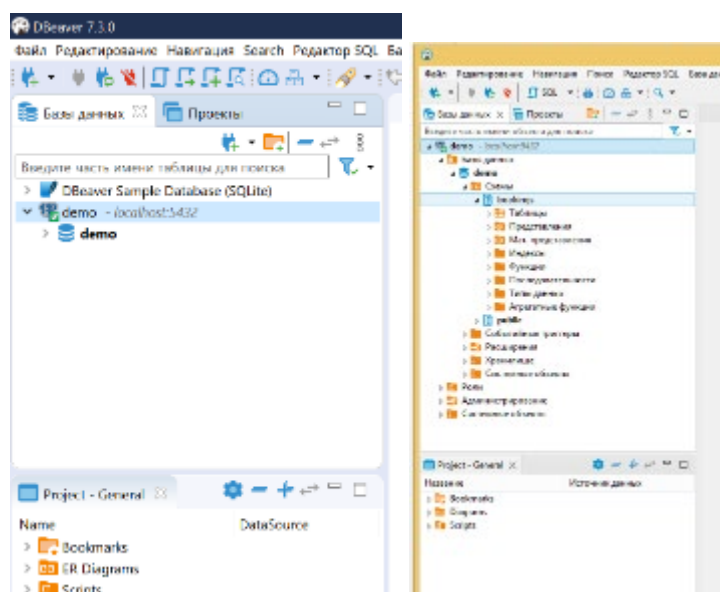


Рисунок 6 – Подключение БД

Раскроем все вложения БД и изучим БД. (рисунок 7) и посмотрим Схему данных. Для этого щелкните правой кнопкой мыши по таблице booking и выберите пункт «Открыть объект Схема» и перейдите на вкладку Диаграмма (рисунок 7).

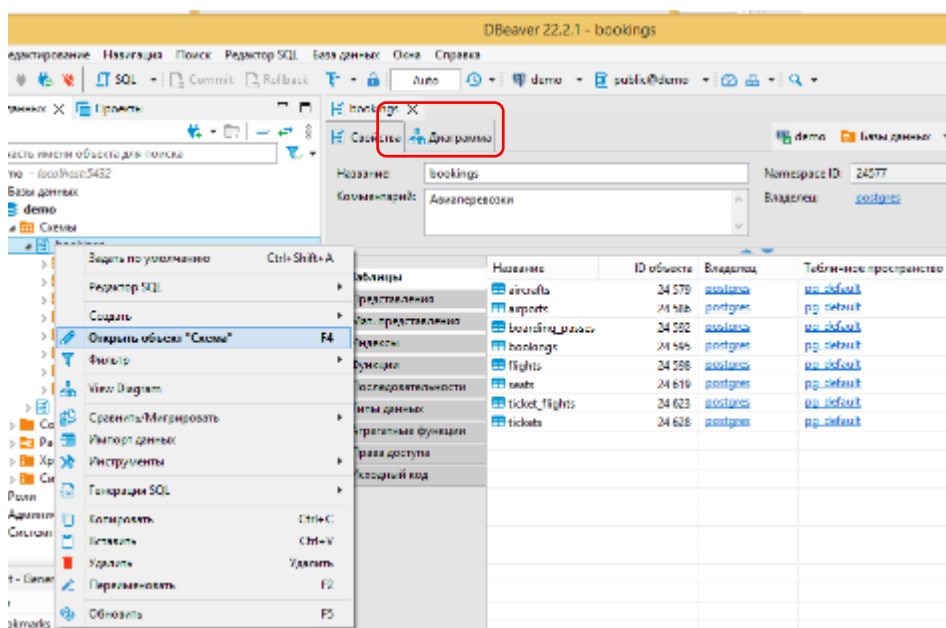


Рисунок 7 – Открытие Схемы данных

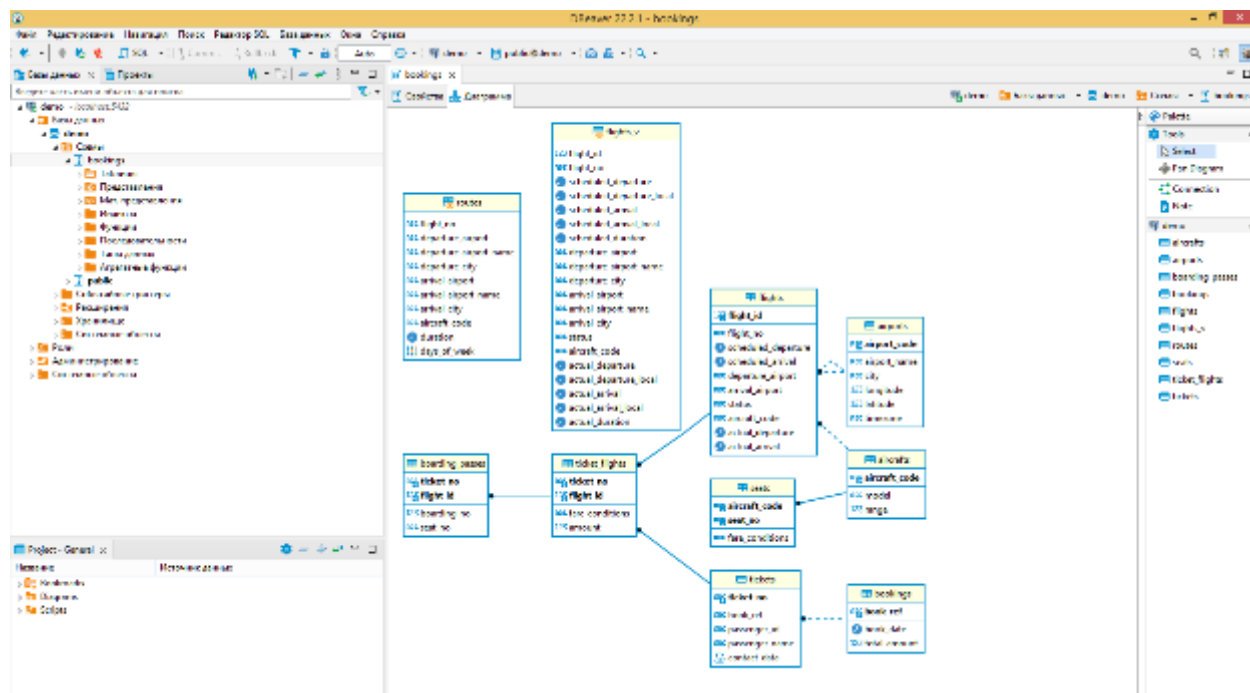


Рисунок 8 – Схема данных БД

Основной сущностью является Бронирование (bookings). В одно бронирование можно включить несколько пассажиров, каждому из которых выписывается отдельный билет (tickets). Билет включает один или несколько перелетов (ticket_flights). Несколько перелетов могут включаться в билет в случаях, когда нет прямого рейса, соединяющего пункты отправления и назначения (полет с пересадками), либо когда билет взят «туда и обратно». Каждый рейс (flights) следует из одного аэропорта (airports) в другой. Рейсы с одним номером имеют одинаковые пункты вылета и назначения, но будут отличаться датой отправления. При регистрации на рейс пассажиру выдается посадочный талон (boarding_passes), в котором указано место в самолете. Пассажир может зарегистрироваться только на тот рейс, который у него в рейсе и места в самолете уникальны.

Количество мест (seats) в самолете и их распределение по классам обслуживания зависит от модели самолета (aircrafts), выполняющего рейс. Предполагается, что каждая модель самолета имеет только одну компоновку салона.

На приведенной схеме (рисунок 8) отмечено только столбцы, соответствующие первичным и внешним ключам.

Рассмотрим основные объекты БД. Откроем таблицу Bookings (рисунок 9-11).

booking_id	ticket_id	book_date	book_status	price
BOOK1	2019-05-10 10:00:00	2019-05-10 10:00:00	1000	1000
BOOK2	2019-05-11 10:00:00	2019-05-11 10:00:00	1000	1000
BOOK3	2019-05-12 10:00:00	2019-05-12 10:00:00	1000	1000
BOOK4	2019-05-13 10:00:00	2019-05-13 10:00:00	1000	1000
BOOK5	2019-05-14 10:00:00	2019-05-14 10:00:00	1000	1000
BOOK6	2019-05-15 10:00:00	2019-05-15 10:00:00	1000	1000
BOOK7	2019-05-16 10:00:00	2019-05-16 10:00:00	1000	1000
BOOK8	2019-05-17 10:00:00	2019-05-17 10:00:00	1000	1000
BOOK9	2019-05-18 10:00:00	2019-05-18 10:00:00	1000	1000
BOOK10	2019-05-19 10:00:00	2019-05-19 10:00:00	1000	1000
BOOK11	2019-05-20 10:00:00	2019-05-20 10:00:00	1000	1000
BOOK12	2019-05-21 10:00:00	2019-05-21 10:00:00	1000	1000
BOOK13	2019-05-22 10:00:00	2019-05-22 10:00:00	1000	1000
BOOK14	2019-05-23 10:00:00	2019-05-23 10:00:00	1000	1000
BOOK15	2019-05-24 10:00:00	2019-05-24 10:00:00	1000	1000
BOOK16	2019-05-25 10:00:00	2019-05-25 10:00:00	1000	1000
BOOK17	2019-05-26 10:00:00	2019-05-26 10:00:00	1000	1000
BOOK18	2019-05-27 10:00:00	2019-05-27 10:00:00	1000	1000
BOOK19	2019-05-28 10:00:00	2019-05-28 10:00:00	1000	1000
BOOK20	2019-05-29 10:00:00	2019-05-29 10:00:00	1000	1000
BOOK21	2019-05-30 10:00:00	2019-05-30 10:00:00	1000	1000
BOOK22	2019-05-31 10:00:00	2019-05-31 10:00:00	1000	1000
BOOK23	2019-06-01 10:00:00	2019-06-01 10:00:00	1000	1000
BOOK24	2019-06-02 10:00:00	2019-06-02 10:00:00	1000	1000
BOOK25	2019-06-03 10:00:00	2019-06-03 10:00:00	1000	1000
BOOK26	2019-06-04 10:00:00	2019-06-04 10:00:00	1000	1000
BOOK27	2019-06-05 10:00:00	2019-06-05 10:00:00	1000	1000
BOOK28	2019-06-06 10:00:00	2019-06-06 10:00:00	1000	1000
BOOK29	2019-06-07 10:00:00	2019-06-07 10:00:00	1000	1000
BOOK30	2019-06-08 10:00:00	2019-06-08 10:00:00	1000	1000
BOOK31	2019-06-09 10:00:00	2019-06-09 10:00:00	1000	1000
BOOK32	2019-06-10 10:00:00	2019-06-10 10:00:00	1000	1000
BOOK33	2019-06-11 10:00:00	2019-06-11 10:00:00	1000	1000
BOOK34	2019-06-12 10:00:00	2019-06-12 10:00:00	1000	1000
BOOK35	2019-06-13 10:00:00	2019-06-13 10:00:00	1000	1000
BOOK36	2019-06-14 10:00:00	2019-06-14 10:00:00	1000	1000
BOOK37	2019-06-15 10:00:00	2019-06-15 10:00:00	1000	1000
BOOK38	2019-06-16 10:00:00	2019-06-16 10:00:00	1000	1000
BOOK39	2019-06-17 10:00:00	2019-06-17 10:00:00	1000	1000
BOOK40	2019-06-18 10:00:00	2019-06-18 10:00:00	1000	1000
BOOK41	2019-06-19 10:00:00	2019-06-19 10:00:00	1000	1000
BOOK42	2019-06-20 10:00:00	2019-06-20 10:00:00	1000	1000
BOOK43	2019-06-21 10:00:00	2019-06-21 10:00:00	1000	1000
BOOK44	2019-06-22 10:00:00	2019-06-22 10:00:00	1000	1000
BOOK45	2019-06-23 10:00:00	2019-06-23 10:00:00	1000	1000
BOOK46	2019-06-24 10:00:00	2019-06-24 10:00:00	1000	1000
BOOK47	2019-06-25 10:00:00	2019-06-25 10:00:00	1000	1000
BOOK48	2019-06-26 10:00:00	2019-06-26 10:00:00	1000	1000
BOOK49	2019-06-27 10:00:00	2019-06-27 10:00:00	1000	1000
BOOK50	2019-06-28 10:00:00	2019-06-28 10:00:00	1000	1000
BOOK51	2019-06-29 10:00:00	2019-06-29 10:00:00	1000	1000
BOOK52	2019-06-30 10:00:00	2019-06-30 10:00:00	1000	1000
BOOK53	2019-07-01 10:00:00	2019-07-01 10:00:00	1000	1000
BOOK54	2019-07-02 10:00:00	2019-07-02 10:00:00	1000	1000
BOOK55	2019-07-03 10:00:00	2019-07-03 10:00:00	1000	1000
BOOK56	2019-07-04 10:00:00	2019-07-04 10:00:00	1000	1000
BOOK57	2019-07-05 10:00:00	2019-07-05 10:00:00	1000	1000
BOOK58	2019-07-06 10:00:00	2019-07-06 10:00:00	1000	1000
BOOK59	2019-07-07 10:00:00	2019-07-07 10:00:00	1000	1000
BOOK60	2019-07-08 10:00:00	2019-07-08 10:00:00	1000	1000
BOOK61	2019-07-09 10:00:00	2019-07-09 10:00:00	1000	1000
BOOK62	2019-07-10 10:00:00	2019-07-10 10:00:00	1000	1000
BOOK63	2019-07-11 10:00:00	2019-07-11 10:00:00	1000	1000
BOOK64	2019-07-12 10:00:00	2019-07-12 10:00:00	1000	1000
BOOK65	2019-07-13 10:00:00	2019-07-13 10:00:00	1000	1000
BOOK66	2019-07-14 10:00:00	2019-07-14 10:00:00	1000	1000
BOOK67	2019-07-15 10:00:00	2019-07-15 10:00:00	1000	1000
BOOK68	2019-07-16 10:00:00	2019-07-16 10:00:00	1000	1000
BOOK69	2019-07-17 10:00:00	2019-07-17 10:00:00	1000	1000
BOOK70	2019-07-18 10:00:00	2019-07-18 10:00:00	1000	1000
BOOK71	2019-07-19 10:00:00	2019-07-19 10:00:00	1000	1000
BOOK72	2019-07-20 10:00:00	2019-07-20 10:00:00	1000	1000
BOOK73	2019-07-21 10:00:00	2019-07-21 10:00:00	1000	1000
BOOK74	2019-07-22 10:00:00	2019-07-22 10:00:00	1000	1000
BOOK75	2019-07-23 10:00:00	2019-07-23 10:00:00	1000	1000
BOOK76	2019-07-24 10:00:00	2019-07-24 10:00:00	1000	1000
BOOK77	2019-07-25 10:00:00	2019-07-25 10:00:00	1000	1000
BOOK78	2019-07-26 10:00:00	2019-07-26 10:00:00	1000	1000
BOOK79	2019-07-27 10:00:00	2019-07-27 10:00:00	1000	1000
BOOK80	2019-07-28 10:00:00	2019-07-28 10:00:00	1000	1000
BOOK81	2019-07-29 10:00:00	2019-07-29 10:00:00	1000	1000
BOOK82	2019-07-30 10:00:00	2019-07-30 10:00:00	1000	1000
BOOK83	2019-07-31 10:00:00	2019-07-31 10:00:00	1000	1000
BOOK84	2019-08-01 10:00:00	2019-08-01 10:00:00	1000	1000
BOOK85	2019-08-02 10:00:00	2019-08-02 10:00:00	1000	1000
BOOK86	2019-08-03 10:00:00	2019-08-03 10:00:00	1000	1000
BOOK87	2019-08-04 10:00:00	2019-08-04 10:00:00	1000	1000
BOOK88	2019-08-05 10:00:00	2019-08-05 10:00:00	1000	1000
BOOK89	2019-08-06 10:00:00	2019-08-06 10:00:00	1000	1000
BOOK90	2019-08-07 10:00:00	2019-08-07 10:00:00	1000	1000
BOOK91	2019-08-08 10:00:00	2019-08-08 10:00:00	1000	1000
BOOK92	2019-08-09 10:00:00	2019-08-09 10:00:00	1000	1000
BOOK93	2019-08-10 10:00:00	2019-08-10 10:00:00	1000	1000
BOOK94	2019-08-11 10:00:00	2019-08-11 10:00:00	1000	1000
BOOK95	2019-08-12 10:00:00	2019-08-12 10:00:00	1000	1000
BOOK96	2019-08-13 10:00:00	2019-08-13 10:00:00	1000	1000
BOOK97	2019-08-14 10:00:00	2019-08-14 10:00:00	1000	1000
BOOK98	2019-08-15 10:00:00	2019-08-15 10:00:00	1000	1000
BOOK99	2019-08-16 10:00:00	2019-08-16 10:00:00	1000	1000
BOOK100	2019-08-17 10:00:00	2019-08-17 10:00:00	1000	1000

Рисунок 9 – Таблица Bookings (данные таблицы)

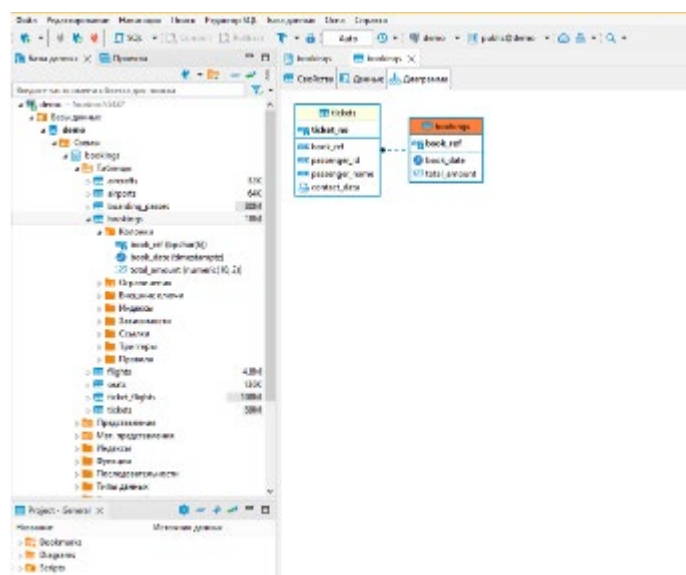


Рисунок 10 – Диаграмма таблицы Bookings

Имя	Тип данных	Инициализация	Примечание	Not Null	По умолчанию	Комментарий
booking_id	INTEGER	1	Primary Key	Yes		Идентификатор записи
ticket_id	INTEGER	2	Foreign Key	Yes		Идентификатор билета
book_date	DATE	3	Foreign Key	Yes		Дата бронирования
book_status	INTEGER	4	Foreign Key	Yes		Статус бронирования
price	DECIMAL	5		No		Цена билета

Рисунок 11 – Поля таблицы Bookings

Пассажир заранее (book_date максимум за месяц до рейса) бронирует билет себе и, возможно, нескольким другим пассажирам. Бронирование идентифицируется номером (book_ref, шестизначная комбинация букв и цифр). Поле total_amount хранит общую стоимость включенных в бронирование перелетов всех пассажиров.

Откроем редактор SQL (рисунок 12).

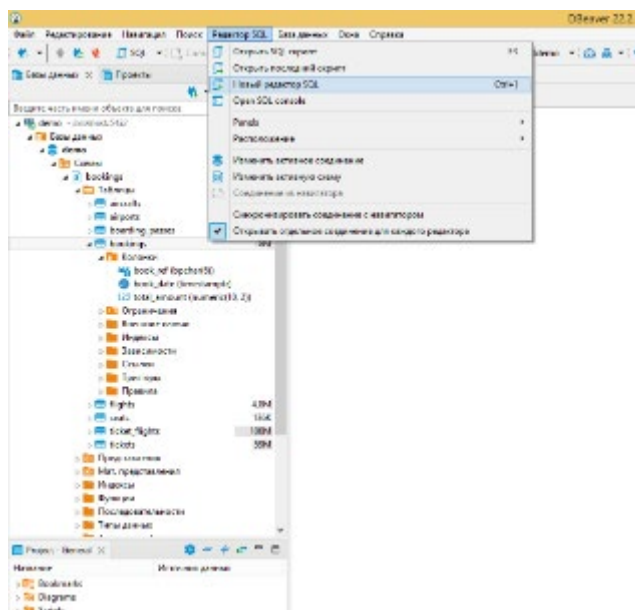


Рисунок 12 – Открытие редактора SQL

Начнем с бронирования и выберем какое-нибудь одно. Для этого напомним следующий запрос:

```
select * from bookings b where b.book_ref = '0824C5'
```

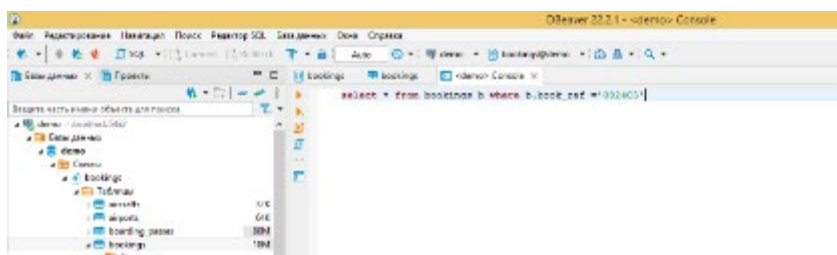


Рисунок 13 – Заброс о бронировании

Мы видим дату бронирования и общую сумму.

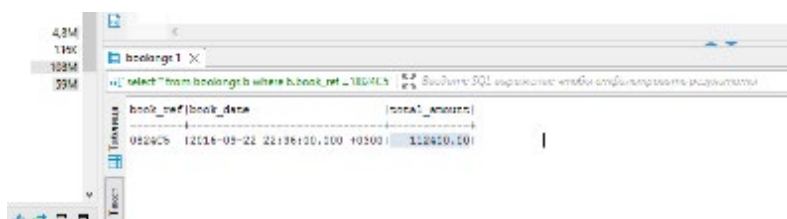


Рисунок 14 – Результат запроса бронирования

Если сравнить дату с текущей, то бронирование сделано довольно давно:

```
select now();
```

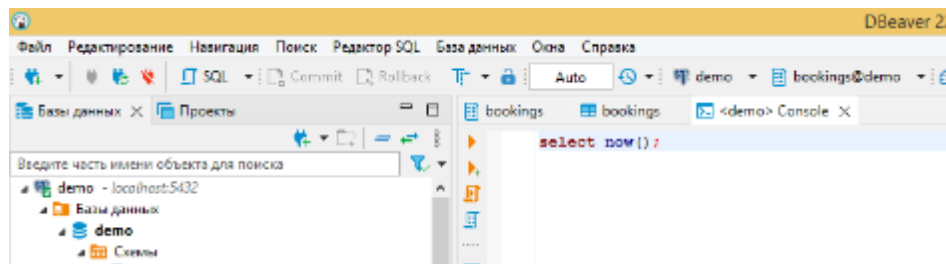


Рисунок 15 – Запрос текущей даты

bookings 1 X

select * from bookings b where b.book_ref = '0824C5'

book_ref	book_date	total_amount
0824C5	2016-09-22 22:36:00.000 +0300	112400.00

Рисунок 16 – Результат запроса текущей даты

Но для демобазы «текущим» моментом является другая дата. Для выяснения этой даты сделаем запрос:

```
select bookings.now();
```

Результат 1 X

select bookings.now();

now
2016-10-18 17:05:00.000 +0300

Рисунок 17 – Результат запроса текущей даты БД

Напишем запрос и выясним, когда была сделана следующая бронь 0824C5.

```
select bookings.now()-b.book_date  
from bookings b  
where b.book_ref='0824C5';
```



Рисунок 18 – Результат запроса

Итак, билеты были забронированы «20 дней назад».

Посмотрим следующую таблицу Билеты (Tickets).

Название	#	Тип данных	Автоматическое	Правильно сортировано	Not Null	По умолчанию	Комментарий
ticket_no	1	varchar(13)	default				Номер билета
book_ref	2	varchar(5)	default				Номер бронирования
passenger_id	3	varchar(20)	default				Идентификатор пассажира
passenger_name	4	text	default				Имя пассажира
contact_data	5	text					Контактные данные пассажира

Рисунок 18 – Таблица Билеты (Tickets).

Билет имеет уникальный номер (ticket-no), состоящий из 13 цифр. Билет содержит идентификатор пассажира (passenger_id) – номер документа, удостоверяющего личность, - его фамилию и имя (passenger_name) и контактную информацию (contact_data).

Ни идентификатор пассажира, ни имя не являются постоянными (можно поменять паспорт, можно сменить фамилию), поэтому однозначно найти все билеты одного и того же пассажира невозможно.

Посмотрим, какие билеты включены в выбранное бронирование.

```
select t.*
from bookings b
join tickets t on t.book_ref=b.book_ref
where b.book_ref = '0824C5'
```

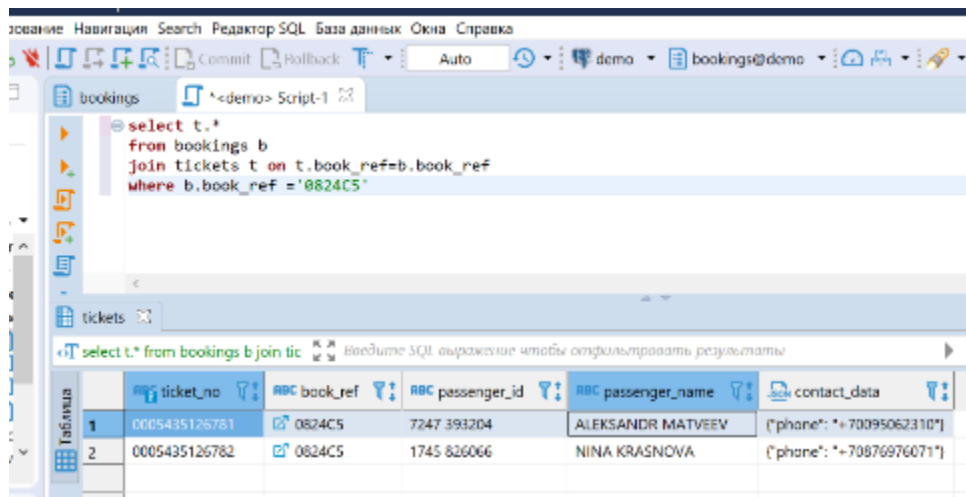


Рисунок 18 – Запрос о билетах бронирования

Как видно по результату запроса, летят два человека. На каждого оформляется собственный билет с информацией о пассажире.

Следующая таблица Перелеты (Ticket_flights).

ticket_flights

ID объекта: 15823

src: pg_default

Владелец: postgres

Перелеты

Дополнительные опции:

Создание:

Название	#	Тип данных	Длина	Точность	Масштаб	Автоувеличение	Правило сортировки	Not Null	По умолчанию	Комментарий
ticket_no	1	bpchar	13	13			default	☒		Номер билета
flight_id	2	int4		10				☒		Идентификатор рейс
fare_conditions	3	varchar	10	10			default	☒		Класс обслуживания
amount	4	numeric		10	2			☒		Стоимость перелета

Рисунок 19 – Структура таблицы Перелеты (Ticket_flights)

Перелет соединяет билет с рейсом и идентифицируется их номерами. Для каждого перелета указывают его стоимость (amount) и класс обслуживания (fare_conditions).

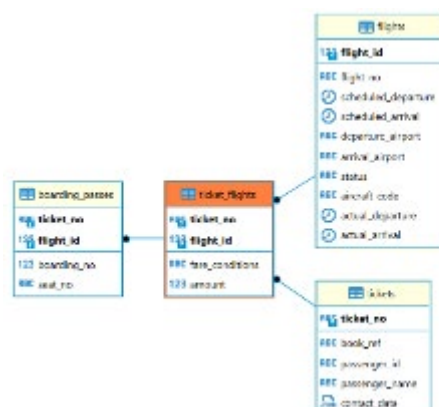
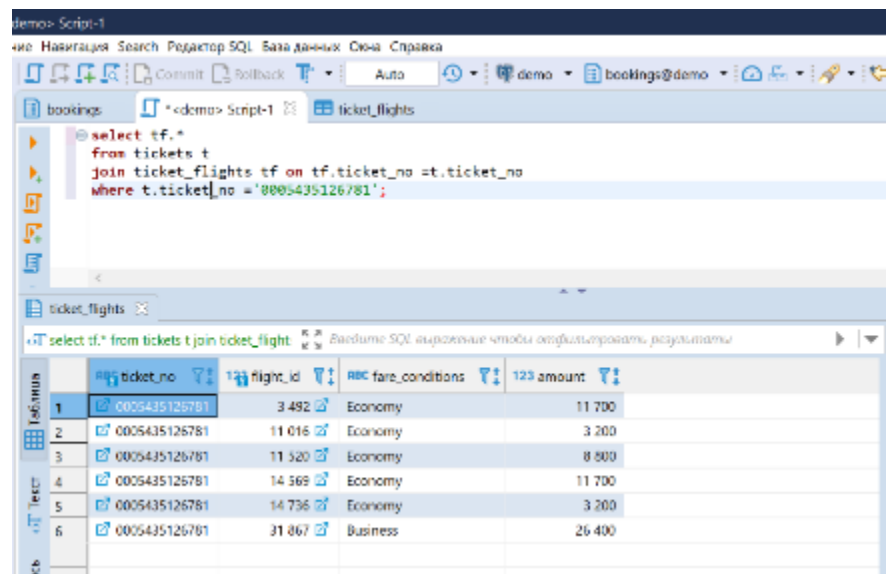


Рисунок 20 – Схема данных Перелеты (Ticket_flights)

Выясним, по какому маршруту летят пассажиры. Добавим в запрос перелеты.

```
select tf.*
from tickets t
join ticket_flights tf on tf.ticket_no = t.ticket_no
where t.ticket_no = '0005435126781';
```



	ticket_no	flight_id	fare_conditions	amount
1	0005435126781	3 492	Economy	11 700
2	0005435126781	11 016	Economy	3 200
3	0005435126781	11 520	Economy	8 800
4	0005435126781	14 369	Economy	11 700
5	0005435126781	14 736	Economy	3 200
6	0005435126781	31 867	Business	26 400

Рисунок 21 – Запрос по перелетам

Здесь мы смотрим только на один билет – все маршруты в одном бронировании всегда совпадают.

Видим, что в билете 6 переплетов. Среди них один – бизнес-классом, другие – эконом.

Следующий запрос сделаем для таблицы Рейсы (Flights). Рейс выполняется по расписанию из одного аэропорта в другой. Естественный ключ – номер рейса и дата отправления, но используется суррогатный ключ.

Рейс соединяет аэропорты вылета и прибытия. Такое понятие, как «рейс с пересадками» отсутствует: если нет прямого рейса, в билет просто включаются несколько рейсов.

flights	ID объекта:	16800								
pg_default	Владелец:	postgres								
Название	#	Тип данных	Длина	Точность	Масштаб	Алгоритм	Правило сортировки	Not Null	По умолчанию	Комментарий
flight_id	1	serial						☒	nextval('book_	Идентификатор рейса
flight_no	2	bpchar	6	6			default	☒		Номер рейса
scheduled_...	3	timestamptz		35	6			☒		Время вылета по расписан...
scheduled_...	4	timestamptz		35	6			☒		Время прилёта по расписан...
departure_...	5	bpchar	3	3			default	☒		Аэропорт отправления
arrival_airp...	6	bpchar	3	3			default	☒		Аэропорт прибытия
status	7	varchar	20	20			default	☒		Статус рейса
aircraft_code	8	bpchar	3	3			default	☒		Код самолета, IATA
actual_dep...	9	timestamptz		35	6			☐		Фактическое время вылета
actual_arrival	10	timestamptz		35	6			☐		Фактическое время прилёта

Рисунок 22 – Структура таблицы Рейсы (Flights)

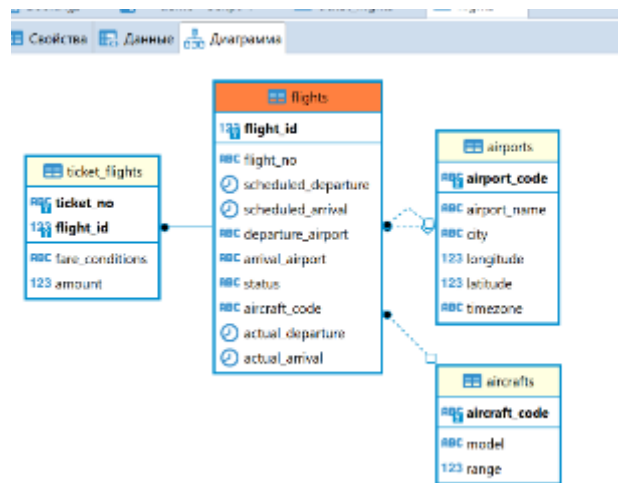


Рисунок 23 – Схема данных таблицы Рейсы (Flights)

Теперь посмотрим, какие рейсы скрываются за выбранными перелетами.

```

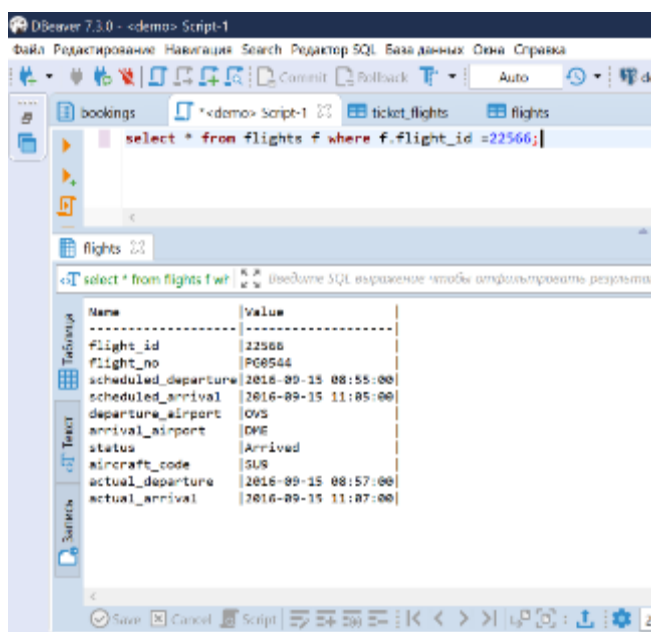
select f.flight_id, f.scheduled_departure,
f.departure_airport dep, f.arrival_airport arr,
f.status, f.aircraft_code aircraft
from tickets t
join ticket_flights tf on tf.ticket_no =t.ticket_no
join flights f on f.flight_id =tf.flight_id
where t.ticket_no ='0005435126781'
order by f.scheduled_departure ;
  
```

Редктирование	Навигация	Search	Редктор SQL	База данных	Сюда	Справка
bookings	*demo	Script-1	ticket_flights	flights		
<pre> select f.flight_id, f.scheduled_departure, f.departure_airport dep, f.arrival_airport arr, f.status, f.aircraft_code aircraft from tickets t join ticket_flights tf on tf.ticket_no =t.ticket_no join flights f on f.flight_id =tf.flight_id where t.ticket_no ='0005435126781' order by f.scheduled_departure ; </pre>						
<pre> select f.flight_id, f.scheduled_departure, f.dep, f.arr, f.status, f.aircraft </pre>						
flights	flight_id	scheduled_departure	dep	arr	status	aircraft
1	3492	2016-10-10 10:00:00	VKO	PEE	Arrived	773
2	14736	2016-10-10 14:30:00	PEE	SVX	Arrived	SU9
3	11520	2016-10-11 10:30:00	SVX	SGC	Arrived	SU9
4	31867	2016-10-13 13:45:00	SGC	SVX	Departed	SU9
5	11016	2016-10-14 07:50:00	SVX	PEE	On Time	SU9
6	14569	2016-10-14 17:55:00	PEE	VKO	Scheduled	773

Рисунок 24 – Результат запроса таблицы Рейсы (Flights)

Мы видим три рейса «туда» и три «обратно». «Туда» все рейсы уже совершены (Arrived), а в настоящее время пассажир летит «обратно» (Departed). Следующий рейс будет по расписанию (On Time), а на последний еще не открыта регистрация (Scheduled).

Посмотрим внимательнее на все столбцы одного из рейсов.



The screenshot shows the DBViewer 7.3.0 interface. The SQL editor contains the query: `select * from flights f where f.flight_id = 22566;`. The results pane displays the following data:

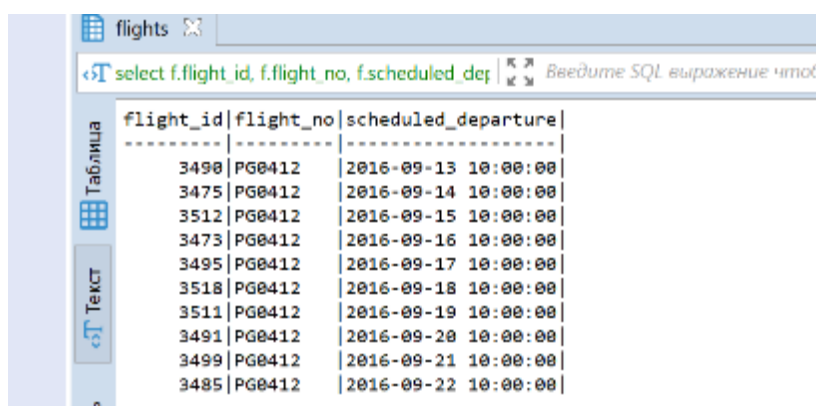
Name	Value
flight_id	22566
flight_no	PG0544
scheduled_departure	2016-09-15 08:55:00
scheduled_arrival	2016-09-15 11:05:00
departure_airport	OVS
arrival_airport	DME
status	Arrived
aircraft_code	SUR
actual_departure	2016-09-15 08:57:00
actual_arrival	2016-09-15 11:07:00

Рисунок 25

Реальное время может отличаться от времени по расписанию (обычно не сильно).

Номер flight_no одинаков для всех рейсов, следующих по одному маршруту по расписанию:

```
select f.flight_id, f.flight_no, f.scheduled_departure
from flights f
where f.flight_no = 'PG0412'
order by f.scheduled_departure
limit 10;
```



The screenshot shows the DBViewer 7.3.0 interface. The SQL editor contains the query: `select f.flight_id, f.flight_no, f.scheduled_dep`. The results pane displays the following data:

flight_id	flight_no	scheduled_departure
3498	PG0412	2016-09-13 10:00:00
3475	PG0412	2016-09-14 10:00:00
3512	PG0412	2016-09-15 10:00:00
3473	PG0412	2016-09-16 10:00:00
3495	PG0412	2016-09-17 10:00:00
3518	PG0412	2016-09-18 10:00:00
3511	PG0412	2016-09-19 10:00:00
3491	PG0412	2016-09-20 10:00:00
3499	PG0412	2016-09-21 10:00:00
3485	PG0412	2016-09-22 10:00:00

Рисунок 26

Следующая таблица «Аэропорты» (Airports)

Название	#	Тип данных	Длина	Точность	Масштаб	Автоувеличение	Правило сортировки	Not Null	По умолчанию	Комментарий
airport_code	1	varchar	3				default	✓		Код аэропорта
airport_name	2	text					default	✓		Название аэропорта
city	3	text					default	✓		Город
longitude	4	float		17	17			✓		Координаты аэропорта: дол.
latitude	5	float		17	17			✓		Координаты аэропорта: ши.
timezone	6	text					default	✓		Временная зона аэропорта

Рисунок 27

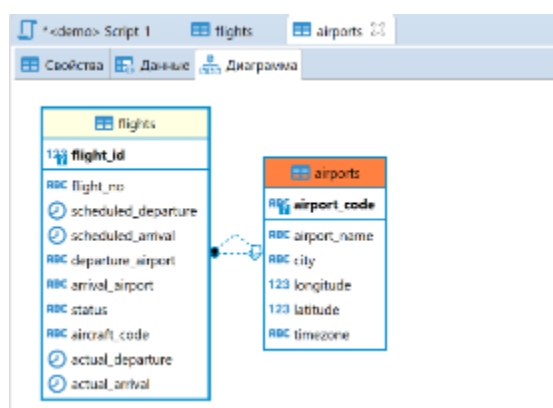


Рисунок 28 – Схема данных таблицы «Аэропорты» (Airports)

Аэропорт идентифицируется трехбуквенным кодом (airport_code) и имеет свое имя (airport_name). Для города не предусмотрено отдельной сущности, но введено поле с названием города (city), позволяющее найти аэропорты одного города. Это представление также включает координаты аэропорта (coordinates) и часовой пояс (timezone).

Значения полей airport_name и city определяются в зависимости от выбранного в конфигурационном параметре bookings.lang языка.

Расшифруем сведения от рейсах.

```
select f.scheduled_departure,
dep.airport_code || ' ' || dep.city || ' (' || dep.airport_name || ')' departure,
arr.airport_code || ' ' || arr.city || ' (' || arr.airport_name || ')' arrival
from tickets t
  join ticket_flights tf on tf.ticket_no = t.ticket_no
  join flights f on f.flight_id = tf.flight_id
  join airports dep on dep.airport_code = f.departure_airport
  join airports arr on arr.airport_code = f.arrival_airport
where t.ticket_no = '0005435126781'
order by f.scheduled_departure;
```

The screenshot shows a database window with a query: `select f.scheduled_departure`. The results are displayed in a table with three columns: `scheduled_departure`, `departure`, and `arrival`. The data shows flight schedules between Moscow (VKO) and Perm (PEE) via Yekaterinburg (SVX) and Surgut (SGC).

scheduled_departure	departure	arrival
2016-10-10 10:00:00	VKO Москва (Внуково)	PEE Пермь (Пермь)
2016-10-10 14:30:00	PEE Пермь (Пермь)	SVX Екатеринбург (Кольцово)
2016-10-11 10:30:00	SVX Екатеринбург (Кольцово)	SGC Сургут (Сургут)
2016-10-13 13:45:00	SGC Сургут (Сургут)	SVX Екатеринбург (Кольцово)
2016-10-14 07:50:00	SVX Екатеринбург (Кольцово)	PEE Пермь (Пермь)
2016-10-14 17:55:00	PEE Пермь (Пермь)	VKO Москва (Внуково)

Рисунок 29

Чтобы не выписывать каждый раз подобный запрос, существует представление `flights_v`:

```
select * from flights_v f where f.flight_id =22566;
```

The screenshot shows a database window with a query: `SELECT * FROM flights_v f WHERE f.f`. The results are displayed in a table with two columns: `Name` and `Value`. The data shows detailed flight information for flight ID 22566, including scheduled and actual times, durations, and airport names.

Name	Value
flight_id	22566
flight_no	PG0544
scheduled_departure	2016-09-15 08:55:00.000 +0300
scheduled_departure_local	2016-09-15 10:55:00.000
scheduled_arrival	2016-09-15 11:05:00.000 +0300
scheduled_arrival_local	2016-09-15 11:05:00.000
scheduled_duration	02:10:00
departure_airport	OVS
departure_airport_name	Советский
departure_city	Советский
arrival_airport	DME
arrival_airport_name	Домодедово
arrival_city	Москва
status	Arrived
aircraft_code	SU9
actual_departure	2016-09-15 08:57:00.000 +0300
actual_departure_local	2016-09-15 10:57:00.000
actual_arrival	2016-09-15 11:07:00.000 +0300
actual_arrival_local	2016-09-15 11:07:00.000
actual_duration	02:10:00

Рисунок 30

Здесь мы видим и локальное время в часовых поясах городов отправления и прибытия, длительность полета, названия аэропортов.

Поскольку в демобазе маршруты не меняются со временем, из таблицы рейсов моно выделить информацию, которая не зависит от конкретной даты вылета. Такая информация собрана в представлении `routes`:

```
select * from routes r where r.flight_no = 'PG0412'
```

The screenshot shows a SQL query window titled "routes 1". The query is `select * from routes r where r.flight_r`. The results are displayed in a table with two columns: "Name" and "Value".

Name	Value
flight_no	PG0412
departure_airport	VKO
departure_airport_name	Внуково
departure_city	Москва
arrival_airport	PEE
arrival_airport_name	Пермь
arrival_city	Пермь
aircraft_code	773
duration	01:25:00
days_of_week	{1,2,3,4,5,6,7}

Рисунок 31

Видно, что рейсы выполняются ежедневно (массив `days_of_week`).

Перейдем к рассмотрению таблицы Самолеты (Aircrafts).

Каждая модель воздушного судна идентифицируется своим трехзначным кодом (`aircraft_code`). Указывается также название модели (`model`) и максимальная дальность полета в километрах (`range`). Значение поля `model` определяется в зависимости от выбранного в конфигурационном параметре `bookings.lang` языка.

The screenshot shows a database management tool interface. The "aircrafts" table is selected. The table structure is displayed in a table below.

Имя	#	Тип данных	Автоувеличение	Правило сортировки	Not Null	По умолчанию	Комментарий
aircraft_code	1	bpchar(3)		default	[x]		Код самолета, IATA
model	2	text		default	[x]		Модель самолета
range	3	int4			[x]		Максимальная дальность

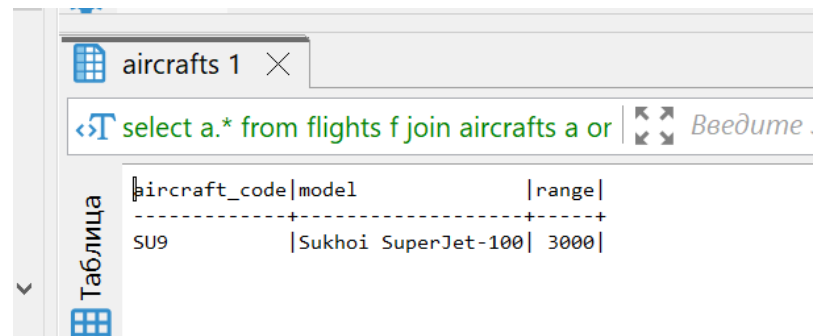
Рисунок 32

Модели самолетов, обслуживающих рейсы, также используют стандартные трехсимвольные коды в качестве первичных ключей.

```

select a.*
from flights f
      join aircrafts a on a.aircraft_code
      =f.aircraft_code
      where f.flight_id =22566;

```

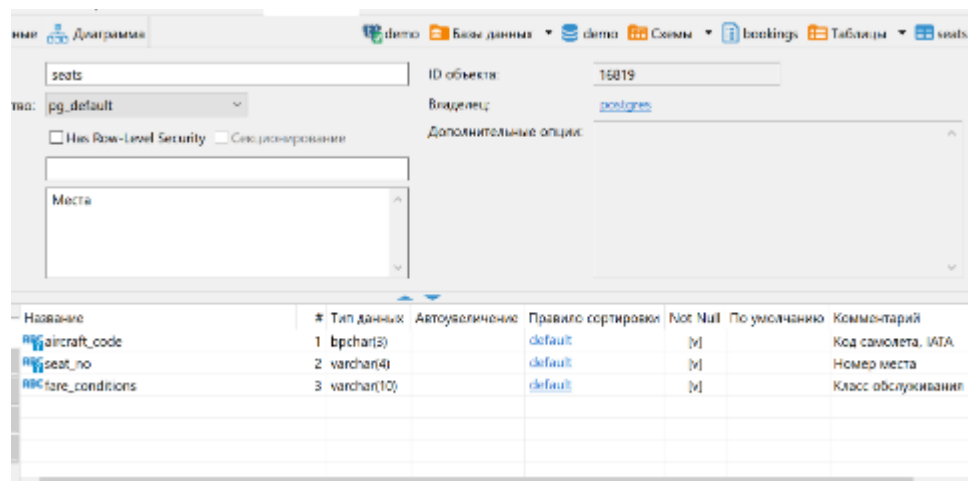


aircraft_code	model	range
SU9	Sukhoi SuperJet-100	3000

Рисунок 33

Следующая таблица Места (Seats)

Места определяют схему салона каждой модели. Каждое место определяется своим номером (seat_no) и имеет закрепленный за ним класс обслуживания (fare_conditions) – Economy, Comfort или Business.



Название	#	Тип данных	Автоувеличение	Правило сортировки	Not Null	По умолчанию	Комментарий
aircraft_code	1	bpchar(3)		default	[x]		Код самолета, IATA
seat_no	2	varchar(4)		default	[x]		Номер места
fare_conditions	3	varchar(10)		default	[x]		Класс обслуживания

Рисунок 34

В демобазе все самолеты одной модели имеют одинаковую конфигурацию самолета. Посмотрим на первый ряд:

```

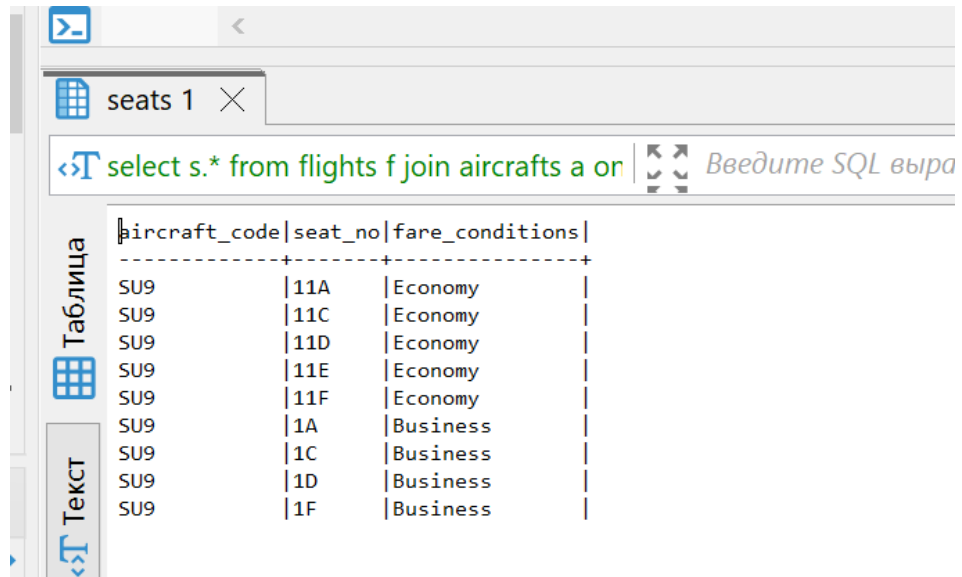
select s.*
from flights f
      join aircrafts a on a.aircraft_code
      =f.aircraft_code

```

```

join seats s on s.aircraft_code = a.aircraft_code
where f.flight_id =22566
and s.seat_no ~ '1.$';

```



The screenshot shows a database client window with a tab labeled 'seats 1'. The SQL editor contains the query: `select s.* from flights f join aircrafts a on f.flight_id = a.flight_id and s.seat_no ~ '1.$';`. The results are displayed in a table view with columns: aircraft_code, seat_no, and fare_conditions.

aircraft_code	seat_no	fare_conditions
SU9	11A	Economy
SU9	11C	Economy
SU9	11D	Economy
SU9	11E	Economy
SU9	11F	Economy
SU9	1A	Business
SU9	1C	Business
SU9	1D	Business
SU9	1F	Business

Рисунок 35

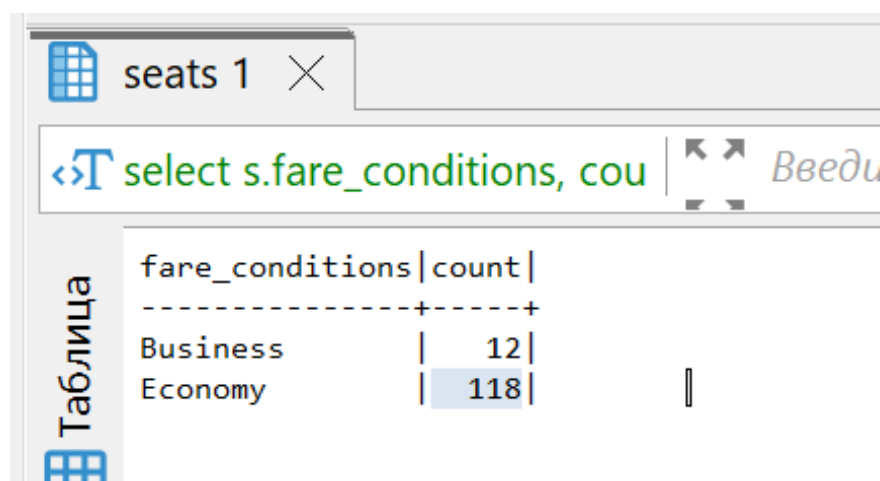
Это бизнес-класс.

А вот общее число мест различных классов обслуживания.

```

select s.fare_conditions, count(*)
from seats s
where s.aircraft_code = '733'
group by s.fare_conditions ;

```



The screenshot shows a database client window with a tab labeled 'seats 1'. The SQL editor contains the query: `select s.fare_conditions, count(*) from seats s where s.aircraft_code = '733' group by s.fare_conditions ;`. The results are displayed in a table view with columns: fare_conditions and count.

fare_conditions	count
Business	12
Economy	118

Рисунок 36

Таблица Посадочные талоны (Boarding_passes).

При регистрации на рейс, которая возможна за сутки до плановой даты отправления, пассажиру выдается посадочный талон. Он идентифицируется также, как и перелет – номером билета и номером рейса.

Посадочным талоном присваиваются последовательные номера (boarding_no) в порядке регистрации пассажиров на рейс (этот номер будет уникальным только в пределах данного рейса). В посадочном талоне указывается номер места (seat_no).

The screenshot shows a database management tool interface. At the top, there's a search bar with 'boarding_passes' entered. To the right, 'ID объекта' is '16794' and 'Владелец' is 'postgres'. Below this, there are checkboxes for 'Has Row-Level Security' and 'Синхронизация', and a section for 'Дополнительные опции'. A list box labeled 'Посадочные талоны' is visible. At the bottom, a table structure is shown with columns: 'ticket_no', 'flight_id', 'boarding_no', and 'seat_no'. The table has 4 columns and 4 rows of data.

ticket_no	flight_id	boarding_no	seat_no
0005435126781	1	1	1
0005435126781	1	2	2
0005435126781	1	3	3
0005435126781	1	4	4

Рисунок 37

Чтобы выяснить на каких местах сидел пассажир, необходимо заглянуть в посадочный талон, который выдается при регистрации на рейс.

```
select f.status, bp.*  
from tickets t  
    join ticket_flights tf on tf.ticket_no  
=t.ticket_no  
    join flights f on f.flight_id =tf.flight_id  
    left join boarding_passes bp  
    on bp.ticket_no =tf.ticket_no and bp.flight_id  
=tf.flight_id  
where t.ticket_no  ='0005435126781'  
    order by f.scheduled_departure;
```

flights(+) 1 X

`<>T select f.status, bp.* from tickets t join tic` Введите SQL выражение

Таблица

status	ticket_no	flight_id	boarding_no	seat_no
Arrived	0005435126781	3492	84	22A
Arrived	0005435126781	14736	71	19D
Arrived	0005435126781	11520	14	8D
Departed	0005435126781	31867	7	3F
On Time				
Scheduled				

Текст

Рисунок 38

На два оставшихся рейса пассажир еще не зарегистрировался.

Самостоятельное решение задач

Напишите следующие запросы:

1. Сколько человек бывает включено в одно бронирование?
2. До каких городов нельзя добраться без пересадок из Москвы?
3. Какая модель самолета выполняет больше всего рейсов, а какая – меньше всего?
4. Какая модель перевозит больше всего пассажиров?

Лабораторная работа № 10-11. Последовательный доступ

Теоретическое обоснование

В распоряжении оптимизатора имеется несколько способов доступа к данным. Самый простой из них – последовательное сканирование таблицы. Файл (или файлы) таблицы читаются постранично от начала до конца. При этом рассматриваются все версии строк на каждой странице: удовлетворяют ли они условиям запроса и соблюдены ли правила видимости.

Напомним, что чтение происходит через буферный кеш. Чтобы большая таблица не вытеснила все полезные данные, для последовательного сканирования создается «кольцо буферов» небольшого размера. При этом другие процессы, одновременно выполняющие последовательное сканирование той же таблицы, «присоединяются» к тому же кольцу и тем

самым экономят дисковое чтение (если процесс присоединился не сразу, он затем отдельно дочитывает начальные страницы таблицы).

Последовательное чтение файла позволяет использовать тот факт, что операционная система обычно читает данные порциями больше, чем размер страницы: с большой вероятностью несколько других страниц уже окажутся в кеше ОС.

Последовательное сканирование эффективно работает, когда надо прочитать всю таблицу или значительную ее часть (если селективность условия низка). Если же из всей таблицы нужна только небольшая часть записей, более предпочтительными являются методы доступа, использующие индекс.

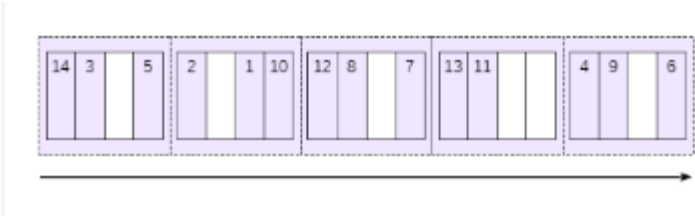


Рисунок 1 – Схема последовательного сканирования

ОСНОВНЫЕ ОПЕРАТОРЫ ПЛАНА PG SQL

Оператор	Пояснение
Seq Scan	Последовательный перебор строк таблицы (возможно с отбором по условию)
Index Only Scan	Поиск по покрывающему индексу без захода в основную таблицу
Index Scan	Поиск по индексу, с заходом в основную таблицу за доп. колонками
Nested Loops	Соединение вложенными циклами
Hash Join	Соединение с помощью хеш-таблицы (Соответствия)
Merge Join	Соединение заранее отсортированных наборов с помощью алгоритма слияния
Sort	Сортировка УПОРЯДОЧИТЬ ПО
Прочие	Агрегаты СГРУППИРОВАТЬ ПО, ПЕРВЫЕ и пр.

Последовательное сканирование

В плане выполнения запроса последовательное сканирование представлено узлом Seq Scan (рисунок 1):

```
explain select * from flights;
```

Результат запроса (рисунок 2)

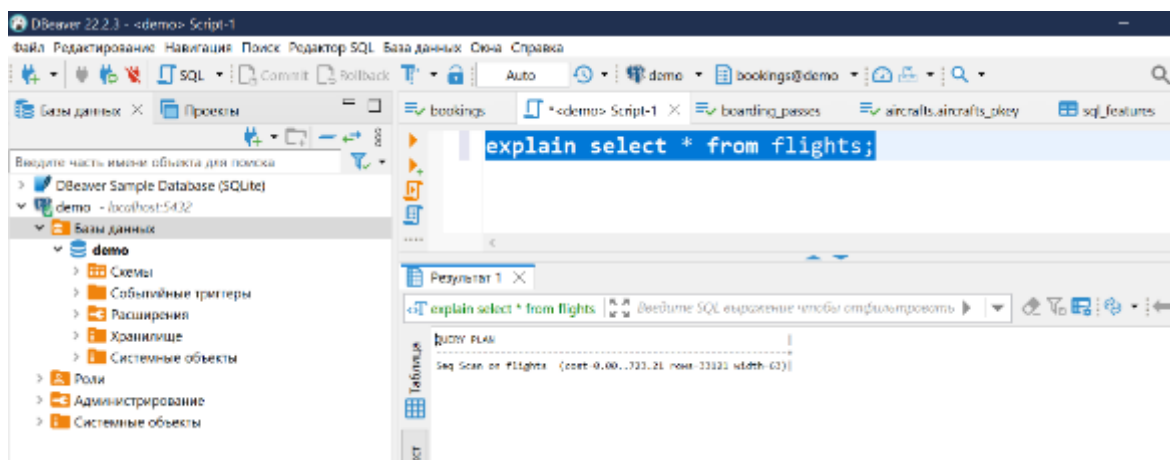


Рисунок 2 – Результат запроса последовательного сканирования

EXPLAIN сообщает, что используется Seq Scan — последовательное, блок за блоком, чтение данных таблицы foo.

В скобках приведены важные значения:

- cost — оценка стоимости;
- rows — оценка числа строк, возвращаемых операцией;
- width — оценка размера одной записи в байтах.

Стоимость указывается в некоторых условных единицах и состоит из двух компонент.

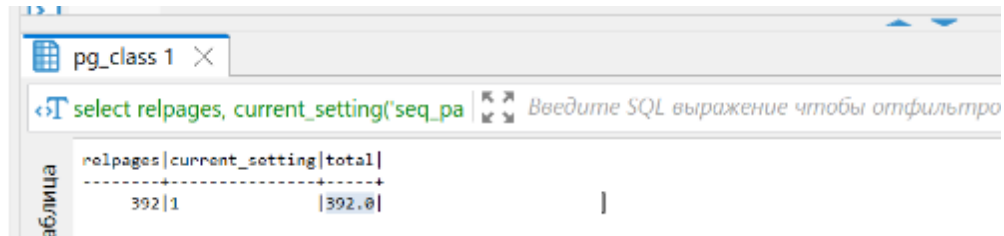
Первое число показывает начальную стоимость вычисления узла. Для последовательного сканирования это ноль — чтобы возвращать данные, никакой подготовки не требуется.

Второе число показывает полную стоимость для получения всех данных. Как оно получается?

Оптимизатор PostgreSQL учитывает дисковый ввод-вывод и ресурсы процессора. Составляющая ввода-вывода рассчитывается как произведение числа страниц в таблице на условную стоимость чтения одной страницы (рисунок 3):

```
select relpages, current_setting('seq_page_cost'),
```

```
relpages * current_setting('seq_page_cost'):: real
AS total
from pg_class where relname='flights';
```

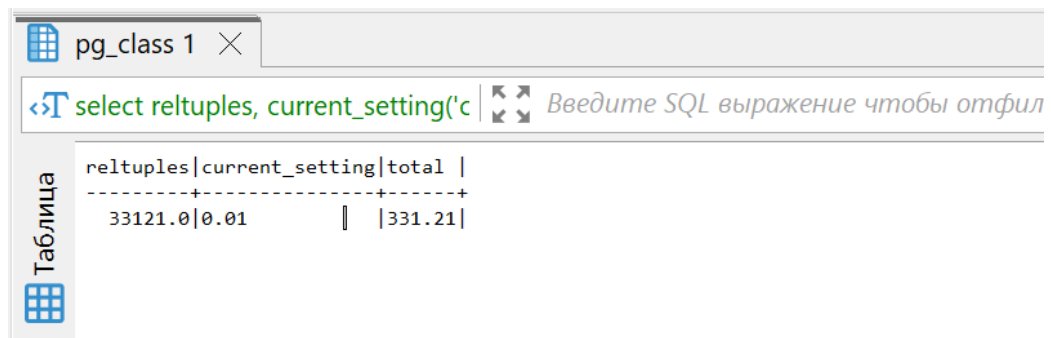


relpages	current_setting	total
392	1	392.0

Рисунок 3 – Результат вычисления составляющей ввода-вывода

Составляющая ресурсов процессора складывается из стоимости обработки каждой строки (рисунок 4):

```
select reltuples, current_setting('cpu_tuple_cost'),
reltuples * current_setting('cpu_tuple_cost'):: real
AS total
from pg_class where relname='flights';
```



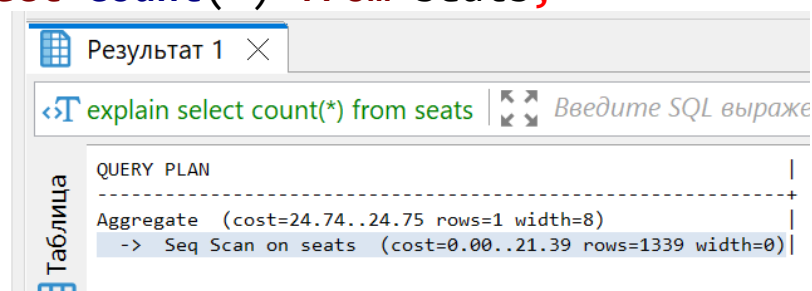
reltuples	current_setting	total
33121.0	0.01	331.21

Рисунок 4 - Результат вычисления составляющей процессора

Агрегация

Рассмотрим более сложный запрос с агрегатной функцией (на небольшой таблице) (рисунок 5):

```
explain select count(*) from seats;
```



QUERY PLAN
Aggregate (cost=24.74..24.75 rows=1 width=8)
-> Seq Scan on seats (cost=0.00..21.39 rows=1339 width=0)

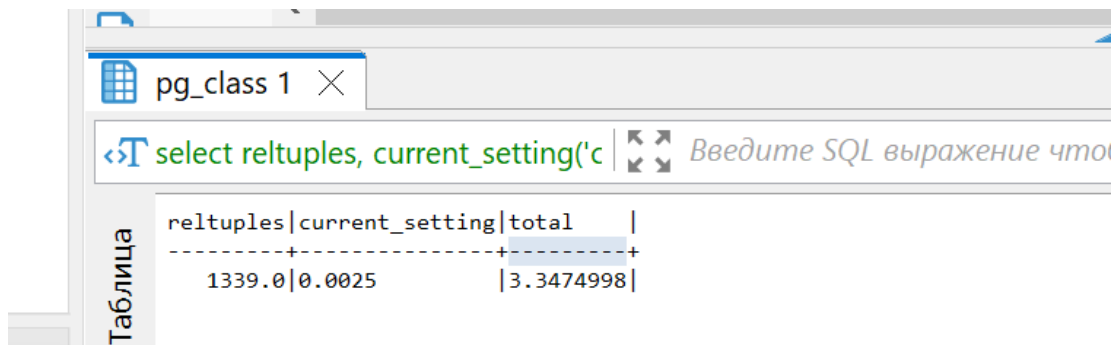
Рисунок 5 - Сложный запрос с агрегатной функцией

План состоит из двух узлов. Верхний – Aggregate, в котором происходит вычисление count, - получает данные от нижнего – Seq Scan.

Обратите внимание на стоимость узла Aggregate: начальная стоимость практически равна полной. Это означает, что узел не может выдать результат, пока не обработает все данные (что вполне логично).

Разница между оценкой Aggregate и верхней оценкой для Seq Scan-стоимость работы собственно узла Aggregate. Он вычисляется исходя из оценки ресурсов на выполнение элементарной операции (рисунок 6):

```
select reltuples,  
current_setting('cpu_operator_cost'),  
reltuples * current_setting('cpu_operator_cost')::  
real AS total  
from pg_class where relname='seats';
```



The screenshot shows a PostgreSQL query editor window titled "pg_class 1". The query entered is:

```
select reltuples, current_setting('cpu_operator_cost'),  
reltuples * current_setting('cpu_operator_cost')::  
real AS total  
from pg_class where relname='seats';
```

The query results are displayed in a table with the following data:

reltuples	current_setting	total
1339.0	0.0025	3.3474998

Рисунок 6 –

Параллельные планы

Ведущий процесс

выполняет последовательную часть плана
запускает рабочие процессы и получает от них данные

Рабочие процессы

одновременно работают над параллельной частью плана

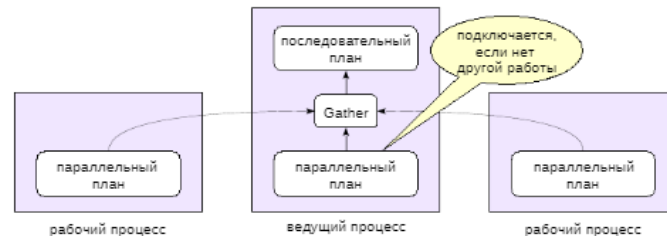


Рисунок 7

PostgreSQL поддерживает параллельное выполнение запросов. Идея состоит в том, что ведущий процесс, выполняющий запрос, порождает (с помощью postmaster) несколько рабочих процессов, которые одновременно выполняют одну и ту же «параллельную» часть плана. Результаты этого выполнения передаются ведущему процессу, который собирает их в узле Gather.

Если рабочие процессы не успевают загрузить ведущего данными, ведущий процесс тоже подключается к выполнению того же самого параллельного плана.

Разумеется, запуск процессов и пересылка данных требуют определенных ресурсов, поэтому далеко не каждый запрос выполняется параллельно.

Кроме того, даже при параллельном выполнении не все шаги плана запроса могут быть распараллелены. Часть операций может выполняться ведущим процессом в одиночку, последовательно.

В PostgreSQL нет другого теоретически возможного режима распараллеливания, при котором несколько процессов составляют конвейер для обработки данных (грубо говоря, отдельные узлы плана выполняются отдельными процессами). Разработчики PostgreSQL сочли такой режим неэффективным.

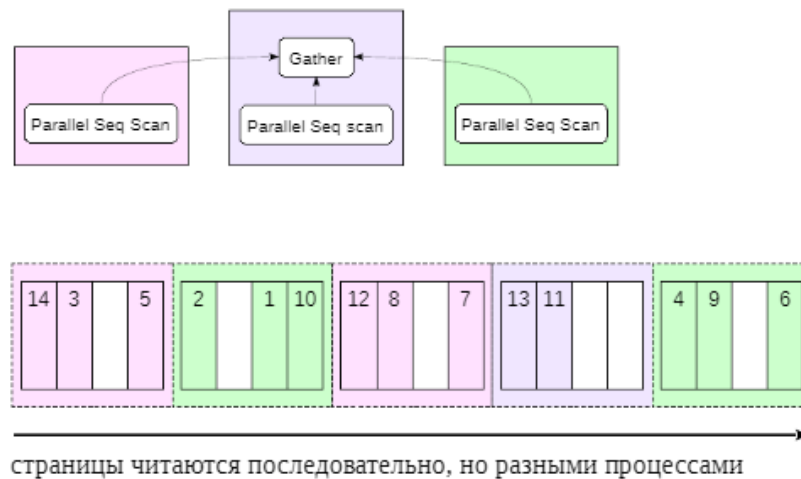


Рисунок 8

Примером параллельного выполнения является Parallel Seq Scan – «параллельное последовательное сканирование». Название звучит противоречиво, но, тем не менее, отражает суть операции. Страницы таблицы читаются последовательно, в том же самом порядке, в котором они читались бы при обычном последовательном сканировании. Однако запросы на чтение выдаются несколькими параллельными работающими процессами. Процессы синхронизируются между собой, чтобы их запросы шли в правильном порядке.

Преимущество такого сканирования появляется в том, что параллельные процессы одновременно обрабатывают свои страницы. Чтобы эффект оправдал дополнительные накладные расходы на пересылку данных от процесса к процессу, обработка должна быть достаточно ресурсоемкой. Хорошим примером является агрегация данных, поскольку для нее необходимы ресурсы процессора, а пересылать требуется только одно итоговое число. В таком случае параллельное выполнение запроса может занять существенно меньше времени, чем последовательное.

Рассмотри пример плана с параллельным последовательным сканированием (рисунок 9):

```
explain select count(*) from bookings b ;
```

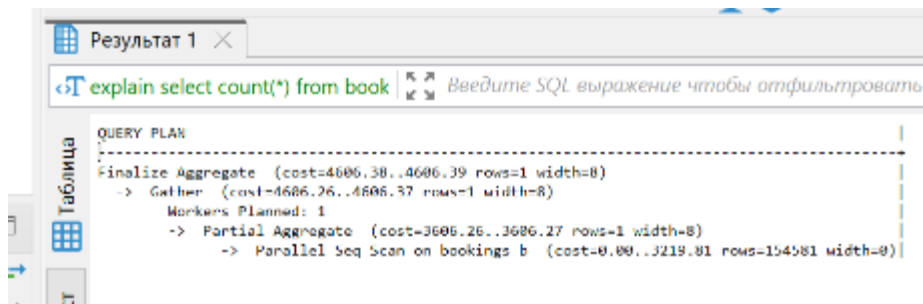


Рисунок 9

Все, что находится ниже узла Gather – параллельная часть плана. Она выполняется в каждом из рабочих процессов (которых запланировано два) и, возможно, в ведущем процессе.

Узел Gather и все узлы выше выполняются только в предыдущем процессе. Это последовательная часть плана.

Начнем разбираться снизу вверх. Узел Parallel Seq Scan представляет сканирование таблицы в параллельном режиме.

В поле rows показана оценка числа строк, которые обработает один рабочий процесс. Всего их запланировано 2, и еще часть работы выполняет ведущий, поэтому общее число строк делится на 2.4 (доля ведущего процесса уменьшается с ростом числа рабочих процессов).

```

select round(reltuples/2.4) "rows"
from pg_class where relname = 'bookings';
  
```

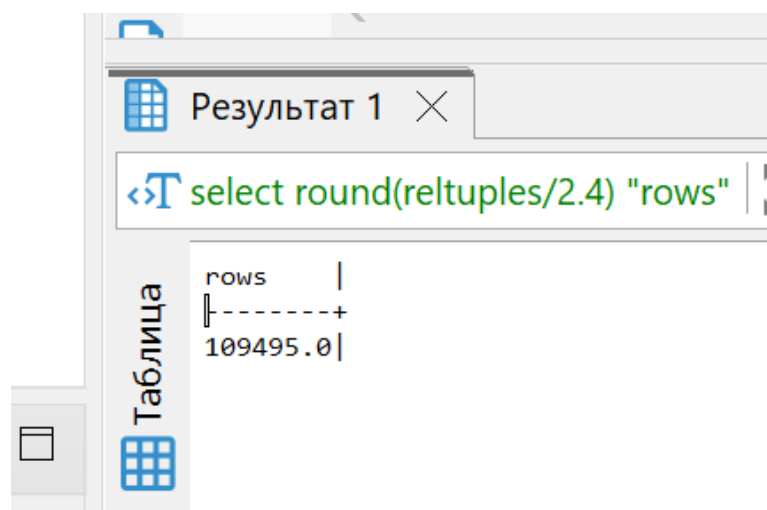


Рисунок 10 –

По умолчанию ведущий процесс участвует в выполнении параллельной части плана (рисунок 11):

```
show parallel_leader_participation;
```

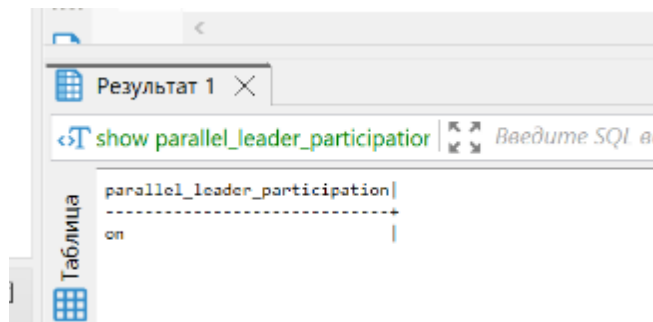


Рисунок 11

Если ведущий процесс становится узким местом, его можно разгрузить (рисунок 12):

```
set parallel_leader_participation=off;
```

The screenshot shows a SQL query window titled 'Статистика 1'. The query entered is 'set parallel_leader_participation=off;'. The result is displayed in a table with the following data:

Name	Value
Updated Rows	0
Query	set parallel_leader_participation=off
Finish time	Tue Nov 01 08:28:27 MSK 2022

Рисунок 12

```
explain select count(*) from bookings;
```

The screenshot shows a SQL query window titled 'Результат 1'. The query entered is 'explain select count(*) from bookings;'. The result is displayed in a table with the following data:

QUERY PLAN
Aggregate (cost=4958.85..4958.86 rows=1 width=8)
-> Seq Scan on bookings (cost=0.00..4301.88 rows=262788 width=0)

Рисунок 13

Общее число строк делится на число запланированных рабочих процессов (рисунок 14)

```
select round(reltuples/2) "rows"  
from pg_class where relname='bookings'
```

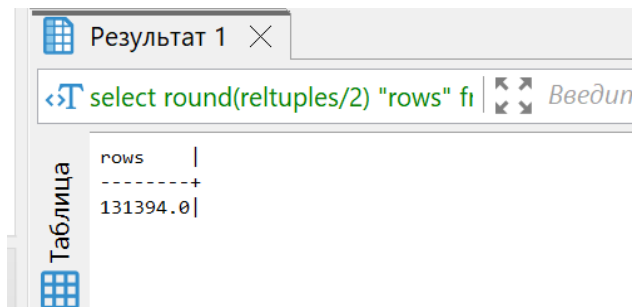


Рисунок 14

Вернем значение по умолчанию для параметра `parallel_leader_participation` (рисунок 15):

`reset parallel_leader_participation;`

Name	Value
Updated Rows	0
Query	reset parallel_leader_participation
Finish time	Tue Nov 01 09:16:56 MSK 2022

Рисунок 15

Для оценки узла Parallel Seq Scan компонента ввода-вывода берется полностью (таблицу все равно придется прочитать страницу за страницей), а ресурсы процессора делятся между процессами (на 2.4 в данном случае).

Лабораторная 12-13. Знакомство с GreenPlum

Цель работы: приобрести навыки работы с GreenPlum и научиться выполнять

Теоретическое обоснование

Некоторое время назад основными системами для аналитической обработки больших объемов данных являлись SMP системы или системы, симметричные с мультипроцессорной архитектурой. Главной особенностью систем с такой архитектурой является наличие общих физических ресурсов, которые разделяются между несколькими процессорами, сравнимой производительности. Память служит, в частности, для передачи сообщений между процессорами. При этом все вычислительные устройства при

обращения к памяти имеют равные права. Именно поэтому SMP архитектура называется симметричной.



Большинство популярных СУБД, таких как Oracle, Sybase и Postgrase, реализованы именно на такой архитектуре.



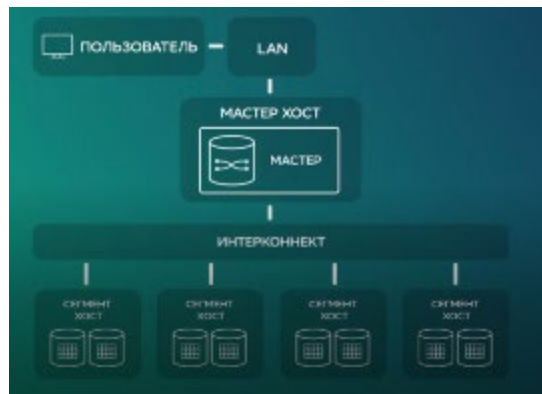
Системы обладают рядом преимуществ, таких как высокая скорость обмена данными между процессорами за счет наличия общей памяти, простота, универсальность обслуживания и относительно невысокая цена.



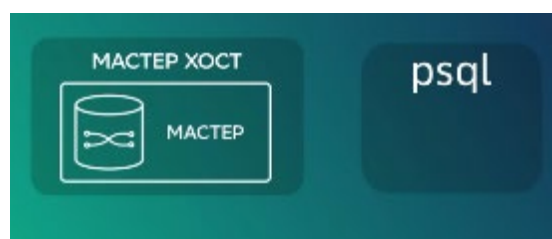
Однако существенным недостатком таких систем является плохая масштабируемость. Каждый раз, когда необходимо увеличить скорость обработки данных, нам необходимо увеличивать мощность сервера за счет увеличения числа процессоров.



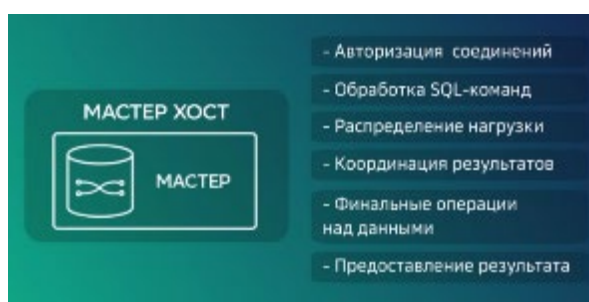
СУБД Greenplum реализована на массивной параллельной архитектуре без совместного использования ресурсов.



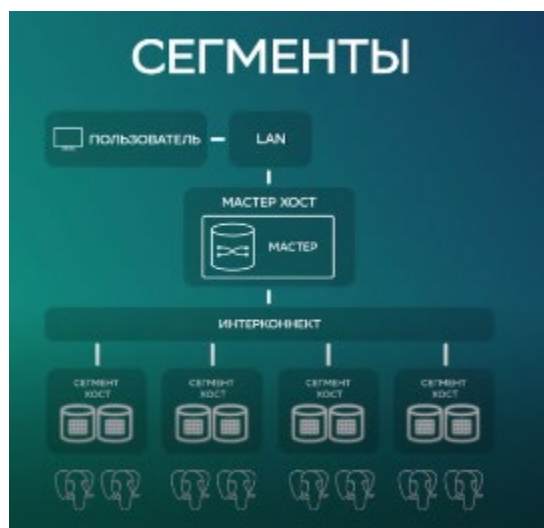
Также мы узнали, что Greenplum представляет из себя множество модифицированных баз данных PostgreSQL, которые расположены на нескольких серверах и взаимодействуют друг с другом, образуя единую цельную базу данных. Логически архитектур Greenplum выглядит следующим образом: мастер. Мастер — это входная точка для Greenplum. Это экземпляры базы данных, Postgres к которому клиенты подключаются и отправляют свои SQL запросы. Мастер координирует работу с другими экземплярами базы данных системы, которые называются сегменты. Пользователи, взаимодействуют с мастером так же, как если бы это была обычная база данных Postgre. Подключение может выполняться при помощи клиентских программ, таких как psql, либо используя стандартные программные интерфейсы приложений, такие как JDBC или ODBC.



На мастере располагается глобальный системный каталог. Это набор системных таблиц, которые содержат метаданные о всех объектах базы данных Greenplum. Важно отметить, что мастер не хранит каких либо пользовательских данных. Все пользовательские данные хранятся на сегментах. Функции мастера заключаются в следующем: мастер выполняет авторизацию клиентских соединений, обрабатывает входящие SQL команды, распределяет нагрузку между сегментами, координирует результаты, возвращаемые с каждым сегментом, выполняет финальные операции над данными, предоставляет конечный результат клиенту.



Сегменты - это отдельные экземпляры базы данных Postgres, которые хранят определенную порцию пользовательских данных и выполняют большую часть обработки запросов.



Когда пользователь подключается к базе данных через мастера и запускает запрос, на каждом сегменте создаются процессы для обработки этого запроса, затем каждый сегмент параллельно обрабатывает свою порцию данных и возвращает финальный результат своей работы на мастер. Сегменты работают на серверах, которые называются сегмент хосты или ноды, на

каждом ноде обычно располагается от 2 до 8 сегментов Greenplum. Количество сегментов фигурирует при первоначальной установке системы и напрямую зависит от профиля рабочих нагрузок системы и технических характеристик сервера, таких как количество и производительность ядер процессора, объема оперативной и постоянной памяти и сетевых интерфейсов. Предполагается, что все ноды системы будут настроены идентично. Ключом к достижению максимальной производительности от базы данных Greenplum является равномерное распределение данных и рабочих нагрузок по большому количеству сегментов с одинаковой производительностью. Это необходимо для того, чтобы все сегменты одновременно начинали работать над задачей и одновременно завершали свою работу. В этом случае ни один из узлов не станет узким горлышко всей системы.

Интерконнект - это сетевой слой архитектуры Greenplum и интерконнект обеспечивает взаимосвязь между мастером, сегментами и сетевой инфраструктурой, в которой развернут кластер.

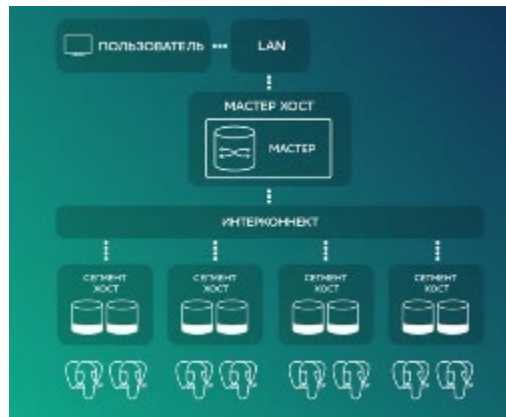


Для обеспечения высокой производительности системы рекомендуемая пропускная способность сети должна быть не менее 10 гигабит. По умолчанию обмен данными происходит по протоколу UDP. Также программное обеспечение Greenplum выполняет дополнительную проверку пакетов поверх.

2. Дистрибьюция

Одним из важнейших факторов быстрой работы системы в Greenplum общее время выполнения запроса измеряется временем выполнения на всех

сегментах, так как обработка на сегментах выполняется в параллель, то максимальная скорость системы ограничивается работой самого медленного сегмента.



Если данные распределены неравномерно, сегменты с большим количеством данных будут выполнять работу дольше, что снижает общую производительность. Таким образом, для повышения производительности необходимо стремиться к равномерному распределению данных по всем сегментам системы.

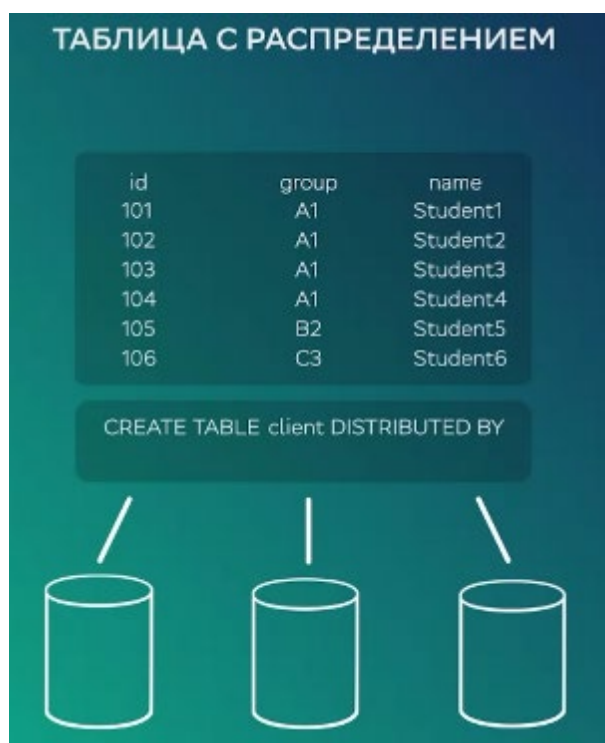
Понимание и правильное использование такой ключевой особенности Greenplum, как распределение данных по всем сегментам с учетом особенностей ваших данных поможет вам добиться наилучшей производительности.

ТАБЛИЦА С РАСПРЕДЕЛЕНИЕМ

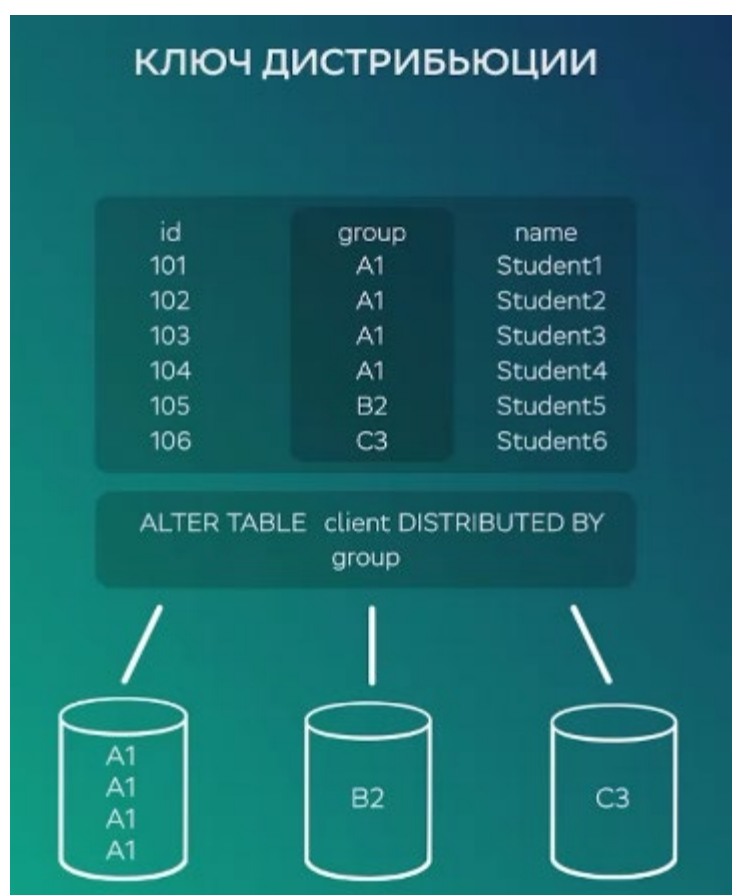
id	group	name
10	A	Student
10	A	Student
10	A	Student
10	A	Student
10	A	Student
10	A	Student
10	A	Student
10	A	Student
10	A	Student
10	A	Student

Равномерное распределение + Учет особенностей данных = Производительность

Даже одна таблица Greenplum хранится не на одном, а на нескольких сегментах. Каждый сегмент хранит уникальную порцию данных. Для каждой таблицы задаётся своя политика дистрибьюции. Способ, согласно которому строки распределяются по сегментам, он задаётся при создании таблицы в команде `CREATE TABLE`, а также может быть изменен впоследствии командой `ALTER TABLE`.

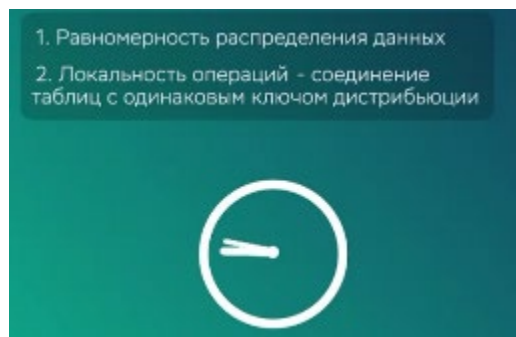


Ключ дистрибьюции это поле или набор полей, по значениям которых определяется, на каком сегменте будет храниться существующая строка таблицы.

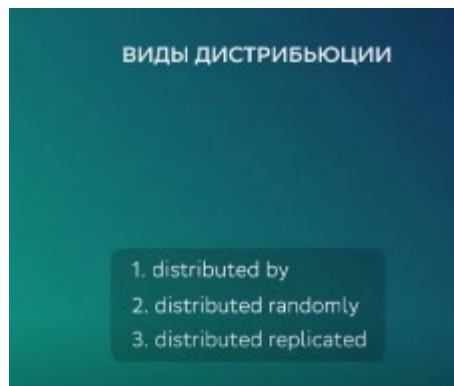


При выборе ключа дистрибьюции необходимо учитывать два основных момента: Ключ должен обеспечивать равномерность распределение данных. Вторая наша цель это добиться локальной сети операций. Под локальными операциями подразумевается ситуация, когда соединяем и таблицы имеют равные ключ дистрибьюции, что позволяет выполнять соединение локально на сегментах без необходимости перераспределять данные, что в свою очередь, даёт существенное ускорение выполнения запроса.

КЛЮЧ ДИСТРИБЬЮЦИИ

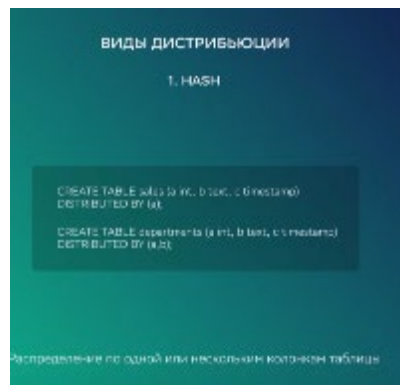


В Greenplum существует три вида дистрибьюции. Это distributed by, distributed randomly, distributed replicated.



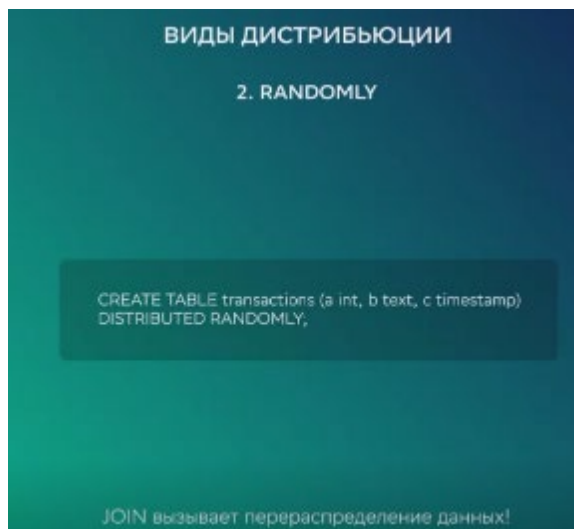
Давайте рассмотрим каждый из них более подробно.

Distributed by - Это дистрибуция по HASH указанных полей. В этом случае и значение колонки или нескольких колонок вычисляется хэш-значение, по которому в дальнейшем определяется, на какой сегмент попадает запись.

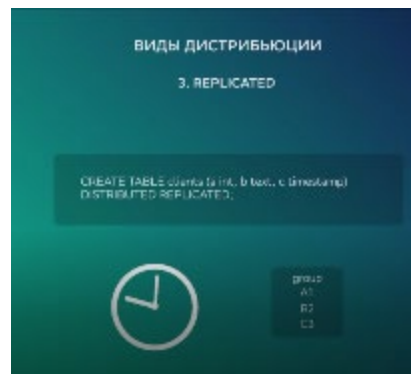


В приведенных примерах вычисления хэш-значения осуществляется по значению одной или двух колонок.

Distributed randomly - Это метод, при котором планировщики сам равномерно размещает данные по всем сегментам. Данный метод лучше всего подходит для таблиц, в которых отсутствует или сложно определить ключ, который обеспечит равномерное распределение данных. Важно учесть, что при таком алгоритме соединение с другими таблицами будет всегда вызывать перераспределение данных, так как строки распределены случайным образом.

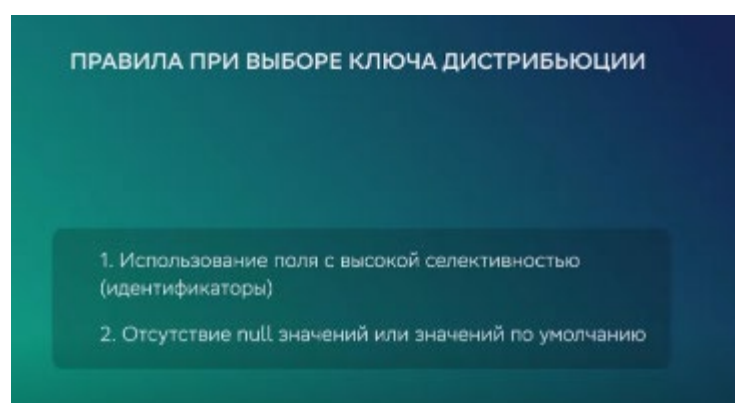


Distributed replicated - В этом случае на каждом сегменте хранится полная копия таблицы. Такой метод подходит для небольших таблиц, например, справочников, которые регулярно участвуют в соединениях с другими таблицами.

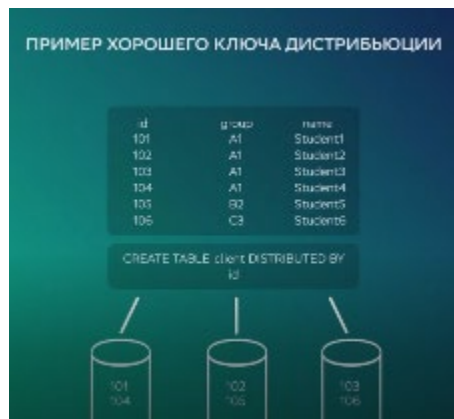


Хранения полной копии таблицы на каждом сегменте позволяет сэкономить время, которое в противном случае было бы потрачено на перераспределение во время выполнения запроса.

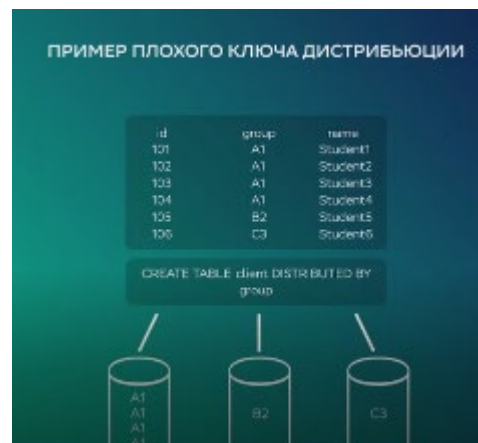
При выборе ключа дистрибьюции необходимо руководствоваться следующими правилами: в качестве ключа дистрибьюции необходимо выбирать полис с большой селективностью, то есть поля с большим количеством уникальных значений. Как правило, под этот критерий подходят поля с идентификаторами. Если данные распределены по ключу с низкой селективностью, то велика вероятность неравномерного распределения данных, так как одинаковое значение будут располагаться на одном сегменте. В ключах не должно быть назначений или значений по умолчанию. Иначе все NULL значения или другие значения по умолчанию будут попадать на один сегмент, что приведет к перекосу в данных.



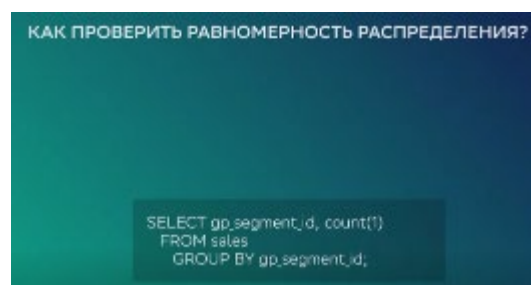
Рассмотрим примеры. В данном примере в качестве ключа дистрибьюции выбрано поле ID. В результате распределение на каждый сегмент падает ровно по две строки. Распределение получается равномерное, а значит, ключ дистрибьюции хороший.



Посмотрим, что изменится. Если в качестве ключа дистрибьюции выбрано поле групп. После распределения мы видим, что на одном из сегментов данных в четыре раза больше, чем на других. Как следствие, и работать этот сегмент будет в четыре раза дольше.



Делаем вывод о том, что ключ выбран не оптимально проверить равномерное распределение строк между сегментами можно, используя служебный столбец `gp_segment_id`, который присутствует в каждой таблице.

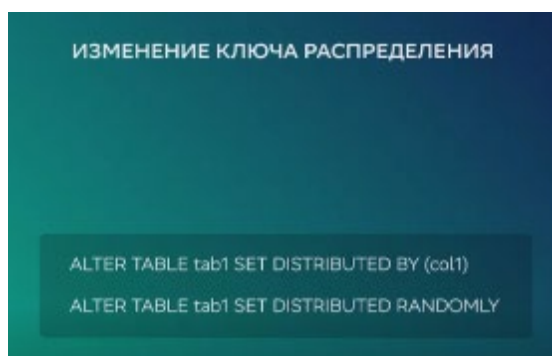


В данном столбце содержится идентификатор сегмента, на котором лежит строка. Пример запроса с группировкой по полю `gp_segment_id` для определения неравномерности распределения строк по сегментам, приведенным на экране.

По результату запроса, видно, что распределение данных таблицы сейчас практически равномерное.

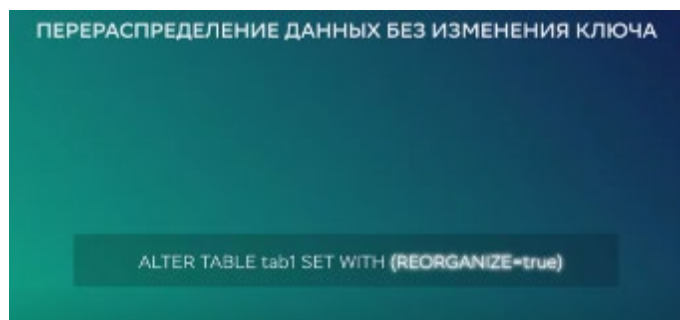


Ключ распределения можно изменить без создания таблицы с помощью предложения ALTER TABLE SET.

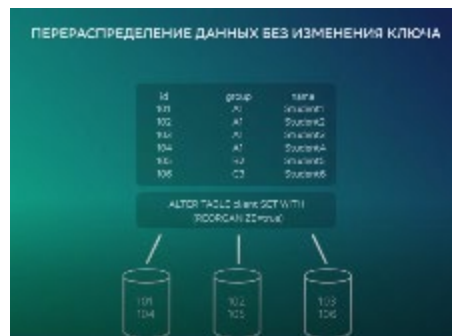


Примеры использования приведены на экране.

Также есть возможность перераспределения данных по прежнему ключа дистрибьюции при помощи опции REORGANIZE=true.



Эта опция будет полезна в двух случаях: при добавлении нового сегмента и для избежания раздутая таблиц. Про раздутие таблицы мы поговорим позже. А при добавлении нового сегмента в кластер возникает необходимость заново перераспределить данные, чтобы избежать перекоса в распределении данных, когда новый сегмент фактически не задействован для хранения данных, а все данные по-прежнему распределены по старым сегментам.



Вот некоторые рекомендации, которые помогут вам правильно распределять данные: явно задавайте способ и ключ распределения при создании таблицы, в том числе для временных таблиц; для небольших таблиц, где нет хороших кандидатов на ключи распределения из одного атрибута используйте RANDOMLY для распределения.

РЕКОМЕНДАЦИИ ПО РАСПРЕДЕЛЕНИЮ ДАННЫХ

Общие рекомендации:

- Явно задавайте способ и ключ распределения при создании таблицы, в том числе для временных таблиц
- Используйте RANDOMLY-распределение для небольших таблиц, где нет хороших кандидатов на ключ распределения из одного атрибута

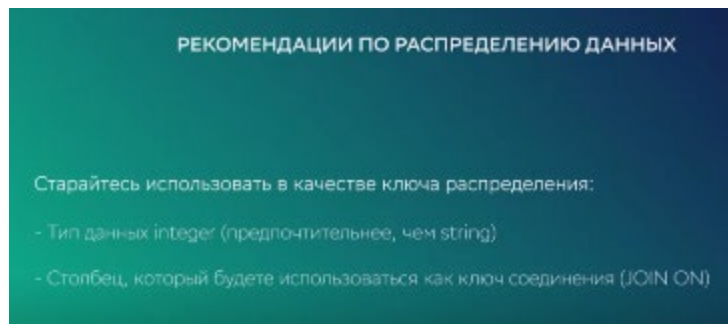
Избегайте использования в качестве ключа распределения атрибутов, которые будете часто указывать в предложении Where или атрибутов с типом данных DATE, TIMESTAMP.

РЕКОМЕНДАЦИИ ПО РАСПРЕДЕЛЕНИЮ ДАННЫХ

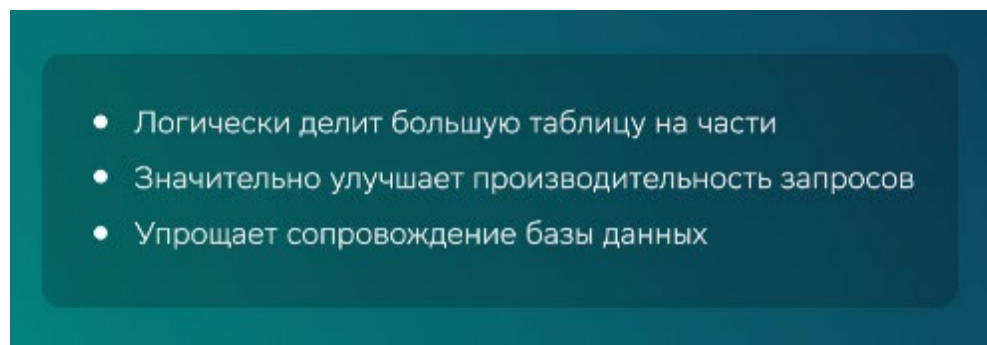
Избегайте использования в качестве ключа распределения:

- Атрибутов, которые будете часто указывать в предложении WHERE
- Атрибутов с типом данных date, timestamp

В качестве ключа распределения, предпочтительно использовать столбцы с типом данных Integer, а также столбцы, которые используются в качестве ключа соединения JOIN ON.



Ещё одна особенность greenplum, улучшающая производительность. Это партиционирование. Партиционирование или секционирование представляет собой логическое разделение таблицы на части по определённому критерию, применение партиционирования на больших таблицах значительно улучшает производительность запросов, а также упрощает сопровождение базы данных благодаря возможности разделять, перемещать и компактно хранить данные в зависимости от их востребованности.

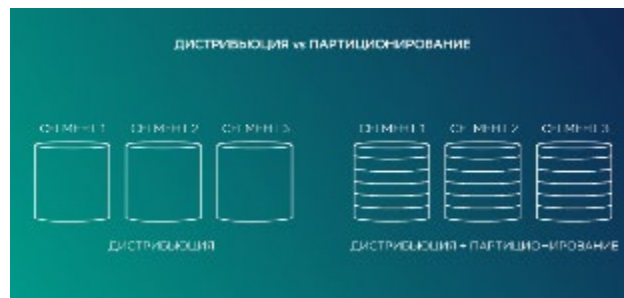


Партиционированная таблица также, как и любые таблицы в greenplum имеют ключ дистрибьюции. Важно различать понятия дистрибьюции и таблиц.

Дистрибьюция - это распределения данных по сегментам с целью равномерно распределить нагрузку по всем узлам кластера.

Партиционирование - это метод оптимизации, при котором на каждом сегменте таблица делится на части с целью увеличения производительности запросов за счёт выборочного сканирования.

Партиционирование распространяется на все сегменты и выполняется на каждом сегменте одинаково. Например, если таблица при создании поделена на 6 порций, то на каждом сегменте кластера будет по 6 партиций, когда таблица партиционирована запрос выполняется гораздо быстрее за счёт того, что на каждом сегменте сканируется только часть таблицы. Дистрибуция обязательно для всех таблиц, то есть все таблицы в greenplum являются распределенными и имеют ключ дистрибуции.



Партиционирование - это лишь опция, которую можно применять к таблице в целях оптимизации в GreenPlum поддерживаются следующие виды партиционирования:

- 1) по диапазону значений RANGE;
- 2) по списку значений;
- 3) лист;
- 4) многоуровневая партиционирование, которое представляет из себя комбинацию, предыдущих типов.



Представим таблицу, содержащую записи о клиентах учебного заведения, например, студентах университета. Согласно заданной политике, дистрибуции, таблица равномерно распределена по сегментам на основе идентификатор студентов. Допустим, что для построения отчетности

периодически требуется анализировать информацию о всех учащихся того или иного возраста. Вместе с тем мы хотим, чтобы система выдавала результаты запросов для разных возрастов как можно быстрее. Эту задачу поможет решить RANCH позиционирование по году рождения. Вид позиционирование задается при создании таблицы с помощью выражения PARTITION BY RANCH.

ПАРТИЦИОНИРОВАНИЕ ПО ДИАПАЗОНУ ЗНАЧЕНИЙ (RANGE)

id	group	name	birthday
101	A1	Student1	20.03.1995
102	A1	Student2	19.01.1999
103	A1	Student3	22.04.1997
104	A1	Student5	12.11.1999
105	C2	Student5	25.04.1992
106	C2	Student6	01.06.2000

```

CREATE TABLE client
(
  id int(11) NOT NULL,
  group varchar(10) NOT NULL,
  name varchar(20) NOT NULL,
  birthday date(10) NOT NULL,
  PRIMARY KEY (id),
  PARTITION BY RANGE (birthday)
(
  PARTITION birthday_1995_01_01_11 INCLUDE
  AND birthday_2000_01_01 EXCLUDE
  EVERY INTERVAL 1 year
);
  
```

ПАРТИЦИОНИРОВАНИЕ ПО ДИАПАЗОНУ ЗНАЧЕНИЙ (RANGE)

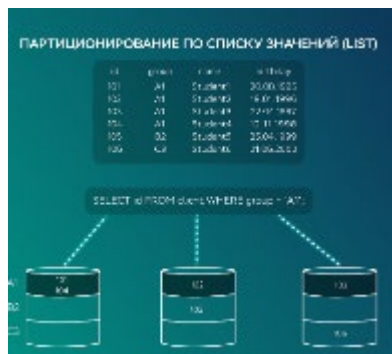
id	group	name	birthday
101	A1	Student1	20.03.1995
102	A1	Student2	19.01.1999
103	A1	Student3	22.04.1997
104	A1	Student5	12.11.1999
105	C2	Student5	25.04.1992
106	C2	Student6	01.06.2000

```

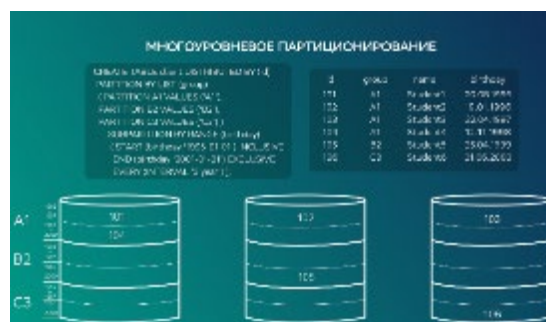
SELECT id FROM client
WHERE birthday <= to_date('01.01.1999', 'DD-MM-YYYY')
and birthday < to_date('01.01.1997', 'DD-MM-YYYY');
  
```

Диапазон значений задан выражением `start and`, а шаг каждой партии выражением `EVERY INTERVAL` интервал после создания таблицы на каждом сегменте будут созданы по шесть партий, хранящие записи по году. Если в запросе к таблице есть условия фильтрации записей по году, то база данных в рамках каждого сегмента сразу обращается к той позиции, которая соответствует нужному году, в связи с чем запрос выполняется намного быстрее.

В случае партиционирования типа лист разделения данных выполняется на основе списка значений.



Например, мы можем применить к той же таблице CLIENT партиционирование по полю группа студентов. Разделение партий по значениям выполняется с помощью выражения PARTITIONAL VALUE. На каждом сегменте создается по три партии акции для групп A1, B-2 и C3. Внутри каждой партии есть студенты с разными годами рождения. Например, запись о студенте четыре хранится в сегменте один в партии A-1. Этот способ позиционирования будет удобен, если требуется часто строить запросы с фильтрации по группе студентов. Кроме того, в одной таблице можно реализовать многоуровневое партиционирование, при котором будут использоваться комбинации обоих типов. В примере партиционирование проводится по группе студентов, а внутри группы по году рождения.



Имейте в виду, что использование многоуровневого партиционирования может привести к созданию избыточного количества партий, что, в свою очередь, может замедлить работу всей базы данных.

При создании партиционированной таблицы можно указывать имя для каждой партий, можно указывать отличающиеся диапазоны для партий, можно указывать отличающиеся опции хранения, ориентации и компрессии.

Можно создать партию по умолчанию, дефолт, в которую будут попадать значения, не удовлетворяющие условиям других партий. Иногда

это удобный способ фильтрации ошибочных записей. При этом невозможно указать разные ключи дистрибуции для партиций в рамках одной таблицы. Нельзя поменять тип партиционирования для существующей таблицы без пересоздания таблицы.

Можно указать:

- Отличающиеся имена
- Отличающиеся ключи
- Разные опции хранения (ориентация, компрессия)

Можно создать:

- Партицию по умолчанию (DEFAULT) для значений, не удовлетворяющих условиям других партиций.

Польза:

- Указать разные ключи дистрибуции для партиций одной таблицы
- Поменять тип партиционирования без пересоздания таблицы

Рассмотрим пример стратегией партиционирования с применением разных опций хранения данных.

ПРИМЕР СТРАТЕГИИ ПАРТИЦИОНИРОВАНИЯ И ХРАНЕНИЯ ДАННЫХ				
Глубина	Ориентация	Партиция	Способ хранения	Компрессия
Сегодня		1 день	HEAP	-
-3 мес		1 мес	AO	Низкая (x2.5) Zstd
-12 мес		1 год	AO	Средняя (x7) Zstd
-24 мес		1 год	AO	Высокая (x15) Zstd
-120 мес		1 год	Внешняя таблица	-

Наиболее свежие данные хранятся без компрессии для того, чтобы можно было быстро выполнять операции обновления. Старые данные хранятся с более сильными алгоритмами сжатия. Например, для данных трехмесячной давности применяется низкая степень сжатия, что обеспечивает хорошую производительность, так как выполняется мало операций, ввода и вывода. Для данных годичной давности применяется колоночная ориентация с более высокой степенью компрессии, что повышает производительность при выборке по подмножеству колонок. Для данных глубиной два года используется колоночная ориентация и максимальная компрессия. При этом производительность ниже, так как при выполнении запросов строки

приходится расжимать. Архивные данные хранятся за пределами базы данных в виде файлов. Напомним, что цель партиционирования состоит в исключении чтения лишних партиций, то есть минимизации количества сканируемых партиций при выполнении запроса. Следовательно, наиболее эффективным будет такое партиционирование, которое максимально приближена к логике выполнения запроса. Последовательность операции при выполнении запроса отражается в плане запроса, который выводится с помощью команды EXPLAIN. Подробное описание запроса будет сказано в заключительной части этого курса.

А сейчас давайте посмотрим на план выполнения запроса в случае таблицы с партиционированием или без него.

ПРИМЕР ХОРОШЕГО КЛЮЧА ПАРТИЦИОНИРОВАНИЯ	
Иллюстрация "хорошего" партиций для table elimination при выполнении запроса	
План запроса EXPLAIN SELECT * FROM sales WHERE date = '01-07-21' AND region = 'usa';	QUERY PLAN БЕЗ партиций Gather Motion 32.1 (dlist, segments: 32) (cost=0.00..5834.89 rows=44906244 width=49) -> Seq Scan on sales (cost=0.00..1679.05 rows=140321 width=49) Filter: ((date = '2020-04-18'::date) AND (region = 'usa'::text)) Optimizer: Parallel Optimizer (CBO/COA)
	QUERY PLAN С партициями Gather Motion 32.1 (dlist, segments: 32) (cost=0.00..6048.26 rows=4689529 width=49) -> Sequence (cost=0.00..853.56 rows=1405611 width=49) -> Partition Selector for table part (dynamic scan (st 1)) (cost=0.00..100.00 rows=4 width=4) Partitions selected: 1 (out of 60) -> Dynamic Seq Scan on salespart (dynamic scan (st 1)) (cost=0.00..863.56 rows=1405611 width=49) Filter: ((date = '2020-04-18'::date) AND (region = 'usa'::text)) Optimizer: Parallel Optimizer (CBO/COA)

В плане запроса для таблицы без партиционирования сканируется вся таблица. Поэтому стоимость выполнения запроса получается высокой. Для таблицы с партиционированием мы видим, что сканируется только одна нужная партиция, в связи с чем стоимость запроса низкая и, как следствие, запрос выполняется намного быстрее.

Рассмотрим примеры синтаксиса.

Для разных партиций можно указывать разные интервалы и опции хранения.

СИНТАКСИС. ИНТЕРВАЛЫ ХРАНЕНИЯ ДЛЯ ПАРТИЦИЙ

```
CREATE TABLE orders
(order_id BIGINT,
order_date TIMESTAMP WITH TIME ZONE,
order_mode VARCHAR(8),
customer_id INTEGER)
DISTRIBUTED BY (customer_id)
PARTITION BY RANGE (order_date)
(start (date '2008-01-01') end (date '2011-01-01') every (interval '1 year')
with (appendonly=true, orientation=column, compresstype=zlib, compresslevel=3),
start (date '2011-01-01') end (date '2013-01-01') every (interval '3 months')
with (appendonly=true, orientation=column, compresstype=zlib, compresslevel=1),
start (date '2013-01-01') end (date '2014-01-01') every (interval '1 months')
with (appendonly=true, orientation=row, compresstype=zstd),
start (date '2014-01-01') end (date '2014-02-01') every (interval '1 day'));
```

Для партии с интервалом год или три месяца задана колоночная ориентация хранения и тип компрессии THE LEAP с разными коэффициентами сжатия. Для партии с интервалом один месяц применяется ориентация по строкам и другой тип компрессии без явного указания коэффициента сжатия. Для партии с интервалом один день используется способ хранения по умолчанию, то есть HEAP без компрессии. Рассмотрим синтаксис изменения партий. Для изменения партий используется команда ALTER TABLE с дополнительными опциями. Для добавления партий используется команда AT PARTITION для обычных партий или AT DEFAULT для партий по умолчанию.

СИНТАКСИС. ПРИМЕРЫ DDL-ОПЕРАЦИЙ

Добавление партии

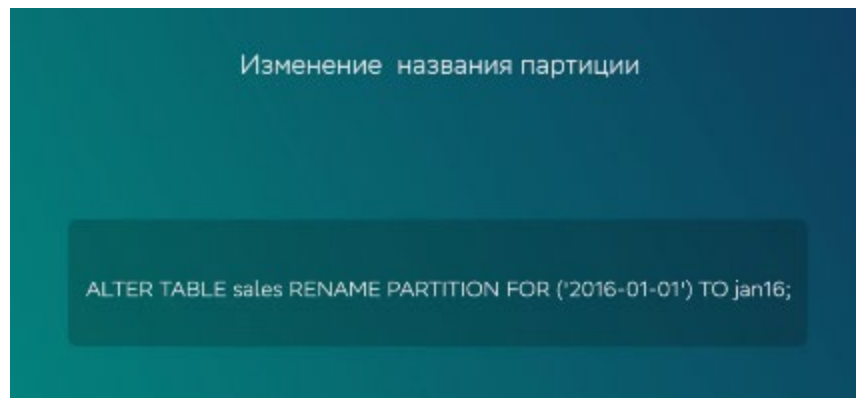
```
ALTER TABLE card_list ADD PARTITION n_a values ('NA');
```

```
ALTER TABLE card_list ADD PARTITION new
START ('2016-01-01'::date) END ('2017-01-01'::date)
WITH (appendonly='true', compresstype=zlib, compresslevel='9');
```

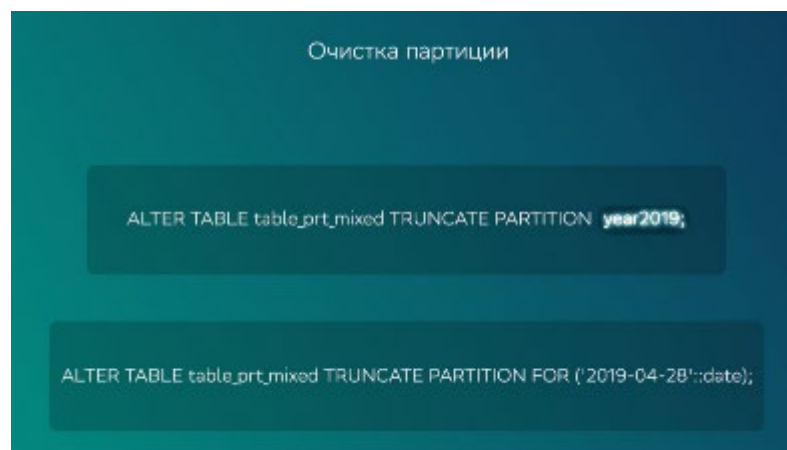
```
ALTER TABLE card_list ADD DEFAULT PARTITION other;
```

Чтобы явно задать имя партий, укажите его при добавлении партии.

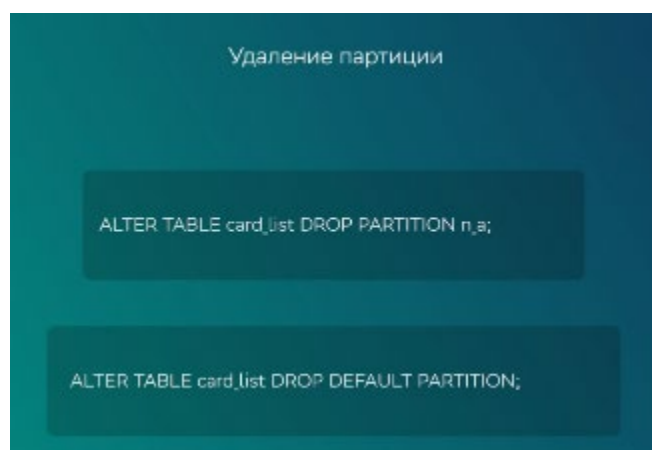
На рисунке показаны примеры для партий по списку, по диапазону значений, и для партий по умолчанию, независимо от типа партиционирования.



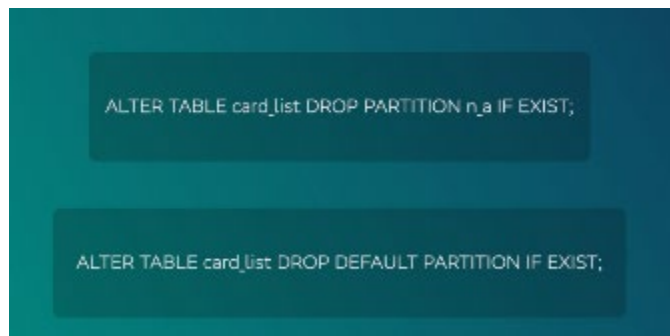
Для изменения названия партиций используется команда RENAME PARTITION. Для очистки партиции применяется команда TRUNCATE PARTITION. На рисунке показано, как очистить партицию с названием EP 2019 или партицию, содержащую дату 28 апреля 2019 года.



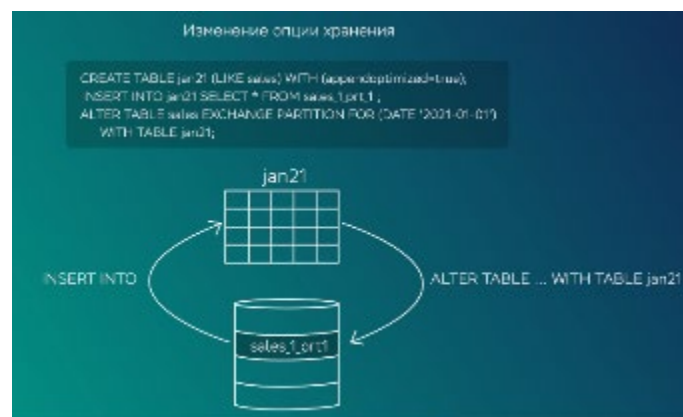
Для удаления партиции используется команда DROP PARTITION. На рисунке приведены примеры удаления обычной и дефолтной партиции.



При желании в тексте команды можно указать опцию IF EXIST, которая поможет избежать сообщения об ошибке и при выполнении команды в случае отсутствия удаляемой партиции.



Для изменения опций и хранения в отдельные партии можно создать таблицу с нужной опцией хранения, вставить в нее данные из этой партии и затем заменить всю прежнюю партию на только что созданную таблицу с помощью команды **ALTER TABLE EXCHANGE PARTITION**.



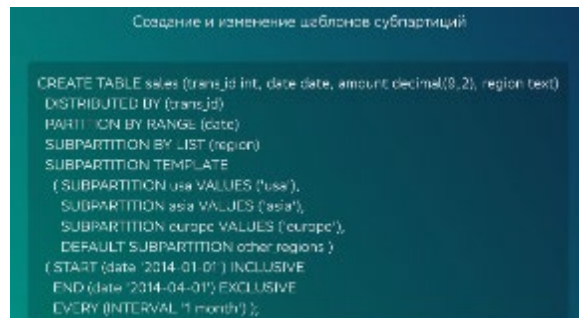
Для разделения одной партии на две части используется команда **ALTER TABLE SPLIT PARTITION**.



В первом примере на рисунке выполняется разделение партии, которые хранятся данные за январь на две партии. В первой будут храниться данные с первого по 15 января, а во второй с 16 го по 31-я. Если в вашей таблице есть дефолт партия, то добавить новую партию в такой таблице можно только путем разделения дефолт партии, как это показано во втором

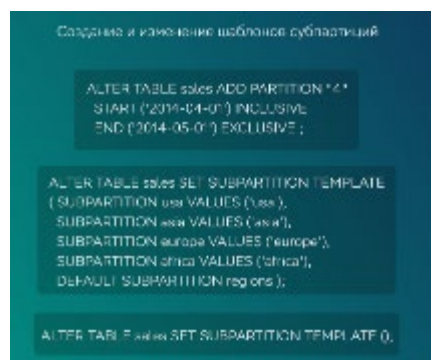
примере. В этом случае в предложении `in to` в качестве второй партиции необходимо указать дефолт партицию.

При создании таблицы возможно определение шаблона, по которому будут автоматически создаваться наборы суб-партиции при добавлении новых позиций.



```
CREATE TABLE sales (trans_id int, date date, amount decimal(8,2), region text)
DISTRIBUTED BY (trans_id)
PARTITION BY RANGE (date)
SUBPARTITION BY LIST (region)
SUBPARTITION TEMPLATE
( SUBPARTITION usa VALUES ('usa'),
  SUBPARTITION asia VALUES ('asia'),
  SUBPARTITION europe VALUES ('europe'),
  DEFAULT SUBPARTITION other regions )
( START (date '2014-01-01') INCLUSIVE
  END (date '2014-04-01') EXCLUSIVE
  EVERY (INTERVAL '1 month') );
```

Например, в следующем скрипт создания шаблон определяется с помощью предложения `SUP-PARTITION TEMPLATE`, в котором мы определяем четыре суб-партиции для разных регионов. Теперь, если мы добавляем в таблицу партицию следующем с скриптом, то эта позиция по шаблону автоматически разделится на четыре суб-партиции и для всех регионов.



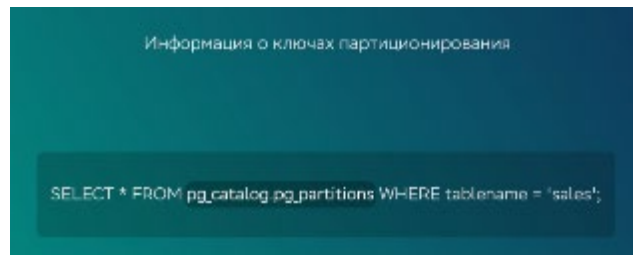
```
ALTER TABLE sales ADD PARTITION "A"
START ('2014-04-01') INCLUSIVE
END ('2014-05-01') EXCLUSIVE ;

ALTER TABLE sales SET SUBPARTITION TEMPLATE
( SUBPARTITION usa VALUES ('usa'),
  SUBPARTITION asia VALUES ('asia'),
  SUBPARTITION europe VALUES ('europe'),
  SUBPARTITION africa VALUES ('africa'),
  DEFAULT SUBPARTITION regions );

ALTER TABLE sales SET SUBPARTITION TEMPLATE ();
```

При желании шаблон партиционирования можно изменить командой `ALTER TABLE SET SUP-PARTITION TEMPLATE`. Следующий скрипт добавит в шаблон партиционирования еще одну партицию для региона Африка, на уже созданные партиции это никак не повлияет. А вновь созданные партиции будут создаваться по новому шаблону. Шаблон партиционирования также можно удалить, выполнив команду `ALTER TABLE SET SUP-PARTITION TEMPLATE` с пустыми скобками.

Информация об используемых партициях содержится в специальных представлениях.



С помощью представления PG_PARTITIONS можно получить подробную информацию об партиционированных таблицах и их иерархии. Представление PG-PARTITIONS COLUMN отображает информацию о колонках, которые используются для партиционирования. Партиционирование следует использовать с учетом следующих рекомендаций:

применяйте партиционирование только на больших таблицах и только если фильтры предполагаемых запросов над таблицей будут исключать некоторые партиции из просмотра, или когда нужно обеспечить так называемое окно данных.

При периодической загрузки в таблицу новых данных и архивации о цене актуальных данных путем компрессии, либо выноса во внешние системы.

Старайтесь выбирать партиционирование по диапазону, а не по списку значений, выполняйте партиционирование на основе часто используемого столбца.

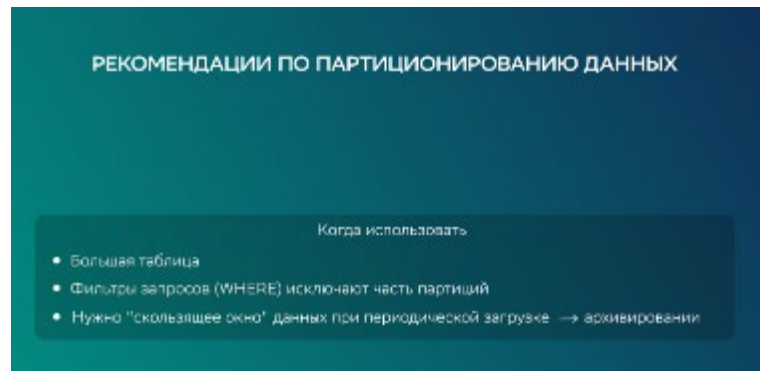
Проверяйте, что запросы выборочно просматривают партиции, исключая ненужные. Для этого используйте план запроса EXPLAIN.

Учитывайте, что запросы, сканирующие все партиции выполняются намного дольше, чем если бы таблица вообще не была бы партиционирована.

Для разных партиций одной таблицы выбирайте оптимальный способ физического хранения, например, построчную или колоночную ориентацию.

Избегайте создания большого количества партиций. Greenplum эффективно обрабатывает операции чтения FULL SKAN, что позволяет уменьшать количество партиций, при этом увеличивать объем данных в

каждой партиции, не используйте дефолт партицию, так как она всегда просматривается при выполнении любого запроса.



Не используйте многоуровневое партиционирование, так как это обычно приводит к созданию чрезмерно большого количества партиций. Никогда не используйте партиции по столбцу, который является ключом дистрибуции. Не создавайте слишком много партиций при колоночной ориентации таблицы, так как это приведет к слишком большому количеству физических файлов на сегменте. Ведь количество файлов — это произведение количества сегментов, столбцов и партиций.

Лабораторная работа № 14. Работа с таблицами в GreenPlum.

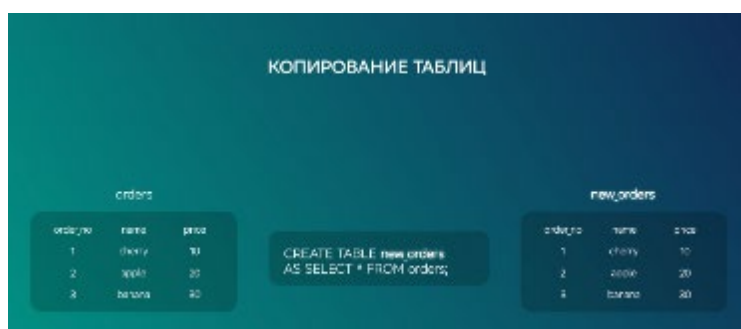
Цель работы: освоить технологию работы с таблицами в GreenPlum

Рассмотрим, как создать таблицу, выясним какие методы хранения данных и способы ориентации для таблиц существуют, а также в каких случаях следует их использовать.

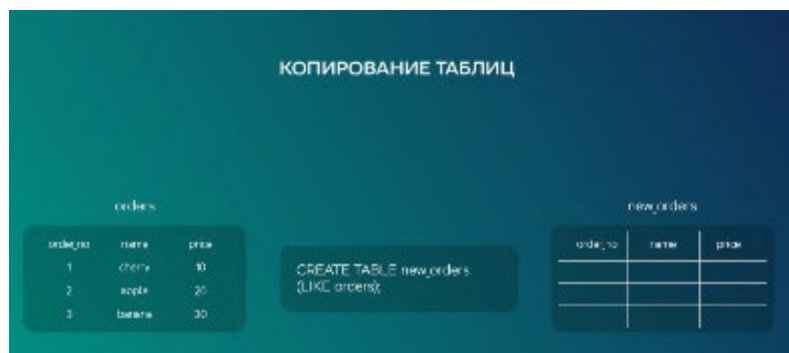
Сама объект таблица, пожалуй, не требует дополнительных пояснений. Для создания таблицы используются команда `CREATE TABLE`, при создании требуется указать ИМЯ новой таблицы и имена типов данных каждого столбца. Создадим таблицу `ЗАКАЗЫ` с номером заказа, его названием и ценой. В результате будет создана пустая таблица, владельцем таблицы будет назначен пользователь от чего имени она была создана.



Таблицы по умолчанию создаются в схеме Public, при этом есть возможность явно задать схему создания, указав ее перед именем таблицы. Имя таблицы должно быть уникальным в текущей схеме не должно сосуществовать других таблиц, внутренних или внешних индексов или представлений с тем же именем. Предположим, что у нас уже есть таблица заказов наполненная данными, мы можем скопировать ее, используя команду `CREATE TABLE SELECTE FROM`, при этом будет создана новая таблица с названием `NEW ORDERS` вместе с данными.



Еще удобная команда `CREATE TABLE LIKE` – она создает новую пустую таблицу и автоматически копирует все имена колонок, типы данных, ограничения и способ распределения.



При использовании обоих способов копирования, новая созданная таблица абсолютно независима от исходной таблицы. Обратите внимание свойства хранения, например, тип партиционирования или метод хранения не копируется. При создании таблиц следует помнить не только то какими данными мы их напомним, но и то как именно мы будем с ними работать, от этого зависит оптимальный метод их хранения. GreenPlum поддерживает два метода хранения данных Heap (куча) и append optimized.

Тип таблиц Heap используется по умолчанию и является стандартной физической структурой со строковой ориентацией, данный тип таблиц унаследован из Postgresql и по своей структуре похож на все привычные нам таблицы.



В каких случаях необходимо выбирать первый тип хранения Heap?

Данный тип лучше всего подходит для таблиц с небольшим объемом данных в большинстве случаев это менее 300 тысяч строк или менее 30 или 100 мегабайт данных. Необходимо учесть, что данные показатели относительны и зависят от размера кластера, также данный тип подходит для таблиц в которых предполагаются частые операции изменения данных update или delete, единичные операции вставки, параллельные операция обновления, удаления и вставки данных, то есть all type нагрузка.

Второй тип это Append Optimized таблицы, данный тип встречается только в GreenPlum, таблицы данного типа обладают особой структурой, являются компактными и обладают рядом специальных возможностей. Для Append Optimized таблиц мы можем выбирать ориентацию строковую или колоночную, в отличии от Heap таблицы Append Optimized таблицы сжимать,

применяя различные алгоритмы компрессии, также Append Optimized таблицы менее ресурсоемкий в обслуживании, например, сбор статистики может быть выполнен инкрементально и таким образом быстрее, вместе с этим у Append Optimized таблиц есть и ограничения, как следует из названия данный тип подходит только для данных, которые редко изменяются, при этом вставка этих данных происходит батчами, заметный рост производительности появляется только на больших таблицах, поэтому использовать данный тип для маленьких таблиц не рекомендуется. Данный тип хранения не поддерживает уникальные индексы и ограничения первичного ключа, такая возможность только для типа Heap.



В каких случаях необходимо выбирать второй тип хранения Append Optimized?

Хранения типа Append Optimized рекомендуется для таблиц, которые содержат много данных, как правило это таблица фактов, также для таблиц, которые создаются однажды и далее используются при выполнении запросов или при вставке данных большими батчами, то есть при all LAP.



Ориентация.

Мы уже упомянули, что для Heap таблиц доступна только строковая ориентация, в то время как для Append Optimized таблиц доступна как строковая, так и колоночная ориентация.



В каких случаях необходимо использовать тот или иной тип?

Построчную ориентацию следует использовать для таблиц над которыми производятся множественные обновления, частые операции вставки, а также частая выборка на временно большого количества столбцов, например, когда в предложении **SELECT** присутствует звездочка(*).



Колоночную ориентацию следует использовать для таблиц, в которых при выборке используется малое количество столбцов, выполняется агрегирующие операции над малым количеством столбцов. Есть единичные столбцы, которые часто обновляются без изменения остальных значений в строке. Использование колоночной ориентации дает прирост в скорости выполнения операций с таблицей, если при этом используется малое количество столбцов относительно общего числа. Также это тип хранения существенно уменьшает количество занимаемого таблицами места, примерно на 30%.

И так вы узнали возможные способы хранения и ориентации данных и получили рекомендации по их выбору в зависимости от ситуации использования и типов запросов, стоит упомянуть, что в GreenPlum допускаются разные виды компрессии данных, при методе **Append Optimized** для строк можно использовать **THE LEAP** и **THE STD**, для колоночного хранения

видов также применима компрессия RLE. При методе хранения HEAP компрессия недоступна.

GreenPlum поддерживает полиморфное хранение данных, часть данных может храниться в строках, а часть в колонках, причем с разными типами компрессии и всё это в рамках одной таблицы – это сделано для оптимального распределения нагрузки и повышения производительности при выполнении аналитических операций, эта тема более подробно рассмотрена в теме партиционирования. Обычно в строках хранятся горячие данные, т.е. постоянно востребованные для оперативного анализа, а в колонках холодные, т.е. архивные.



Компрессия

Ниже приведены описания типов компрессии, доступных в Greenplum.

Виды компрессии

Ориентация	К каким объектам применима	Алгоритм компрессии
Строки	Таблицы	ZLIB, ZSTD
Колонки	Колонки. Таблицы	RLE_TYPE, ZLIB, ZSTD

Описание алгоритмов компрессии

Алгоритм	Уровни сжатия	Описание
ZSTD	от 1 до 9	Позволяет выбрать оптимальное соотношение степени сжатия и скорости обработки данных. Используется чаще всего
RLE_TYPE	от 1 до 4	Более эффективен, чем ZLIB и ZSTD, в случаях, когда в данных есть много повторяющихся значений в большом количестве идущих подряд строк
ZLIB	от 1 до 19	Возможен более высокий уровень сжатия, но при низкой скорости

При выборе алгоритма компрессии необходимо учитывать баланс следующих аспектов:

- степень сжатия (экономия дискового пространства);
- потребление CPU;
- скорость сжатия;
- скорость декомпрессии (сканирования) при чтении данных.

В заключение, упомянем еще несколько базовых понятий, оставшихся за рамками приведенного выше видео.

Схема как объект БД

При просмотре объектов базы данных (как, например, показано здесь), вы сразу заметите, что база данных содержит одну или несколько именованных схем, которые в свою очередь содержат таблицы.

Схема базы данных (Database Schema) — это именованная коллекция таблиц. Схема также может содержать представления, индексы, последовательности, типы данных, операторы и функции. Схемы аналогичны каталогам на уровне операционной системы, за исключением того, что схемы не могут быть вложенными.

Одно и то же имя объекта можно свободно использовать в разных схемах, например, и schema1, и schema2 могут содержать таблицы с именем mytable.

Цели использования схем:

объединение объектов базы данных в логические группы для облегчения управления ими;

работа разных приложений с одной БД без конфликтов имён;

независимая работа несколько пользователей с одной БД.

Констрейнты

Констрейнты (constraint) — это правила, применяемые к столбцам данных таблицы. Они используются, чтобы ограничить типы и значения данных, которые могут храниться в таблице. Это дает возможность привязать возможные значения таблицы к бизнес-правилу: например, указать, что цены товаров могут быть только положительными, даже если тип данных теоретически допускает отрицательные значения. Если пользователь попытается сохранить значение, нарушающее указанные ограничения, возникнет ошибка.

Виды констрейнтов в Greenplum

Значение	Описание
NOT NULL	Данная колонка не может иметь значение NULL
DEFAULT	Обеспечивает значение по умолчанию для колонки в случае, если данные не указаны

UNIQUE	Все значения в данной колонке уникальны. Замечание: доступно только в heap-таблицах
PRIMARY KEY	Уникальный идентификатор каждой записи в таблице БД. Замечание: это комбинация NOT NULL и UNIQUE, доступно только в heap-таблицах.
FOREIGN KEY	Уникальный идентификатор записи в другой таблице БД. Замечание: допускается в синтаксисе, но не работает
CHECK	Гарантирует, что все значения в колонке соответствуют определённому условию
INDEX	Используется для быстрого создания и получения данных из БД

Например,

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

```
CREATE TABLE products
( product_no integer NOT NULL,
  name text NOT NULL,
  price numeric );
```

```
CREATE TABLE products
( product_no integer UNIQUE,
  name text,
  price numeric)  DISTRIBUTED BY (product_no);
```

```
CREATE TABLE products
( product_no integer PRIMARY KEY,
  name text,
  price numeric)  DISTRIBUTED BY (product_no);
```

Примеры синтаксиса создания таблиц

Создание схемы и таблицы в схеме:

```
CREATE SCHEMA myschema;
```

```
CREATE TABLE myschema.company(
  ID INT          NOT NULL,
  NAME VARCHAR (20)  NOT NULL,
  AGE INT          NOT NULL,
  ADDRESS CHAR (25),
  PRIMARY KEY (ID)
);
```

Неар-таблицы:

```
CREATE TABLE foo (a int, b text) DISTRIBUTED BY (a);
```

```
CREATE TABLE foo (a int, b text) WITH (appendonly=false)
DISTRIBUTED BY (a);
```

```
CREATE TABLE foo (a int, b text) WITH (appendoptimized=false)
DISTRIBUTED BY (a);
```

АО-таблицы:

```
CREATE TABLE bar (a int, b text)
  WITH (appendoptimized=true)
  DISTRIBUTED BY (a);
```

```
CREATE TABLE bar (a int, b text)
  WITH (appendonly=true)
  DISTRIBUTED BY (a);
```

```
CREATE TABLE bar (a int, b text)
  WITH (appendoptimized=true, orientation=row)
  DISTRIBUTED BY (a);
```

```
CREATE TABLE bar (a int, b text)
  WITH (appendoptimized=true, orientation=column)
  DISTRIBUTED BY (a);
```

```
CREATE TABLE bar (a int, b text)
  WITH (appendonly=true, orientation=column)
  DISTRIBUTED BY (a);
```

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания

по организации и проведению самостоятельной работы
по дисциплине

«Проектирование систем искусственного интеллекта»

для студентов направления подготовки

09.03.02 «Информационные системы и технологии»

направленность (профиль)

**«Прикладное программирование в интеллектуальных информационных
системах»**

Ставрополь, 2026

1. Общие положения

Самостоятельная работа – планируемая учебная, учебно-исследовательская, научно-исследовательская работа студентов, выполняемая во внеаудиторное (аудиторное) время по заданию и при методическом руководстве преподавателя, но без его непосредственного участия (при частичном непосредственном участии преподавателя, оставляющем ведущую роль за работой студентов).

Самостоятельная работа студентов (СРС) в ВУЗе является важным видом учебной и научной деятельности студента. Самостоятельная работа студентов играет значительную роль в рейтинговой технологии обучения.

К основным видам самостоятельной работы студентов относятся:

- формирование и усвоение содержания конспекта лекций на базе рекомендованной лектором учебной литературы, включая информационные образовательные ресурсы (электронные учебники, электронные библиотеки и др.);
- написание докладов;
- подготовка к семинарам, практическим и лабораторным работам, их оформление;
- составление аннотированного списка статей из соответствующих журналов по отраслям знаний (педагогических, психологических, методических и др.);
- выполнение учебно-исследовательских работ, проектная деятельность;
- подготовка практических разработок и рекомендаций по решению проблемной ситуации;
- выполнение домашних заданий в виде решения отдельных задач, проведения типовых расчетов, расчетно-компьютерных и индивидуальных работ по отдельным разделам содержания дисциплин и т.д.;
- компьютерный текущий самоконтроль и контроль успеваемости на базе электронных обучающих и аттестующих тестов;
- выполнение курсовых работ (проектов) в рамках дисциплин;
- выполнение выпускной квалификационной работы и др.

Методика организации самостоятельной работы студентов зависит от структуры, характера и особенностей изучаемой дисциплины, объема часов на ее изучение, вида заданий для самостоятельной работы студентов, индивидуальных качеств студентов и условий учебной деятельности.

Процесс организации самостоятельной работы студентов включает в себя следующие этапы:

- подготовительный (определение целей, составление программы, подготовка методического обеспечения, подготовка оборудования);
- основной (реализация программы, использование приемов поиска информации, усвоения, переработки, применения, передачи знаний, фиксирование результатов, самоорганизация процесса работы);
- заключительный (оценка значимости и анализ результатов, их систематизация, оценка эффективности программы и приемов работы, выводы о направлениях оптимизации труда).

Самостоятельная работа по дисциплине «Проектирование систем искусственного интеллекта» направлена на формирование следующих **компетенций**:

Код, формулировка компетенции	Код, формулировка индикатора
ОПК-8 Способен применять математические модели, методы и средства проектирования информационных и автоматизированных систем	ОПК-8 _{ид-8.1} – Демонстрирует знания методологий и основных методов математического моделирования
	ОПК-8 _{ид-8.2} – Выбирает математические модели, классифицирует их и определяет условия применения
	ОПК-8 _{ид-8.3} – Осуществляет выбор методов и средств проектирования информационных и

	автоматизированных систем
	ОПК-8 _{ид-8.4} – Применяет инструментальные средства моделирования и проектирования информационных и автоматизированных систем
	ОПК-8 _{ид-8.5} – Проводит моделирование процессов и систем с применением современных инструментальных средств

2. Цель и задачи самостоятельной работы

Ведущая цель организации и осуществления СРС совпадает с целью обучения студента – формирование набора общенаучных, профессиональных и специальных компетенций будущего бакалавра по соответствующему направлению подготовки

При организации СРС важным и необходимым условием становятся формирование умения самостоятельной работы для приобретения знаний, навыков и возможности организации учебной и научной деятельности. Целью самостоятельной работы студентов является овладение фундаментальными знаниями, профессиональными умениями и навыками деятельности по профилю, опытом творческой, исследовательской деятельности. Самостоятельная работа студентов способствует развитию самостоятельности, ответственности и организованности, творческого подхода к решению проблем учебного и профессионального уровня.

Задачами СРС являются:

- систематизация и закрепление полученных теоретических знаний и практических умений студентов;
- углубление и расширение теоретических знаний;
- формирование умений использовать нормативную, правовую, справочную документацию и специальную литературу;
- развитие познавательных способностей и активности студентов: творческой инициативы, самостоятельности, ответственности и организованности;
- формирование самостоятельности мышления, способностей к саморазвитию, самосовершенствованию и самореализации;
- развитие исследовательских умений;
- использование материала, собранного и полученного в ходе самостоятельных занятий на семинарах, на практических и лабораторных занятиях, при написании курсовых и выпускной квалификационной работ, для эффективной подготовки к итоговым зачетам и экзаменам.

3. Технологическая карта самостоятельной работы студента

Коды реализуемых компетенций	Вид деятельности студентов	Средства и технологии оценки	Объем часов, в том числе (акад.)		
			СРС	Контактная работа с преподавателем	Всего
7 семестр					
	Самостоятельное изучение литературы и источников	Собеседование	24	8	32
	Подготовка лабораторным занятиям	Защита ЛР	120	10	130
Итого за 7 семестр			144	18	162
Итого			144	18	162

4. Порядок выполнения самостоятельной работы студентом

4.1. Методические рекомендации по работе с учебной литературой

При работе с книгой необходимо подобрать литературу, научиться правильно ее читать, вести записи. Для подбора литературы в библиотеке используются алфавитный и систематический каталоги.

Важно помнить, что рациональные навыки работы с книгой - это всегда большая экономия времени и сил.

Правильный подбор учебников рекомендуется преподавателем, читающим лекционный курс. Необходимая литература может быть также указана в методических разработках по данному курсу.

Изучая материал по учебнику, следует переходить к следующему вопросу только после правильного уяснения предыдущего, описывая на бумаге все выкладки и вычисления (в том числе те, которые в учебнике опущены или на лекции даны для самостоятельного вывода).

При изучении любой дисциплины большую и важную роль играет самостоятельная индивидуальная работа.

Особое внимание следует обратить на определение основных понятий курса. Студент должен подробно разбирать примеры, которые поясняют такие определения, и уметь строить аналогичные примеры самостоятельно. Нужно добиваться точного представления о том, что изучаешь. Полезно составлять опорные конспекты. При изучении материала по учебнику полезно в тетради (на специально отведенных полях) дополнять конспект лекций. Там же следует отмечать вопросы, выделенные студентом для консультации с преподавателем.

Выводы, полученные в результате изучения, рекомендуется в конспекте выделять, чтобы они при перечитывании записей лучше запоминались.

Опыт показывает, что многим студентам помогает составление листа опорных сигналов, содержащего важнейшие и наиболее часто употребляемые формулы и понятия. Такой лист помогает запомнить формулы, основные положения лекции, а также может служить постоянным справочником для студента.

Чтение научного текста является частью познавательной деятельности. Ее цель – извлечение из текста необходимой информации. От того на сколько осознанна читающим собственная внутренняя установка при обращении к печатному слову (найти нужные сведения, усвоить информацию полностью или частично, критически проанализировать материал и т.п.) во многом зависит эффективность осуществляемого действия.

Выделяют **четыре основные установки в чтении научного текста**:

информационно-поисковый (задача – найти, выделить искомую информацию)

усваивающая (усилия читателя направлены на то, чтобы как можно полнее осознать и запомнить как сами сведения излагаемые автором, так и всю логику его рассуждений)

аналитико-критическая (читатель стремится критически осмыслить материал, проанализировав его, определив свое отношение к нему)

творческая (создает у читателя готовность в том или ином виде – как отправной пункт для своих рассуждений, как образ для действия по аналогии и т.п. – использовать суждения автора, ход его мыслей, результат наблюдения, разработанную методику, дополнить их, подвергнуть новой проверке).

Основные виды систематизированной записи прочитанного:

Аннотирование – предельно краткое связное описание просмотренной или прочитанной книги (статьи), ее содержания, источников, характера и назначения;

Планирование – краткая логическая организация текста, раскрывающая содержание и структуру изучаемого материала;

Тезирование – лаконичное воспроизведение основных утверждений автора без привлечения фактического материала;

Цитирование – дословное выписывание из текста выдержек, извлечений, наиболее существенно отражающих ту или иную мысль автора;

Конспектирование – краткое и последовательное изложение содержания прочитанного.

Конспект – сложный способ изложения содержания книги или статьи в логической последовательности. Конспект аккумулирует в себе предыдущие виды записи, позволяет всесторонне охватить содержание книги, статьи. Поэтому умение составлять план, тезисы, делать выписки и другие записи определяет и технологию составления конспекта.

Методические рекомендации по составлению конспекта:

1. Внимательно прочитайте текст. Уточните в справочной литературе непонятные слова. При записи не забудьте вынести справочные данные на поля конспекта;

2. Выделите главное, составьте план;

3. Кратко сформулируйте основные положения текста, отметьте аргументацию автора;

4. Законспектируйте материал, четко следуя пунктам плана. При конспектировании старайтесь выразить мысль своими словами. Записи следует вести четко, ясно.

5. Грамотно записывайте цитаты. Цитируя, учитывайте лаконичность, значимость мысли.

В тексте конспекта желательно приводить не только тезисные положения, но и их доказательства. При оформлении конспекта необходимо стремиться к емкости каждого предложения. Мысли автора книги следует излагать кратко, заботясь о стиле и выразительности написанного. Число дополнительных элементов конспекта должно быть логически обоснованным, записи должны распределяться в определенной последовательности, отвечающей логической структуре произведения. Для уточнения и дополнения необходимо оставлять поля.

Овладение навыками конспектирования требует от студента целеустремленности, повседневной самостоятельной работы.

4.2. Методические рекомендации по подготовке к практическим и лабораторным занятиям

Для того чтобы практические и лабораторные занятия приносили максимальную пользу, необходимо помнить, что упражнение и решение задач проводятся по вычитанному на лекциях материалу и связаны, как правило, с детальным разбором отдельных вопросов лекционного курса. Следует подчеркнуть, что только после усвоения лекционного материала с определенной точки зрения (а именно с той, с которой он излагается на лекциях) он будет закрепляться на практических занятиях как в результате обсуждения и анализа лекционного материала, так и с помощью решения проблемных ситуаций, задач. При этих условиях студент не только хорошо усвоит материал, но и научится применять его на практике, а также получит дополнительный стимул (и это очень важно) для активной проработки лекции.

При самостоятельном решении задач нужно обосновывать каждый этап решения, исходя из теоретических положений курса. Если студент видит несколько путей решения проблемы (задачи), то нужно сравнить их и выбрать самый рациональный. Полезно до начала вычислений составить краткий план решения проблемы (задачи). Решение проблемных задач или примеров следует излагать подробно, вычисления располагать в строгом порядке, отделяя вспомогательные вычисления от основных. Решения при необходимости нужно сопровождать комментариями, схемами, чертежами и рисунками.

Следует помнить, что решение каждой учебной задачи должно доводиться до окончательного логического ответа, которого требует условие, и по возможности с выводом. Полученный ответ следует проверить способами, вытекающими из существа данной задачи. Полезно также (если возможно) решать несколькими способами и сравнить полученные результаты. Решение задач данного типа нужно продолжать до приобретения твердых навыков в их решении.

4.3. Методические рекомендации по самопроверке знаний

После изучения определенной темы по записям в конспекте и учебнику, а также решения достаточного количества соответствующих задач на практических занятиях и самостоятельно студенту рекомендуется, провести самопроверку усвоенных знаний, ответив на контрольные вопросы по изученной теме.

В случае необходимости нужно еще раз внимательно разобраться в материале.

Иногда недостаточность усвоения того или иного вопроса выясняется только при изучении дальнейшего материала. В этом случае надо вернуться назад и повторить плохо усвоенный материал. Важный критерий усвоения теоретического материала - умение решать задачи или пройти тестирование по пройденному материалу. Однако следует помнить, что правильное решение задачи может получиться в результате применения механически заученных формул без понимания сущности теоретических положений.

4.4. Методические рекомендации по написанию научных текстов (докладов, докладов, эссе, научных статей и т.д.)

Перед тем, как приступить к написанию научного текста, важно разобраться, какова истинная цель вашего научного текста - это поможет вам разумно распределить свои силы и время.

Во-первых, сначала нужно определиться с идеей научного текста, а для этого необходимо научиться либо относиться к разным явлениям и фактам несколько критически (своя идея – как иная точка зрения), либо научиться увлекаться какими-то известными идеями, которые нуждаются в доработке (идея – как оптимистическая позиция и направленность на дальнейшее совершенствование уже известного). Во-вторых, научиться организовывать свое время, ведь, как известно, свободное (от всяких глупостей) время –

важнейшее условие настоящего творчества, для него наконец-то появляется время. Иногда именно на организацию такого времени уходит немалая часть сил и талантов.

Писать следует ясно и понятно, стараясь основные положения формулировать четко и недвусмысленно (чтобы и самому понятно было), а также стремясь структурировать свой текст. Каждый раз надо представлять, что ваш текст будет кто-то читать и ему захочется сориентироваться в нем, быстро находить ответы на интересующие вопросы (заодно представьте себя на месте такого человека). Понятно, что работа, написанная «сплошным текстом» (без заголовков, без выделения крупным шрифтом наиболее важных мест и т. п.), у культурного читателя должна вызывать брезгливость и даже жалость к автору (исключения составляют некоторые древние тексты, когда и жанр был иной и к текстам относились иначе, да и самих текстов было гораздо меньше – не то, что в эпоху «информационного взрыва» и соответствующего «информационного мусора»).

Объем текста и различные оформительские требования во многом зависят от принятых в конкретном учебном заведении порядков.

Доклад – это самостоятельное исследование студентом определенной проблемы, комплекса взаимосвязанных вопросов.

Доклад не должна составляться из фрагментов статей, монографий, пособий. Кроме простого изложения фактов и цитат, в доклад е должно проявляться авторское видение проблемы и ее решения.

Рассмотрим основные этапы подготовки
а студентом.

Выполнение доклада начинается с выбора темы.

Затем студент приходит на первую консультацию к руководителю, которая предусматривает:

- обсуждение цели и задач работы, основных моментов избранной темы;
- консультирование по вопросам подбора литературы;
- составление предварительного плана.

Следующим этапом является работа с литературой. Необходимая литература подбирается студентом самостоятельно.

После подбора литературы целесообразно сделать рабочий вариант плана работы. В нем нужно выделить основные вопросы темы и параграфы, раскрывающие их содержание.

Составленный список литературы и предварительный вариант плана уточняются, согласуются на очередной консультации с руководителем.

Затем начинается следующий этап работы – изучение литературы. Только внимательно читая и конспектируя литературу, можно разобраться в основных вопросах темы и подготовиться к самостоятельному (авторскому) изложению содержания доклада. Конспектируя первоисточники, необходимо отразить основную идею автора и его позицию по исследуемому вопросу, выявить проблемы и наметить задачи для дальнейшего изучения данных проблем.

Систематизация и анализ изученной литературы по проблеме исследования позволяют студенту написать работу.

Рабочий вариант текста доклада предоставляется руководителю на проверку. На основе рабочего варианта текста руководитель вместе со студентом обсуждает возможности доработки текста, его оформление. После доработки доклад сдается на кафедру для его оценивания руководителем.

Требования к написанию доклада

Написание 1 доклада является обязательным условием выполнения плана СРС по любой дисциплине профессионального цикла.

Тема доклада может быть выбрана студентом из предложенных в рабочей программе или фонде оценочных средств дисциплины, либо определена самостоятельно, исходя из интересов студента (в рамках изучаемой дисциплины). Выбранную тему необходимо согласовать с преподавателем.

Доклад должен быть написан научным языком.

Объем доклада должен составлять 20-25 стр.

Структура доклада:

- Введение (не более 3-4 страниц). Во введении необходимо обосновать выбор темы, ее актуальность, очертить область исследования, объект исследования, основные цели и задачи исследования.

- Основная часть состоит из 2-3 разделов. В них раскрывается суть исследуемой проблемы, проводится обзор мировой литературы и источников Интернет по предмету исследования, в котором дается характеристика степени разработанности проблемы и авторская аналитическая оценка основных теоретических подходов к ее решению. Изложение материала не должно ограничиваться лишь описательным подходом к раскрытию выбранной темы. Оно также должно содержать собственное видение рассматриваемой проблемы и изложение собственной точки зрения на возможные пути ее решения.

- Заключение (1-2 страницы). В заключении кратко излагаются достигнутые при изучении проблемы цели, перспективы развития исследуемого вопроса

- Список использованной литературы (не меньше 10 источников), в алфавитном порядке, оформленный в соответствии с принятыми правилами. В список использованной литературы рекомендуется включать работы отечественных и зарубежных авторов, в том числе статьи, опубликованные в научных журналах в течение последних 3-х лет и ссылки на ресурсы сети Интернет.

- Приложение (при необходимости).

Требования к оформлению:

- текст с одной стороны листа;
- шрифт Times New Roman;
- кегль шрифта 14;
- межстрочное расстояние 1,5;
- поля: сверху 2,5 см, снизу – 2,5 см, слева - 3 см, справа 1,5 см;
- доклад должен быть представлен в сброшюрованном виде.

Порядок защиты доклада:

Защита доклада проводится на практических занятиях, после окончания работы студента над ним и исправления всех недочетов, выявленных преподавателем в ходе консультаций. На защиту доклада отводится 5-7 минут времени, в ходе которого студент должен показать свободное владение материалом по заявленной теме. При защите доклада приветствуется использование мультимедиа-презентации.

Оценка доклада

Доклад оценивается по следующим критериям:

- соблюдение требований к его оформлению;
- необходимость и достаточность для раскрытия темы приведенной в тексте доклада информации;
- умение студента свободно излагать основные идеи, отраженные в докладе;
- способность студента понять суть задаваемых преподавателем и сокурсниками вопросов и сформулировать точные ответы на них.

Критерии оценки:

Оценка «отлично» выставляется студенту, если в докладе студент исчерпывающе, последовательно, четко и логически стройно излагает материал; свободно справляется с задачами, вопросами и другими видами применения знаний; использует для написания доклада современные научные материалы; анализирует полученную информацию; проявляет самостоятельность при написании доклада.

Оценка «хорошо» выставляется студенту, если качество выполнения доклада достаточно высокое. Студент твердо знает материал, грамотно и по существу излагает его, не допуская существенных неточностей в ответе на вопросы по теме доклада.

Оценка «удовлетворительно» выставляется студенту, если материал доклада излагается частично, но пробелы не носят существенного характера, студент допускает неточности и ошибки при защите доклада, дает недостаточно правильные формулировки, наблюдаются нарушения логической последовательности в изложении материала.

Оценка «неудовлетворительно» выставляется студенту, если он не подготовил доклад или допустил существенные ошибки. Студент неуверенно излагает материал доклада, не отвечает на вопросы преподавателя.

4.5. Методические рекомендации по выполнению исследовательских проектов

Исследовательская проектная работа – это групповая работа, для выполнения которой необходим выбор и приложение научной методики к поставленной задаче, получение собственного теоретического или экспериментального материала, на основании которого необходимо провести анализ и сделать выводы об исследуемом явлении. Выполнение проекта – это всегда коллективная, творческая практическая работа, предназначенная для получения определенного продукта или научно-технического результата. Такая работа подразумевает четкое, однозначное формирование поставленной задачи, определение сроков выполнения намеченного, определение требований к разрабатываемому объекту.

Выполнение 1 группового проекта является обязательным условием выполнения самостоятельной работы по любой дисциплине профессионального цикла. Тема проектного задания может быть выбрана студентом из предложенных в рабочей программе или фонде оценочных средств дисциплины, либо определена самостоятельно, исходя из интересов студента (в рамках изучаемой дисциплины). Выбранную тему необходимо согласовать с преподавателем.

Требования по выполнению и оформлению проекта

При выполнении проекта приветствуется работа в группе (2-3 человека). Проект – это исследовательская работа, в ходе которой студенты должны продемонстрировать владение навыками научного исследования, умения проводить анализ, обобщать информацию, делать выводы, предлагать свои решения проблемы, рассматриваемой в проекте.

При подготовке материалов проекта студенты должны продемонстрировать владение современными методами компьютерной обработки данных.

Критерии оценки работы участника проекта.

Для каждого из участников проекта оцениваются:

- профессиональные теоретические знания в соответствующей области;
- умение работать со справочной и научной литературой, осуществлять поиск необходимой информации в Интернет;
- умение работать с техническими средствами;
- умение пользоваться соответствующими выполняемому проекту информационными технологиями;
- умение готовить материалы проекта для презентации: составлять и редактировать тексты, формировать презентацию проекта;
- умение работать в команде;
- умение публично представлять результаты собственной деятельности;
- коммуникабельность, инициативность, творческие способности.

Критерии выставления оценки участникам проекта

Оценка	Профессиональные компетенции	Компетенции, связанные с использованием соответствующих выполняемому проекту технических средств и информационных технологий	Иные универсальные компетенции (коммуникабельность, инициативность, умение работать в «команде», управленческие навыки и т.д.)	Отчетность
«Отлично»	Работа выполнена на высоком профессиональном уровне. Представленный материал в основном фактически верен, допускаются негрубые фактические неточности. Студент свободно отвечает на вопросы, связанные с проектом.	Технические средства и информационные технологии освоены и использованы для реализации проекта полностью	Студент проявил инициативу, творческий подход, способность к выполнению сложных заданий, навыки работы в коллективе, организационные способности.	Проект представлен полностью и в срок.
«Хорошо»	Работа выполнена на достаточно высоком профессиональном уровне. Допущено до 4–5 фактических ошибок. Студент отвечает на вопросы, связанные с проектом, но недостаточно полно.	Обнаруживаются некоторые ошибки в использовании соответствующих технических средств и информационных технологий	Студент достаточно полно, но без инициативы и творческих находок выполнил возложенные на него задачи.	Проект представлен достаточно полно и в срок, но с некоторыми недоработками.
«Удовлетворительно»	Уровень недостаточно высок. Допущено до 8 фактических ошибок. Студент может ответить лишь на некоторые из заданных вопросов, связанных с проектом.	Обнаруживает недостаточное владение навыками работы с техническими средствами и соответствующим и информационным и технологиями	Студент выполнил большую часть возложенной на него работы.	Проект сдан со значительным опозданием (более недели) и не полностью
«Неудовлетворительно»	Работа не выполнена или выполнена на низком уровне. Допущено более 8 фактических ошибок. Ответы на связанные с проектом вопросы	Навыков работы с техническими средствами нет, информационные технологии не освоены	Студент практически не работал, не выполнил свои задачи или выполнил лишь отдельные не существенные	Проект не сдан.

Оценка	Профессиональные компетенции	Компетенции, связанные с использованием соответствующих выполняемому проекту технических средств и информационных технологий	Иные универсальные компетенции (коммуникабельность, инициативность, умение работать в «команде», управленческие навыки и т.д.)	Отчетность
	обнаруживают непонимание предмета и отсутствие ориентации в материале проекта.		поручения в групповом проекте.	

Студенты должны: защитить проект в режиме презентации, предъявить файлы выполненного проекта, уметь рассказать о технологиях, использованных ими при выполнении проекта, дать оценку работы каждого члена группы (если проект групповой).

4.6. Методические рекомендации по подготовке к экзаменам и зачетам

Изучение многих общепрофессиональных и специальных дисциплин завершается экзаменом. Подготовка к экзамену способствует закреплению, углублению и обобщению знаний, получаемых, в процессе обучения, а также применению их к решению практических задач. Готовясь к экзамену, студент ликвидирует имеющиеся пробелы в знаниях, углубляет, систематизирует и упорядочивает свои знания. На экзамене студент демонстрирует то, что он приобрел в процессе обучения по конкретной учебной дисциплине.

Экзаменационная сессия - это серия экзаменов, установленных учебным планом. Между экзаменами интервал 3-4 дня. Не следует думать, что 3-4 дня достаточно для успешной подготовки к экзаменам.

В эти 3-4 дня нужно систематизировать уже имеющиеся знания. На консультации перед экзаменом студентов познакомят с основными требованиями, ответят на возникшие у них вопросы. Поэтому посещение консультаций обязательно.

Требования к организации подготовки к экзаменам те же, что и при занятиях в течение семестра, но соблюдаться они должны более строго. Во-первых, очень важно соблюдение режима дня; сон не менее 8 часов в сутки, занятия заканчиваются не позднее, чем за 2-3 часа до сна. Оптимальное время занятий - утренние и дневные часы. В перерывах между занятиями рекомендуются прогулки на свежем воздухе, неустойчивые занятия спортом. Во-вторых, наличие хороших собственных конспектов лекций. Даже в том случае, если была пропущена какая-либо лекция, необходимо во время ее восстановить (переписать ее на кафедре), обдумать, снять возникшие вопросы для того, чтобы запоминание материала было осознанным. В-третьих, при подготовке к экзаменам у студента должен быть хороший учебник или конспект литературы, прочитанной по указанию преподавателя в течение семестра. Здесь можно эффективно использовать листы опорных сигналов.

Вначале следует просмотреть весь материал по сдаваемой дисциплине, отметить для себя трудные вопросы. Обязательно в них разобраться. В заключение еще раз целесообразно повторить основные положения, используя при этом листы опорных сигналов.

Систематическая подготовка к занятиям в течение семестра позволит использовать время экзаменационной сессии для систематизации знаний.

Контроль самостоятельной работы студентов

Контроль самостоятельной работы проводится преподавателем в аудитории.

Предусмотрены следующие виды контроля: собеседование, оценка доклада, оценка презентации, оценка участия в круглом столе, оценка выполнения проекта.

Подробные критерии оценивания компетенций приведены в Фонде оценочных средств для проведения текущей и промежуточной аттестации.

Список литературы для выполнения СРС

Перечень основной литературы:

1. Башмакова,, Е. И. Информатика и информационные технологии. Технология работы в MS WORD 2016 : учебное пособие / Е. И. Башмакова. - Информатика и информационные технологии. Технология работы в MS WORD 2016,Весь срок охраны авторского права. - Электрон. дан. (1 файл). - Москва : Ай Пи Ар Медиа, 2020. - 90 с. - электронный. - Книга находится в премиум-версии ЭБС IPR BOOKS. - ISBN 978-5-4497-0515-0, экземпляров неограничено
2. Башмакова,, Е. И. Информатика и информационные технологии. Умный Excel 2016: библиотека функций : учебное пособие / Е. И. Башмакова. - Информатика и информационные технологии. Умный Excel 2016: библиотека функций,Весь срок охраны авторского права. - Электрон. дан. (1 файл). - Москва : Ай Пи Ар Медиа, 2020. - 109 с. - электронный. - Книга находится в премиум-версии ЭБС IPR BOOKS. - ISBN 978-5-4497-0516-7, экземпляров неограничено
3. Мандра,, А. Г. Информатика и информационные технологии : лабораторный практикум / А. Г. Мандра, А. В. Попов, А. И. Дьяконов. - Информатика и информационные технологии,2026-09-20. - Электрон. дан. (1 файл). - Самара : Самарский государственный технический университет, ЭБС АСВ, 2020. - 64 с. - электронный. - Книга находится в премиум-версии ЭБС IPR BOOKS. - ISBN 2227-8397, экземпляров неограничено
4. Цветкова, А. В. Информатика и информационные технологии Электронный ресурс : Учебное пособие для СПО / А. В. Цветкова. - Информатика и информационные технологии,2020-08-30. - Саратов : Научная книга, 2019. - 190 с. - Книга находится в премиум-версии ЭБС IPR BOOKS. - ISBN 978-5-9758-1891-1, экземпляров неограничено

Перечень дополнительной литературы:

1. Современные информационные технологии Электронный ресурс : Сборник трудов по материалам 3-й межвузовской научно-технической конференции с международным участием 29 сентября 2017 г. / В. И. Воловач [и др.] ; ред. В. М. Артюшенко. - Королёв : Научный консультант, МГОТУ, 2017. - 191 с. - Книга находится в премиум-версии ЭБС IPR BOOKS. - ISBN 978-5-9500999-7-7, экземпляров неограниченно
2. Современные мультимедийные информационные технологии Электронный ресурс : учебное пособие / С.С. Мытько / Д.А. Репечко / А.П. Алексеев / А.Р. Ванютин / И.А. Королькова. - Современные мультимедийные информационные технологии,2019-05-25. - Москва : СОЛОН-ПРЕСС, 2017. - 108 с. - Книга находится в базовой версии ЭБС IPRbooks. - ISBN 978-5-91359-219-4, экземпляров неограниченно
3. Современные информационные технологии Электронный ресурс : учебное пособие / С.С. Мытько / Д.А. Репечко / И.А. Королькова / А.Р. Ванютин / А.П. Алексеев ; ред. А.П. Алексеев. - Самара : Поволжский государственный университет телекоммуникаций и информатики, 2019. - 101 с. - Книга находится в базовой версии ЭБС

IPRbooks., экземпляров неограниченно

4. Адлер, Ю.П. Статистическое управление процессами. «Большие данные» Электронный ресурс : учебное пособие / Е.А. Черных / Ю.П. Адлер. - Статистическое управление процессами. «Большие данные», 2019-09-01. - Москва : Издательский Дом МИСиС, 2016. - 52 с. - Книга находится в базовой версии ЭБС IPRbooks. - ISBN 978-5-87623-969-3, экземпляров неограниченно

Методическая литература:

1. Методические рекомендации для самостоятельной работы студентов по дисциплине «Проектирование систем искусственного интеллекта»
2. Методические указания к лабораторным работам по дисциплине «Проектирование систем искусственного интеллекта»

Интернет-ресурсы:

1. <http://el.ncfu.ru/> – система управления обучением ФГАОУ ВО СКФУ. Дистанционная поддержка дисциплины «Проектирование систем искусственного интеллекта».
2. <http://www.un.org> - Сайт ООН Информационно-коммуникационные технологии.
3. <http://www.intuit.ru> – Интернет-Университет Компьютерных технологий.