

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 «**Инжиниринг данных**»
для студентов направления 09.03.02 Информационные системы и
технологии
направленность (профиль) «Прикладное программирование и
интернет-технологии»

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.

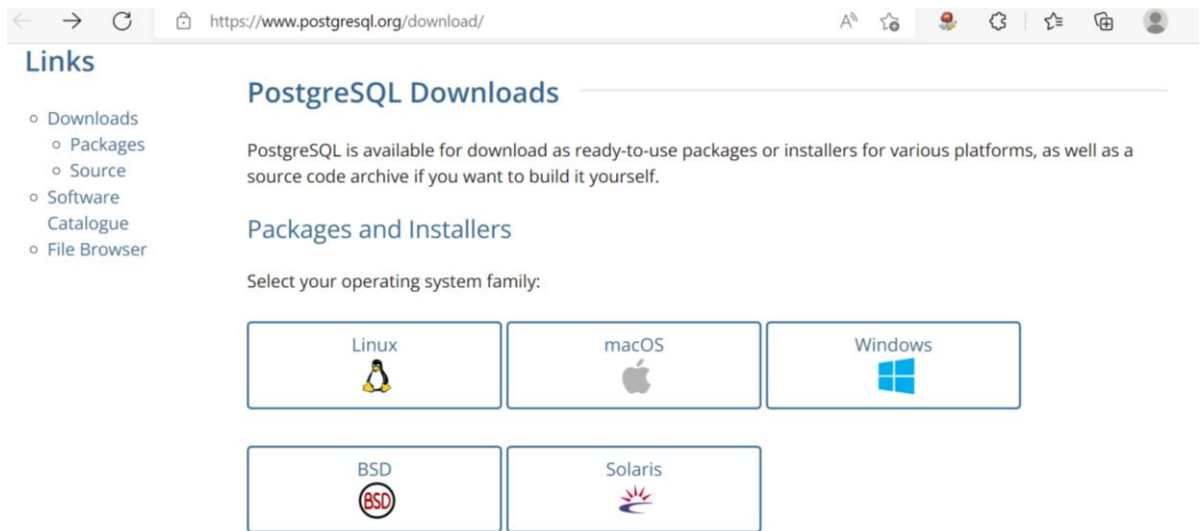
Кроме этого есть и другие требования:

- 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.

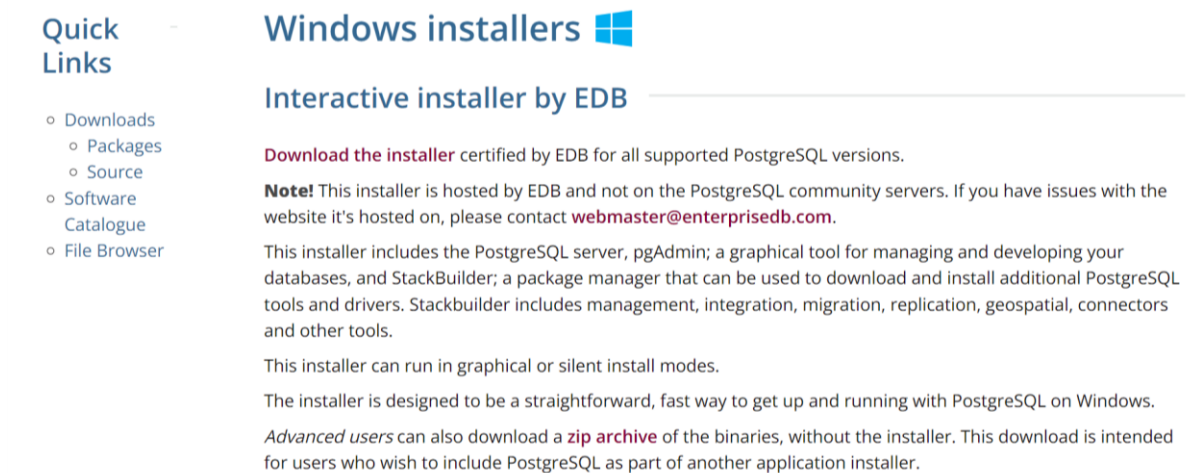
Ссылка для скачивания

[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):

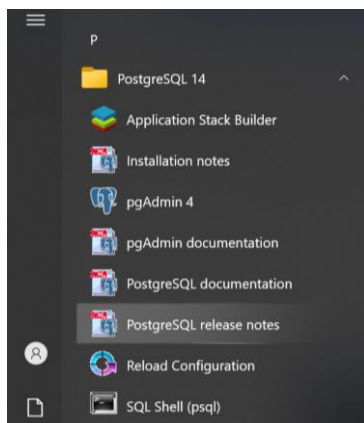
- Главный серверный процесс, управляющий файлами баз данных, принимающий подключения клиентских приложений и выполняющий различные запросы клиентов к базам данных. Эта программа сервера БД называется postgres.
- Клиентское приложение пользователя, желающее выполнять операции в базе данных. Клиентские 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.

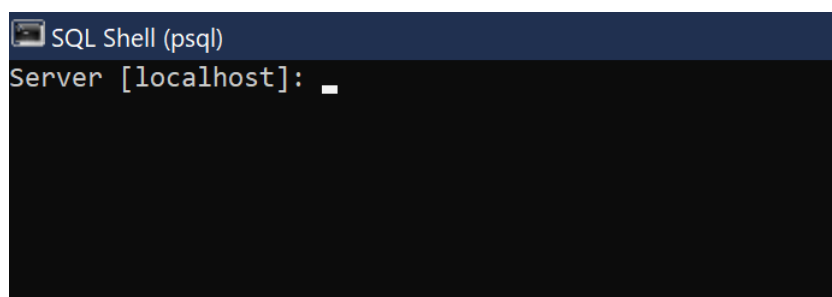
Сервер PostgreSQL может обслуживать одновременно несколько подключений клиентов. Для этого он запускает («порождает») отдельный процесс для каждого подключения. Можно сказать, что клиент и серверный процесс общаются, не затрагивая главный процесс postgres. Таким образом, главный серверный процесс всегда 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.

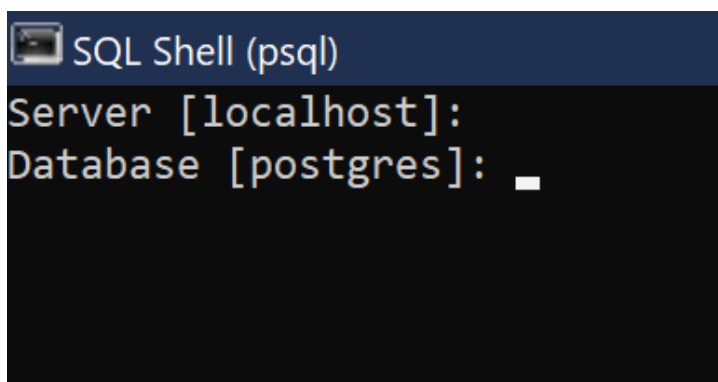


После запуска SQL Shell нас встречает надпись Server [local|host]



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.

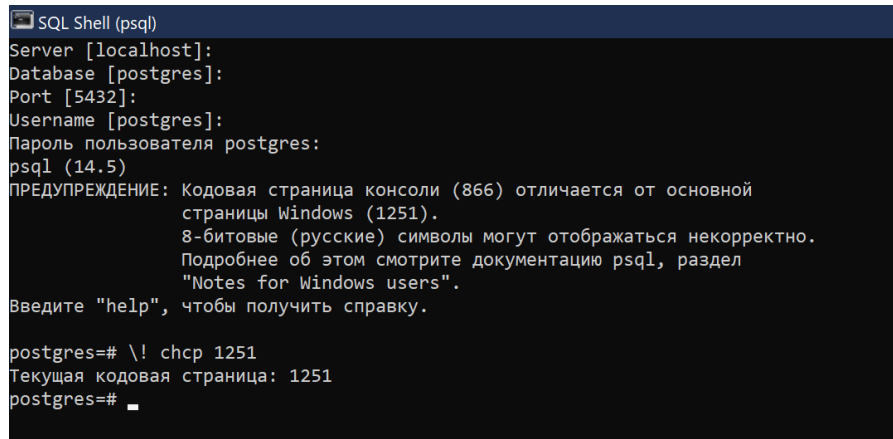
Нажимает Enter.

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
```



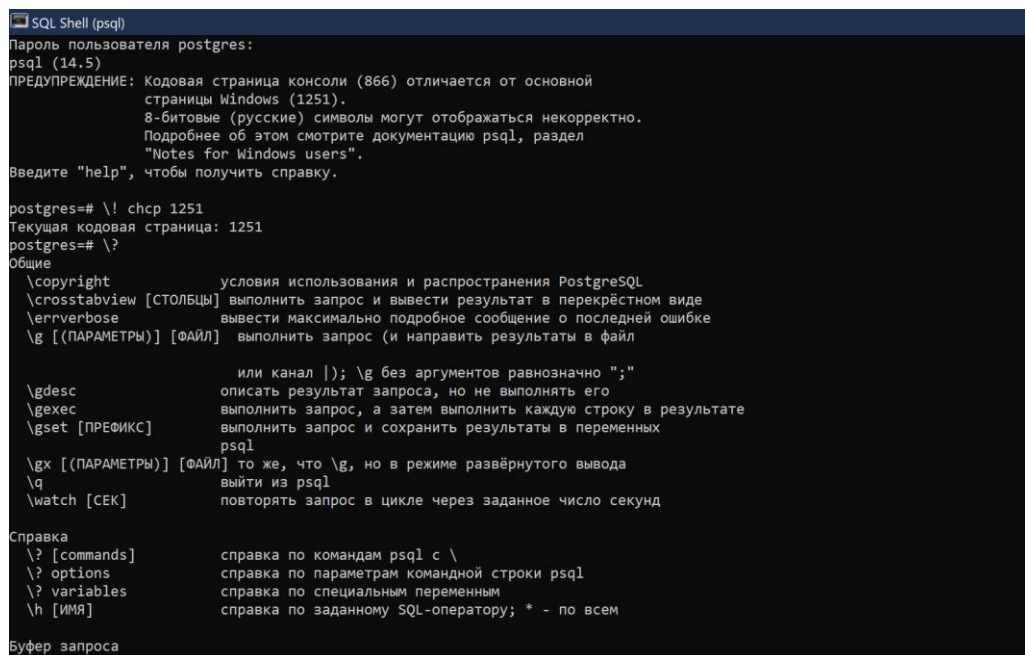
```
SQL Shell (psql)
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Пароль пользователя postgres:
psql (14.5)
ПРЕДУПРЕЖДЕНИЕ: Кодовая страница консоли (866) отличается от основной
                  страницы Windows (1251).
                  8-битовые (русские) символы могут 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
                  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=# \?
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 [ПРЕФИКС]    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

```

Имя	Владелец	Кодировка	Список баз данных 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 + postgres=CTc/postgres
template1	postgres	UTF8	Russian_Russia.1251	Russian_Russia.1251	=c/postgres + postgres=CTc/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

```

Имя	Владелец	Кодировка	Список баз данных 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 + postgres=CTc/postgres
template1	postgres	UTF8	Russian_Russia.1251	Russian_Russia.1251	=c/postgres + postgres=CTc/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

```

Имя	Владелец	Кодировка	Список баз данных 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 + postgres=CTc/postgres
template1	postgres	UTF8	Russian_Russia.1251	Russian_Russia.1251	=c/postgres + postgres=CTc/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".
starter=#
```

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 "C:\Program Files\PostgreSQL\13\bin\psql.exe" -h %server% -U %username% -d %database% -p %port% 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: CREATE DATABASE starter;
DROP DATABASE	Команда DROP DATABASE [имя_базы_данных] удаляет

	существующую базу данных с названием [имя_базы_данных]. Пример: <code>DROP DATABASE starter;</code>
\c	Команда \c [имя_бд] 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

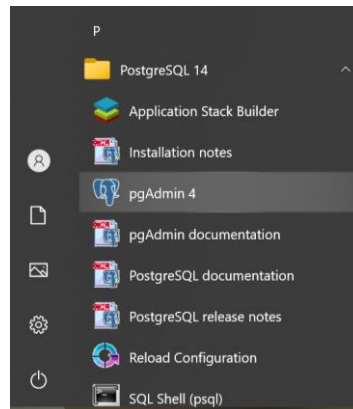
Теоретическое обоснование

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.

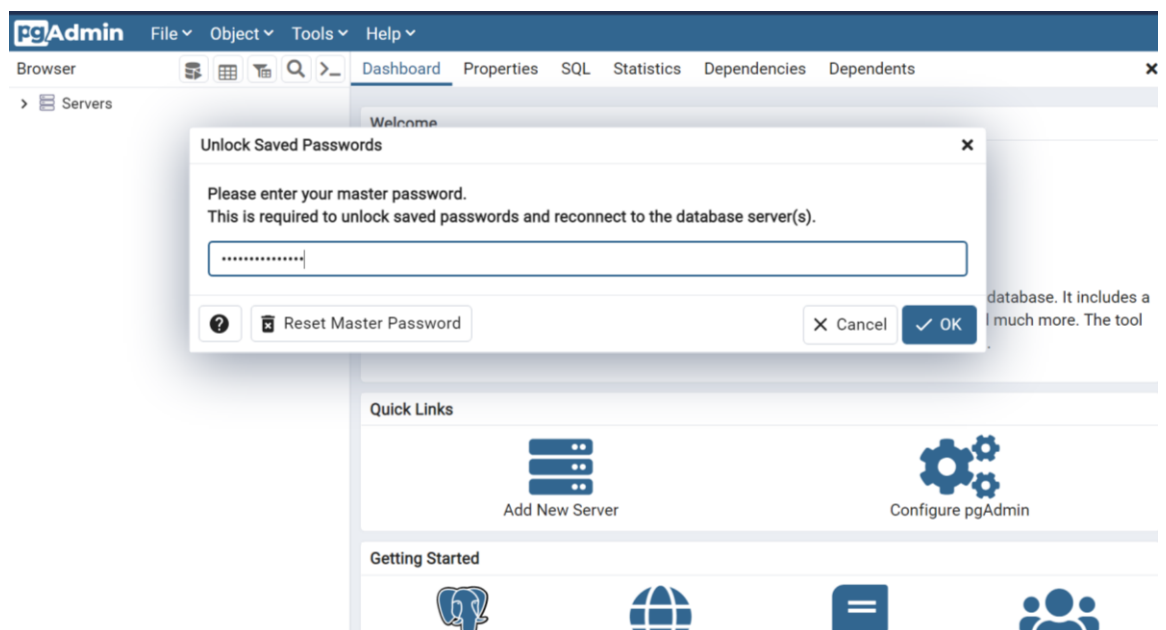
Этот инструмент был установлен в комплекте инструментов postgres. Давайте посмотрим на то, как мы начать работать с ним.

Методика выполнения

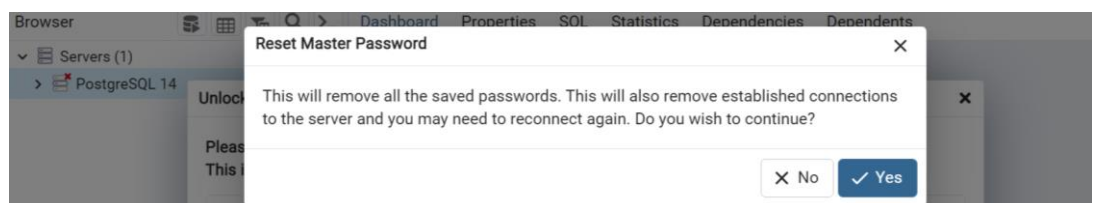
1. Запустить pgAdmin4 можно через меню пуск. Он находится на вкладке Пуск-PostgreSQL14-pgAdmin4.



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

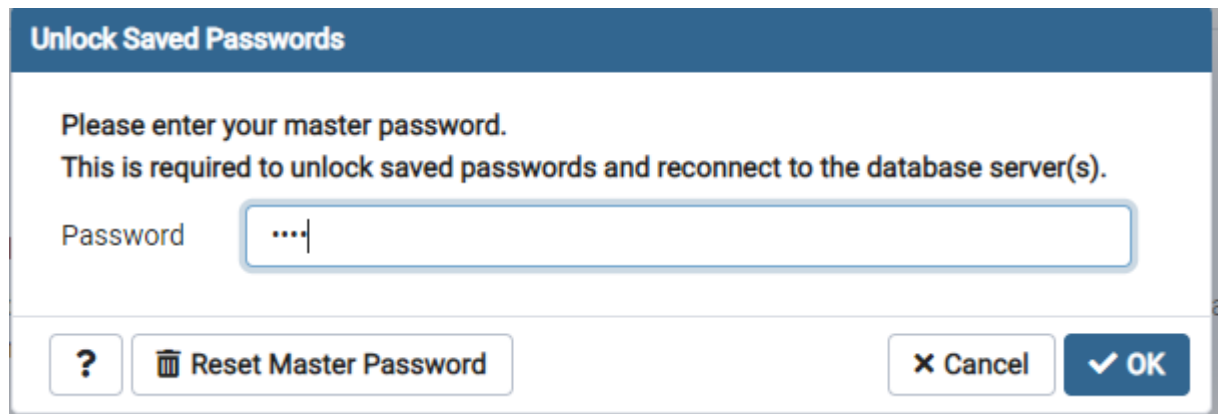


3. Если вы забыли пароль, то его можно поменять, нажав на кнопку Reset Master Password. В окне Reset Master Password вводим новый пароль и применяем изменения.



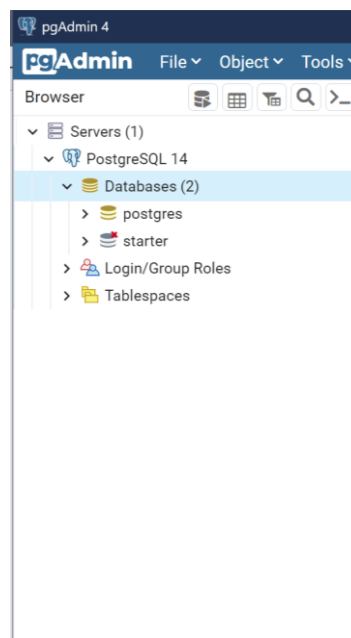
4. После входа, вас встретит пустой экран, без какой-либо информации. Мы не видим её так как не подключены ни к одному серверу. Нажмите на вкладку Servers в левой части экрана, а затем сделайте двойной щелчок по вкладке PostgreSQL14.

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

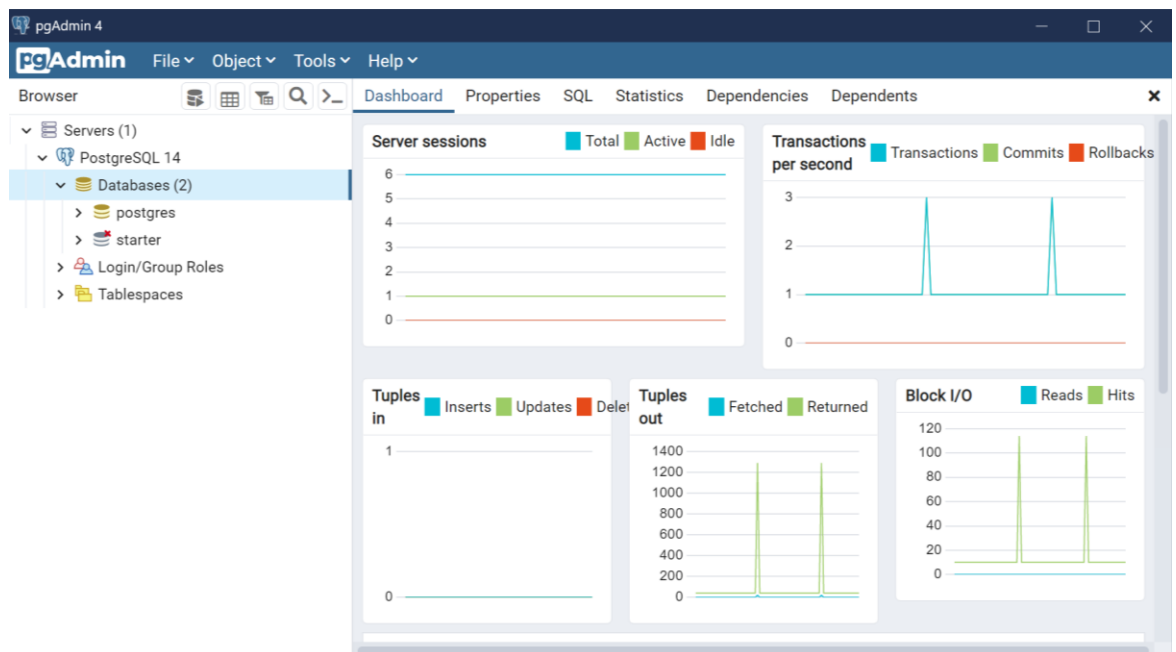


6. После ввода пароля мы сможем увидеть все созданные нами базы данных. Для отображения метрик по нужной вам базе данных, нажмите на неё.

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=# \! chcp 1251
Текущая кодовая страница: 1251
postgres=# \c starter
Вы подключены к базе данных "starter" как пользователь "postgres".
postgres=# CREATE TABLE student(
starter(# id INT, name VARCHAR(50), surname VARCHAR(50), gender VARCHAR(50),
starter(# mobile VARCHAR(50), birthday DATE);
CREATE TABLE
postgres=#

```

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

postgres=#

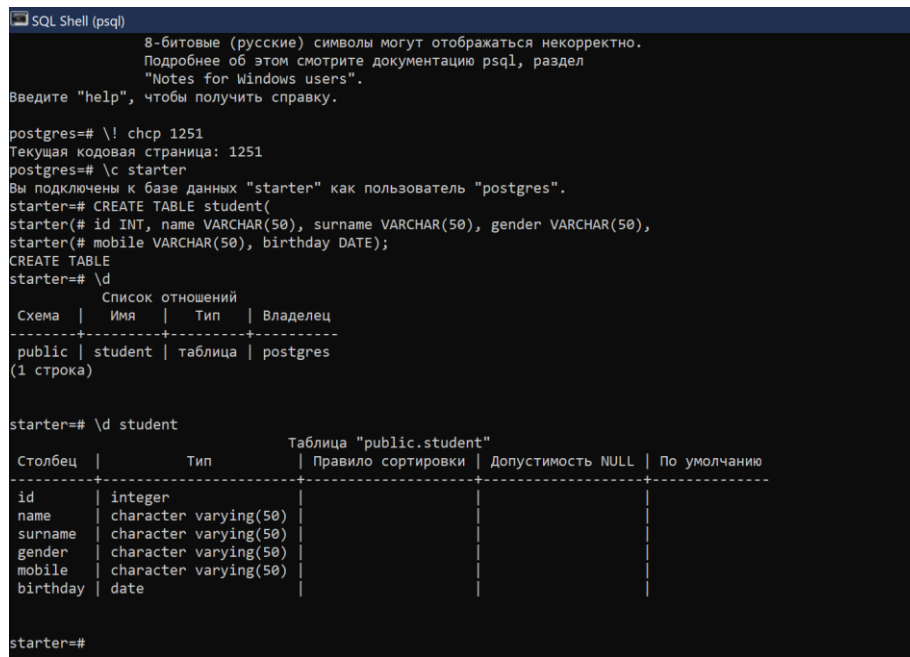
```

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".
postgres=# CREATE TABLE student(
postgres=# id INT, name VARCHAR(50), surname VARCHAR(50), gender VARCHAR(50),
postgres=# mobile VARCHAR(50), birthday DATE);
CREATE TABLE
postgres=# \d
      Список отношений
+-----+-----+-----+-----+
| Схема | Имя  | Тип  | Владелец |
+-----+-----+-----+-----+
| public | student | таблица | postgres |
+-----+-----+-----+-----+
(1 строка)

postgres=# \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         |                     |                     |              |
+-----+-----+-----+-----+-----+

postgres=#
```

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
starter=# INSERT INTO student(name, surname, gender, mobile, birthday) VALUES ('Michael', 'Scott', 'male', '+79884561237', '2000-04-12');
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', '2000-04-12');
INSERT 0 1
starter=# SELECT * FROM student;
 id | name  | surname | gender | mobile  | birthday
-----+-----+-----+-----+-----+-----
  1 | Michael | Scott  | male  | +79884561237 | 2000-04-12
(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
-----+-----+-----+-----+-----+-----
    | Michael | Scott | male | +79884561237 | 2000-04-12
(1 строка)

starter=# \i 'D:/ИнжинирингДанных/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
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 | Correy | Humberstone | Bigender | 601-336-8268 | 2021-05-06
  2 | Rina | Leaves | Non-binary | 444-842-7249 | 2020-12-18
  3 | Leyla | Culpen | Non-binary | 790-158-3796 | 2020-12-01
  4 | Karna | Hosby | Genderqueer | 399-767-3350 | 2021-02-07
  5 | Berkie | Yule | Non-binary | 779-336-6742 | 2021-05-30
  6 | Avrit | Dahl | Genderfluid | 633-484-1428 | 2021-08-07
  7 | Andrei | Cockney | Genderfluid | 953-996-3483 | 2021-06-10
  8 | Gates | Dobbie | Agender | 161-421-5731 | 2021-03-19
  9 | Ulrich | Bridge | Female | 975-401-0123 | 2021-01-15
 10 | Georgiana | Behnke | Bigender | 568-165-8199 | 2021-02-26
 11 | Emma | Cokly | Polygender | 289-747-5136 | 2021-07-12
 12 | Chicky | Schonfelder | Genderfluid | 903-625-6677 | 2021-08-07
 13 | Rodina | Denness | Male | 586-183-6597 | 2021-08-19
 14 | Jacqueline | Micah | Female | 765-924-7203 | 2021-05-03
 15 | Othelia | Kernley | 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 | Balduin | Cordall | Polygender | 314-694-9253 | 2021-05-22
 20 | Lia | Kermott | Genderfluid | 220-178-8717 | 2021-04-12
 21 | Aurilia | Densumbe | Polygender | 378-864-9864 | 2021-05-21
 22 | Delcine | Pendered | Female | 988-699-8710 | 2021-04-20
 23 | Lewis | Yanyshev | 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 | Nemchinov | Female | 822-216-7261 | 2021-03-28
 29 | Rosabelle | Fenich | Agender | 829-323-5779 | 2020-10-31
 30 | Ameline | Lardiner | Polygender | 598-751-2283 | 2021-05-19
 31 | Emlyn | 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;
```

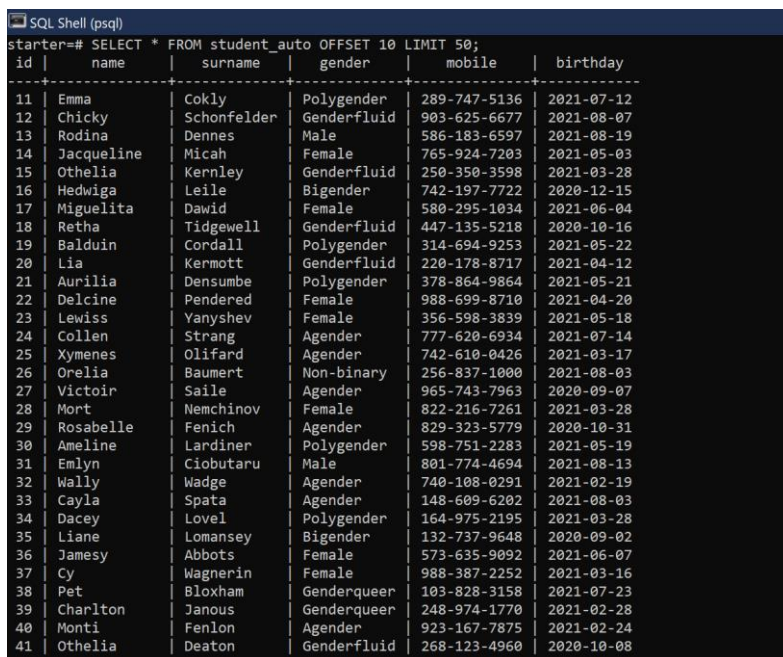
Выборка данных. Скрипты

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

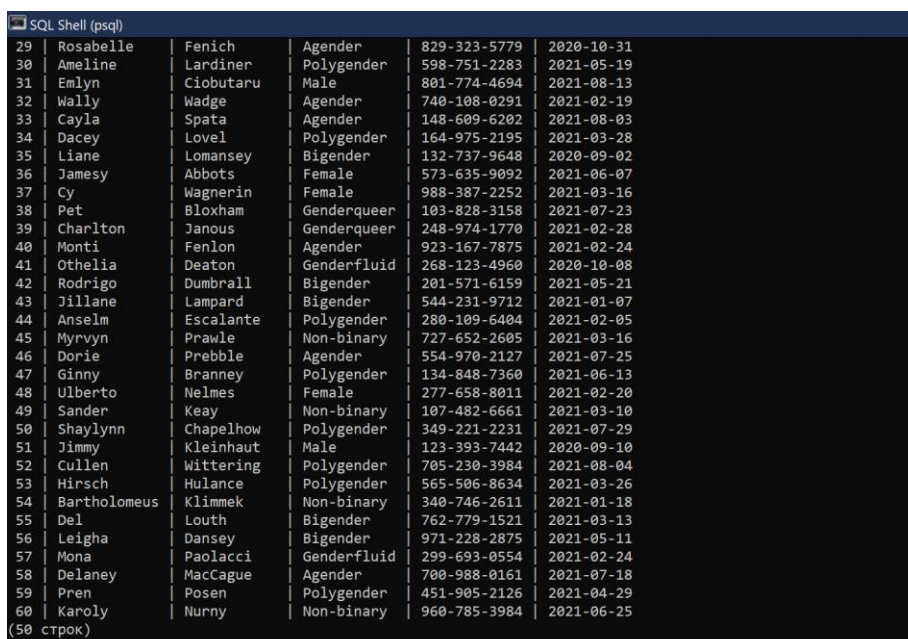
`SELECT *FROM student_auto OFFSET 10 LIMIT 50:`

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	Emma	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	765-924-7203	2021-05-03
15	Othelia	Kernley	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	Balduin	Cordall	Polygender	314-694-9253	2021-05-22
20	Lia	Kermott	Genderfluid	220-178-8717	2021-04-12
21	Aurilia	Densumbe	Polygender	378-864-9864	2021-05-21
22	Delcine	Pendered	Female	988-699-8710	2021-04-20
23	Lewiss	Yanyshv	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	Nemchinov	Female	822-216-7261	2021-03-28
29	Rosabelle	Fenich	Agender	829-323-5779	2020-10-31
30	Ameline	Lardiner	Polygender	598-751-2283	2021-05-19
31	Emlyn	Ciobutaru	Male	801-774-4694	2021-08-13
32	Wally	Wadge	Agender	740-108-0291	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	Lomansey	Bigender	132-737-9648	2020-09-02
36	Jamesy	Abbots	Female	573-635-9092	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)
```

29	Rosabelle	Fenich	Agender	829-323-5779	2020-10-31
30	Ameline	Lardiner	Polygender	598-751-2283	2021-05-19
31	Emlyn	Ciobutaru	Male	801-774-4694	2021-08-13
32	Wally	Wadge	Agender	740-108-0291	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	Lomansey	Bigender	132-737-9648	2020-09-02
36	Jamesy	Abbots	Female	573-635-9092	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	Dumbrall	Bigender	201-571-6159	2021-05-21
43	Jillane	Lampard	Bigender	544-231-9712	2021-01-07
44	Anselm	Escalante	Polygender	280-109-6404	2021-02-05
45	Myrvyn	Prawle	Non-binary	727-652-2605	2021-03-16
46	Dorie	Prebble	Agender	554-970-2127	2021-07-25
47	Ginny	Branney	Polygender	134-848-7360	2021-06-13
48	Ulberto	Nelmes	Female	277-658-8011	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	Hulance	Polygender	565-506-8634	2021-03-26
54	Bartholomeus	Klimmek	Non-binary	340-746-2611	2021-01-18
55	Del	Louth	Bigender	762-779-1521	2021-03-13
56	Leigha	Dansey	Bigender	971-228-2875	2021-05-11
57	Mona	Paolacci	Genderfluid	299-693-0554	2021-02-24
58	Delaney	MacCague	Agender	700-988-0161	2021-07-18
59	Pren	Posen	Polygender	451-905-2126	2021-04-29
60	Karoly	Nurny	Non-binary	960-785-3984	2021-06-25

(60 строк)

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 | Livvy    | 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
Emma   | Cokly
Chicky | Schonfelder
Rodina | Dennes
Jacqueline | Micah
Othelia | Kernley
Hedwiga | Leile
Miguelita | Dawid
Retha  | Tidgewell
Balduin | Cordall
Lia    | Kermott
Aurilia | Densumbe
Delcine | Pendered
Lewiss  | Yanyshev
Collen  | Strang
Xymenes | Olifard
Orelia  | Baumert
Victoir | Saile
Mort    | Nemchinov
Rosabelle | 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
2021-05-03
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
2021-08-03
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
400	Elihu	Stent	Female	458-777-2977	2021-08-28
400	Elihu	Stent	Female	458-777-2977	2021-08-28
399	Seka	Harly	Male	853-826-0288	2021-08-28
400	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	Eymor	Non-binary	687-135-4156	2021-08-27
296	Morse	Albery	Female	621-496-6036	2021-08-27
74	Llywellyn	Eymor	Non-binary	687-135-4156	2021-08-27
74	Llywellyn	Eymor	Non-binary	687-135-4156	2021-08-27
313	Andras	Cutill	Genderqueer	304-653-9770	2021-08-25
313	Andras	Cutill	Genderqueer	304-653-9770	2021-08-25
313	Andras	Cutill	Genderqueer	304-653-9770	2021-08-25
412	Shea	Lapworth	Male	914-307-1809	2021-08-24
177	Chad	Boat	Genderfluid	146-531-5558	2021-08-24
177	Chad	Boat	Genderfluid	146-531-5558	2021-08-24
177	Chad	Boat	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	Wille	Bigender	957-734-3893	2021-08-20

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

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

SQL Shell (psql)

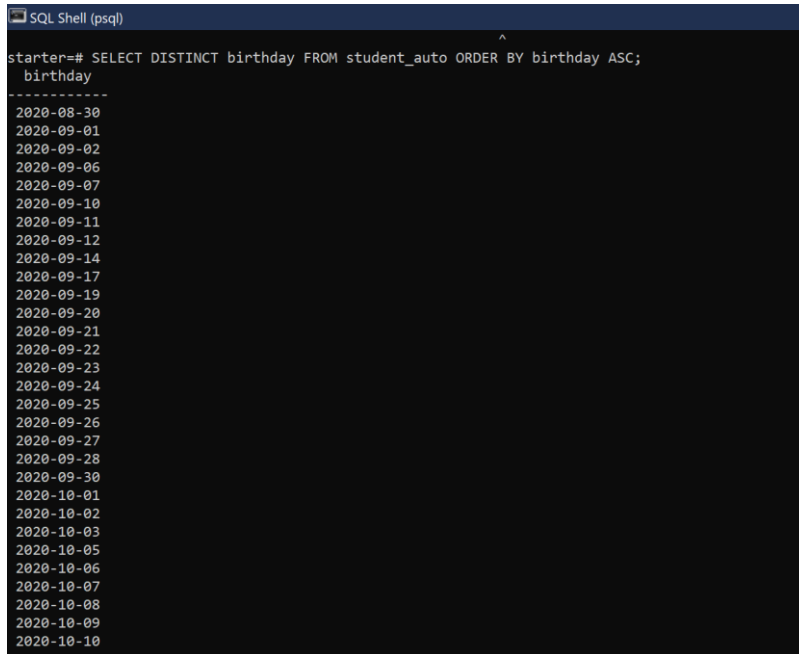
```
starter=# SELECT name FROM student_auto ORDER BY birthday ASC;
```

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 (psql)

```

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

```

id	name	surname	gender	mobile	birthday
9	Ulrich	Bridge	Female	975-401-0123	2021-01-15
14	Jacqueline	Micah	Female	765-924-7203	2021-05-03
17	Miguelita	Dawid	Female	580-295-1034	2021-06-04
22	Delcine	Pendered	Female	988-699-8710	2021-04-20
23	Lewis	Yanyshv	Female	356-598-3839	2021-05-18
28	Mort	Nemchinov	Female	822-216-7261	2021-03-28
36	Jamesy	Abbots	Female	573-635-9092	2021-06-07
37	Cy	Wagnerin	Female	988-387-2252	2021-03-16
48	Ulberto	Nelmes	Female	277-658-8011	2021-02-20
62	Lainey	Swaffield	Female	555-568-4213	2020-10-28
70	Alisander	Burdin	Female	518-741-9227	2021-07-14
81	Fred	Pimme	Female	750-803-0513	2021-03-24
87	Luise	Coleshill	Female	500-191-4154	2020-11-20
93	Katrinka	Casey	Female	584-649-8258	2021-04-22
98	Beryle	Pedrick	Female	520-340-0038	2020-09-14
102	Cammy	Cogin	Female	372-739-5886	2020-12-26
140	Grace	Sedgwick	Female	356-808-0361	2021-03-02
153	Cosmo	Cowlishaw	Female	334-581-1646	2021-01-23
156	Huntley	Inkster	Female	979-954-7883	2020-09-28
158	Vitoria	Haet	Female	633-657-3550	2021-04-03
160	Chico	Seviour	Female	873-409-9343	2021-02-08
169	Dill	Jendrys	Female	922-351-0904	2021-01-15
173	Bendicty	Mizen	Female	404-849-6163	2021-04-01
179	Janos	Mazillius	Female	182-141-9569	2021-01-31
183	Justinn	Rycraft	Female	210-262-5928	2021-04-24
188	Kinny	Le Gallo	Female	470-596-8113	2020-09-20
193	Salaith	English	Female	305-951-9427	2021-06-20

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

```
SELECT * FROM student_auto WHERE gender = 'Female' and
name='Stavros';
```

позволяет сузить диапазон запроса.

```
SQL Shell (psql)
starter=# SELECT * FROM student_auto WHERE gender = 'Female' and name='Stavros';
 id | name  | surname | gender | mobile  | birthday
-----+-----+-----+-----+-----+-----
 396 | Stavros | Coite   | Female | 975-911-6462 | 2021-03-03
 396 | Stavros | Coite   | Female | 975-911-6462 | 2021-03-03
 396 | Stavros | Coite   | Female | 975-911-6462 | 2021-03-03
(3 строки)

starter=#
```

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

```
\i 'D:/ИнжинирингДанных/travelers.sql'
```

Вывод INSERT 0 1 говорит об успешном импорте данных.

```
Выбрать SQL Shell (psql)
starter=# \i 'D:/ИнжинирингДанных/travelers.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
```

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

```
SELECT * FROM travelers;
```

```
SQL Shell (psql)
starter=# SELECT * FROM travelers;
 id | name      | surname | email | gender | country
-----+-----+-----+-----+-----+-----
 1 | Cornelius | Dutch   | cdutch0@joomla.org | Female | Russia
 2 | Yettie    | Gruby   | ygruby1@time.com   | Male   | Russia
 3 | Konstantine | Yakovl | kyakovl2@tripadvisor.com | Female | Indonesia
 4 | Darrel    | Siddaley | dsiddaley3@businessweek.com | Female | China
 5 | Olva      | Leopard | oleopard4@skyrock.com | Male   | China
 6 | Dalli     | Chapelhow | dchapelhow5@godaddy.com | Female | China
 7 | Kerwinn   | Lammas  | klammas6@histats.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   | Comfort | lcomfortb@plala.or.jp | Male   | Uganda
13 | Ranique   | Kulver  | rkulverc@weebly.com | Male   | Brazil
14 | Mallory   | Warham  | mwarhamd@ebay.co.uk | Female | Yemen
15 | Godwin    | Gratrex | ggratrexe@networksolutions.com | Male   | China
16 | Flory     | Mayhead | fmayheadf@gnu.org   | Female | Panama
17 | Hunter    | Anniwell | hanniwel1g@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    | sdykaj@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     | Naiara  | bnaiara@wix.com     | Male   | China
```

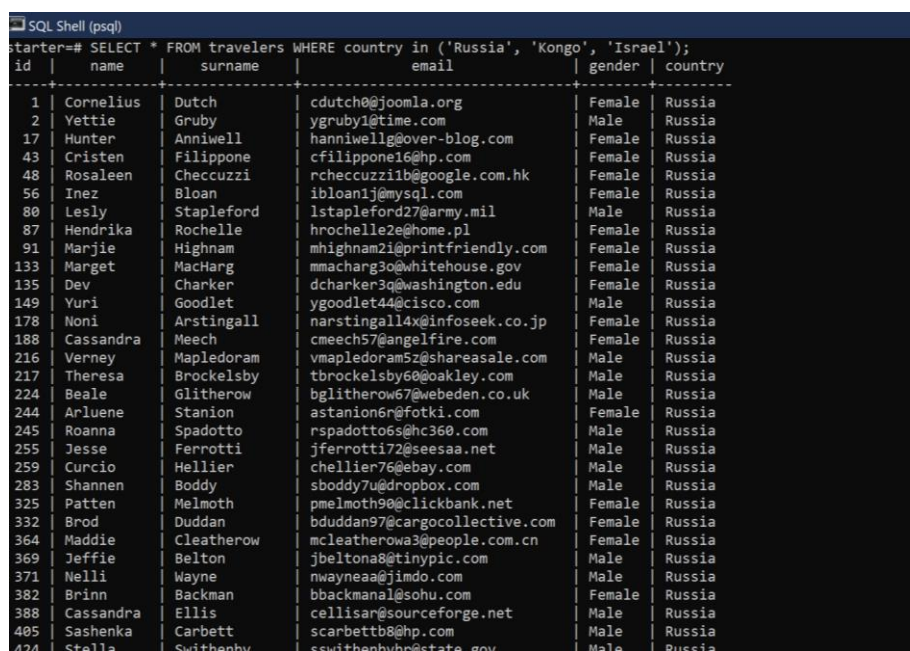
Как результат, мы видим точную копию таблицы, которая хранится на сервере.

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	cdutch0@joomla.org	Female	Russia
2	Yettie	Gruby	ygruby1@time.com	Male	Russia
17	Hunter	Anniwell	hanniwellg@over-blog.com	Female	Russia
43	Cristen	Filippone	cfilippone16@hp.com	Female	Russia
48	Rosaleen	Checuzzi	rcheccuzzi1b@google.com.hk	Female	Russia
56	Inez	Bloan	ibloan1j@mysql.com	Female	Russia
80	Lesly	Stapleford	lstapleford27@army.mil	Male	Russia
87	Hendrika	Rochelle	hrochelle2e@home.pl	Female	Russia
91	Marjie	Highnam	mhighnam2i@printfriendly.com	Female	Russia
133	Marget	MacHarg	mmacharg3o@whitehouse.gov	Female	Russia
135	Dev	Charker	dcharker3q@washington.edu	Female	Russia
149	Yuri	Goodlet	ygoodlet44@cisco.com	Male	Russia
178	Nori	Arstingall	narstingall4x@infoseek.co.jp	Female	Russia
188	Cassandra	Meech	cmeech57@angelfire.com	Female	Russia
216	Verney	Mapledoram	vmappedoram5z@shareasale.com	Male	Russia
217	Theresa	Brockelsby	tbrockelsby6@oakley.com	Male	Russia
224	Beale	Glitherow	bglitherow67@webden.co.uk	Male	Russia
244	Arluene	Stanion	astanion6r@fotki.com	Female	Russia
245	Roanna	Spadotto	rspadotto6s@hc360.com	Male	Russia
255	Jesse	Ferrotti	jferrotti72@seesaa.net	Male	Russia
259	Curcio	Hellier	chellier76@ebay.com	Male	Russia
283	Shannen	Boddy	sboddy7u@dropbox.com	Male	Russia
325	Patten	Melmoth	pmelmoth9@clickbank.net	Female	Russia
332	Brod	Duddan	bduddan97@cargocollective.com	Female	Russia
364	Maddie	Cleatherow	mcleatherowa3@people.com.cn	Female	Russia
369	Jeffie	Belton	jbeltona8@tinypic.com	Male	Russia
371	Nelli	Wayne	nwayneaa@jimdo.com	Male	Russia
382	Brinn	Backman	bbackmanal@sohu.com	Female	Russia
388	Cassandra	Ellis	cellisar@sourceforge.net	Male	Russia
405	Sashenka	Carbett	scarbettb8@hp.com	Male	Russia
424	Stella	Swithenby	sswithenbyhr@state.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	Darrel	Siddaley	dsiddaley3@businessweek.com	Female	China
5	Olva	Leopard	oleopard4@skyrock.com	Male	China
6	Dalli	Chapelhow	dchapelhow5@godaddy.com	Female	China
10	Jedidiah	Cayser	jcayser9@wordpress.com	Female	China
11	Clary	Rea	creaa@godaddy.com	Male	China
13	Ranique	Kulver	rkulverc@weebly.com	Male	Brazil
15	Godwin	Gratrex	ggratrexe@networksolutions.com	Male	China
18	Brannon	Cottisford	bcottisfordh@yellowpages.com	Female	China
24	Jody	Rallin	jrallinn@exblog.jp	Female	Brazil
25	Berti	Najara	bnajara0@wix.com	Male	China
32	Trudie	Westwater	twestwater@rambler.ru	Male	China
42	Margaretha	Bauckham	mbauckham15@mapquest.com	Female	Colombia
45	Marc	Sant	msant18@devhub.com	Male	China
52	Lammond	Shefton	lshefton1f@1688.com	Female	China
58	Brigit	Swatten	bswatten1l@unicef.org	Male	Brazil
60	Hadria	Chipp	hchipp1n@nydailynews.com	Male	China
62	Court	Klambt	cklambt1p@ifeng.com	Female	China
66	Sherwin	Ivakin	sivakin1t@blinklist.com	Male	Canada
69	Queenie	Brodest	qbrodest1w@comcast.net	Female	Czech Republic
71	Horatius	Fargher	hfargher1y@vimeo.com	Male	Colombia
78	Toni	Exall	texall125@instagram.com	Male	China
79	Sheffy	Radclyffe	sradclyffe26@usa.gov	Male	China
81	Heather	Colbourne	hcolbourne28@hud.gov	Male	China
89	Ardelle	McGurgan	amcgurgan2g@marriott.com	Female	China
90	Ellie	Broske	ebroske2h@imgur.com	Female	Colombia
95	Saree	Terrett	sterrett2m@google.co.jp	Female	China
96	Lorianne	Cicccitti	lcicccitti2n@ezinearticles.com	Female	China
98	Giraud	Kiddle	gkiddle2p@zimbio.com	Female	China
101	Sammy	Devonish	sdevonish2s@nbcnews.com	Male	Brazil
105	Florie	Puckinghorne	fpuckinghorne2w@blog.com	Female	Brazil
111	Clovis	Bilam	cbilam32@google.ru	Male	China

В результате в консоль выведутся значения, которые начинаются с А, В, С, 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';`

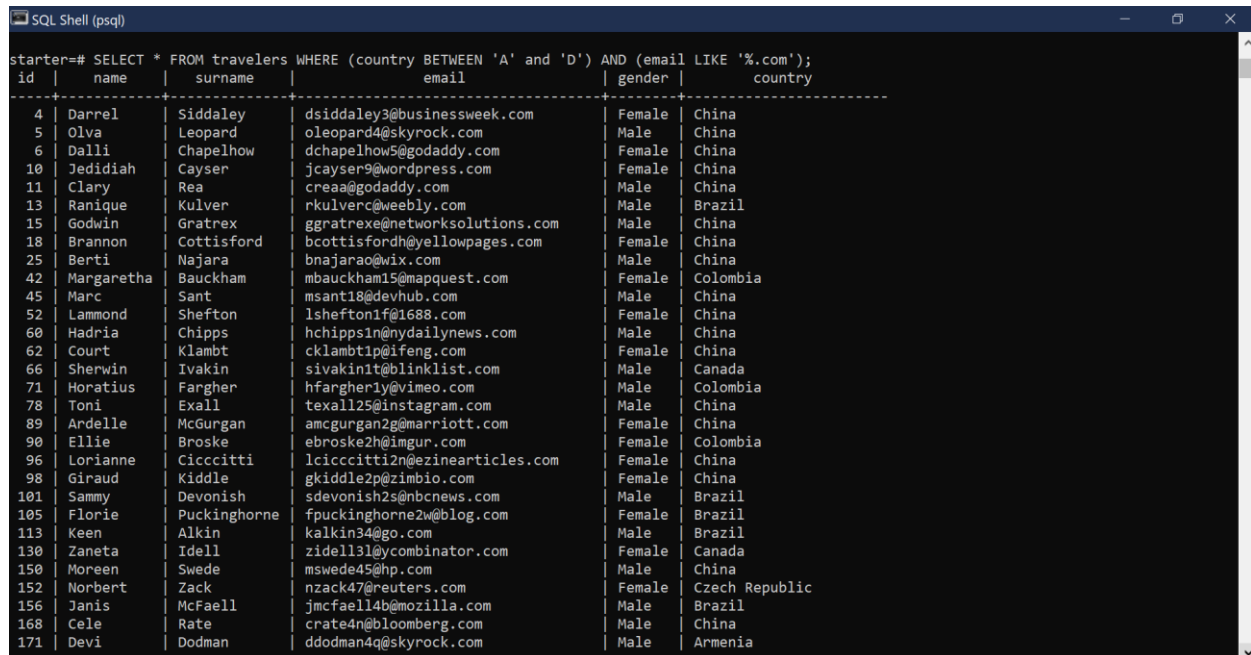
SQL Shell (psql)

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

	id	name	surname	email	gender	country
19	Gale	Di Nisco	gdinisco1@umich.edu	Male	United Kingdom	
26	Sebastien	Vines	svinesp@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	dcharker3q@washington.edu	Female	Russia	
151	Diandra	Aleixo	daleixo46@cornell.edu	Female	Palau	
158	Welsh	Brockherst	wbrockherst4d@tamu.edu	Male	Azerbaijan	
172	Jana	Cuttriss	jcuttriss4r@berkeley.edu	Female	Bulgaria	
191	Rosana	Ratnege	rratnege5a@princeton.edu	Male	Czech Republic	
214	Neda	Litchmore	nlitchmore5x@tamu.edu	Female	China	
215	Charita	Shadfourth	cshadfourth5y@upenn.edu	Female	Vietnam	
233	Ferdinande	Stockhill	fstockhill16g@washington.edu	Female	Thailand	
241	Larry	Cliburn	lcliburn60@si.edu	Female	Hungary	
252	Britte	Barents	bbarents6z@washington.edu	Male	China	
295	Sigfrid	Banbridge	sbanbridge86@ucla.edu	Male	China	
315	Iorgos	Boschmann	iboschmann8q@washington.edu	Male	Democratic Republic of the Congo	
330	Kathi	Baptie	kbaptie95@unc.edu	Female	Nicaragua	
345	Jemimah	Chanson	jchanson9k@utexas.edu	Male	Brazil	
367	Joby	Inglesant	jinglesanta6@harvard.edu	Male	Malaysia	
374	Fern	Irnis	firnisad@ucsd.edu	Female	Greece	
378	Harriett	Zecchetti	hzecchetti@washington.edu	Female	Canada	
410	Cortney	Daudray	cdaudraybd@cmu.edu	Female	Croatia	
452	Philippe	Madgewick	pmadgewickc@ucsd.edu	Female	Philippines	
462	Humbert	Parell	hparell1ct@mit.edu	Female	Canada	
463	Gavan	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');



The screenshot shows a terminal window titled "SQL Shell (psql)". The prompt is "sql>". The user has entered the query: "SELECT * FROM travelers WHERE (country BETWEEN 'A' and 'D') AND (email LIKE '%.com');". The results are displayed in a table with columns: id, name, surname, email, gender, and country. The results are sorted by id.

id	name	surname	email	gender	country
4	Darrel	Siddaley	dsiddaley3@businessweek.com	Female	China
5	Olva	Leopard	oleopard4@skyrock.com	Male	China
6	Dalli	Chapelhow	dchapelhow5@godaddy.com	Female	China
10	Jedidiah	Cayser	jcayser9@wordpress.com	Female	China
11	Clary	Rea	creaa@godaddy.com	Male	China
13	Ranique	Kulver	rkulverc@weebly.com	Male	Brazil
15	Godwin	Gratrex	ggratrexe@networksolutions.com	Male	China
18	Brannon	Cottisford	bcottisfordh@yellowpages.com	Female	China
25	Berti	Najara	bnajara@wix.com	Male	China
42	Margaretha	Bauckham	mbauckham15@mapquest.com	Female	Colombia
45	Marc	Sant	msant18@devhub.com	Male	China
52	Lammond	Shefton	lshefton1f@1688.com	Female	China
60	Hadria	Chipps	hchipps1n@nydailynews.com	Male	China
62	Court	Klambt	cklambt1p@ifeng.com	Female	China
66	Sherwin	Ivakin	sivakin1t@blinklist.com	Male	Canada
71	Horatius	Fargher	hfargher1y@vimeo.com	Male	Colombia
78	Toni	Exall	texall25@instagram.com	Male	China
89	Ardelle	McGurgan	amcgurgan2g@marriott.com	Female	China
90	Ellie	Broske	ebroske2h@imgur.com	Female	Colombia
96	Lorianne	Ciccitti	lciccitti1n@ezinearticles.com	Female	China
98	Giraud	Kiddle	gkiddle2p@zimbio.com	Female	China
101	Sammy	Devonish	sdevonish2s@nbcnews.com	Male	Brazil
105	Florie	Puckinghorne	fpuckinghorne2w@blog.com	Female	Brazil
113	Keen	Alkin	kalkin34@go.com	Male	Brazil
130	Zaneta	Idell	zidell13l@combinator.com	Female	Canada
150	Moreen	Swede	mswede45@hp.com	Male	China
152	Norbert	Zack	nzack47@reuters.com	Female	Czech Republic
156	Janis	McFaell	jmcfaell4b@mozilla.com	Male	Brazil
168	Cele	Rate	crate4n@bloomberg.com	Male	China
171	Devi	Dodman	ddodman4q@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
   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
```

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    | 40
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   | 40
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) | | | 
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;
```

```
SQL Shell (psql)

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) | | | 

starter=# SELECT id As n, surname As s FROM travelers;
 n | s
---+---
 1 | Dutch
 2 | Gruby
 3 | Yakovl
 4 | Siddaley
 5 | Leopard
 6 | Chapelhow
 7 | Lammass
 8 | MacDougall
 9 | McCritchie
10 | Cayser
11 | Rea
12 | Comfort
13 | Kulver
14 | Warham
15 | Gratrex
16 | Mayhead
17 | Anniwell
18 | Cottisford
19 | Di Nisco
20 | Dyka
```

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;
```

SQL Shell (psql)

```
public | student_auto | таблица | postgres
public | travelers    | таблица | postgres
(4 строки)
```

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

id	name	surname	departure_from	arriving_to	price
1	Reider	MacBarron	Brazil	Portugal	414
2	Alisun	Rothschild	France	Russia	169
3	Barr	Le Fleming	Argentina	Philippines	255
4	Tanny	Jobbins	Indonesia	Indonesia	781
5	Rurik	Semeniuk	Japan	Peru	392
6	Briana	Yare	China	Japan	900
7	Gert	Crickmoor	Tajikistan	Dominican Republic	188
8	Florinda	Japp	Brazil	Argentina	464
9	Valerye	Yesipov	Poland	China	75
10	Jeniffer	Goosey	Mexico	Thailand	447
11	Phedra	Moralas	Bangladesh	Macedonia	124
12	Nikolas	Parnham	Honduras	China	374
13	Erroll	Iacobucci	Serbia	Russia	469
14	Tomlin	Lghan	China	Myanmar	849
15	Fairfax	Dark	China	Russia	394
16	Gaby	Yesson	Cuba	Colombia	740
17	Ryley	Petrasek	Philippines	Mexico	991
18	Kelly	Elvy	China	Russia	96
19	Alisun	Posnette	China	Japan	537
20	Gilemette	Waggatt	Indonesia	China	545
21	Jimmie	Broomer	France	China	147

Рисунок 1

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;
```

```
Задание 7. Базовая арифметика

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=# select max(price) from booking_n;
max
-----
 999
(1 строка)
```

Рисунок 3

Для поиска самой маленькой цены используем функцию min:

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

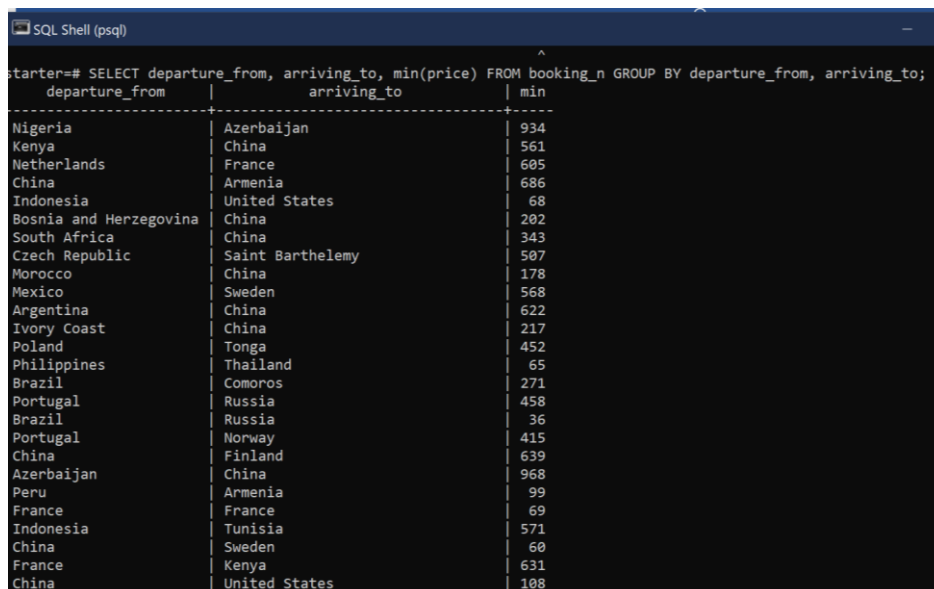
```
starter=# select min(price) from booking_n;
min
-----
 25
(1 строка)

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;
```



```

starter=# SELECT departure_from, arriving_to, min(price) FROM booking_n GROUP BY departure_from, arriving_to;

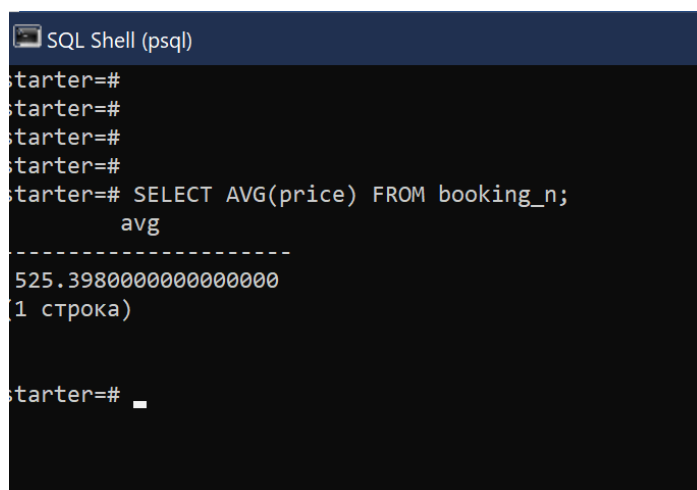
```

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	202
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;
```



```

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

```

avg
525.3980000000000000

(1 строка)

```

starter=#

```

Рисунок 6

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

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

```
SQL Shell (psql)
starter=#
starter=#
starter=#
starter=#
starter=# SELECT AVG(price) FROM booking_n;
      avg
-----
525.3980000000000000
(1 строка)

starter=# SELECT SUM(price) FROM booking_n;
      sum
-----
525398
(1 строка)
```

Рисунок 7

И последнее, давайте узнаем у какой страны самые дорогие вылеты:

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

```
SQL Shell (psql)
starter=# SELECT departure_from, max(price) FROM booking_n GROUP BY departure_from ORDER BY max(price) DESC;
 departure_from | max
-----+-----
Peru            | 999
Nigeria        | 998
Kazakhstan      | 998
Indonesia       | 997
China           | 995
Palestinian Territory | 994
Afghanistan     | 993
Philippines     | 991
Russia          | 989
Brazil          | 987
Thailand        | 987
France          | 987
Togo            | 985
Portugal        | 983
Myanmar         | 979
Sweden          | 978
Venezuela       | 977
United States   | 976
Greece          | 974
Dominican Republic | 973
South Africa    | 971
Honduras        | 968
Azerbaijan      | 968
Latvia          | 951
Bosnia and Herzegovina | 949
Ghana           | 948
Malaysia        | 947
Ukraine         | 941
Canada          | 940
Japan           | 933
```

Рисунок 8

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	Humberstone	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	Dobbie	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
Emma	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	Densumbe	1 year 5 mons 17 days 19:05:12.206245
Delcine	Pendered	1 year 6 mons 17 days 19:05:12.206245
Leviss	Yanushev	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

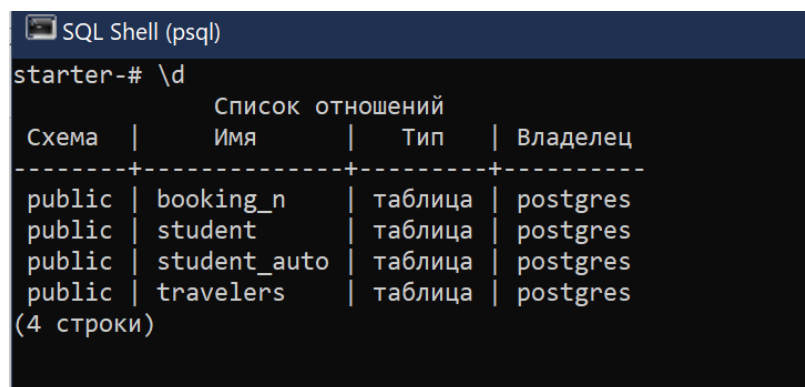
Команда ALTER TABLE имя_таблицы DROP

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
          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

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
Ewan	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
Patten	3
Shelly	2
Darya	3
Ellynn	2
Morry	2
Salomo	2
Spenser	2
Rafaelita	2
Corrie	2
Suzann	2
Amata	2
Heriberto	2
Theresa	2
Marc	2
Fina	2
Mile	2
Christabel	2
Marthe	2
Cesare	2
Gino	2
Delcine	2
Cassandra	2
Kylie	2
Berti	2
Kameko	2
Ferdinande	2
Charlot	2
Hugh	2
Velvet	2

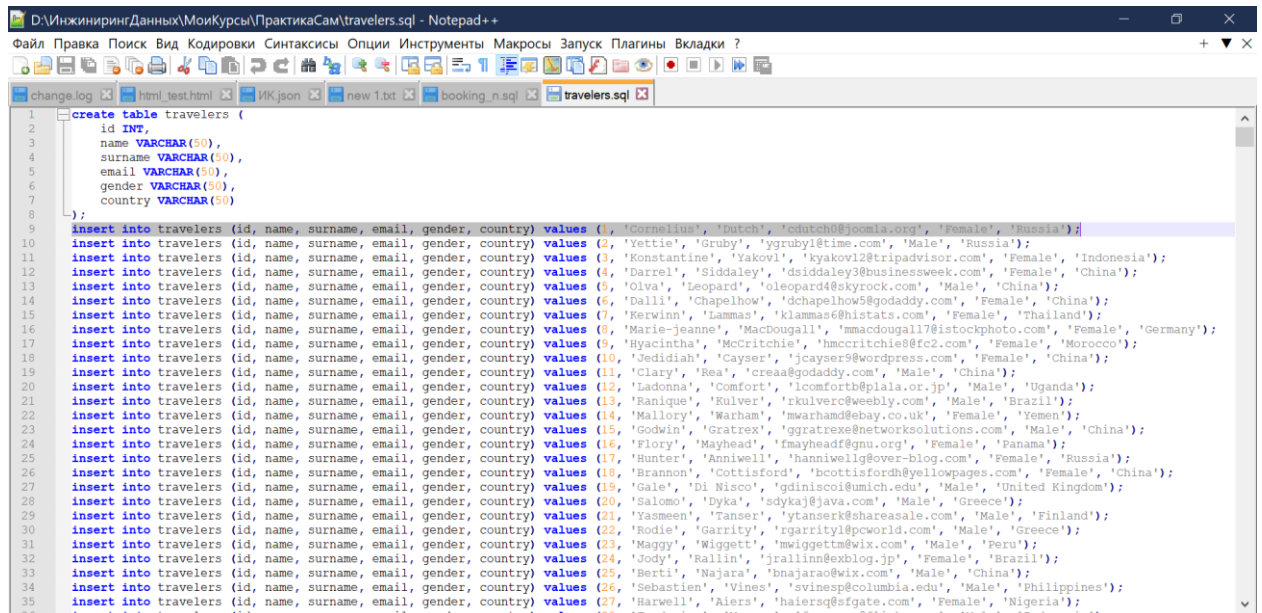
Рисунок 3

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');`

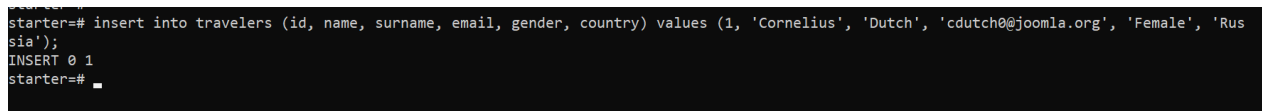
и вставим эту строку в консоль postgresql



```
1 create table travelers (
2     id INT,
3     name VARCHAR(50),
4     surname VARCHAR(50),
5     email VARCHAR(50),
6     gender VARCHAR(50),
7     country VARCHAR(50)
8 );
9 insert into travelers (id, name, surname, email, gender, country) values (1, 'Cornelius', 'Dutch', 'cdutch0@joomla.org', 'Female', 'Russia');
10 insert into travelers (id, name, surname, email, gender, country) values (2, 'Yettie', 'Gruby', 'ygrubyl@time.com', 'Male', 'Russia');
11 insert into travelers (id, name, surname, email, gender, country) values (3, 'Konstantine', 'Yakovl', 'kyakovl2@tripadvisor.com', 'Female', 'Indonesia');
12 insert into travelers (id, name, surname, email, gender, country) values (4, 'Darrel', 'Siddaley', 'dsiddaley3@businessweek.com', 'Female', 'China');
13 insert into travelers (id, name, surname, email, gender, country) values (5, 'Olva', 'Leopard', 'oleopard4@skyrock.com', 'Male', 'China');
14 insert into travelers (id, name, surname, email, gender, country) values (6, 'Dalli', 'Chapelhow', 'dchapelhow5@godaddy.com', 'Female', 'China');
15 insert into travelers (id, name, surname, email, gender, country) values (7, 'Kervinn', 'Lamas', 'klamas6@histats.com', 'Female', 'Thailand');
16 insert into travelers (id, name, surname, email, gender, country) values (8, 'Marie-jeanne', 'MacDougall', 'mmacdougall7@istockphoto.com', 'Female', 'Germany');
17 insert into travelers (id, name, surname, email, gender, country) values (9, 'Hyacintha', 'McCritchie', 'hmccritchie8@fc2.com', 'Female', 'Morocco');
18 insert into travelers (id, name, surname, email, gender, country) values (10, 'Jedidiah', 'Cayser', 'jcayser9@wordpress.com', 'Female', 'China');
19 insert into travelers (id, name, surname, email, gender, country) values (11, 'Clary', 'Rea', 'creaa@godaddy.com', 'Male', 'China');
20 insert into travelers (id, name, surname, email, gender, country) values (12, 'Ladonna', 'Comfort', 'lcomfortb@plala.or.jp', 'Male', 'Uganda');
21 insert into travelers (id, name, surname, email, gender, country) values (13, 'Ranique', 'Kulver', 'rkulverc@weebly.com', 'Male', 'Brazil');
22 insert into travelers (id, name, surname, email, gender, country) values (14, 'Mallory', 'Warham', 'mwarhamd@ebay.co.uk', 'Female', 'Yemen');
23 insert into travelers (id, name, surname, email, gender, country) values (15, 'Godwin', 'Catrex', 'ggatrexe@networksolutions.com', 'Male', 'China');
24 insert into travelers (id, name, surname, email, gender, country) values (16, 'Flory', 'Mayhead', 'fmayheadf@gnu.org', 'Female', 'Panama');
25 insert into travelers (id, name, surname, email, gender, country) values (17, 'Hunter', 'Anniwell', 'hanniwellg@over-blog.com', 'Female', 'Russia');
26 insert into travelers (id, name, surname, email, gender, country) values (18, 'Brannon', 'Cottisford', 'bcottisfordh@yellowpages.com', 'Female', 'China');
27 insert into travelers (id, name, surname, email, gender, country) values (19, 'Gale', 'Di Nisco', 'gdinisoi@umich.edu', 'Male', 'United Kingdom');
28 insert into travelers (id, name, surname, email, gender, country) values (20, 'Salomo', 'Dyka', 'sdykaj@java.com', 'Male', 'Greece');
29 insert into travelers (id, name, surname, email, gender, country) values (21, 'Yasmeen', 'Tanser', 'ytanserk@shareasale.com', 'Male', 'Finland');
30 insert into travelers (id, name, surname, email, gender, country) values (22, 'Rodie', 'Garritty', 'rgarrittyl@pcworld.com', 'Male', 'Greece');
31 insert into travelers (id, name, surname, email, gender, country) values (23, 'Maggie', 'Wiggett', 'mwiggettm@wix.com', 'Male', 'Peru');
32 insert into travelers (id, name, surname, email, gender, country) values (24, 'Oody', 'Ballin', 'jballin@exblog.jp', 'Female', 'Brazil');
33 insert into travelers (id, name, surname, email, gender, country) values (25, 'Berti', 'Najara', 'bnajara@wix.com', 'Male', 'China');
34 insert into travelers (id, name, surname, email, gender, country) values (26, 'Sebastien', 'Vines', 'svinesp@columbia.edu', 'Male', 'Philippines');
35 insert into travelers (id, name, surname, email, gender, country) values (27, 'Harwell', 'Aiers', 'haiersq@sfgate.com', 'Female', 'Nigeria');
```

Рисунок 4

Как результат, успешная вставка в таблицу (рисунок 5)

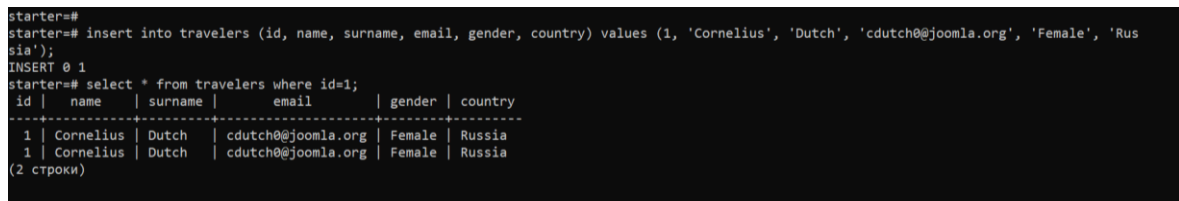


```
starter=# insert into travelers (id, name, surname, email, gender, country) values (1, 'Cornelius', 'Dutch', 'cdutch0@joomla.org', 'Female', 'Russia');
INSERT 0 1
starter=#
```

Рисунок 5

Сделаем запрос на вывод всех записей с данным id. Увидим 2 записи.

```
select * from travelers where id=1;
```

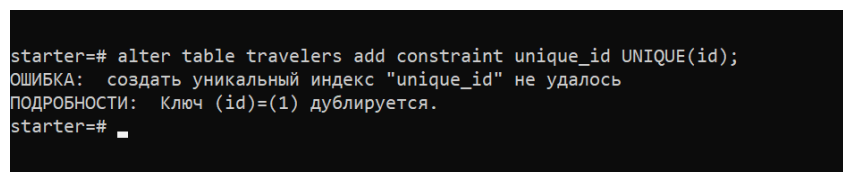


```
starter=#
starter=# insert into travelers (id, name, surname, email, gender, country) values (1, 'Cornelius', 'Dutch', 'cdutch0@joomla.org', 'Female', 'Russia');
INSERT 0 1
starter=# select * from travelers where id=1;
 id | name  | surname | email          | gender | country
-----+-----+-----+-----+-----+-----
  1 | Cornelius | Dutch | cdutch0@joomla.org | Female | Russia
  1 | Cornelius | Dutch | cdutch0@joomla.org | Female | Russia
(2 строки)
```

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
```

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)			

```
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', 'Dutch', 'cdutch@joomla.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.

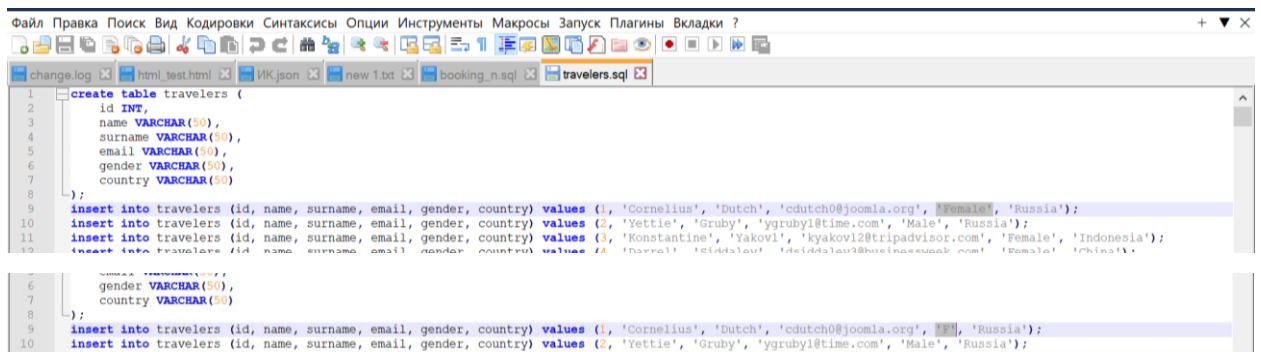
Ограничения можно накладывать не только на уникальные значения в столбцах, но и на сам формат написания этих значений в принципе.

```
select * from travelers;
```

id	name	surname	email	gender	country
2	Yettie	Gruby	ygruby1@time.com	Male	Russia
3	Konstantine	Yakovl	kyakovl2@tripadvisor.com	Female	Indonesia
4	Darrel	Siddale	dsiddale3@businessweek.com	Female	China
5	Olva	Leopard	oleopard4@skyrock.com	Male	China
6	Dalli	Chapelhow	dchapelhow5@godaddy.com	Female	China
7	Kerwinn	Lammas	klammas6@histats.com	Female	Thailand
8	Marie-jeanne	MacDougall	mmacdougall7@istockphoto.com	Female	Germany
9	Hyacintha	McCritchie	hmcritchie8@fc2.com	Female	Morocco
10	Jedidiah	Cayser	jcayser9@wordpress.com	Female	China
11	Clary	Rea	creaa@godaddy.com	Male	China
12	Ladonna	Comfort	lcomfortb@plala.or.jp	Male	Uganda
13	Ranique	Kulver	rkulverc@weebly.com	Male	Brazil
14	Mallory	Warham	mwarhamd@ebay.co.uk	Female	Yemen
15	Godwin	Gratrex	ggratrexe@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	sdykaj@java.com	Male	Greece
21	Yasmeen	Tanser	ytanserk@shareasale.com	Male	Finland

Рисунок 12

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



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';
```

```
SQL> delete from travelers where gender='F';
DELETE 1
SQL>
```

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');
ALTER TABLE
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:

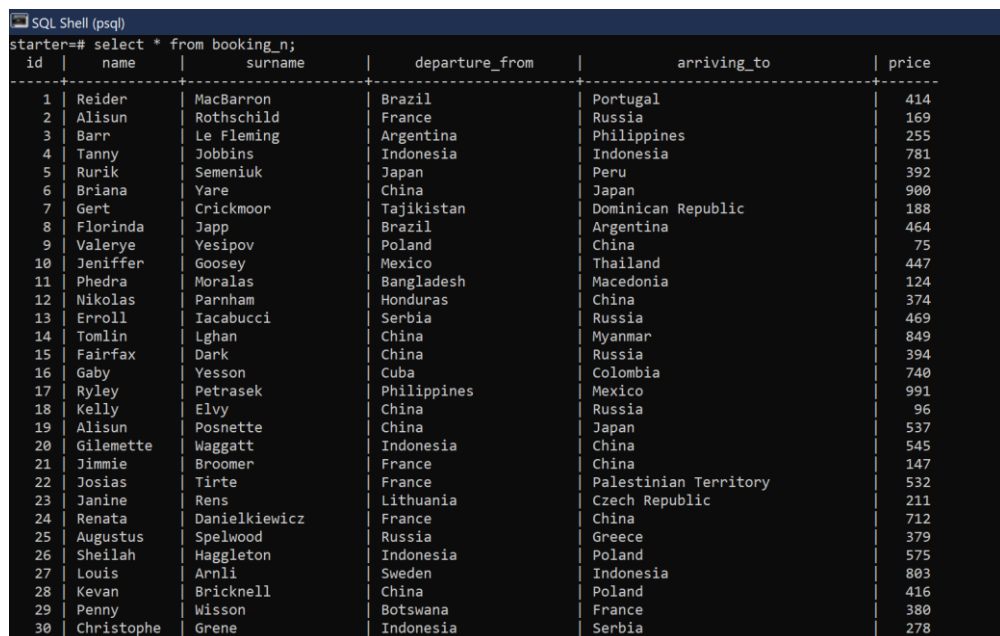
```
ALTER TABLE travelers ADD CONSTRAINT gender_check  
CHECK(gender = 'Female' OR gender = 'Male');
```

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;
```



id	name	surname	departure_from	arriving_to	price
1	Reider	MacBarron	Brazil	Portugal	414
2	Alisun	Rothschild	France	Russia	169
3	Barr	Le Fleming	Argentina	Philippines	255
4	Tanny	Jobbins	Indonesia	Indonesia	781
5	Rurik	Semeniuk	Japan	Peru	392
6	Briana	Vare	China	Japan	900
7	Gert	Crickmoor	Tajikistan	Dominican Republic	188
8	Florinda	Japp	Brazil	Argentina	464
9	Valerye	Yesipov	Poland	China	75
10	Jeniffer	Goosey	Mexico	Thailand	447
11	Phedra	Moralas	Bangladesh	Macedonia	124
12	Nikolas	Parnham	Honduras	China	374
13	Erroll	Iacobucci	Serbia	Russia	469
14	Tomlin	Lghan	China	Myanmar	849
15	Fairfax	Dark	China	Russia	394
16	Gaby	Yesson	Cuba	Colombia	740
17	Ryley	Petrasek	Philippines	Mexico	991
18	Kelly	Elvy	China	Russia	96
19	Alisun	Posnette	China	Japan	537
20	Gilemette	Waggatt	Indonesia	China	545
21	Jimmie	Broomer	France	China	147
22	Josias	Tirte	France	Palestinian Territory	532
23	Janine	Rens	Lithuania	Czech Republic	211
24	Renata	Danielkiewicz	France	China	712
25	Augustus	Spelwood	Russia	Greece	379
26	Sheilah	Haggleton	Indonesia	Poland	575
27	Louis	Arnli	Sweden	Indonesia	803
28	Kevan	Bricknell	China	Poland	416
29	Penny	Wisson	Botswana	France	380
30	Christophe	Grene	Indonesia	Serbia	278

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



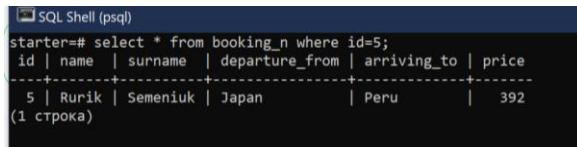
5	Rurik	Semeniuk	Japan
---	-------	----------	-------

Необходимо изменить это имя на настоящее.

Сделать мы это можем при помощи команды UPSERT. Во время выполнения этой команды мы можем уточнить где конкретно происходить

изменение с помощью той же логики, что мы использовали для вывода всей строки под номером 5.

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

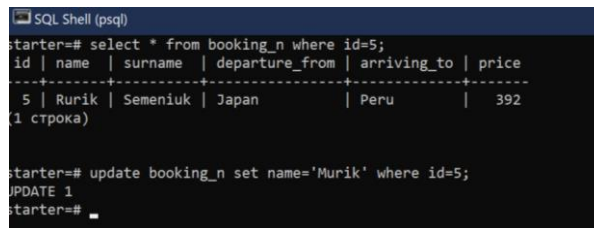


```
SQL Shell (psql)
starter=# select * from booking_n where id=5;
 id | name | surname | departure_from | arriving_to | price
-----+-----+-----+-----+-----+-----
  5 | Rurik | Semeniuk | Japan          | Peru        | 392
(1 строка)
```

Изменяем именно Rurik.

Введем запрос на изменение имени:

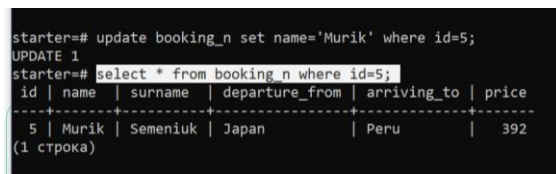
```
update booking_n set name='Murik' where id=5;
```



```
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.

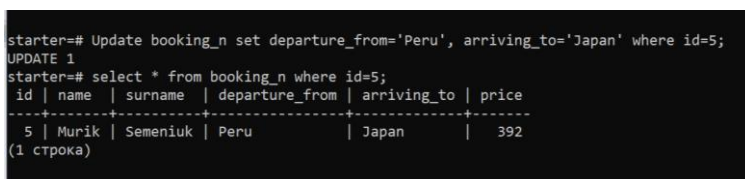


```
starter=# update booking_n set name='Murik' where id=5;
UPDATE 1
starter=# select * from booking_n where id=5;
 id | name | surname | departure_from | arriving_to | price
-----+-----+-----+-----+-----+-----
  5 | Murik | Semeniuk | Japan          | Peru        | 392
(1 строка)
```

Подобным образом можно изменить сразу несколько полей в таблице.

Допустим были перепутаны места вылета и прибытия

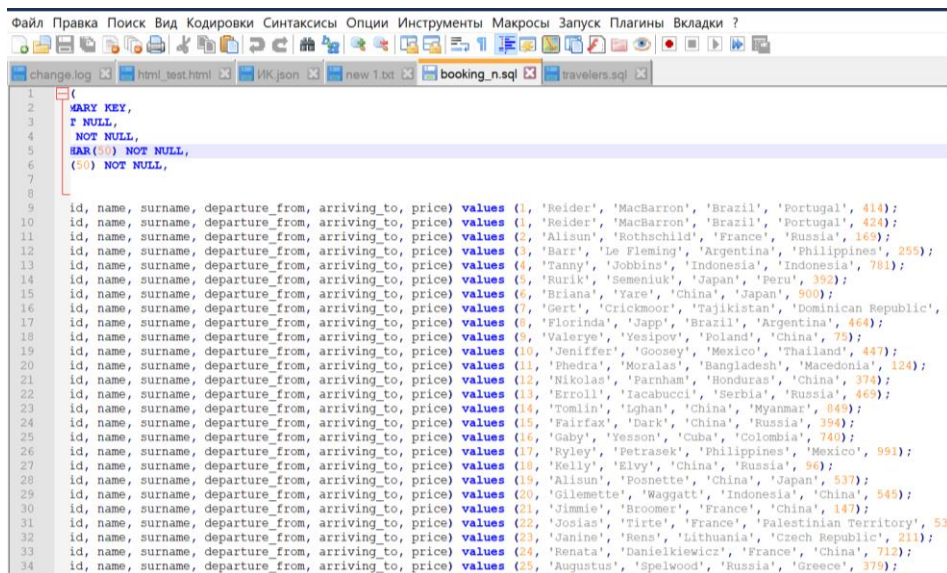
```
Update booking_n set departure_from='Peru', arriving_to='Japan' where id=5;
```



```
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 | Semeniuk | 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.



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=#
```

```

SQL Shell (psql)
id | name | surname | departure_from | arriving_to | price
-----+-----+-----+-----+-----+-----
5 | Murik | Semenik | Japan | Peru | 392
(1 строка)

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 | Semenik | Peru | Japan | 392
(1 строка)

starter=# insert into booking_n (id, name, surname, departure_from, arriving_to, price) values (5, 'Rurik', 'Semenik',
'Japan', 'Peru', 392);
ОШИБКА: повторяющееся значение ключа нарушает ограничение уникальности "booking_n_pkey"
ПОДРОБНОСТИ: Ключ "(id)=(5)" уже существует.
starter=# insert into booking_n (id, name, surname, departure_from, arriving_to, price) values (5, 'Rurik', 'Semenik',
'Japan', 'Peru', 392) on conflict (id) do nothing;
INSERT 0 0
starter=# select * from booking_n where id=5;
id | name | surname | departure_from | arriving_to | price
-----+-----+-----+-----+-----+-----
5 | Murik | Semenik | Peru | Japan | 392
(1 строка)

```

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

```

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

```

D:\ИнжинирингДанных\МоиКурсы\ПрактикаСам\booking_n.sql - Notepad++
Файл Правка Поиск Вид Кодировки Синтаксисы Опции Инструменты Макросы Запуск Плагины Вкладки ?
change.log html_test.html IK.json new.txt booking_n.sql travelers.sql
1 create table booking_n (
2     id int not null primary key,
3     name varchar(50) not null,
4     surname varchar(50) not null,
5     departure_from varchar(50) not null,
6     arriving_to varchar(50) not null,
7     price int not null
8 );
9 insert into booking_n (id, name, surname, departure_from, arriving_to, price) values (0, 'Rurik', 'Semenik', 'Japan', 'Peru', 392);
10 insert into booking_n (id, name, surname, departure_from, arriving_to, price) values (1, 'Reider', 'MacBarron', 'Brazil', 'Portugal', 414);
11 insert into booking_n (id, name, surname, departure_from, arriving_to, price) values (1, 'Reider', 'MacBarron', 'Brazil', 'Portugal', 424);
12 insert into booking_n (id, name, surname, departure_from, arriving_to, price) values (2, 'Alisun', 'Rothschild', 'France', 'Russia', 169);
13 insert into booking_n (id, name, surname, departure_from, arriving_to, price) values (3, 'Barr', 'Le Fleming', 'Argentina', 'Philippines', 255);
14 insert into booking_n (id, name, surname, departure_from, arriving_to, price) values (4, 'Tanny', 'Jobbins', 'Indonesia', 'Indonesia', 781);
15 insert into booking_n (id, name, surname, departure_from, arriving_to, price) values (5, 'Rurik', 'Semenik', 'Japan', 'Peru', 392);
16 insert into booking_n (id, name, surname, departure_from, arriving_to, price) values (6, 'Briana', 'Yare', 'China', 'Japan', 900);

```

```
SQL Shell (psql)
'Japan', 'Peru', 392) on conflict (id) do nothing;
INSERT 0 0
starter=# select * from booking_n where id=5;
 id | name | surname | departure_from | arriving_to | price 
-----+-----+-----+-----+-----+-----
  5 | Murik | Semeniyuk | Peru           | Japan       | 392
(1 строка)

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

starter=# insert into booking_n (id, name, surname, departure_from, arriving_to, price) values (0, 'Rurik', 'Semeniyuk',
'Japan', 'Peru', 392);
INSERT 0 1
starter=#
```

Но стоит нам заменить переменную `id`, например, на `surname`, не связанный ни какими правилами, то консоль сразу выдаст ошибку.

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

```
starter=# insert into booking_n (id, name, surname, departure_from, arriving_to, price) values (0, 'Rurik', 'Semeniyuk',
'Japan', 'Peru', 392);
INSERT 0 1
starter=# insert into booking_n (id, name, surname, departure_from, arriving_to, price) values (5, 'Rurik', 'Semeniyuk',
'Japan', 'Peru', 392) on conflict (surname) do nothing;
ОШИБКА: нет уникального ограничения или ограничения-исключения, соответствующего указанию ON CONFLICT
starter=#
```

Во время использования команды `on conflict (id) do` мы можем применить команду `UPSERT` для обновления некоторых полей таблиц.

В текстовом редакторе создадим дубликат строки с `id=0` и меняем страну отправления и заносим в таблицу два раза.

В команде `on conflict (id) do` меняем на то действие, которое нам будет необходимо применить при появлении пассажира с дубликатом `id`.

```
name, surname, departure_from, arriving_to, price) values (0, 'Rurik', 'Semeniyuk', 'Japan', 'Peru', 392);
RE id = 0;
om | arriving_to | price 
-----+-----+-----
 | Peru         | 392

name, surname, departure_from, arriving_to, price) values (0, 'Rurik', 'Semeniyuk', 'Spain', 'Peru', 392);
нарушает ограничение уникальности "booking_n_pkey"
есть.
name, surname, departure_from, arriving_to, price) values (0, 'Rurik', 'Semeniyuk', 'Spain', 'Peru', 392) ON CONFLICT (id) DO UPDATE SET departure_from =
```

insert into booking_n (id, name, surname, departure_from, arriving_to, price) values (5, 'Rurik', 'Semeniuk', 'Japan', 'Peru', 392) on conflict (id) do update set departure_from=excluded.departure_from;

```
starter=# insert into booking_n (id, name, surname, departure_from, arriving_to, price) values (5, 'Rurik', 'Semeniuk', 'Japan', 'Peru', 392) on conflict (id) do update set departure_from=excluded.departure_from;
INSERT 0 1
starter=#
```

UPDATE

Команда **UPDATE** *имя_таблицы* **SET** *имя_столбца = выражение* **WHERE** *логическое_утверждение*

позволит вносить изменения в уже существующие записи внутри таблицы данных.

Примеры :

```
UPDATE booking_n SET name = 'Max' WHERE id = 1;
UPDATE booking_n SET departure_from = 'Russia', arriving_to = 'Kazakhstan'
WHERE id = 1;
```

Для задания с внешними ключами нам нужно создать сравнительно не большую таблицу с адресами путешественников из таблицы travelers.

Как пример, создаем таблицу address.

```
SQL Shell (psql)
starter=# CREATE TABLE address(
starter(# id BIGSERIAL NOT NULL PRIMARY KEY,
starter(# street VARCHAR(50),
starter(# house VARCHAR(50));
CREATE TABLE
starter=#
```

Наша цель связать эти две таблицы. Используем уникальное id из таблицы travelers и id из таблицы address.

```

SQL Shell (psql)
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) |                     |                     |
Индексы:
 "unique_id" UNIQUE CONSTRAINT, btree (id)
Ограничения-проверки:
 "gender_constraint" CHECK (gender::text = 'Female'::text OR gender::text = 'Male'::text)

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)

starter=#

```

Для этого в таблице `travelers` создаем второй столбец с `id`, который будет использован для соединения двух таблиц.

Данные изменения мы делаем с помощью команды `alter table` добавляем поле с именем `address_id`. В команде также `REFERENCES`

Команда ADD [имя_столбца] [тип_данных] REFERENCES [имя_таблицы](имя_столбца_в_таблице) - создает внешний ключ в таблице в виде нового столбца. Он указывает на уже существующий столбец в другой таблице.

Пример :

```
ALTER TABLE booking_n ADD new_booking_id BIGSERIAL REFERENCES booking(id);
```

```

starter=# ALTER TABLE travelers ADD address_id BIGINT REFERENCES address (id);
ALTER TABLE
starter=#

```

Это создаст правило по ограничениям внешнего ключа. Если мы посмотрим на текущее правило таблицы `travelers`, то увидим его ограничение внешнего ключа.

```

starter=# ALTER TABLE travelers ADD address_id BIGINT REFERENCES address (id);
ALTER TABLE
starter=# \d 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)			
address_id	bigint			

```

Индексы:
"unique_id" UNIQUE CONSTRAINT, btree (id)
Ограничения-проверки:
"gender_constraint" CHECK (gender::text = 'Female'::text OR gender::text = 'Male'::text)
Ограничения внешнего ключа:
"travelers_address_id_fkey" FOREIGN KEY (address_id) REFERENCES address(id)

starter=#

```

В текущих правилах таблицы address также появляется информация о новой созданной ссылке из вне.

```

SQL Shell (psql)
starter=# \d 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 извне:
TABLE "travelers" CONSTRAINT "travelers_address_id_fkey" FOREIGN KEY (address_id) REFERENCES address(id)

starter=#

```

Для примера создадим несколько записей для таблицы address.

```

starter=# INSERT INTO address (id, street, house) VALUES (1, 'Groove Street', 'Central House');
INSERT 0 1
starter=# INSERT INTO address (id, street, house) VALUES (2, 'Davis Avenue', 'Left House');
INSERT 0 1
starter=# INSERT INTO address (id, street, house) VALUES (3, 'Capital Boulevard', 'Right House');
INSERT 0 1
starter=# SELECT * FROM address;

```

id	street	house
1	Groove Street	Central House
2	Davis Avenue	Left House
3	Capital Boulevard	Right House

(3 строки)

Таблица address ссылается на столбец address_id в таблице travelers. Но на данный момент о нем нет никакой информации.

```

starter=# SELECT * FROM travelers LIMIT (3);

```

id	name	surname	email	gender	country	address_id
2	Yettie	Gruby	ygruby1@time.com	Male	Russia	
3	Konstantine	Yakovl	kyakovl2@tripadvisor.com	Female	Indonesia	
4	Darrel	Siddaley	dsiddaley3@businessweek.com	Female	China	

(3 строки)

```

starter=# UPDATE travelers SET address_id = 1 WHERE id = 2;
UPDATE 1
starter=#

```

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 | kyakov12@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 извне:
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	Yakov	kyskov12@tripadvisor.com	Female	Indonesia				
4	Darrel	Siddale	dsiddale3@businessweek.com	Female	China				
5	Olva	Leopard	oleopard4@skyrock.com	Male	China				
6	Dalli	Chapelhow	dchapelhow5@godaddy.com	Female	China				
7	Kerwinn	Lammas	klammas6@histats.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	Comfort	lcomfort@plala.or.jp	Male	Uganda				
13	Ranique	Kulver	rkulverc@weebly.com	Male	Brazil				
14	Mallory	Warham	mwarhamd@ebay.co.uk	Female	Yemen				
15	Godwin	Gratrex	ggratrexe@networksolutions.com	Male	China				
16	Flory	Mayhead	fmayheadf@gnu.org	Female	Panama				
17	Hunter	Anniwell	hanniwel1@over-blog.com	Female	Russia				
18	Brannon	Cottisford	bcottisfordh@yellowpages.com	Female	China				
19	Gale	Dj Nisco	gdjniscoi@umich.edu	Male	United Kingdom				
20	Salomo	Dyka	sdykaj@java.com	Male	Greece				
21	Yasmeen	Tanser	ytanser@shareasale.com	Male	Finland				
22	Rodie	Garritty	rgarritty1@pcworld.com	Male	Greece				
23	Maggy	Wiggett	mwiggett@wix.com	Male	Peru				
24	Jody	Rallin	jrallin@exblog.jp	Female	Brazil				
25	Berti	Najara	bnajara@wix.com	Male	China				
26	Calvarion	Viner	cviner@calvinia.edu	Male	Philippines				

Как мы видим, таблица путешественников показалась полностью, создав три столбца из таблицы адресов. Данные столбцы пусть из 999 случаев из 1000 и в основном, только, мешают общему восприятию.

Можно отдельно указать для команды Left join, что нам нужны поля только с заполненными полями address_id и получим тот же вывод, что и обычный join

```
starter=# SELECT * FROM travelers
starter=# LEFT JOIN address ON address.id = travelers.address_id WHERE address_id IS NOT NULL;
```

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)

Команда right join полностью покажет ту таблицу, которую присоединяют

```
starter=# SELECT * FROM travelers RIGHT JOIN address ON address.id = travelers.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
							2	Davis Avenue	Left House
							3	Capital Boulevard	Right House

(3 строки)

```
starter=# _
```

Последние две команды можно соединить командой full join

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

Она отобразит полностью и правую и левую таблицы.

SQL Shell (psql)										
id	name	surname	email	gender	country	address_id	id	street	house	
3	Konstantine	Yakovl	kyakov12@tripadvisor.com	Female	Indonesia					
4	Darrel	Siddaley	dsiddaley3@businessweek.com	Female	China					
5	Olva	Leopard	oleopard4@skyrock.com	Male	China					
6	Dall	Chapelhow	dchapelhow5@godaddy.com	Female	China					
7	Kerwinn	Lamas	klamas6@histats.com	Female	Thailand					
8	Marie-jeanne	MacDougall	mmacdougall7@istockphoto.com	Female	Germany					
9	Hyacintha	McCritchie	hmccritchie8@fc2.com	Female	Morocco					
10	Jedidah	Cayser	jcayser9@wordpress.com	Female	China					
11	Clary	Rea	creaa@godaddy.com	Male	China					
12	Ladonna	Comfort	lcomfortb@plala.or.jp	Male	Uganda					
13	Ranique	Kulver	rkulverc@weebly.com	Male	Brazil					
14	Mallory	Karham	mwarhamd@ebay.co.uk	Female	Yemen					
15	Godwin	Gratrex	ggratrexe@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	sdykaj@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@wix.com	Male	China					
26	Sebastien	Vines	svinesp@columbia.edu	Male	Philippines					
27	Harwell	Aiers	haiersq@fgate.com	Female	Nigeria					
28	Frederigo	Moorey	fmooreyr@flickr.com	Male	Indonesia					
29	Phyllis	Leyborne	pleybornes@foxnews.com	Male	Peru					
30	Sharia	Pantin	spantin@flavors.me	Female	France					
31	Rhonda	Ever	reverduy@cominator.com	Male	Sierra Leone					
32	Trudie	Westwater	twestwater@rambler.ru	Male	China					
33	Wallis	Featherston	wfeatherstonw@facebook.com	Male	Philippines					
34	Cyb	Mollon	cmollonx@discovery.com	Male	Poland					
35	Sara	Phillcox	sphillcox@ebc.ca	Male	Portugal					
36	Eolande	Cadwell	ecadwellz@nydailynews.com	Male	United States					
37	Darya	Crackett	dcrackett10@netvibes.com	Male	Indonesia					
38	Tessie	Griffen	igriffen11@auda.org.au	Male	Philippines					

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 уместается 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	cdutch0@joomla.org	Female	Russia	1
2	Yettie	Gruby	ygruby1@time.com	Male	Russia	
3	Konstantine	Yakovl	kyakov12@tripadvisor.com	Female	Indonesia	
4	Darrel	Siddaley	dsiddaley3@businessweek.com	Female	China	
5	Olva	Leopard	oleopard4@skyrock.com	Male	China	
6	Dalli	Chapelhow	dchapelhow5@godaddy.com	Female	China	
7	Kerwinn	Lammas	klammas6@histats.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	Comfort	lcomfortb@pala.or.jp	Male	Uganda	
13	Ranique	Kulver	rkulverc@eebly.com	Male	Brazil	
14	Mallory	Warham	mwarhamd@ebay.co.uk	Female	Yemen	
15	Godwin	Gratrex	ggratrexe@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	sdykaj@java.com	Male	Greece	
21	Yasmeen	Tanser	ytanserk@shareasale.com	Male	Finland	
22	Rodie	Garrity	rgarrityl@pcworld.com	Male	Greece	
23	Maggy	Wiggett	mwiggettm@wix.com	Male	Peru	
24	Jody	Rallin	jrallinn@exblog.jp	Female	Brazil	
25	Berti	Najara	bnajarao@wix.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	twestwaterv@rambler.ru	Male	China	
33	Wallis	Featherston	wfeatherstom@facebook.com	Male	Philippines	
34	Cyb	Mollon	cmollonx@discovery.com	Male	Poland	
35	Sara	Phillcox	sphillcoxy@cbsc.ca	Male	Portugal	
36	Eolande	Cadwell	ecadwell2@nydailynews.com	Male	United States	
37	Darya	Crackett	dcrackett10@netvibes.com	Male	Indonesia	
38	Jessee	Griffen	jgriffen11@auda.org.au	Male	Philippines	
39	Sargent	Lipman	slipman12@mit.edu	Female	Uruguay	
40	Karin	Millroy	kmilroy13@yolasite.com	Female	Indonesia	
41	Bartholemy	Hilliam	bhilliam14@merriam-webster.com	Female	Indonesia	
42	Margaretha	Bauckham	mbauckham15@mapquest.com	Female	Colombia	
43	Cristen	Filippone	cfilippone16@hp.com	Female	Russia	
44	Britta	Alfonso	balfonso17@paypal.com	Male	Indonesia	
45	Marc	Sant	msant18@devhub.com	Male	China	
46	Rasia	Arnold	rarnold19@msu.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	
ны	Общие видеозаписи	19.03.2019 7:52	Папка с файлами	
ения	Общие документы	10.09.2021 11:42	Папка с файлами	
ive	Общие загруженные файлы	19.03.2019 7:52	Папка с файлами	
oud Files	Общие изображения	19.03.2019 7:52	Папка с файлами	
ж	Общий рабочий стол	02.09.2021 15:53	Папка с файлами	
нотер				
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.



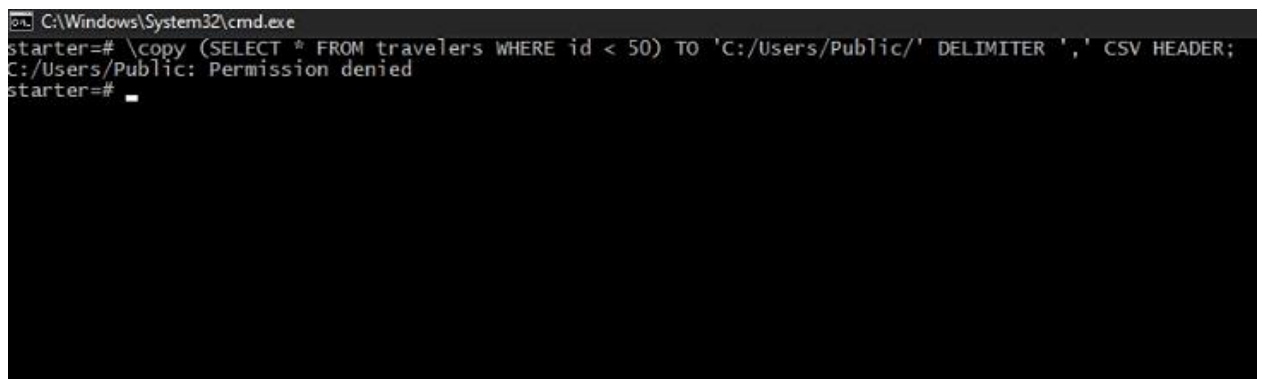
4. Для экспорта мы используем функцию `\copy`, с командой выборки, заключенной в круглые скобки. Команда **TO** указывает на папку, куда будет скопирована выборка. После неё мы вставляем видоизмененный нами полный путь к файлу.

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
postgres=# \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**”.

A screenshot of a Windows command prompt window. The title bar reads 'C:\Windows\System32\cmd.exe'. The command prompt shows the following text: 'starter=# \copy (SELECT * FROM travelers WHERE id < 50) TO 'C:/Users/Public/' DELIMITER ',' CSV HEADER;', 'C:/Users/Public: Permission denied', and 'starter=#' followed by a cursor. The background is black and the text is white.

```
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.txt	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,creaa@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,ytanser@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,Federigo,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,twestwater@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,dcrackettl0@netvibes.com,Male,Indonesia,
39	38,Jessee,Griffen,jgriffenl1@auda.org.au,Male,Philippines,
40	39,Sargent,Lipman,slipmanl2@mit.edu,Female,Uruguay,
41	40,Karin,Milroy,kmilroyl3@yolasite.com,Female,Indonesia,
42	41,Bartholemy,Hilliam,bhilliaml4@merriam-webster.com,Female,Indonesia,
43	42,Margaretha,Bauckham,mbauckhaml5@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,msantl8@devhub.com,Male,China,
47	46,Rasia,Arnold,rarnoldl9@msu.edu,Male,Egypt,
48	47,Christian,Stonham,cstonhamla@mozilla.com,Female,Philippines,
49	48,Rosaleen,Checuzzi,rcheccuzzi1b@google.com.hk,Female,Russia,
50	49,Lib,Bilson,lbilsonlc@de.vu,Female,Greece,
51	

Лабораторная работа № 8-9.

Знакомство с базой данных «Авиаперевозки»

Простые запросы

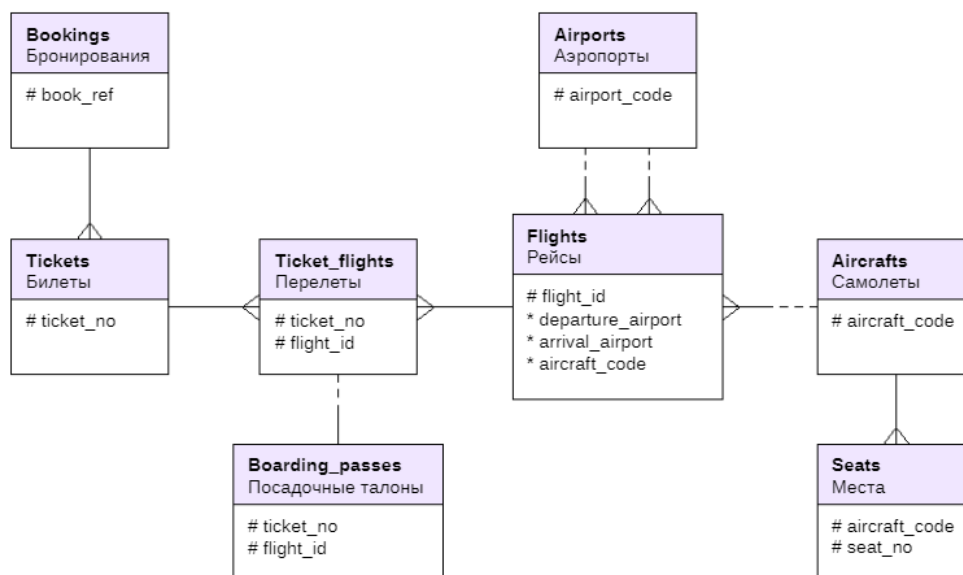
Цель работы: изучить схему данных БД «Авиаперевозки». Познакомится с платформой DBeaver.

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.

```

PS 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	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	представление	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.

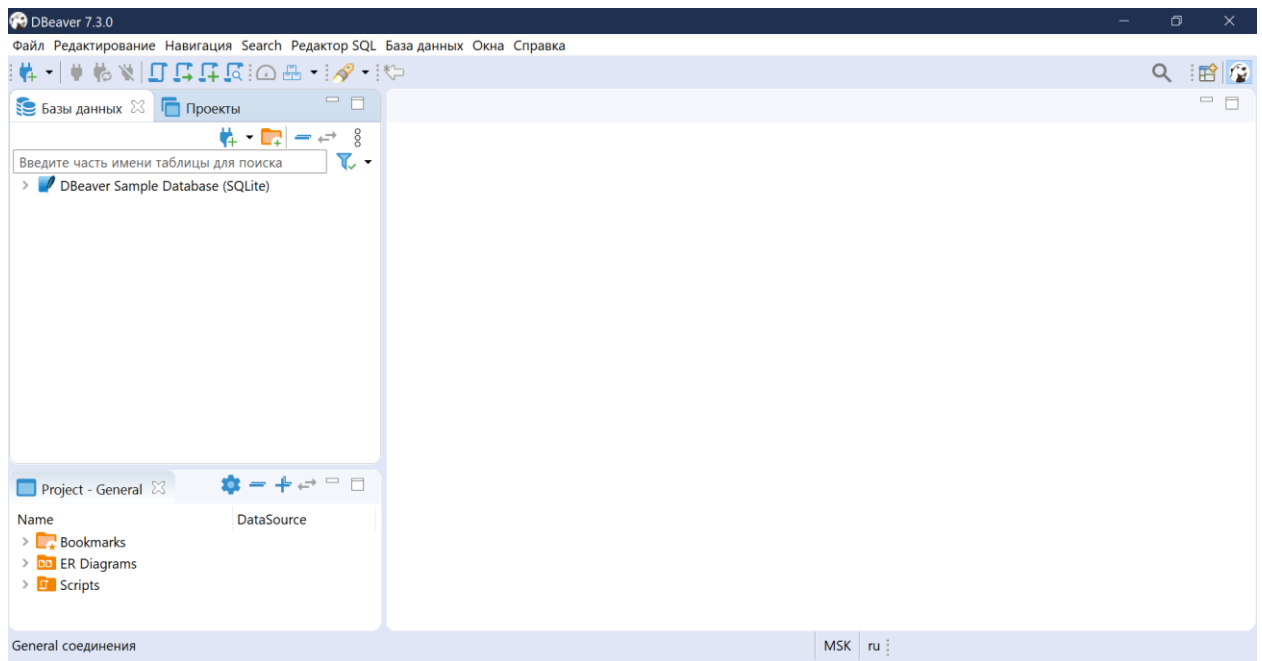


Рисунок 1 - Окно интерфейса DBeaver

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

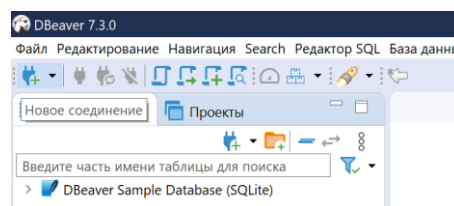


Рисунок 2 – Подключение БД

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

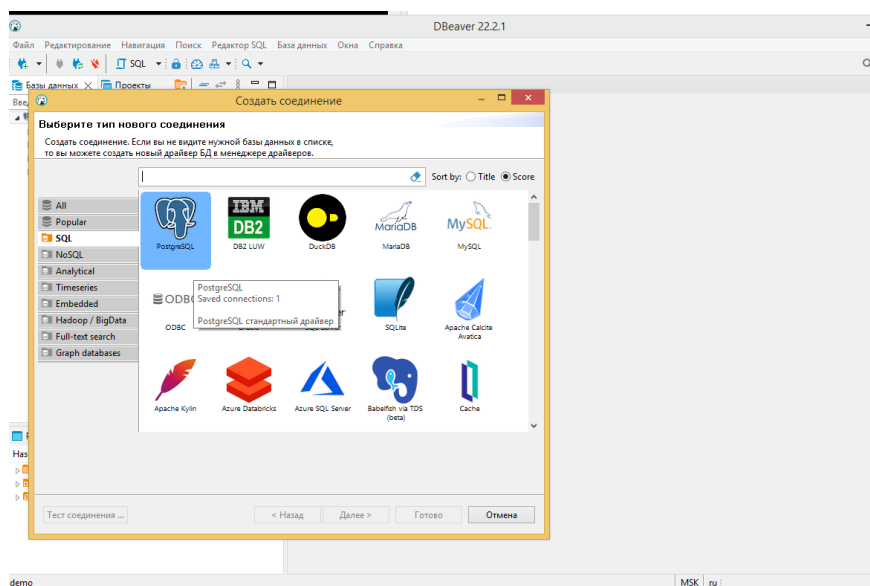


Рисунок 3 – Окно выбора нового соединения

7. В следующем окне (рисунок 4) необходимо провести настройку соединения. В поле База данных введите название БД demo как на рисунке 4. Далее нажмите кнопку «Настройка драйвера». И выберите драйвер как на рисунке 5

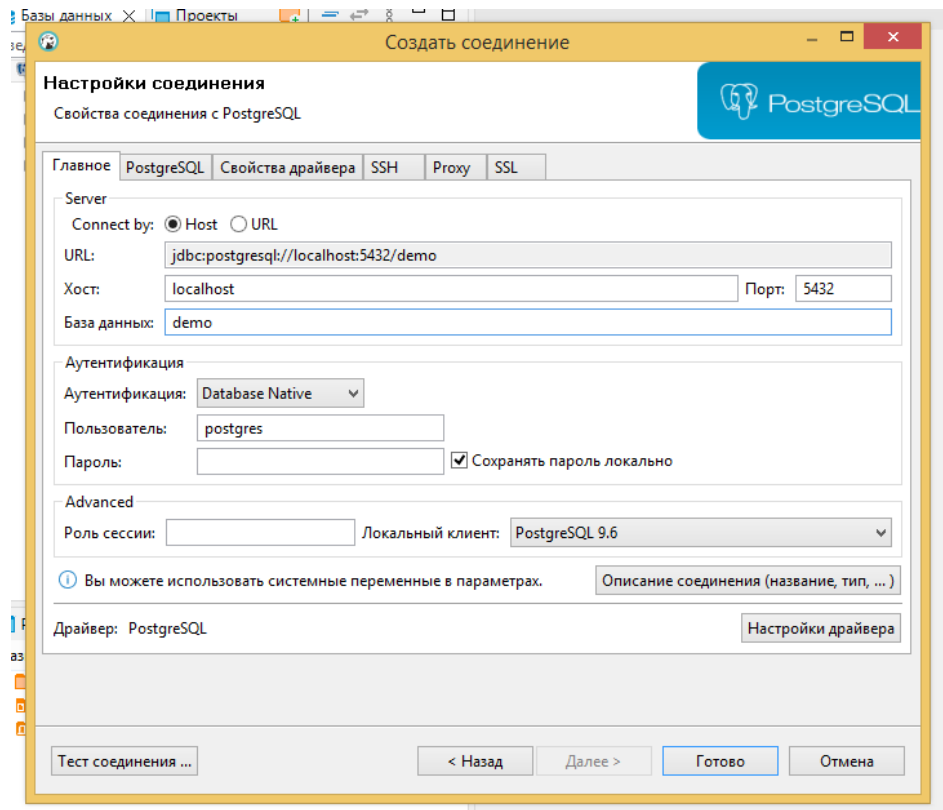


Рисунок 4 – Настройка соединения

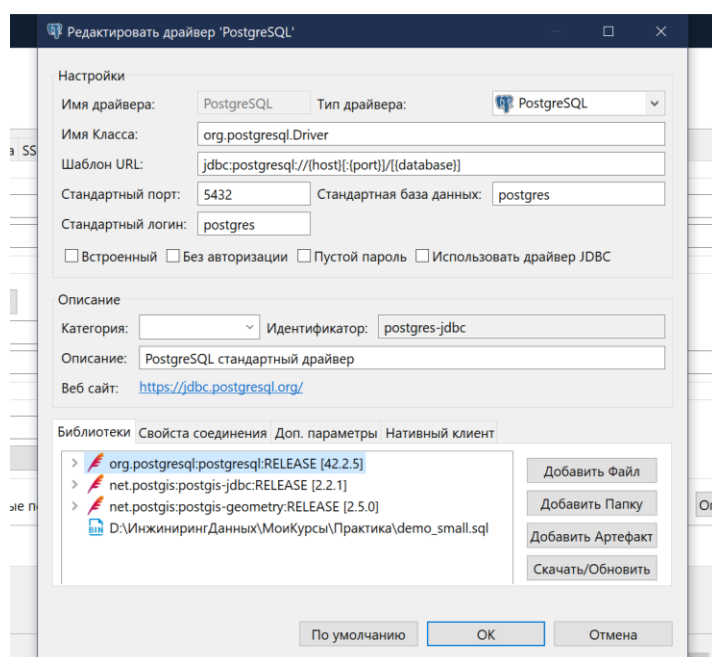


Рисунок 5 – Выбор драйвера

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).

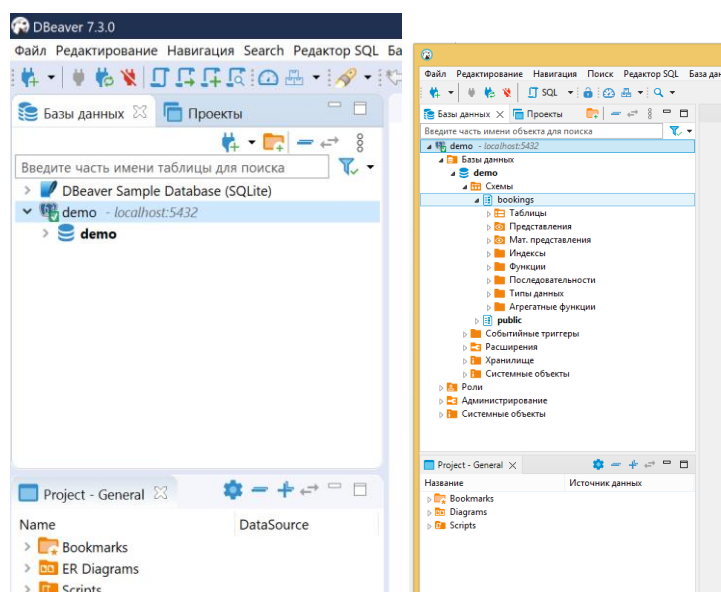


Рисунок 6 – Подключение БД

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).

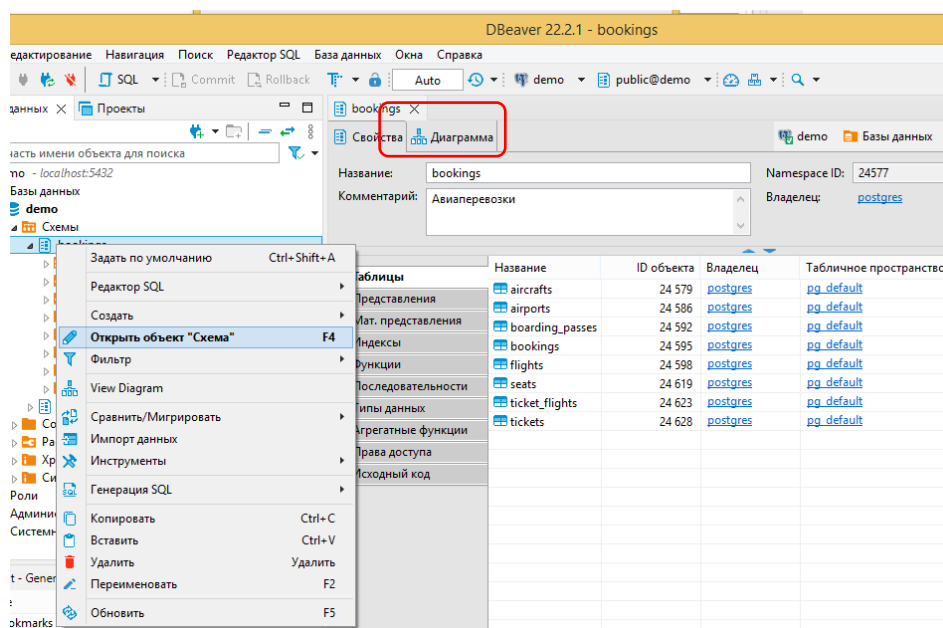


Рисунок 7 – Открытие Схемы данных

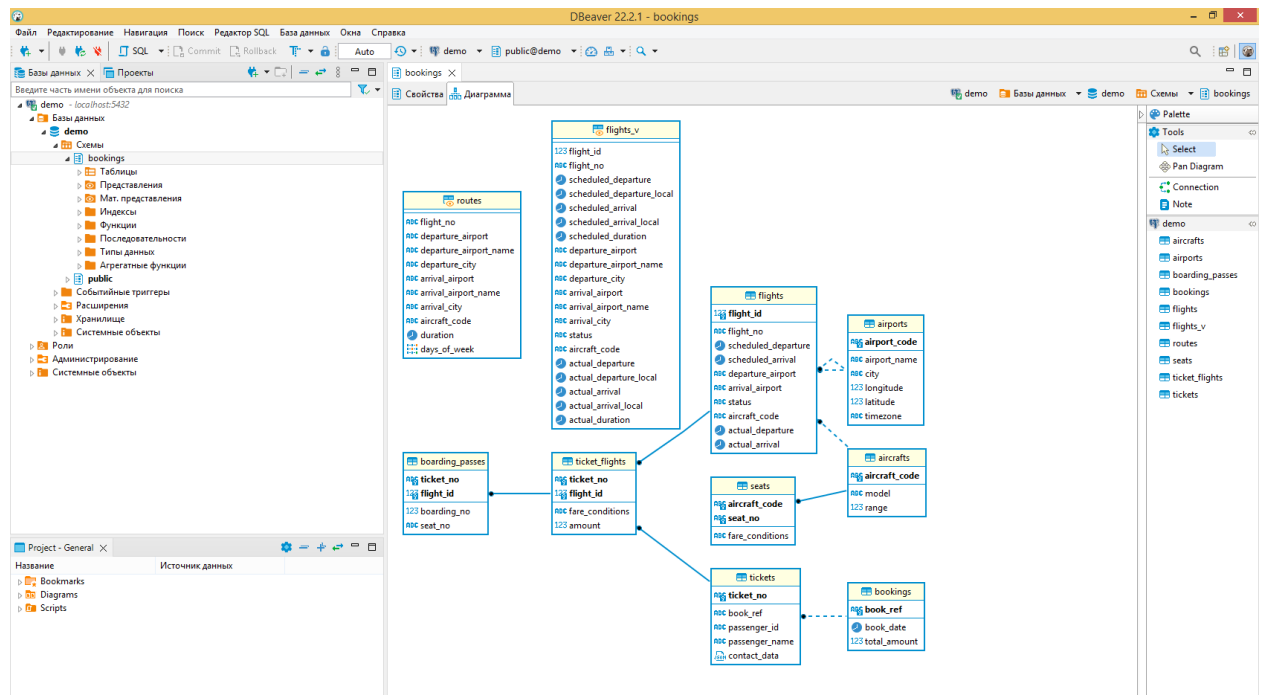


Рисунок 8 – Схема данных БД

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).

book_ref	book_date	total_amount
000012	2016-08-02 00:12:00.000 +0300	265 700
000008	2016-08-11 08:02:00.000 +0300	37 900
000008	2016-10-13 13:27:00.000 +0300	16 100
000101	2016-10-08 12:28:00.000 +0300	131 800
000026	2016-10-25 20:40:00.000 +0300	23 800
000028	2016-09-29 03:30:00.000 +0300	101 500
000020	2016-08-08 15:09:00.000 +0300	89 800
000073	2016-09-27 04:31:00.000 +0300	49 800
000046	2016-10-02 15:52:00.000 +0300	73 900
000032	2016-09-03 01:02:00.000 +0300	109 500
000074	2016-10-10 08:13:00.000 +0300	136 200
000400	2016-09-29 23:34:00.000 +0300	6 800
000406	2016-09-14 04:39:00.000 +0300	140 100
000480	2016-09-25 08:00:00.000 +0300	12 000
000481	2016-09-28 18:34:00.000 +0300	139 900
000011	2016-08-29 02:40:00.000 +0300	26 700
000039	2016-10-04 00:15:00.000 +0300	6 000
000046	2016-09-12 04:54:00.000 +0300	50 700
000027	2016-08-31 08:25:00.000 +0300	28 800
000044	2016-10-07 01:14:00.000 +0300	95 400
000039	2016-09-15 02:27:00.000 +0300	56 900
000078	2016-09-09 01:35:00.000 +0300	296 700
0000C3	2016-09-30 10:42:00.000 +0300	59 400
0000F3	2016-10-02 15:10:00.000 +0300	60 200
000075	2016-09-05 11:19:00.000 +0300	51 800
000079	2016-09-03 17:05:00.000 +0300	32 800
000074	2016-09-28 05:15:00.000 +0300	28 800
000749	2016-09-11 07:17:00.000 +0300	43 200
000750	2016-09-07 18:18:00.000 +0300	137 700
0007C5	2016-09-01 13:09:00.000 +0300	75 400
000036	2016-10-10 22:28:00.000 +0300	23 400
000042	2016-09-06 21:55:00.000 +0300	81 700
000039	2016-09-02 02:12:00.000 +0300	234 800
000062	2016-10-05 13:23:00.000 +0300	45 500
0000DF	2016-09-22 11:33:00.000 +0300	92 400
000044	2016-09-29 23:33:00.000 +0300	29 100
0000FD	2016-09-01 14:36:00.000 +0300	24 400
000009	2016-09-29 03:30:00.000 +0300	99 700
000077	2016-09-22 22:01:00.000 +0300	28 000
000048	2016-10-13 02:38:00.000 +0300	67 000
000048	2016-09-30 01:05:00.000 +0300	38 800

Рисунок 9 – Таблица Bookings (данные таблицы)

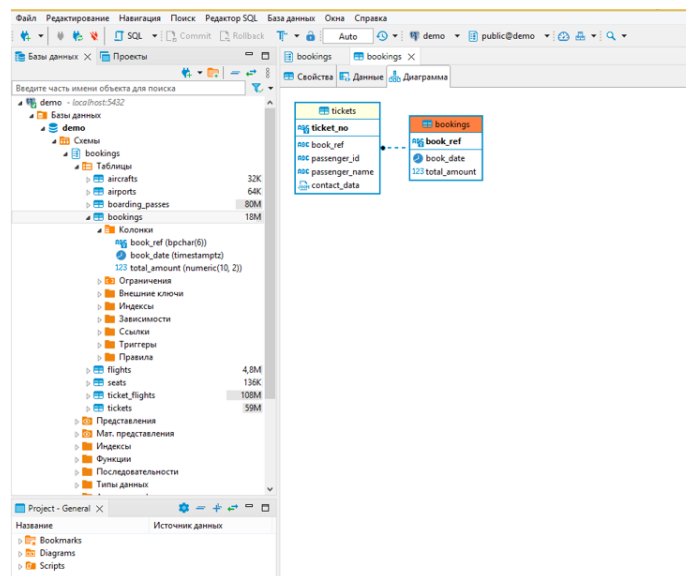


Рисунок 10 – Диаграмма таблицы Bookings

Колонки	Название	# Тип данных	Автоувеличение	Правило сортировки	Not Null	По умолчанию	Комментарий
Ограничения	book_ref	1 bpchar(6)		default	[v]		Номер бронирования
Внешние ключи	book_date	2 timestampz			[v]		Дата бронирования
Индексы	total_amount	3 numeric(10, 2)			[v]		Полная сумма бронирования

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).

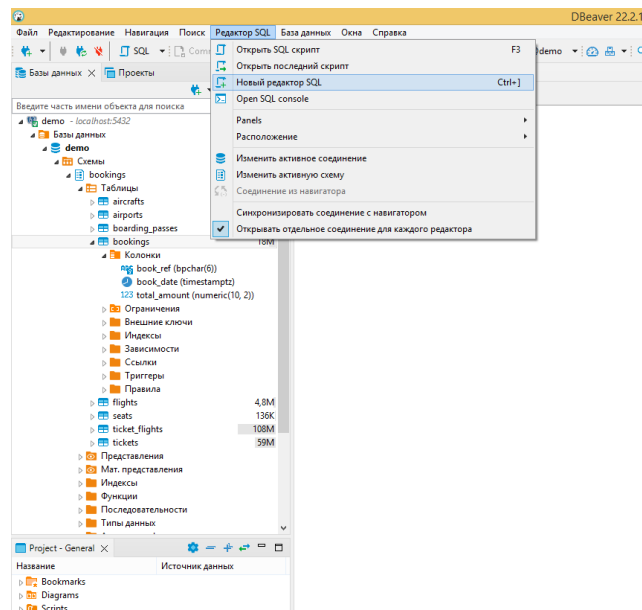


Рисунок 12 – Открытие редактора SQL

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

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

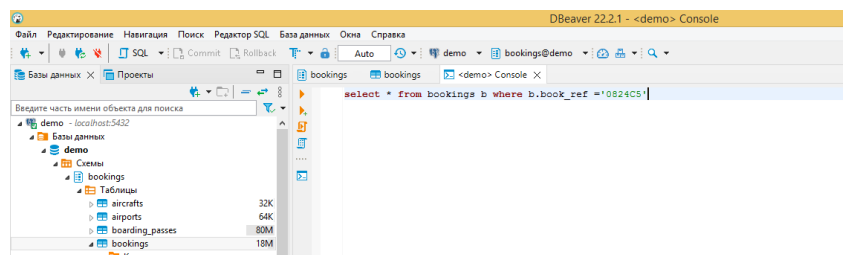


Рисунок 13 – Запрос о бронировании

Мы видим дату бронирования и общую сумму.

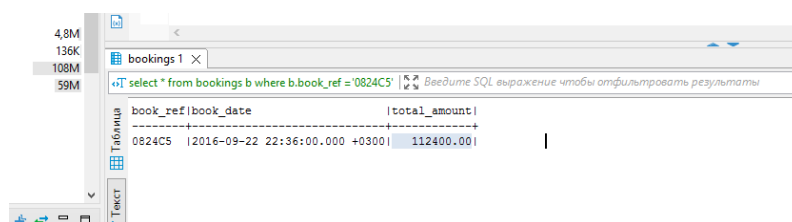


Рисунок 14 – Результат запроса бронирования

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

```
select now();
```

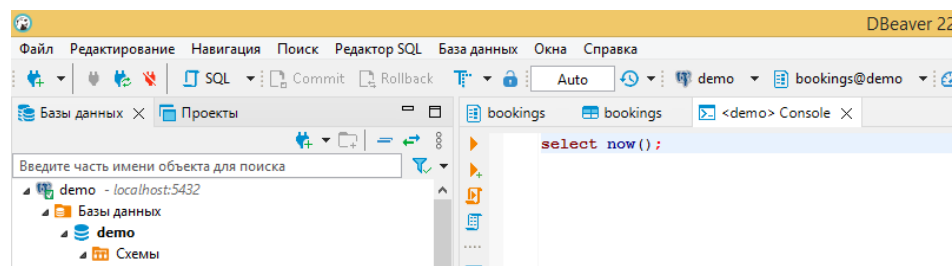


Рисунок 15 – Запрос текущей даты

The screenshot shows the DBBeaver 22 interface with the SQL editor containing the query 'select * from bookings b where b.book_ref = '0824C5''. The result set is displayed in a table with three columns: 'book_ref', 'book_date', and 'total_amount'.

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();
```

The screenshot shows the DBBeaver 22 interface with the SQL editor containing the query 'select bookings.now()'. The result set is displayed in a table with one column: 'now'.

now
2016-10-13 17:00:00.000 +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';
```

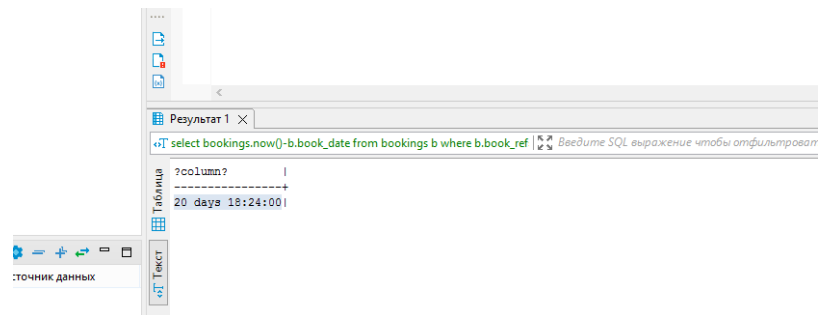


Рисунок 18 – Результат запроса

Итак, билеты были забронированы «20 дней назад».

Посмотрим следующую таблицу Билеты (Tickets).

DBeaver 22.2.1 - tickets

tickets

ID объекта: 16574

Владелец: postgres

Дополнительные опции:

Название	#	Тип данных	Автоувеличение	Правило сортировки	Not Null	По умолчанию	Комментарий
ticket_no	1	bpchar(13)		default	[v]		Номер билета
book_ref	2	bpchar(6)		default	[v]		Номер бронирования
passenger_id	3	varchar(20)		default	[v]		Идентификатор пассажира
passenger_name	4	text		default	[v]		Имя пассажира
contact_data	5	jsonb			[]		Контактные данные пассаж...

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'
```

Результат запроса:

	ticket_no	book_ref	passenger_id	passenger_name	contact_data
1	0005435126781	0824C5	7247 393204	ALEKSANDR MATVEEV	{"phone": "+70095062310"}
2	0005435126782	0824C5	1745 826066	NINA KRASNOVA	{"phone": "+70876976071"}

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).

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	10			default	☒		Идентификатор рейса
fare_conditions	3	varchar	10	10			default	☒		Класс обслуживания
amount	4	numeric		10	2			☒		Стоимость перелета

Рисунок 19 – Структура таблицы Перелеты (Ticket_flights)

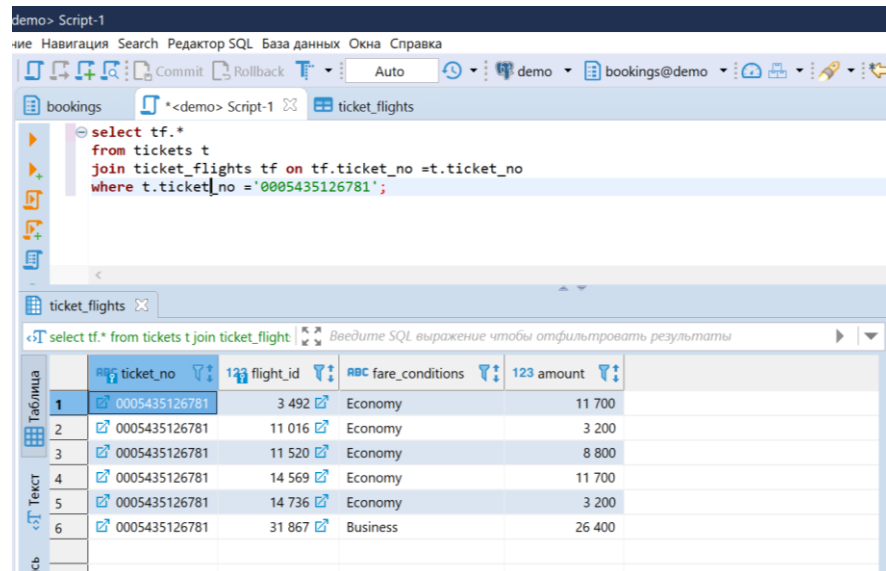
Перелет соединяет билет с рейсом и идентифицируется их номерами. Для каждого перелета указывают его стоимость (amount) и класс обслуживания (fare_conditions).



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';
```



The screenshot shows a database interface with a query editor and a results pane. The query is:

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

The results pane displays the following data:

	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	14 569	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

во: pg_default

Владелец: postgres

Название	#	Тип данных	Длина	Точность	Масштаб	Автоувеличение	Правило сортировки	Not Null	По умолчанию	Комментарий
 flight_id	1	serial						☒	nextval('booki...	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)

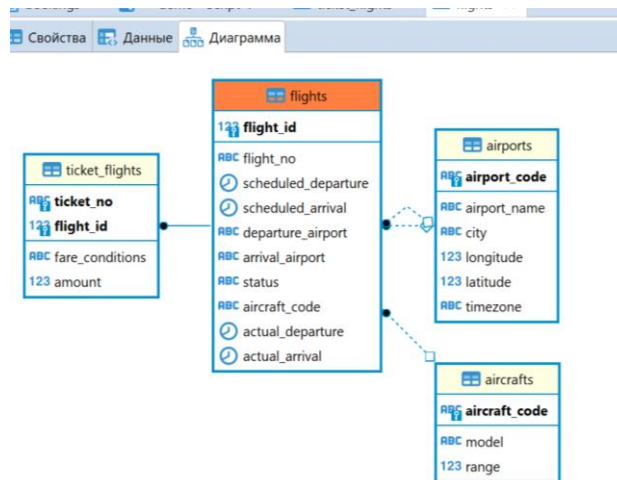


Рисунок 23 – Схема данных таблицы Рейсы (Flights)

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 ;
  
```

```

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 ;

```

	flight_id	scheduled_departure	dep	arr	status	aircraft
1	3 492	2016-10-10 10:00:00	VKO	PEE	Arrived	773
2	14 736	2016-10-10 14:30:00	PEE	SVX	Arrived	SU9
3	11 520	2016-10-11 10:30:00	SVX	SGC	Arrived	SU9
4	31 867	2016-10-13 13:45:00	SGC	SVX	Departed	SU9
5	11 016	2016-10-14 07:50:00	SVX	PEE	On Time	SU9
6	14 569	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.

```

select * from flights f where f.flight_id =22566;

```

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	SU9
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 a database interface with a query window and a results table. The query is: `select f.flight_id, f.flight_no, f.scheduled_dep`. The results table has three columns: `flight_id`, `flight_no`, and `scheduled_departure`. It displays 10 rows of data for flight PG0412, showing flight IDs ranging from 3490 to 3485 and scheduled departure times from 2016-09-13 10:00:00 to 2016-09-22 10:00:00.

flight_id	flight_no	scheduled_departure
3490	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)

The screenshot shows the 'airports' table properties. It includes fields for 'airport_code', 'airport_name', 'city', 'longitude', 'latitude', and 'timezone'. The table has 6 columns. The 'airport_code' field is a 3-character string. The 'city' field is a text field. The 'longitude' and 'latitude' fields are float8 types. The 'timezone' field is a text field. The table is owned by 'postgres' and has a primary key on 'airport_code'.

Название	#	Тип данных	Длина	Точность	Масштаб	Автоувеличение	Правило сортировки	Not Null	По умолчанию	Комментарий
airport_code	1	bpchar	3	3			default	☒		Код аэропорта
airport_name	2	text					default	☒		Название аэропорта
city	3	text					default	☒		Город
longitude	4	float8		17	17			☒		Координаты аэропорта: дол...
latitude	5	float8		17	17			☒		Координаты аэропорта: ши...
timezone	6	text					default	☒		Временная зона аэропорта

Рисунок 27

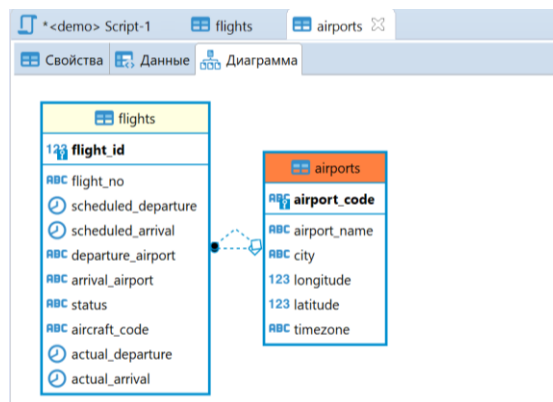


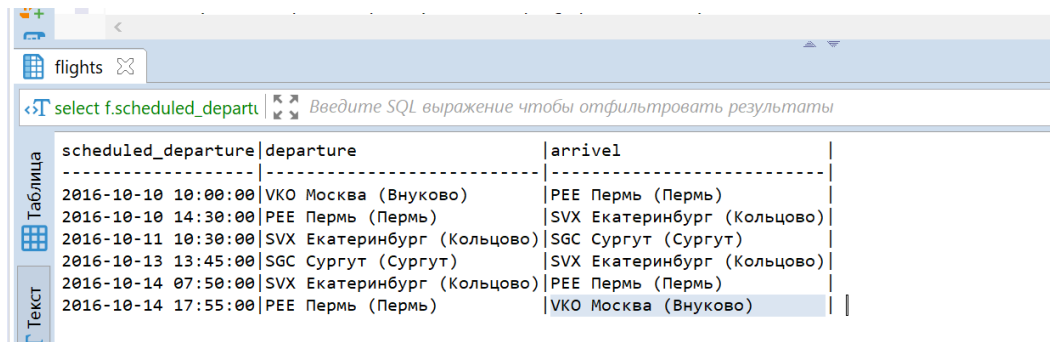
Рисунок 28 – Схема данных таблицы «Аэропорты» (Airports)

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;
```

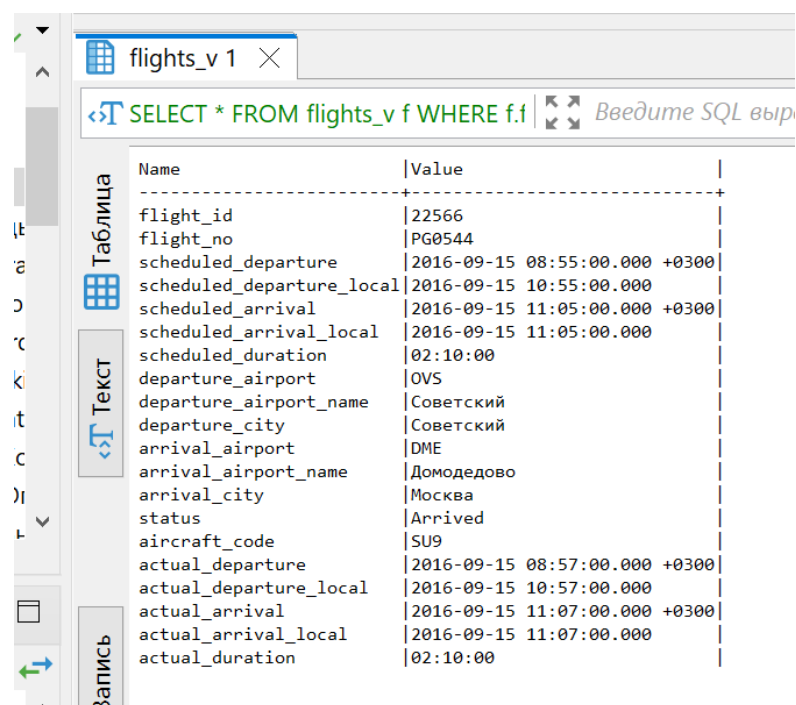


scheduled_departure	departure	arrival
2016-10-10 10:00:00	VKO Москва (Внуково)	PEE Пермь (Пермь)
2016-10-10 14:30:00	PEE Пермь (Пермь)	SVX Екатеринбург (Кольцово)
2016-10-11 10:30:00	SVX Екатеринбург (Кольцово)	SGC Сургут (Сургут)
2016-10-13 13:45:00	SGC Сургут (Сургут)	SVX Екатеринбург (Кольцово)
2016-10-14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



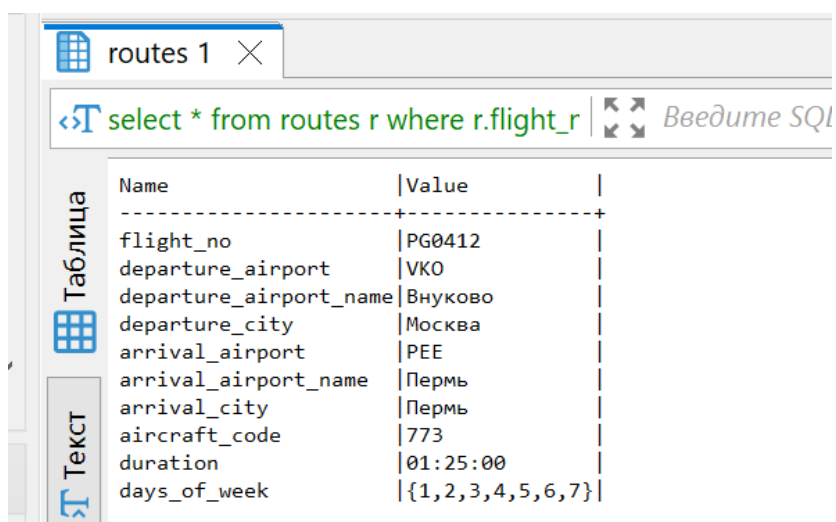
Name	Value
flight_id	22566
flight_no	PG0544
scheduled_departure	2016-09-15 08:55:00.000 +0300
scheduled_departure_local	2016-09-15 10:55:00.000
scheduled_arrival	2016-09-15 11:05:00.000 +0300
scheduled_arrival_local	2016-09-15 11:05:00.000
scheduled_duration	02:10:00
departure_airport	OV5
departure_airport_name	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'
```



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.

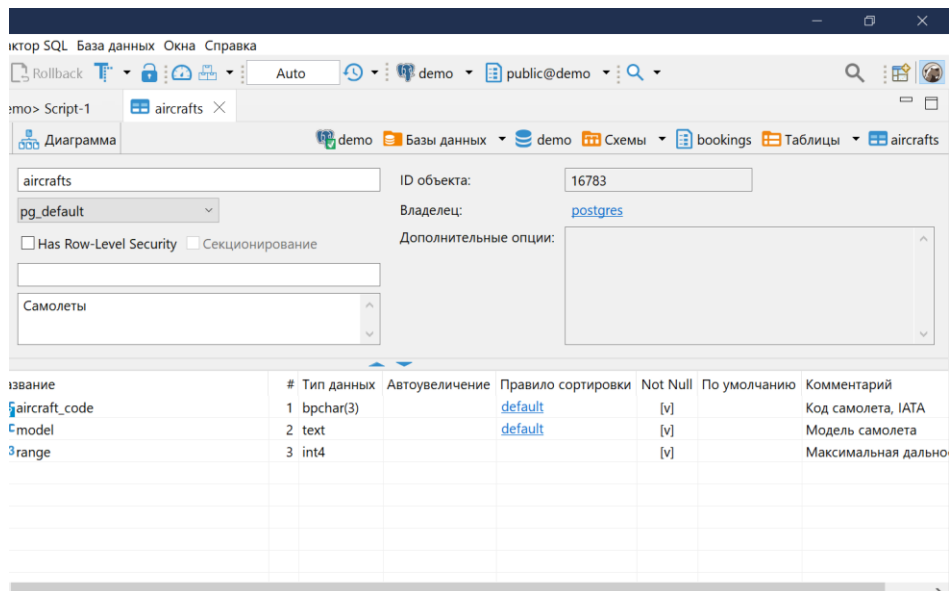


Рисунок 32

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;
```

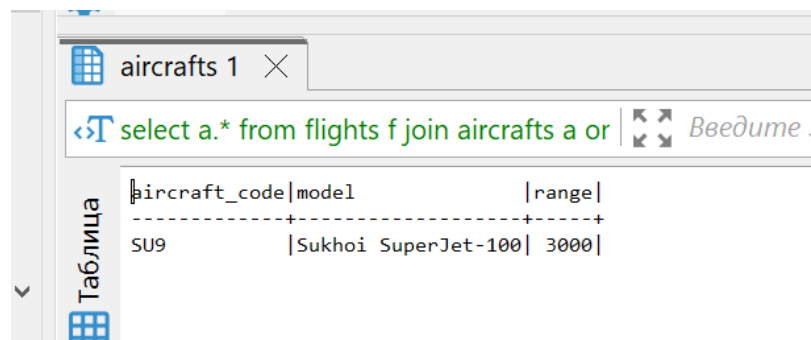


Рисунок 33

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.

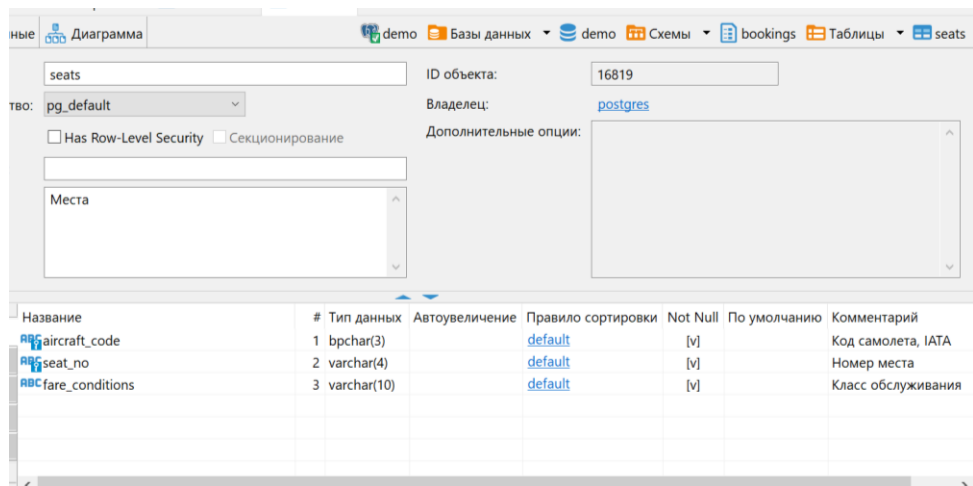


Рисунок 34

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.$';
```

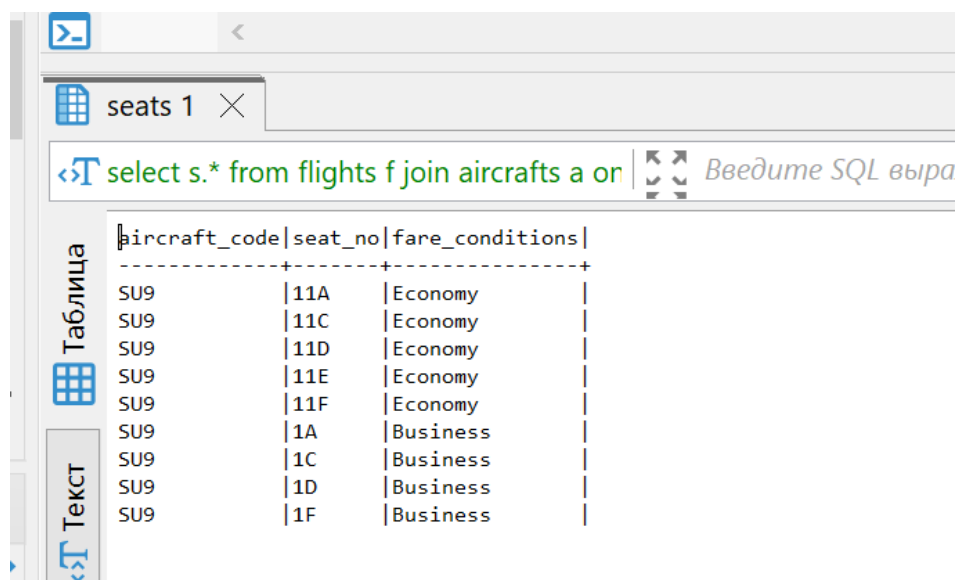
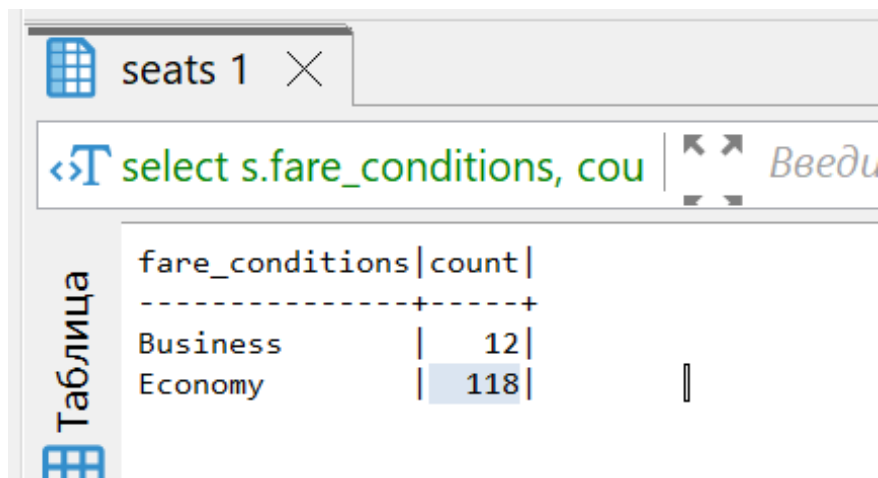


Рисунок 35

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 ;
```



seats 1

select s.fare_conditions, cou

Введи

Таблица

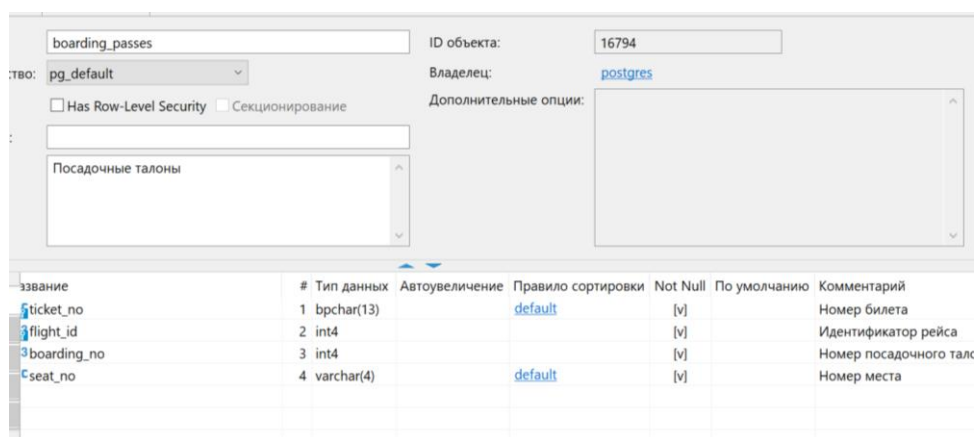
fare_conditions	count
Business	12
Economy	118

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).



boarding_passes

ID объекта: 16794

Владелец: postgres

Дополнительные опции:

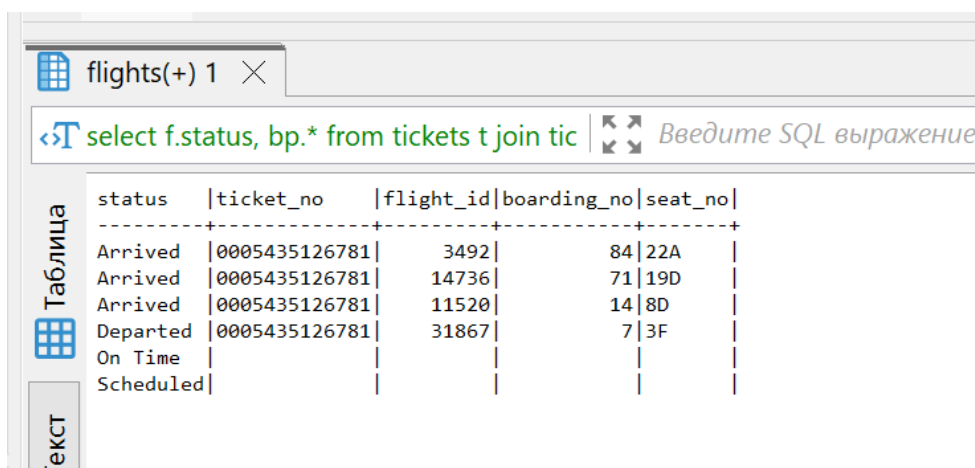
Посадочные талоны

Имя	#	Тип данных	Автоувеличение	Правило сортировки	Not Null	По умолчанию	Комментарий
ticket_no	1	bpchar(13)		default	[v]		Номер билета
flight_id	2	int4			[v]		Идентификатор рейса
boarding_no	3	int4			[v]		Номер посадочного талона
seat_no	4	varchar(4)		default	[v]		Номер места

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;
```



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				

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.

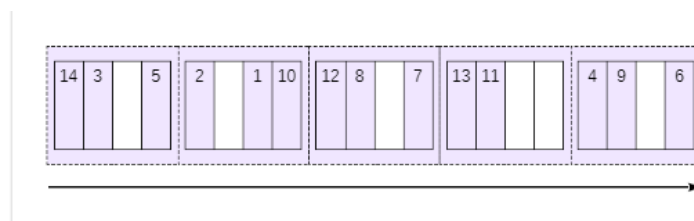


Рисунок 1 – Схема последовательного сканирования

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)

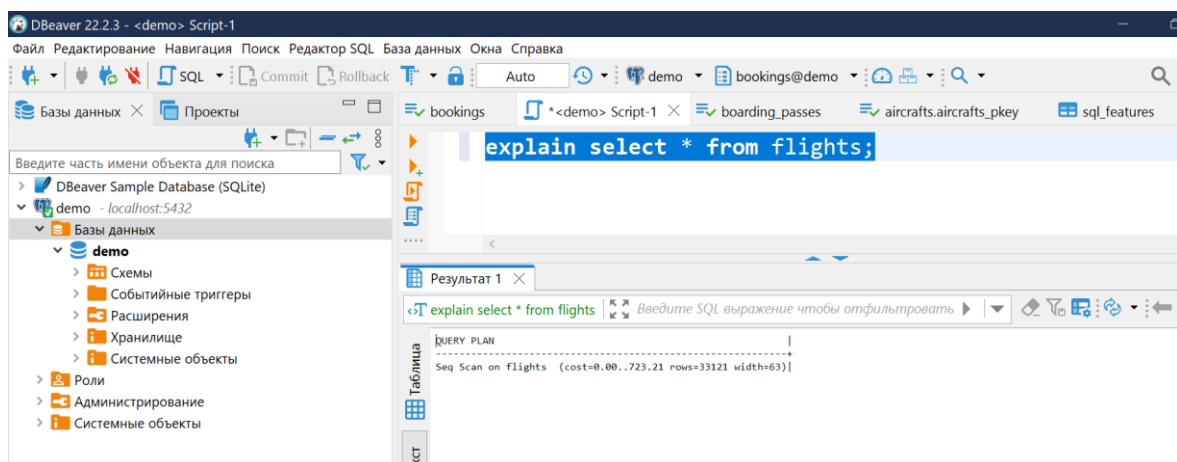


Рисунок 2 – Результат запроса последовательного сканирования

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';
```

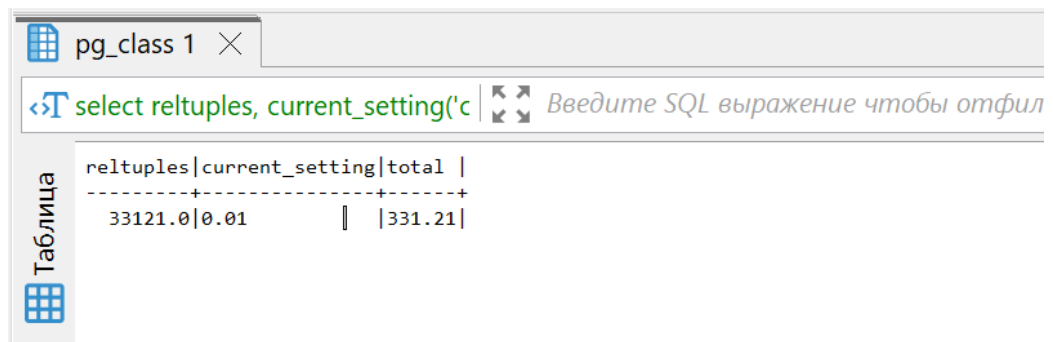


relpages	current_setting	total
392	1	392.0

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';
```



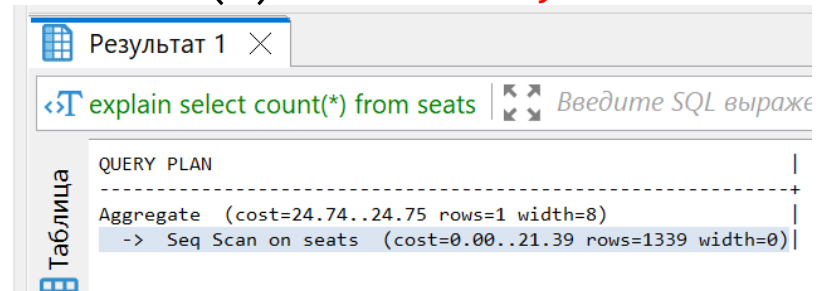
reltuples	current_setting	total
33121.0	0.01	331.21

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';
```

pg_class 1 X

`select reltuples, current_setting('c` Введите SQL выражение что

Таблица

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

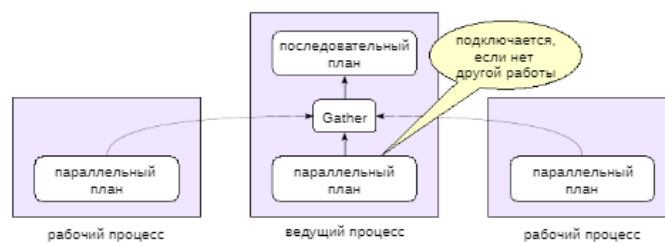


Рисунок 7

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.

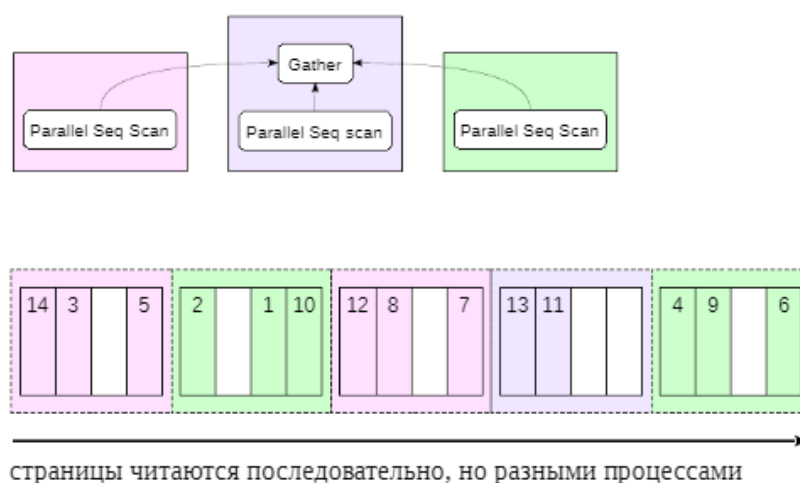


Рисунок 8

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 ;
```

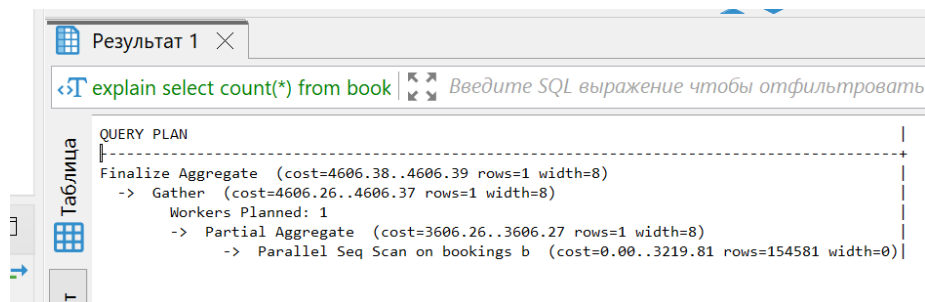


Рисунок 9

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';
```

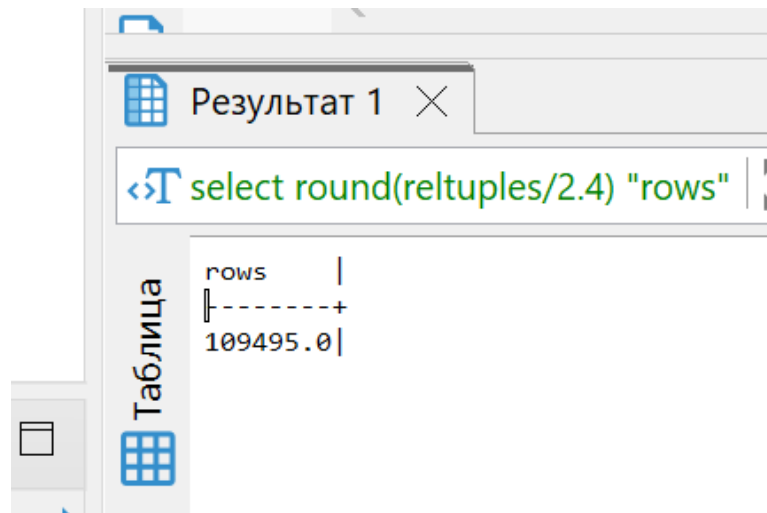


Рисунок 10 –

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

```
show parallel_leader_participation;
```

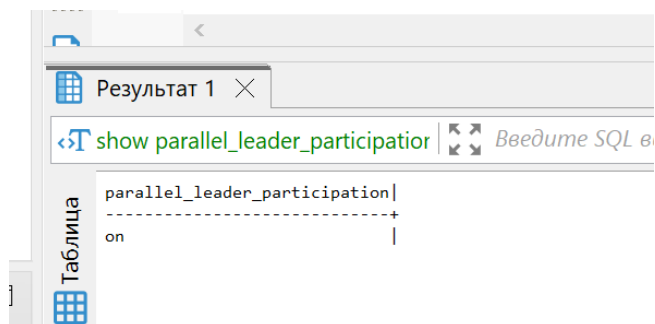


Рисунок 11

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

```
set parallel_leader_participation=off;
```

Статистика 1	
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;
```

Результат 1	
explain select count(*) from book	
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'
```

Результат 1	
select round(reltuples/2) "rows" fi	
rows	
131394.0	

Рисунок 14

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

```
reset parallel_leader_participation;
```

Статистика 1	
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 обращении 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.



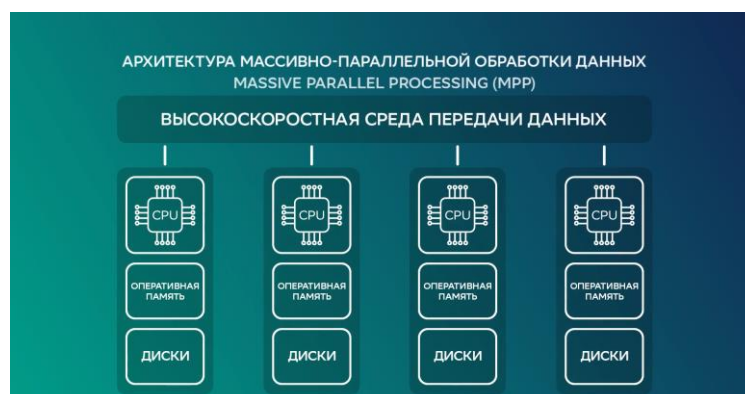
Системы обладают рядом преимуществ, таких как высокая скорость обмена данными между процессорами за счет наличия общей памяти, простота, универсальность обслуживания и относительно невысокая цена.



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.



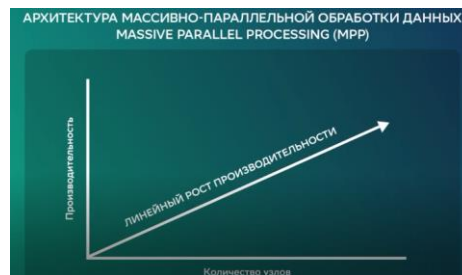
Как правило, данная операция является дорогостоящей, а в некоторых случаях и вовсе невозможно из-за физических ограничений в конфигурации сервера. Для решения проблемы масштабируемости в системах аналитической обработки данных стали применять архитектуру MPP или архитектуру массивно-параллельной обработки данных.



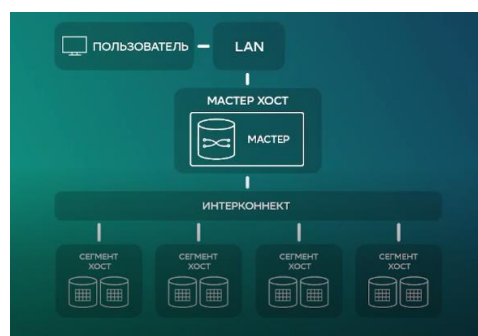
В такой архитектуре система состоит из нескольких независимых узлов соединенных по сети. При этом в каждом вычислительном узле процессор

обладает своими собственными физическими ресурсами, такими как память и диски, которые не разделяются с другими узлами. Именно поэтому такая архитектура также называется shared nothing.

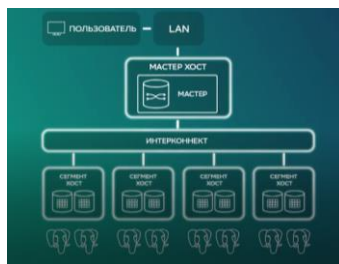
В MPP системах вычислительная мощность и объем хранения данных увеличиваются за счет добавления в систему дополнительных вычислительных узлов. Данный подход позволяет достигнуть линейный рост производительности в зависимости от количества узлов в системе.



Greenplum - это революционная СУБД с архитектурой MPP без разделения ресурсов. Greenplum предназначен для хранения и обработки больших объемов данных методом распределения данных и обработки запросов на нескольких серверах. Данная СУБД лучше всего подходит для построения корпоративных хранилищ данных, решение аналитических задач и задачи машинного обучения и искусственного интеллекта.



В основе Greenplum лежит СУБД PostgreSQL. По сути GreenPlum представляет из себя множество модифицированных экземпляров дисковых баз данных Postgres, работающих совместно как одна связанная система управления базами данных.



GreenPlum в большинстве случаев похож на Postgres. Например, в части синтаксиса SQL функций параметров конфигурации и функциональности для конечного пользователя. Пользователи базы данных взаимодействуют с Greenplum так же, как с обычной СУБД Postgres. Внутреннее устройство Postgres было изменено или дополнена для поддержки параллельной структуры баз данных GreenPlum. Например, компоненты системного каталога, оптимизатора, исполнителя запросов и диспетчера. Транзакции были изменены и улучшены, чтобы они могли выполнять запросы одновременно в параллель во всех экземплярах базы данных Postgres. Greenplum был разработан в 2005 году и изначально являлся проприетарным продуктом.

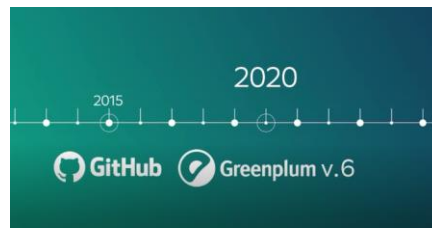


Позже, в 2015 году компания, разработчик данной СУБД Pivotal, выпустила исходный код ядра GreenPlum под свободные лицензии и выложила его на публичный репозиторий GitHub.



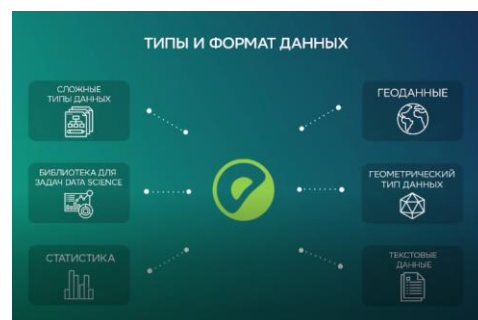
С тех пор, Greenplum является open-source продуктом, который развивается не только основной компанией разработчиком, но и участниками open-source сообщества. На основе открытого исходного кода многие компании делают свои собственные сборки и продают коммерческие версии

СУБД, добавляя в них свои компоненты и осуществляя поддержку пользователей.



Greenplum является исключительно программным решением. Он поставляется в виде дистрибутива, который может быть установлен на различные серверные платформы. Производительность напрямую зависит от оборудования, на котором он установлен.

GreenPlum предоставляет широкий выбор инструментов для решения аналитических задач. Например, поддержка различных сложных типов данных, библиотек для задач Data Science, статистики, поддерживает географические и геометрические типы данных, а также текстовые данные в виде документов.

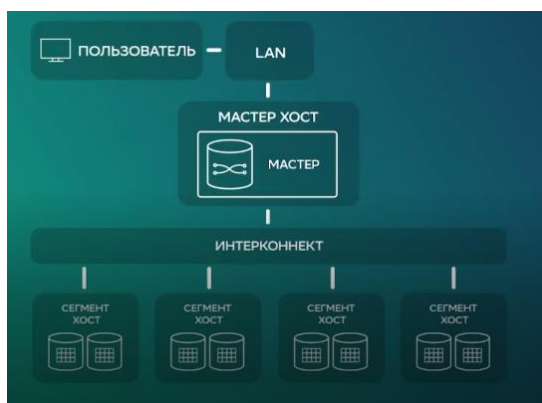


Greenplum обладает большим количеством различных коннекторов, которые предоставляют доступ к другим источникам данных, таким как другие СУБД, компоненты экосистемы Hadoop и новые источники данных, например, Kafka.

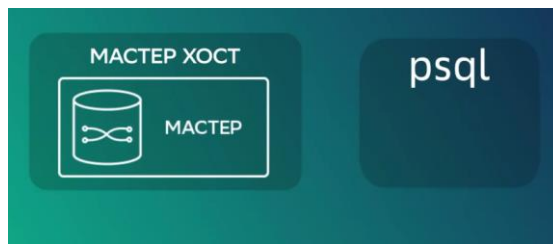


Также в GreenPlum можно разрабатывать процедуры не только на встроенным процедурным языке pl/pgSQL, но и на многих других популярных языках программирования, которые будут компилироваться и выполняться внутри базы данных. Также есть возможность разрабатывать процедуры, которые будут работать в контейнерах. Такой подход позволяет сделать разработку безопасной, путем изолирования исполняемого кода от операционной системы, на которой установлена СУБД.

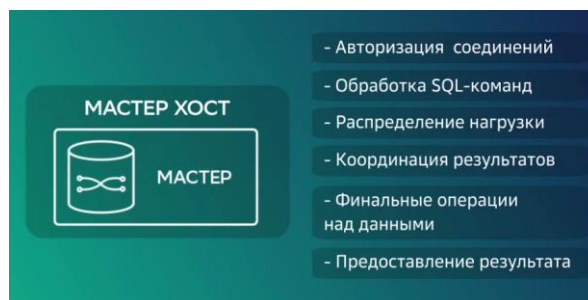
СУБД Greenplum реализована на массивной параллельной архитектуре без совместного использования ресурсов.



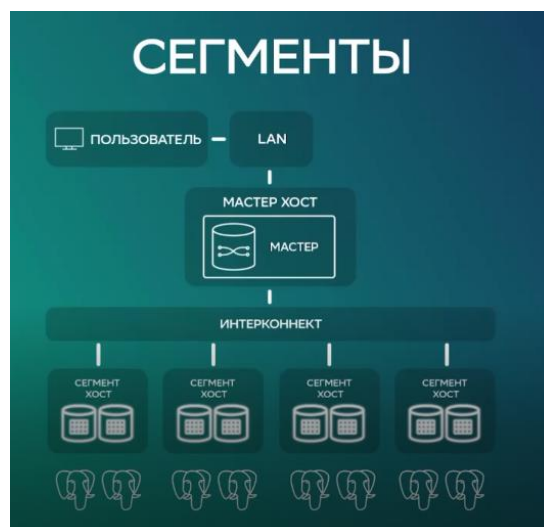
Также мы узнали, что Greenplum представляет из себя множество модифицированных баз данных PostgreSQL, которые расположены на нескольких серверах и взаимодействуют друг с другом, образуя единую цельную базу данных. Логически архитектура Greenplum выглядит следующим образом: мастер. Мастер — это входная точка для Greenplum. Это экземпляр базы данных PostgreSQL, 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 PostgreSQL. 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.

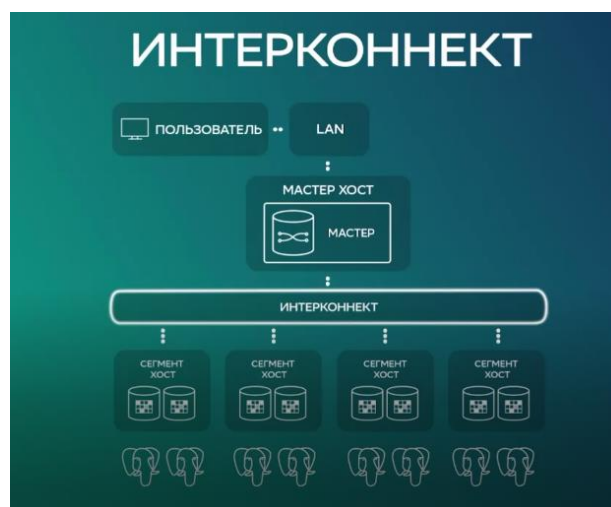


Сегменты - это отдельные экземпляры базы данных Postgres, которые хранят определенную порцию пользовательских данных и выполняют большую часть обработки запросов.



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 горлышком всей системы.

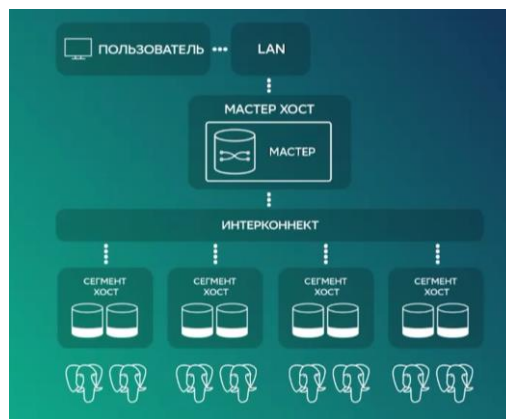
Интерконнект - это сетевой слой архитектуры Greenplum и интерконнект обеспечивает взаимосвязь между мастером, сегментами и сетевой инфраструктурой, в которой развернут кластер.



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 быстродействия 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.

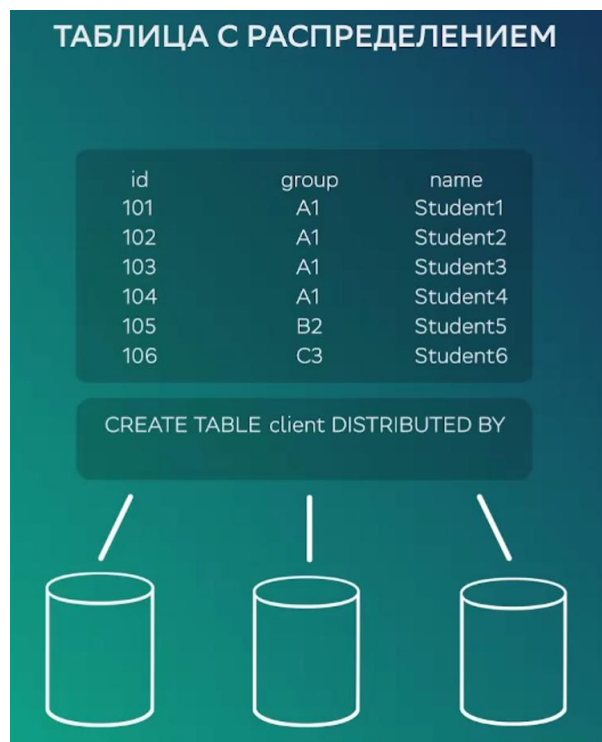
Понимание и правильное использование такой ключевой особенности Greenplum, как распределение данных по всем сегментам с учетом особенностей ваших данных поможет вам добиться наилучшей производительности.

ТАБЛИЦА С РАСПРЕДЕЛЕНИЕМ

id	group	name
101	A1	Student1
102	A1	Student2
103	A1	Student3
104	A1	Student4
105	B2	Student5
106	C3	Student6

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.

КЛЮЧ ДИСТРИБЬЮЦИИ

id	group	name
101	A1	Student1
102	A1	Student2
103	A1	Student3
104	A1	Student4
105	B2	Student5
106	C3	Student6

```
ALTER TABLE client DISTRIBUTED BY  
group
```



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

1. Равномерность распределения данных
2. Локальность операций - соединение таблиц с одинаковым ключом дистрибьюции

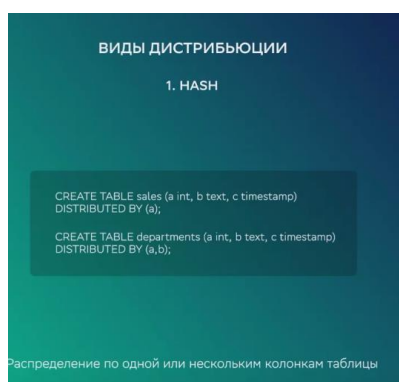


В Greenplum существует три вида дистрибьюции. Это distributed by, distributed randomly, distributed replicated.



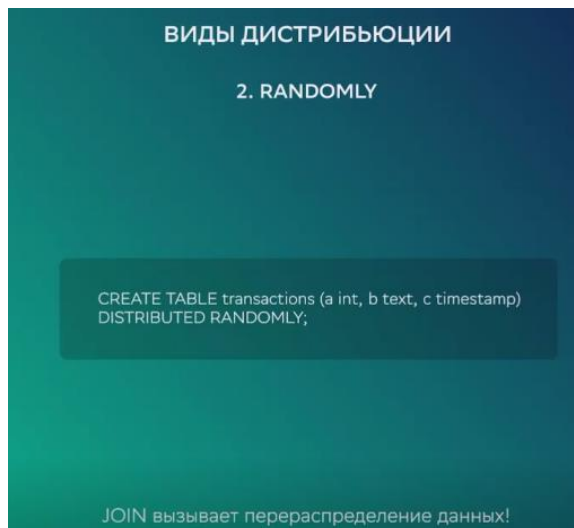
Давайте рассмотрим каждый из них более подробно.

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.

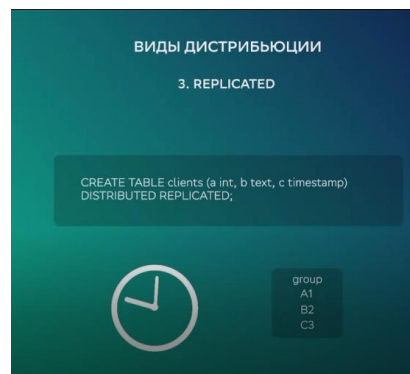


В приведенных примерах вычисления хэш-значения осуществляется по значению одной или двух колонок.

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.



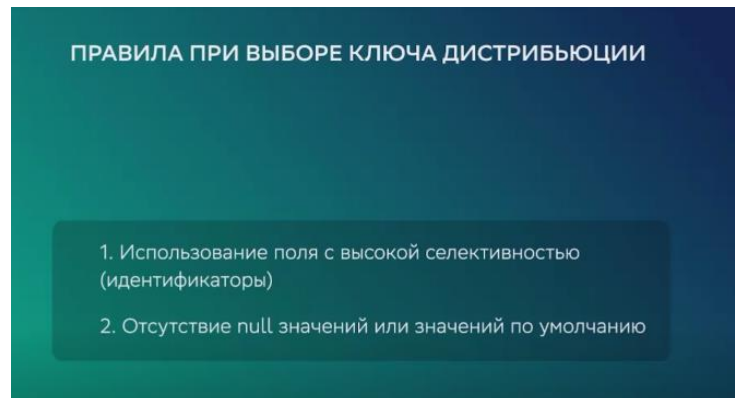
Distributed replicated - В этом случае на каждом сегменте хранится полная копия таблицы. Такой метод подходит для небольших таблиц, например, справочников, которые регулярно участвуют в соединениях с другими таблицами.



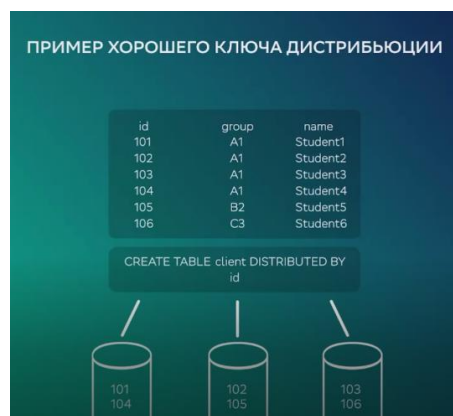
Хранения полной копии таблицы на каждом сегменте позволяет сэкономить время, которое в противном случае было бы потрачено на перераспределение во время выполнения запроса.

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

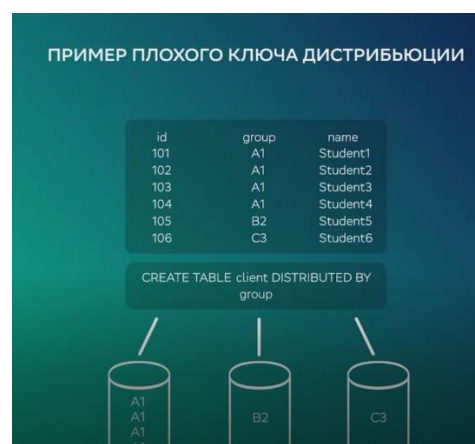
NULL значения или другие значения по умолчанию будут попадать на один сегмент, что приведет к перекосу в данных.



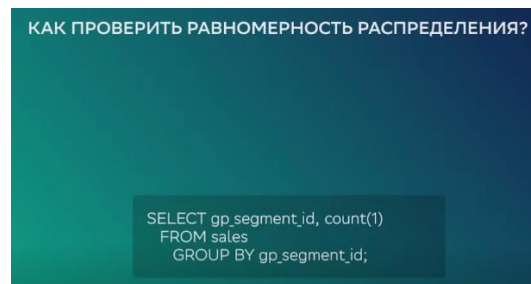
Рассмотрим примеры. В данном примере в качестве ключа дистрибьюции выбрано поле ID. В результате распределение на каждый сегмент падает ровно по две строки. Распределение получается равномерное, а значит, ключ дистрибьюции хороший.



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.

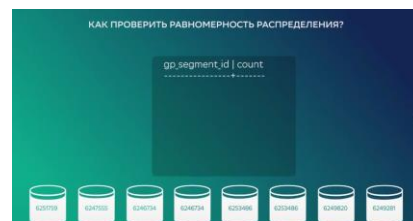


Делаем вывод о том, что ключ выбран не оптимально проверить равномерное распределение строк между сегментами можно, используя служебный столбец `gr_segment_id`, который присутствует в каждой таблице.

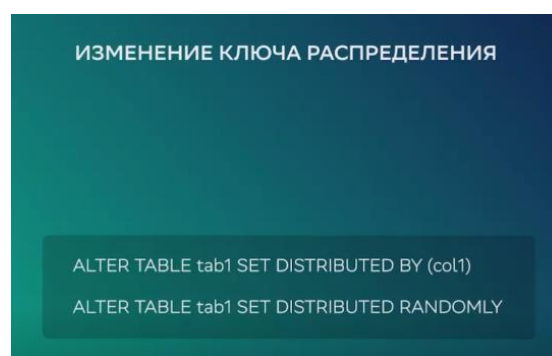


В данном столбце содержится идентификатор сегмента, на котором лежит строка. Пример запроса с группировкой по полю `gp_segment_id` для определения неравномерности распределения строк по сегментам, приведенным на экране.

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

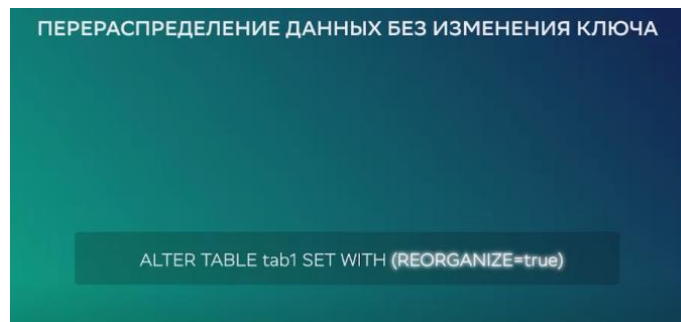


Ключ распределения можно изменить без создания таблицы с помощью предложения ALTER TABLE SET.

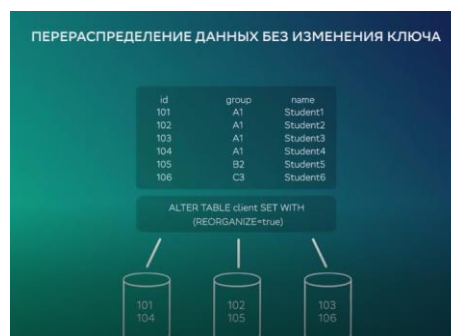


Примеры использования приведены на экране.

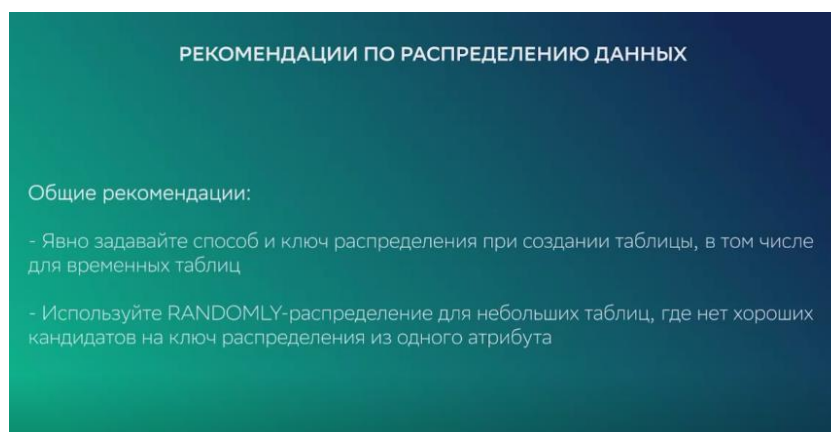
Также есть возможность перераспределения данных по прежнему ключа дистрибьюции при помощи опции REORGANIZE=true.



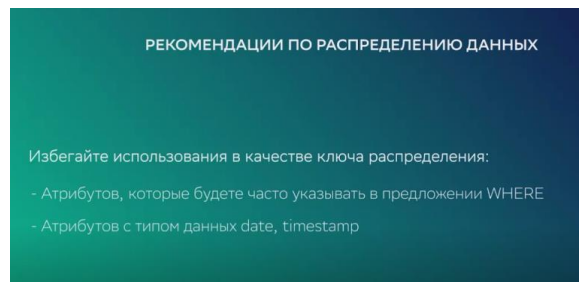
Эта опция будет полезна в двух случаях: при добавлении нового сегмента и для избежания раздутая таблиц. Про раздутие таблицы мы поговорим позже. А при добавлении нового сегмента в кластер возникает необходимость заново перераспределить данные, чтобы избежать перекоса в распределении данных, когда новый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



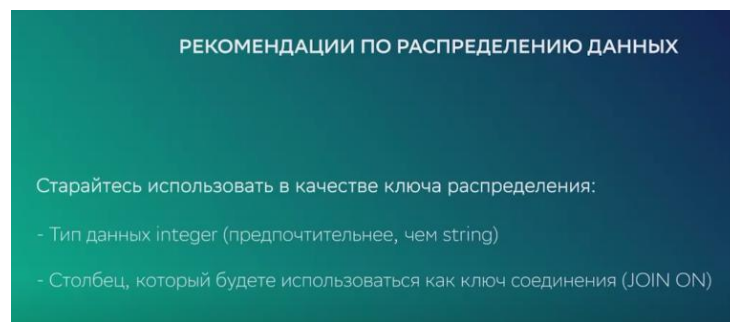
Вот некоторые рекомендации, которые помогут вам правильно распределять данные: явно задавайте способ и ключ распределения при создании таблицы, в том числе для временных таблиц; для небольших таблиц, где нет хороших кандидатов на ключи распределения из одного атрибута используйте RANDOMLY для распределения.



Избегайте использования в качестве ключа распределения атрибутов, которые будете часто указывать в предложении Where или 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.

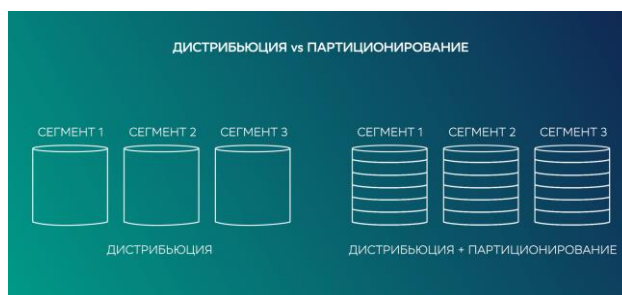
- Логически делит большую таблицу на части
- Значительно улучшает производительность запросов
- Упрощает сопровождение базы данных

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. Дистрибьюция обязательно для всех таблиц, то есть все таблицы в greenplum являются распределёнными и имеют ключ дистрибьюции.

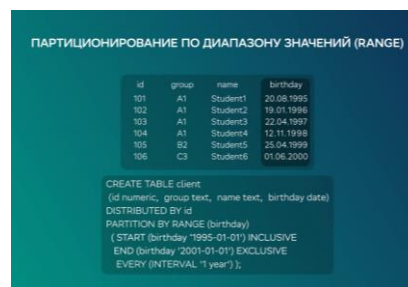


Партиционирование - это лишь опция, которую можно применять к таблице в целях оптимизации в GreenPlum поддерживаются следующие виды партиционирования:

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



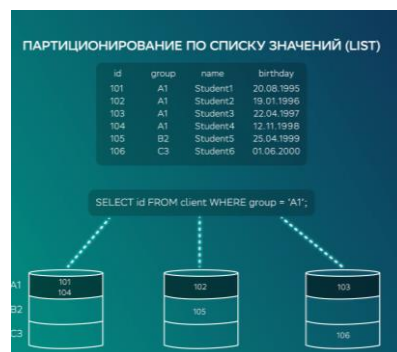
Представим таблицу, содержащую записи о клиентах учебного заведения, например, студентах университета. Согласно заданной политике, дистрибьюции, таблица равномерно распределена по сегментам на основе идентификатор студентов. Допустим, что для построения отчетности периодически требуется анализировать 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 RANGE партиционирование по году рождения. Вид партиционирования задается при создании таблицы с помощью выражения PARTITION BY RANGE.





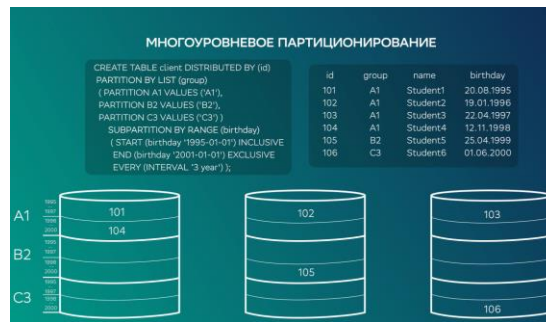
Диапазон значений задан выражением `start и and`, а шаг каждой партии выражением `EVERY INTERVAL` интервал после создания таблицы на каждом сегменте будут созданы по шесть партий, хранящие записи по году. Если в запросе к таблице есть условия фильтрации записей по году, то база данных в рамках каждого 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 партиций, что, в свою очередь, может замедлить работу всей базы данных.

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

Можно создать партицию по умолчанию, дефолт, в которую будут попадать значения, не удовлетворяющие условиям других партиций. Иногда это удобный способ фильтрации ошибочных записей. При этом невозможно указать разные ключи дистрибьюции для партиций в рамках одной таблицы. Нельзя поменять тип партиционирования для существующей таблицы без пересоздания таблицы.

Можно указать:

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

Можно создать:

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

Нельзя:

- Указать разные ключи дистрибьюции 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	Партиции	Способ хранения	Компрессия
Сегодня		1 день	HEAP	-
-3 мес		1 мес	АО	Низкая (x2,5) Zstd
-12 мес		1 год	АО	Средняя (x7) Zstd
-24 мес		1 год	АО	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
Исключение "лишних" партиций (partition 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 (slice1; segments: 32) (cost=0.00..6834.88 rows=44906244 width=49)

-> Seq Scan on sales (cost=0.00..1679.05 rows=1403321 width=49)

Filter: ((date = '2020-04-18'::date) AND (region = 'usa'::text))

Optimizer: Pivotal Optimizer (GPORCA)

QUERY PLAN С партициями

Gather Motion 32:1 (slice1; segments: 32) (cost=0.00..6048.26 rows=46899539 width=49)

-> Sequence (cost=0.00..663.58 rows=1465611 width=49)

-> Partition Selector for tloan_prt (dynamic scan id: 1) (cost=10.00..100.00 rows=4 width=4)

Partitions selected: 1 (out of 60)

-> Dynamic Seq Scan on sales_prt (dynamic scan id: 1) (cost=0.00..663.58 rows=1465611 width=49)

Filter: ((date = '2020-04-18'::date) AND (region = 'usa'::text))

Optimizer: Pivotal Optimizer (GPORCA)

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, compressstype=zlib, compresslevel=3),
   start (date '2011-01-01') end (date '2013-01-01') every (interval '3 months')
   with (appendonly=true, orientation=column, compressstype=zlib, compresslevel=1),
   start (date '2013-01-01') end (date '2014-01-01') every (interval '1 months')
   with (appendonly=true, orientation=row, compressstype=zstd),
   start (date '2014-01-01') end (date '2014-02-01') every (interval '1 day'));
```

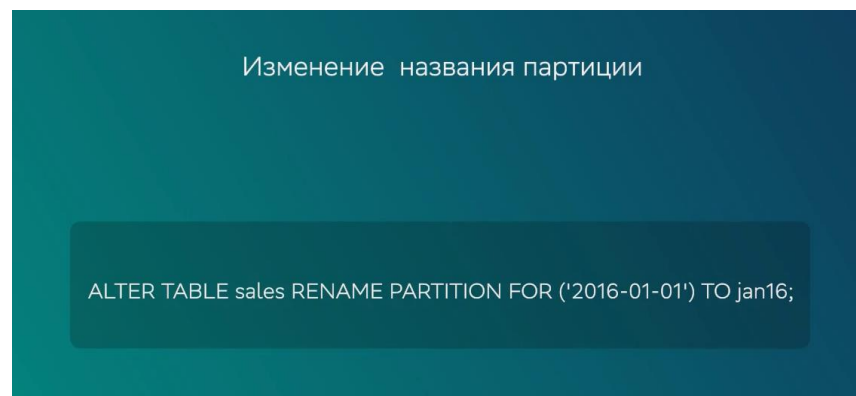
Для партиции с интервалом год или три месяца задана колоночная ориентация хранения и тип компрессии THE LEAP с разными коэффициентами сжатия. Для партиции с интервалом один месяц применяется ориентация по строкам и другой тип компрессии без явного указания коэффициента сжатия. Для партиции с интервалом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

добавления партиций используется команда `AT PARTITION` для обычных партиций или `AT DEFAULT` для партиций по умолчанию.

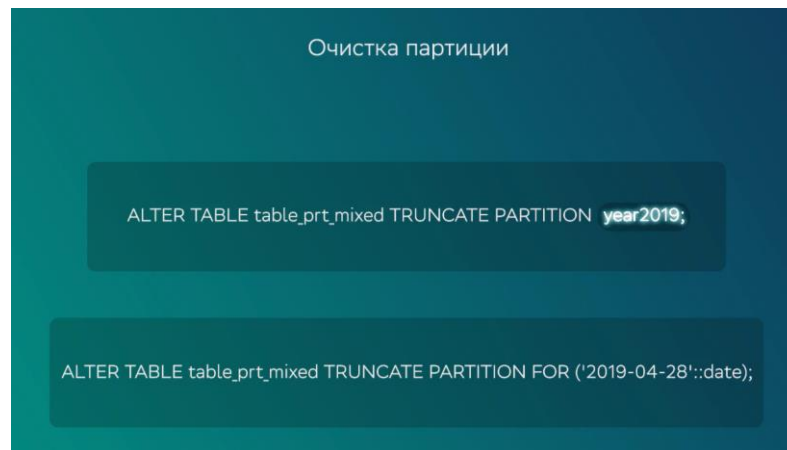


Чтобы явно задать имя партиций, укажите его при добавлении партиции.

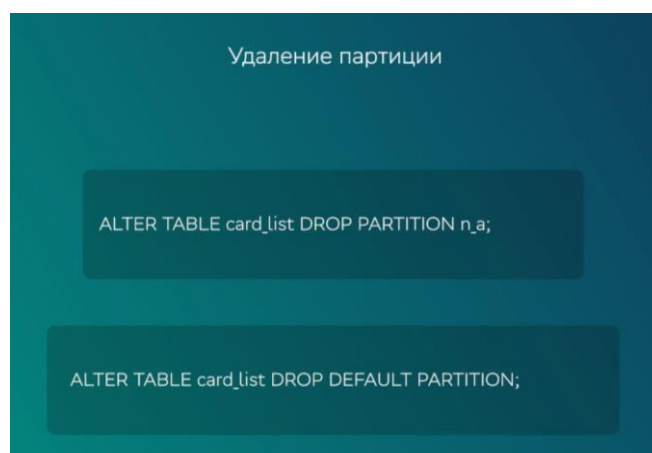
На рисунке показаны примеры для партиций по списку, по диапазону значений, и для партиций по умолчанию, независимо от типа партиционирования.



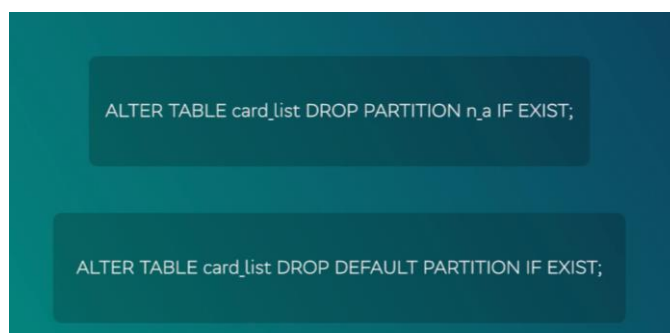
Для изменения названия партиций используется команда `RENAME PARTITION`. Для очистки партиции применяется команда `TRUNCATE PARTITION`. На рисунке показано, как очистить партицию с названием `EP 2019` или партицию, содержащую дату 28 апреля 2019 года.



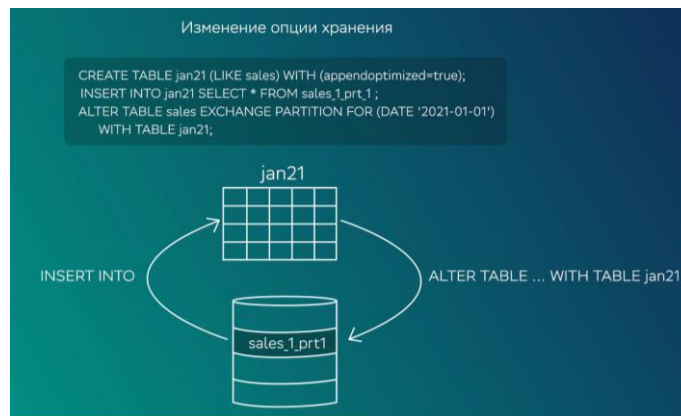
Для удаления партиции используется команда DROP PARTITION. На рисунке приведены примеры удаления обычной и дефолтной партиции.



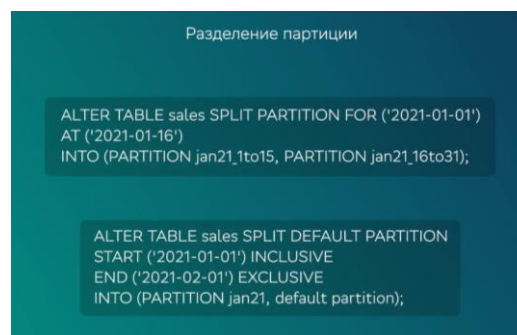
При желании в тексте команды можно указать опцию IF EXIST, которая поможет избежать сообщения об ошибке и при выполнении команды в случае отсутствия удаляемой партиции.



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

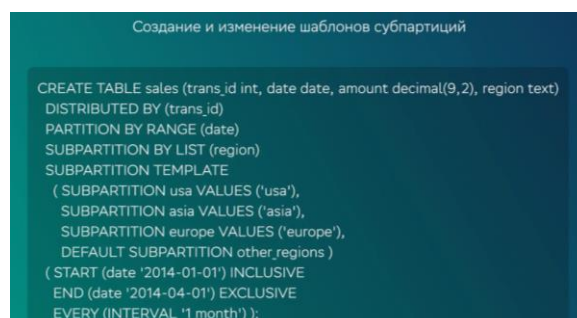


Для разделения одной партии на две части используется команда **ALTER TABLE SPLIT PARTITION**.

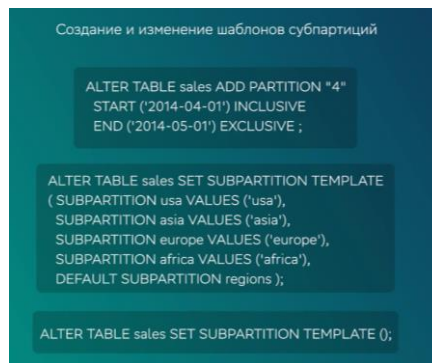


В первом примере на рисунке выполняется разделение партии, которые хранятся данные за январь на две партии. В первой будут храниться данные с первого по 15 января, а во второй с 16 го по 31-я. Если в вашей таблице есть дефолт партия, то добавить новую партию в такой таблице можно только путем разделения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

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

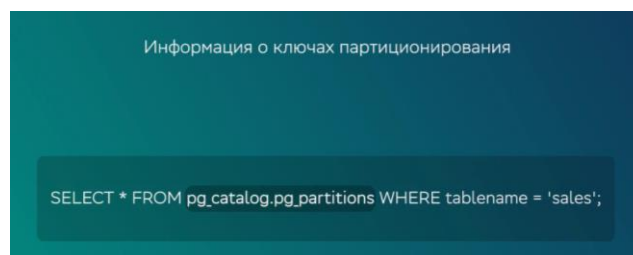


Например, в следующем скрипт создания шаблон определяется с помощью предложения SUP-PARTITION TEMPLATE, в котором мы определяем четыре суб-партиции для разных регионов. Теперь, если мы добавляем в таблицу партицию следующем с скриптом, то эта позиция по шаблону автоматически разделится на четыре суб-партиции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



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 партии из просмотра, или когда нужно обеспечить так называемое окно данных.

При периодической загрузке в таблицу новых данных и архивации оценке актуальных данных путем компрессии, либо выноса во внешние системы.

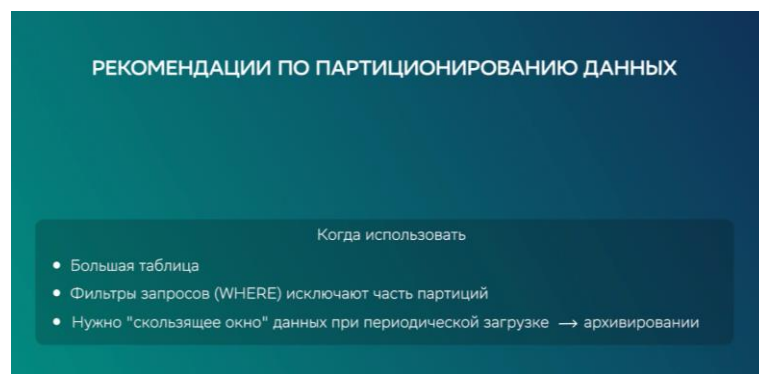
Старайтесь выбирать партиционирование по диапазону, а не по списку значений, выполняйте партиционирование на основе часто используемого столбца.

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

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

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

Избегайте создания большого количества партий. Greenplum эффективно обрабатывает операции чтения FULL SKAN, что позволяет уменьшать количество партий, при этом увеличивать объем данных в каждой партии, не используйте дефолт партию, 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 дистрибьюции. 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.

КОПИРОВАНИЕ ТАБЛИЦ																																
<table><thead><tr><th colspan="3">orders</th></tr><tr><th>order_no</th><th>name</th><th>price</th></tr></thead><tbody><tr><td>1</td><td>cherry</td><td>10</td></tr><tr><td>2</td><td>apple</td><td>20</td></tr><tr><td>3</td><td>banana</td><td>30</td></tr></tbody></table>	orders			order_no	name	price	1	cherry	10	2	apple	20	3	banana	30	<pre>CREATE TABLE new_orders AS SELECT * FROM orders;</pre>	<table><thead><tr><th colspan="3">new_orders</th></tr><tr><th>order_no</th><th>name</th><th>price</th></tr></thead><tbody><tr><td>1</td><td>cherry</td><td>10</td></tr><tr><td>2</td><td>apple</td><td>20</td></tr><tr><td>3</td><td>banana</td><td>30</td></tr></tbody></table>	new_orders			order_no	name	price	1	cherry	10	2	apple	20	3	banana	30
orders																																
order_no	name	price																														
1	cherry	10																														
2	apple	20																														
3	banana	30																														
new_orders																																
order_no	name	price																														
1	cherry	10																														
2	apple	20																														
3	banana	30																														

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

КОПИРОВАНИЕ ТАБЛИЦ																																
<table><thead><tr><th colspan="3">orders</th></tr><tr><th>order_no</th><th>name</th><th>price</th></tr></thead><tbody><tr><td>1</td><td>cherry</td><td>10</td></tr><tr><td>2</td><td>apple</td><td>20</td></tr><tr><td>3</td><td>banana</td><td>30</td></tr></tbody></table>	orders			order_no	name	price	1	cherry	10	2	apple	20	3	banana	30	<pre>CREATE TABLE new_orders (LIKE orders);</pre>	<table><thead><tr><th colspan="3">new_orders</th></tr><tr><th>order_no</th><th>name</th><th>price</th></tr></thead><tbody><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr></tbody></table>	new_orders			order_no	name	price									
orders																																
order_no	name	price																														
1	cherry	10																														
2	apple	20																														
3	banana	30																														
new_orders																																
order_no	name	price																														

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 `appent optimize`.

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 отличие 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 Optimaze 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);
```

Лабораторная работа № 15. Индексы и представления

Цель работы: научиться работать с индексами и представлениями

Давайте вспомним, что такое индекс и зачем он нужен. Представьте, что вам нужно быстро найти термин или фамилию ученого в многотомной энциклопедии. Чтобы не перелистывать каждую книгу, вы обратитесь к индексу предметному указателю. Это список ключевых слов с номерами томов и страниц, где они упомянуты.



Указатель терминов		Указатель имён	
А		А	
алгоритм	... т.10, стр. 5,15,20	Адлер А. (Adler A.)	... т.10, стр.2
архитектура	... т.3, стр. 561	...	
архитектура ПО	... т.10, стр. 5,15,20	...	
Б		Б	
база данных	... т.10, стр.8,17	Буш В. (Bush, W.)	... т.10, стр.3
В		В	
...		Винер, Н. (N. Wiener)	... т.10, стр.5

Для терминов используется один указатель, а для персоналий другой указатель ускоряет поиск. Индекс в базе данных это аналог указателя в бумажной книге, только вместо домашней библиотеки мы имеем дело с данными в таблице.

ИНДЕКС В БД

clients		
строка	id клиента	имя
...	...	Василий
9	101	Борис
...	...	...
1009	102	Алексей

Индекс может быть построен на основе одного или нескольких столбца. Фактически он состоит из значения атрибута. Например, идентификатор клиента, который сопоставим с местоположением записи в таблице.

ИНДЕКС В БД

Индекс по «id клиента»		clients		
id клиента	Адрес строки	строка	id клиента	имя
...	...	...	...	Василий
101	9	9	101	Борис
...	...	...	...	...
102	1009	1009	102	Алексей

- Индекс — указатель, сопоставляющий атрибуты и их местоположение

Во время выполнения запроса индекс помогает быстро найти строку, связанную со значениями атрибута.

Greenplum поддерживает несколько типов индексов: BTREE, BITMAP и GIST.

ТИПЫ ИНДЕКСОВ В GREENPLUM

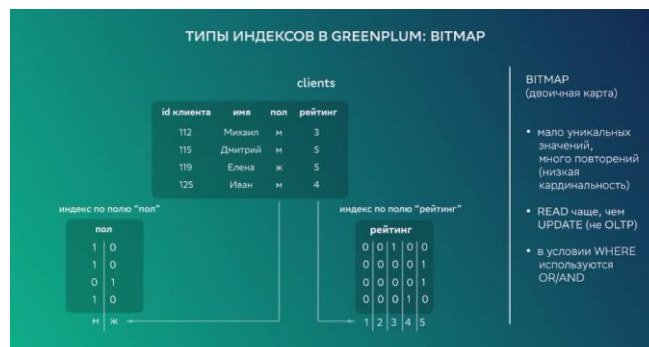
- BTREE
- BITMAP
- GIST

Индекс BTREE используется по умолчанию, его рекомендуется использовать, если к конденсированному полю применяются такие операции, как сравнение, больше, меньше, равно, between и is null, is not null, like.



Кстати, помните, что фильтр по сочетанию символов в начале и в конце фрагмента могут вести себя по-разному.

Индекс BITMAP в следующих случаях, когда колонка содержит относительно небольшое количество уникальных значений, обычно от 100 до 100000, и при этом много повторений. То есть имеет низкую кардинальность, когда чтение намного больше, чем update, то есть тип загрузки не OLTP, когда в запросах к таблице в условии WHERE используются логические операции и/или с полями с BITMAP индексом.



Индекс Gist, по сути, представляет из себя BTREE индекс и применяется для редких типов данных, таких как геоданные текстовые и графические документы.



Рассмотрим примеры создания индекса для таблицы клиентов с использованием BTREE индекса по полю имени и BITMAP индекса по полю Rating.

ПРИМЕРЫ СОЗДАНИЯ ИНДЕКСОВ

client_id	client_name	gender	rating
112	Михаил	м	3
115	Дмитрий	м	5
119	Елена	ж	5
125	Иван	м	3

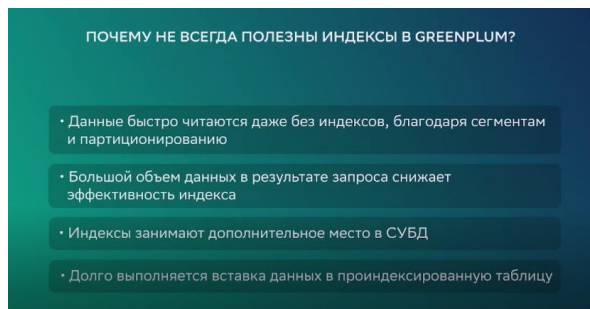
`CREATE INDEX client_name_index ON clients USING btree (client_name);`

client name	Адрес строки
Дмитрий	2
Елена	3
Иван	4
Михаил	1

`CREATE INDEX client_name_index ON clients USING bitmap (rating);`

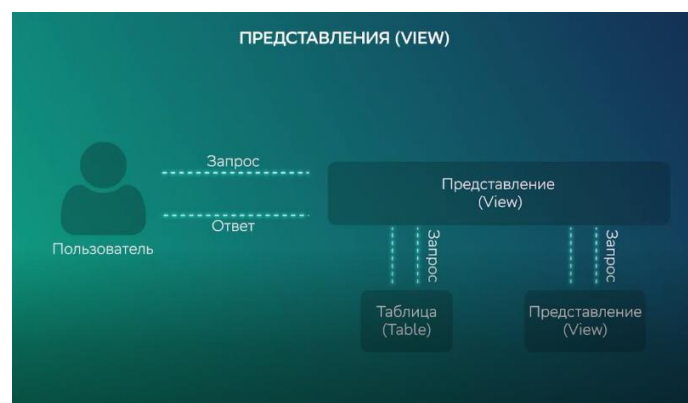
rating	Адрес строки
3	1
3	4
5	2
5	3

Используйте выражение Using, чтобы задать способ индексирования и поле для индексации. В Greenplum индексы нужно использовать с осторожностью по нескольким причинам: Во-первых, Greenplum умеет быстро считывать данные в связи с тем, что данные распределены по сегментам, и каждый сегмент считывает только свою порцию данных. Скорость чтения может быть увеличена еще больше, если использовать партиционирование. Во-вторых, в GreenPlum аналитические запросы могут возвращать большой объем данных. В этом случае использование индексов неэффективно. Не забывайте, что индексы занимают дополнительное место в СУБД и требуют обслуживания. И, наконец, вставка данных в проиндексированную таблицу выполняется намного дольше.



Придерживайтесь следующих правил при работе с индексами в Greenplum. В первую очередь используйте другие способы оптимизации, применения индексов для оптимизации в хранилищах данных необходимо выполнять в последнюю очередь, индексы подходят для близкой OLTP-нагрузки, частый выбор небольшого числа строк с WHERE. Уменьшая стоимость запроса, можно форсировать использование индекса планировщиком, про планирование запросов и их оптимизацию мы будем говорить в следующих частях курса. Если данные в таблице сжаты, то запрос к такой таблице с использованием индекса будет расжимать только те строки, которые запрашиваются. Что может сэкономить время и ресурсы процессора при загрузке данных в таблицу с индексами удалите индексы, загрузите данные, затем создайте индексы заново. Для часто обновляемых таблиц снижайте значение коэффициента заполнения индекса FILLFACTOR. Этот параметр определяет степень заполненности блоков данных таблиц на диске и связан со скоростью и компактностью записей.

В Greenplum, как и во многих других СУБД, существуют объекты типа view представление. В этой теме мы рассмотрим данный тип объектов.



VIEW - Это объект базы данных, который, по сути, представляет из себя сохраненный запрос, в момент обращения к которому пользователю возвращается результат данного запроса. Представление хранят не сами данные, а только ссылки на них. Представление может ссылаться на одну или несколько таблиц, либо на другие представления. Предположим, нас интересует составной список из погодных записей и городов, которые можно получить следующим запросом.

ПРЕДСТАВЛЕНИЯ (VIEW)

```
SELECT cities.id, cities.name, weather.temp_lo, weather.temp_hi, weather.date
FROM weather, cities
WHERE weather.city_id= cities.id;
```

id	name
1	Moscow
2	Kazan
3	Novosibirsk
4	Saint-Petersburg

city_id	temp_lo	temp_hi	date
1	0	15	01.05.2021
2	1	13	01.05.2021
3	-1	4	01.05.2021
3	0	5	02.05.2021

id	name	temp_lo	temp_hi	date
1	Moscow	0	15	01.05.2021
2	Kazan	1	13	01.05.2021
3	Novosibirsk	-1	4	01.05.2021
3	Novosibirsk	0	5	02.05.2021

Мы довольно часто запрашиваем эту информацию, поэтому не хотим каждый раз вводить весь запрос целиком. В этом случае мы можем создать представление **MYVIEW** на основе данного запроса, фактически присвоив имя запросу, а затем обращаться к нему как к обычной таблице.

Рассмотрим синтаксис создания изменения и удаления представлений. Для создания представления предназначен оператор **CREATE VIEW**. Для изменения существующего представления необходимо добавить после оператора **CREATE** предложение **OR REPLACE**.

СИНТАКСИС

Изменение представления:

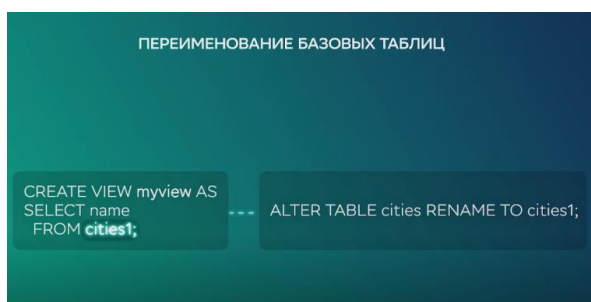
```
CREATE OR REPLACE VIEW <имя_представления>
AS
<SELECT ... >
```

Для удаления представления используется оператор DROP VIEW. Следует учесть, что в случаях, когда необходимо удалить базовые таблицы, то есть таблицы, участвующие в запросе представления, удаление необходимо выполнять с дополнительной опцией CASCADE.

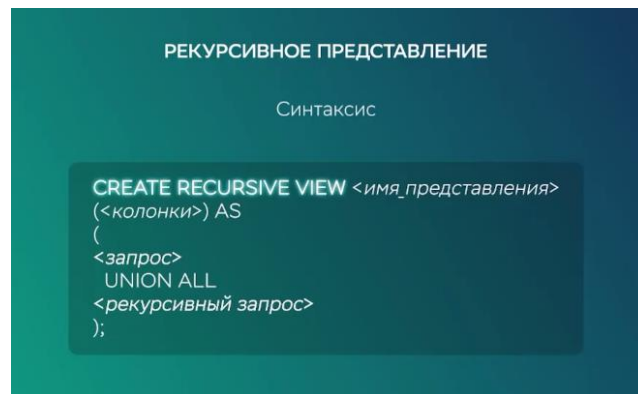


В противном случае вы получите сообщение об ошибке, при удалении базовой таблицы с опцией и вместе с ней автоматически удаляться все представления, в которых она участвует.

GreenPlum позволяет переименовывать базовые таблицы без необходимости пересоздания представлений. К примеру, существует представление myview, основанное на таблицы cities после того, как мы переименуем базовую таблицу cities в cities1 следующей командой, новое имя таблицы автоматически обновится в тексте запроса представления.

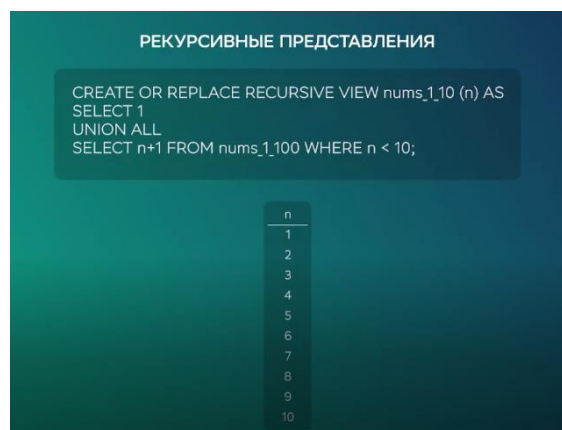


Рекурсивным называется представление, в тексте запроса которого используется ссылка на это же представление. Это способ вызова цикла прямо из SQL-выражения внутри определения представления. Обратите внимание на синтаксис определения рекурсивного представления на рисунке.

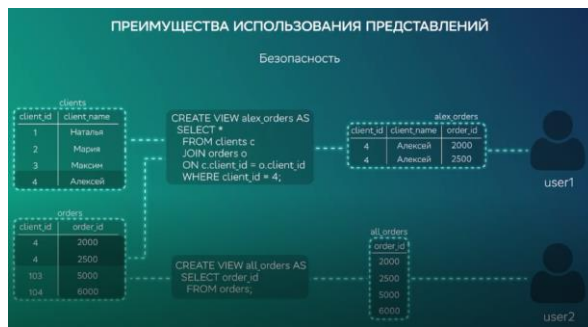


Оно создается с помощью команды CREATE RECURSIVE VIEW. Далее, в первой части представления мы указываем обычный запрос. После следует UNION ALL и часть, в которой осуществляется рекурсивный вызов.

Простой пример репрессивного запроса вы видите на рисунке.



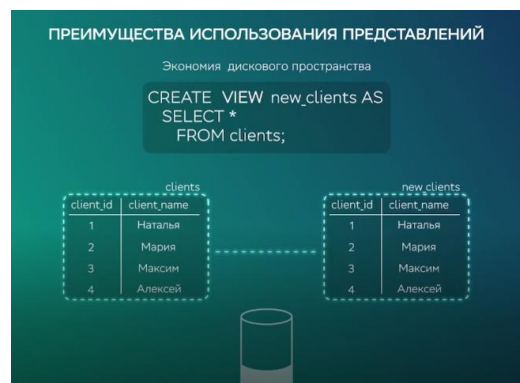
Данный запрос после выполнения выведет последовательность цифр от 1 до 10. Рассмотрим преимущества использования представлений. С помощью представлений можно обеспечить дополнительный уровень безопасности данных. Пользователи могут не иметь прямого доступа к базовым таблицам. При этом каждому пользователю можно разрешить доступ к определенным представлениям, содержащим только ту информацию, которую ему разрешено просматривать. Например, в представлении может использоваться специальный фильтр на данные условия WHERE. Или же в представлении может выводиться только ограниченное количество колонок. Аналогично при помощи и представлении для каждого пользователя можно создать собственную структуру базы данных, определив ее как множество доступных пользователю виртуальных таблиц.



С помощью представления можно скрыть сложность запросов и сложность структур базовых таблиц. Например, можно скрыть всевозможные вычисления соединения, а также исключить избыточные поля из внутреннего определения представления. Эта особенность добавляет модульность и лаконичность при дальнейшем использовании сложно устроенных объектов. Представление обеспечивает разделение схем представления и хранения данных. Представление возвращает не противоречивый и не измененный образ структуры используемых данных даже если исходные таблицы изменяются по структуре или переименовываются.

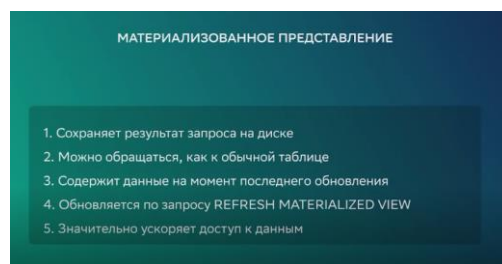


И, наконец, с представлением не требуется дополнительное дисковое пространство. Они не хранят каких-либо данных, а извлекают их по запросу из базовых таблиц.



Теперь давайте рассмотрим такой вид представлений, как материализованные представления. Главной особенностью

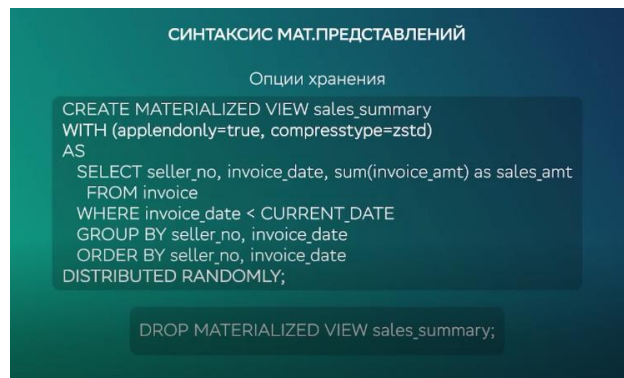
материализованных представлений является то, что в отличие от обычных представлений, они сохраняют результат запроса на диске и в дальнейшем позволяют обращаться к ним как к обычной таблице. При этом данные актуальны на момент последнего обновления материализованного представления. Данные в мат представлении могут быть обновлены по запросу пользователя командой `REFRESH MATERIALIZED VIEW`. Материализованные представления могут значительно ускорить доступ к данным за счет предварительного расчета.



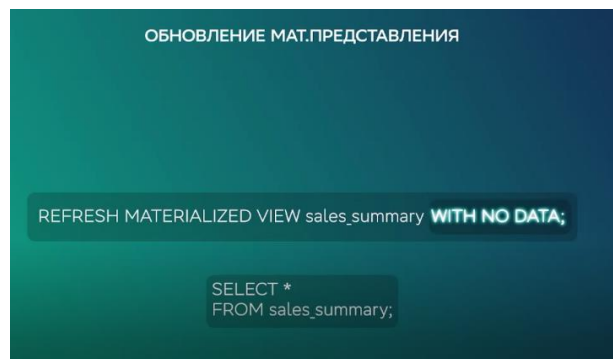
Давайте рассмотрим синтаксис создания мат представлений. Как видно на примере, в отличие от обычных представлений, после слова `CREATE` добавляется слово `MATERIALIZED`.



В Greenplum для мат представления есть возможность указывать опции хранения, как для обычных таблиц. В представленном на слайде скрипте после имени указан тип хранения данных, `APENT OPTIMIZED` с применением компрессии типа `ZSTD` в конце после запроса указан тип распределения данных `RANDOMLY`. Для удаления мат представлении используется оператор `DROP MATERIALIZED VIEW`.



Запросы к мат представлениям выполняются так же, как к обычным таблицам. Также в Greenplum есть возможность создания индекса на основе мат представления как на основе обычной таблицы. Приведем примеры обновления мат представления. Стандартный способ удаляет данные и вставляет новые на основе запроса представления.

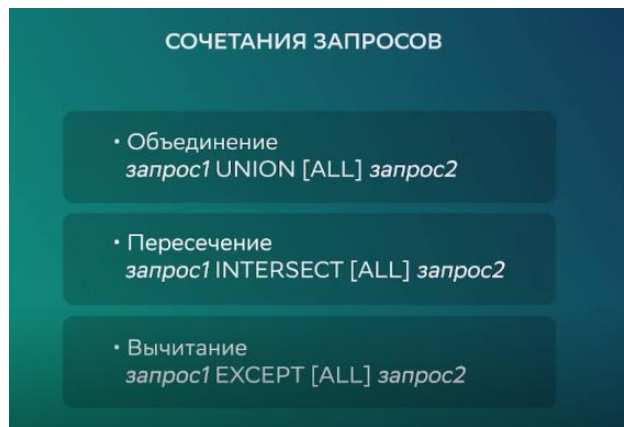


Способ с опцией WITH NO DATA удалит данные в мат представлении и приведет его в состояние UNSCANNABLE запросы к нему будут возвращать ошибку до тех пор, пока представление не будет обновлено стандартным способом. Это дает возможность временно ограничить доступ пользователей к неактуальным данным.

Лабораторная работа № 16. Запросы. Подзапросы

Цель работы: научиться работать с запросами и подзапросами

Поговорим о сочетании запросов и операциях над множествами. В Greenplum SQL-запросы можно комбинировать между собой и получая результат в виде операции над множествами такой, как объединение, пересечение и вычитание.



Общий синтаксис сочетания запросов показан на рисунке. Здесь *запрос1* и *запрос2* это запросы, в которых могут использоваться все возможности, рассмотренные до этого. При этом оба запроса должны содержать одинаковое количество столбца, а также соответствующие столбцы должны иметь одинаковые либо совместимые типы данных. Например, Except и Varchar или NewMeric и Integer.

Давайте рассмотрим каждую операцию по отдельности.

UNION ALL

• объединение результатов запросов

```
SELECT
client_id,
client_name
FROM clients
UNION ALL
SELECT
client_id,
client_name
FROM orders;
```

clients	
client_id	client_name
1	Наталья
1	Наталья
2	Мария
3	Максим
4	Алексей

orders		
order_id	client_id	client_name
2000	2	Лариса
3000	3	Максим
4000	4	Алексей
5000	103	Генрих
6000	104	Ринард

Команда Union позволяет объединить результаты нескольких запросов Select, например, выполнив следующую команду, мы получим все пары значения, идентификатор и имя. Если нас интересует объединение с абсолютно всеми значениями, то мы используем команду UNION ALL - она выводит комбинацию всех значений из обеих таблиц, даже повторы, если необходимо вывести только уникальные значения, то необходимо использовать команду Union. Она исключает все дублирующие строки. Ее поведение идентично применению оператора UNION ALL вместе с DISTINCT. Команда INTERSECT возвращает пересечение результата первого и второго запросов. Например, если выполнить следующую команду, в результате получим только те строки, которые присутствуют как в результате

первого, так и второго запроса. По аналогии с командой Union для вывода пересечения без фильтрации дубликатов, необходимо использовать команду INTERSECT ALL, если требуется оставить только уникальные записи, то используется команда INTERSECT.

INTERSECT ALL

- пересечение результатов запросов
- все значения

```
SELECT
  client_id,
  client_name
FROM clients
INTERSECT ALL
SELECT
  client_id,
  client_name
FROM orders;
```

client_id	client_name
1	Наталья
1	Наталья
2	Мария
3	Максим
4	Алексей

order_id	client_id	client_name
2000	2	Лариса
3000	3	Максим
4000	4	Алексей
5000	103	Генрих
6000	102	Ричард

client_id	client_name
3	Максим
4	Алексей
3	Максим
4	Алексей

И, наконец, команда EXCEPT выполняет операцию вычитания результатов запросов. Приведенный пример команды возвращает все строки, которые есть в результате первого запроса, но отсутствуют в результате второго.

EXCEPT ALL

- вычитание результатов запросов
- все значения

```
SELECT
  client_id,
  client_name
FROM clients
EXCEPT ALL
SELECT
  client_id,
  client_name
FROM orders;
```

client_id	client_name
1	Наталья
1	Наталья
2	Мария
3	Максим
4	Алексей

order_id	client_id	client_name
2000	2	Лариса
3000	3	Максим
4000	4	Алексей
5000	103	Генрих
6000	104	Ричард

client_id	client_name
1	Наталья
1	Наталья
2	Мария

Здесь, как и в предыдущих командах EXCEPT с опцией ALL оставляет повторяющиеся строки, а без такой опции отфильтрует дубликаты. Синтаксически сочетание запросов используется так же, как обычный запрос Select.

СИНТАКСИС

```
DELETE FROM clients
WHERE (client_id, client_name) in (
  SELECT
    client_id,
    client_name
  FROM clients
  INTERSECT
  SELECT
    client_id,
    client_name
  FROM orders);
```

clients	
client_id	client_name
1	Наталья
2	Мария
3	Максим
4	Алексей

orders		
order_id	client_id	client_name
2000	2	Лариса
3000	3	Максим
4000	4	Алексей
5000	103	Генрих
6000	104	Ричард

Например, показанное на рисунке выражение позволяет удалить из таблицы `CLIENTS` все записи, которые пересекаются в ней с таблицей `ORDERS`. Операции над множествами также можно вкладывать и соединять, например, если необходимо соединить три запроса можно написать запрос следующим образом, он будет выполняться в такой последовательности. Сначала выполняется команда `UNION ALL`. Затем из результата вычитается результат третьего запроса.

(запрос1 UNION ALL запрос2) EXCEPT ALL запрос3

Поскольку поиск и удаление дублирующихся строк требует дополнительных вычислительных ресурсов базы данных, рекомендуется использовать сочетания с ключевым словом `ALL`, используйте выражения без `ALL`, только если требуется выводить уникальные значения.

РЕКОМЕНДАЦИИ

Все значения

```
SELECT
  client_id,
  client_name
FROM clients
UNION ALL
SELECT
  client_id,
  client_name
FROM orders;
```

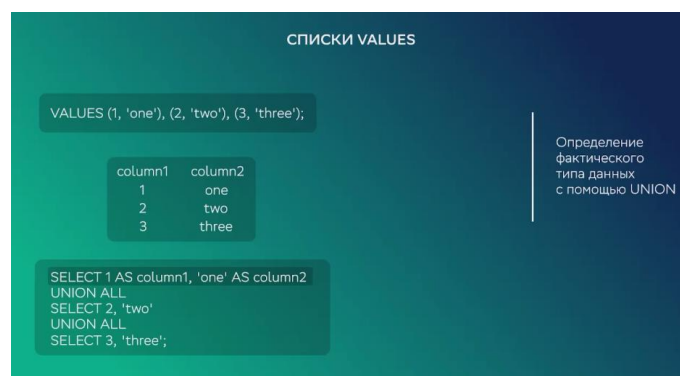
← Рекомендуется

познакомимся с ключевым словом `VALUES`. Предложение `VALUES` позволяет создать постоянную таблицу, которую можно использовать в

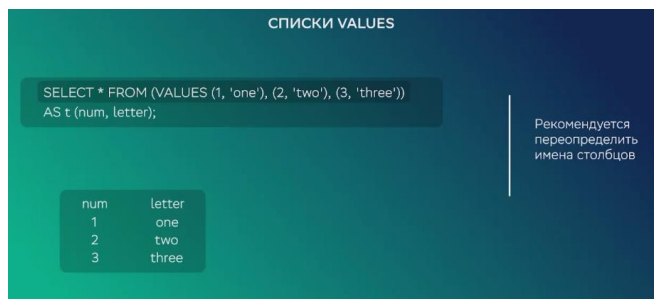
запросе, не создавая и не наполняя таблицу в базе данных. VALUES вычисляет значение строки или множество значений строк, заданное выражениями. Чаще всего эта команда используется для формирования таблицы констант в большой команде, но ее можно использовать и отдельно. В составе других команд синтаксис допускает использование VALUES везде, где допускается Select, так как грамматически она воспринимается как Select. С командой VALUES можно использовать предложения ORDER BY, LIMIT или равнозначные FETCH и OFFSET. Для каждого списка выражений в скобках создается строка таблицы. Все списки должны иметь одинаковое число элементов. То есть число столбцов таблицы и соответствующие элементы во всех списках должны иметь совместимые типы данных. Фактически тип данных столбцов результата определяется по тем же правилам, что и для Union.



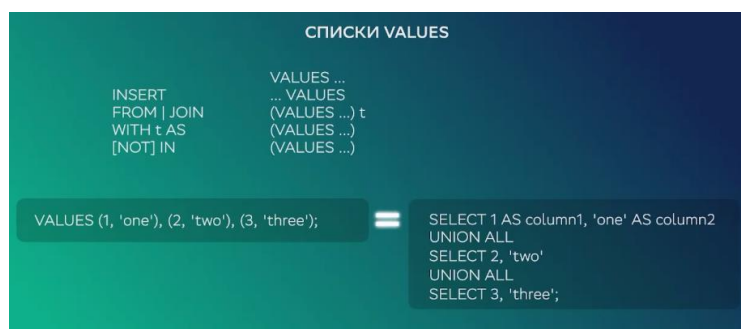
Например, следующие выражения Values, состоящие из трех элементов, вернет таблицу из двух столбца и трех строк. Такое выражение равносильно следующему запросу с применением конструкции UNION ALL. По умолчанию Greenplum назначает столбцам таблицы Values имена Column1, Column2 и так далее.



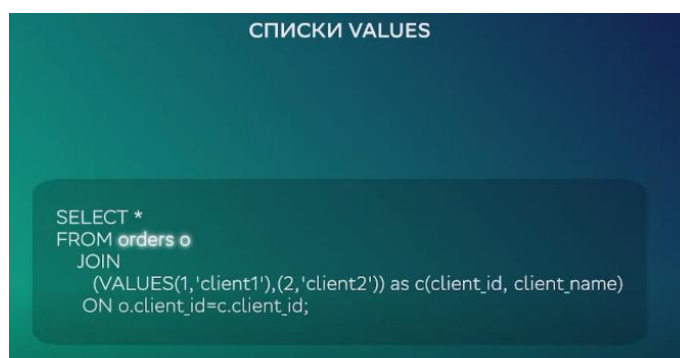
Имена столбцов не определены в стандарте SQL и в другой СУБД они могут быть другими. Поэтому лучше переопределить имена списком псевдонимов. Например, так мы присвоим столбцам названия NUM и LETTER.



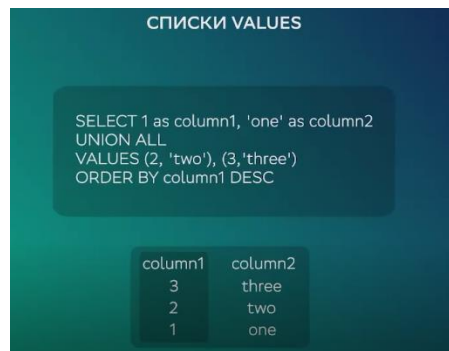
Синтаксически список VALUES с набором выражений равнозначен выражению SELECT FROM. Например, данное выражение VALUES эквивалентно следующему выражению SELECT.



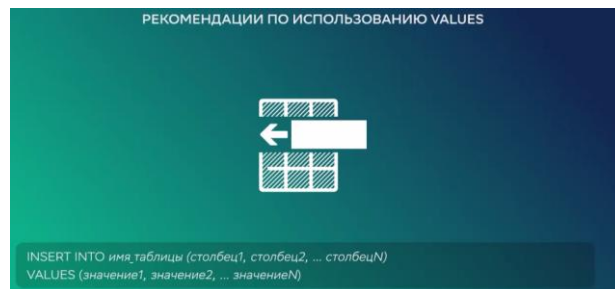
Поэтому применение списка VALUES допускается везде, где допустим SELECT. Например, вы можете использовать его в составе запроса, соединив с другой таблицей следующим образом. Здесь список клиентов, сформированный при помощи ключевого слова VALUES соединяется с таблицей Orders по полю ClientID.



Либо, к примеру, вы можете использовать список VALUES в составе UNION, как в следующем запросе с применением сортировки ORDER BY.



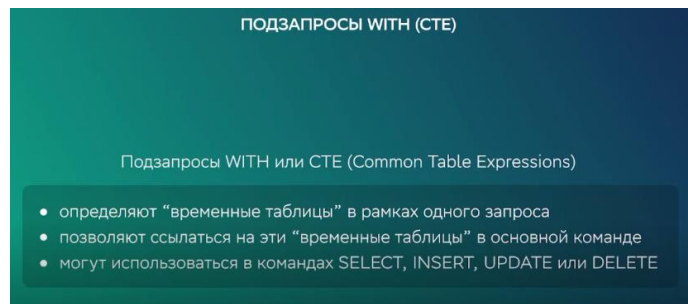
VALUES чаще всего используется как источник данных для команды INSERT, а также как подзапрос.



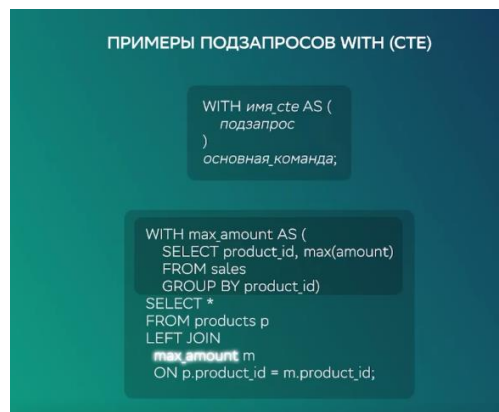
Следует избегать использования VALUES для операции со значительным количеством строк. Единственное исключение из этого правила команда INSERT, она позволяет вставить таким образом много строк, так как тип данных в колонках наследуется из уже существующей таблицы.



Под запросы WITH или общие табличные выражения CTE - удобный способ записывать дополнительные подзапросы для применения в больших запросах. Общее табличное выражение можно представить, как определение временных таблиц, которые существуют только для одного запроса. CTE используются в конструкции WITH и позволяют ссылаться на эти временные таблицы в основных командах, например, SELECT, INSERT, UPDATE или DELETE.



Общий синтаксис CTE выглядит следующим образом: Здесь имя CTE это имя временной таблицы CTE, к которой можно обратиться из основной команд, подзапрос - это запрос, формирующий временную таблицу и основная команда SELECT, INSERT, UPDATE или DELETE. Рассмотрим пример запроса с использованием CTE. И здесь, в качестве основной команды используется запрос соединения двух таблиц PRODUCTS и MAX_AMOUNT, где MAX_AMOUNT это подзапрос CTE, который предварительно рассчитывается в конструкции WITH на основе сгруппированного запроса к таблице SALES.



Основное предназначение конструкции WITH заключается в разбиении сложных запросов на простые части. Например, следующее запрос выводит итоги по продажам только для передовых регионов.

ПРИМЕРЫ ПОДЗАПРОСОВ WITH (CTE)

```

WITH regional_sales AS (
  SELECT region, SUM(amount) AS total_sales
  FROM orders
  GROUP BY region
),
top_regions AS (
  SELECT region
  FROM regional_sales
  WHERE total_sales > (SELECT SUM(total_sales)/10
                      FROM regional_sales)
)
SELECT region,
       product,
       SUM(quantity) AS product_units,
       SUM(amount) AS product_sales
FROM orders
WHERE region IN (SELECT region
                 FROM top_regions)
GROUP BY region, product;

```

Здесь предложение WITH определяет два дополнительных подзапроса CTE, REGIONAL_SALES и TOP_REGIONS. Обратите внимание на то, что результат REGIONAL_SALES используется в TOP_REGIONS, а затем результат TOP_REGIONS используется в основном запросе Selected.

Таким образом, мы можем сослаться на запросы CTE объявленные выше. Такой запрос можно переписать и без конструкции улий WITH. В этом случае подзапросы CTE перемещаются в основную часть запроса. Обратите внимание на то, что нам понадобилось использовать два уровня, вложенных подзапроса в Select, что значительно усложнило восприятие текста запроса.

Подзапросы WITH имеют еще одно полезные свойства. Обычно они вычисляются только один раз для всего родительского запроса. Даже если этот запрос или соседняя подзапросы WITH обращаются к ним неоднократно.

ПРИМЕРЫ ПОДЗАПРОСОВ WITH (CTE)

```

WITH regional_sales AS (
  SELECT region, SUM(amount) AS total_sales
  FROM orders
  GROUP BY region
),
top_regions AS (
  SELECT region
  FROM regional_sales
  WHERE total_sales > (SELECT SUM(total_sales)/10
                      FROM regional_sales)
)
SELECT region,
       product,
       SUM(quantity) AS product_units,
       SUM(amount) AS product_sales
FROM orders
WHERE region IN (SELECT region
                 FROM top_regions)
GROUP BY region, product;

```

→

```

SELECT region,
       product,
       SUM(quantity) AS product_units,
       SUM(amount) AS product_sales
FROM orders
WHERE region IN
  (SELECT region
   FROM (
     SELECT region, SUM(amount) AS total_sales
     FROM orders
     GROUP BY region
   ) AS regional_sales
   WHERE total_sales >
     (SELECT SUM(total_sales)/10
      FROM (SELECT region, SUM(amount) AS total_sales
              FROM orders
              GROUP BY region)
     )
  )
GROUP BY region, product;

```

Например, в нашем запросе мы можем добавить еще одно соединение с под запросом TOP_REGIONS. В этом случае, несмотря на то, что TOP_REGIONS упоминается в основном запросе два раза, он будет вычислен

только один раз для всего запроса. Таким образом, сложные вычисления, результаты которых нужны в нескольких местах, можно вносить в запросы WITH в целях оптимизации.

CTE также можно использовать внутри конструкции INSERT. Так SELECT часть в конструкции INSERT SELECT может быть заменена на подзапрос CTE выборкой из него.



Аналогично CTE можно использовать в таких командах, как Update и Delete. Примеры конструкции вы видите на рисунке.



Теперь давайте поговорим о такой возможности, как рекурсивные запросы CTE. В отличие от обычного запроса, рекурсивный запрос может циклично обращаться к собственному же результату. Благодаря этому появляется возможность строить запросы, которые отображают иерархические данные. В общем виде рекурсивный запрос WITH всегда записывается как не рекурсивная часть потом UNION или UNION ALL, а затем рекурсивная часть, где только в рекурсивной части можно обратиться к результату запроса.

РЕКУРСИВНЫЕ ЗАПРОСЫ CTE

- Могут циклично обращаться к собственному результату
- Отображают иерархические данные

```
WITH RECURSIVE имя_cte AS
(
  (нерекурсивная часть)
  UNION [ ALL ]
  (рекурсивная часть)
  Основная команда
)
```

Приведем пример. Для начала рассмотрим структуру таблицы, которая будет использоваться в запросе. Это таблица сотрудников с колонками, идентификатор сотрудника, фамилия сотрудника и идентификатор менеджера сотрудника. Целью написания рекурсивного запроса будет вывести всех менеджеров Кандинского, то есть его менеджера, менеджера его менеджера и так далее. Перейдем теперь непосредственно к структуре рекурсивного CTE. Здесь после слова WITH, добавляется новое служебное слово RECURSIVE. Затем следует не рекурсивная часть запроса, здесь выбирается сотрудник Кандинский, его фамилия и идентификатор его менеджера. Представим себе некую виртуальную таблицу, которая содержит результаты запроса CTE после выполнения SELECT части в ней будет храниться информация о Кандинском. Далее начинается выполнение рекурсивной части. Она связана с не рекурсивной частью через UNION ALL и будет повторяться, пока будут получаться новые строки. Не рекурсивная часть больше выполняться не будет.

РЕКУРСИВНЫЕ ЗАПРОСЫ CTE. ПРИМЕР

```
WITH RECURSIVE cte1 AS (
  SELECT e.last_name, e.mgr_id,
  FROM employees e
  WHERE e.last_name = 'Кандинский'
  UNION ALL
  SELECT e.last_name, e.mgr_id,
  FROM employees AS e, cte1
  WHERE cte1.mgr_id = e.id
)
SELECT *
FROM cte1;
```

id	last_name	mgr_id
110	Верещагин	408
118	Малевич	408
234	Кандинский	118
305	Врубель	110
408	Репин	NULL

last_name	mgr_id
Кандинский	118

Итак, в рекурсивной части запроса будут найдены сотрудники, которые являются менеджерами сотрудников, полученных на предыдущих шагах рекурсии. В запросе это соединение между таблицей Employees и CTE1 через равенство Id MANAGER из CTE1 и ID сотрудника из Employees. Так на

первом шаге рекурсии будет взята строка, полученная после выполнения не рекурсивной части (Информация О Кандинском) в таблице Employees будет произведен поиск менеджера Кандинского с ID_118 это Малевич. Именно он добавляется в CTE и после выполнения первого шага рекурсии через объединение результата рекурсивного запроса с CTE1 с помощью UNION ALL. Затем рекурсия повторяется, ищутся менеджеры сотрудников из CTE1, добавленная в CTE на предыдущем шаге, то есть для Малевича, его менеджером является Репин с ID_408. Информация о Репине добавляется в CTE, на следующем шаге рекурсии ищется менеджер для Репина. Менеджера у Репина нет, новые строки в CTE не добавляются и создание CTE завершается.

РЕКУРСИВНЫЕ ЗАПРОСЫ CTE. ПРИМЕР

```

WITH RECURSIVE cte1 AS (
  SELECT e.last_name, e.mgr_id,
  FROM employees e
  WHERE e.last_name = 'Кандинский'
  UNION ALL
  SELECT e.last_name, e.mgr_id,
  FROM employees AS e, cte1
  WHERE cte1.mgr_id = e.id
)
SELECT *
FROM cte1;

```

employees		
id	last_name	mgr_id
110	Верещагин	408
118	Малевич	408
234	Кандинский	118
305	Врубель	110
408	Репин	NULL

CTE	
last_name	mgr_id
Кандинский	118
Малевич	408
Репин	NULL

На последнем шаге запроса собирается информация из CTE, пользователю возвращается все содержимое CTE, то есть информация о Кандинском, его менеджере Малевиче и менеджере его менеджера Репине. Так происходит выполнение рекурсивных запросов в GreenPlum. Также важно отметить, что связь между рекурсивной и не рекурсивной частью запроса происходит с помощью выражения UNION ALL, что может приводить к появлению дубликатов, если для рекурсии не задано четкое условие остановки.

Предположим, что в таблице Employees информация о менеджере Репине оказалась ошибочной, и менеджером Репина является он сам. На каждом шаге рекурсии после того, как Репин впервые оказался в CTEЮ, будет добавляться информация о Репине, так как он является менеджером

самого себя. А UNION ALL не исключает дубликаты при объединении CTE1 и результатов рекурсивного шага.

РЕКУРСИВНЫЕ ЗАПРОСЫ CTE. ПРИМЕР																							
<pre>WITH RECURSIVE cte1 AS (SELECT e.last_name, e.mgr_id, FROM employees e WHERE e.last_name = 'Кандинский' UNION ALL SELECT e.last_name, e.mgr_id, FROM employees AS e, cte1 WHERE cte1.mgr_id = e.id) SELECT * FROM cte1;</pre>																							
<table><tr><th colspan="3">employees</th></tr><tr><th>id</th><th>last_name</th><th>mgr_id</th></tr><tr><td>110</td><td>Ворещагин</td><td>408</td></tr><tr><td>118</td><td>Малевиц</td><td>408</td></tr><tr><td>234</td><td>Кандинский</td><td>118</td></tr><tr><td>305</td><td>Врубель</td><td>110</td></tr><tr><td>408</td><td>Репин</td><td>408</td></tr></table>			employees			id	last_name	mgr_id	110	Ворещагин	408	118	Малевиц	408	234	Кандинский	118	305	Врубель	110	408	Репин	408
employees																							
id	last_name	mgr_id																					
110	Ворещагин	408																					
118	Малевиц	408																					
234	Кандинский	118																					
305	Врубель	110																					
408	Репин	408																					
<table><tr><th colspan="3">CTE</th></tr><tr><th>last_name</th><th>mgr_id</th><th></th></tr><tr><td>Кандинский</td><td>118</td><td></td></tr><tr><td>Малевиц</td><td>408</td><td></td></tr><tr><td>Репин</td><td>408</td><td></td></tr><tr><td>Репин</td><td>408</td><td></td></tr><tr><td>Репин</td><td>408</td><td></td></tr></table>			CTE			last_name	mgr_id		Кандинский	118		Малевиц	408		Репин	408		Репин	408		Репин	408	
CTE																							
last_name	mgr_id																						
Кандинский	118																						
Малевиц	408																						
Репин	408																						
Репин	408																						
Репин	408																						

Для остановки можно использовать подсчет шагов рекурсии. Для этого добавим в определение запроса новое поле STEP - количество шагов, для не рекурсивной части оно равно нулю. Далее на каждом шаге рекурсии число шагов STEP будет увеличиваться. И когда оно достигнет четырех, запрос завершится. Ограничение по количеству шагов выбирается пользователем самостоятельно, оптимальным для остановки рекурсии является самый глубокий уровень рекурсии, если он известен заранее.

РЕКУРСИВНЫЕ ЗАПРОСЫ CTE. ПРИМЕР																							
<pre>WITH RECURSIVE cte1 AS (SELECT e.last_name, e.mgr_id, 0 AS step FROM employees e WHERE e.last_name = 'Кандинский' UNION ALL SELECT e.last_name, e.mgr_id, step + 1 FROM employees AS e, cte1 WHERE cte1.mgr_id = e.id AND step < 4) SELECT * FROM cte1;</pre>																							
<table><tr><th colspan="3">employees</th></tr><tr><th>id</th><th>last_name</th><th>mgr_id</th></tr><tr><td>110</td><td>Ворещагин</td><td>408</td></tr><tr><td>118</td><td>Малевиц</td><td>408</td></tr><tr><td>234</td><td>Кандинский</td><td>118</td></tr><tr><td>305</td><td>Врубель</td><td>110</td></tr><tr><td>408</td><td>Репин</td><td>408</td></tr></table>			employees			id	last_name	mgr_id	110	Ворещагин	408	118	Малевиц	408	234	Кандинский	118	305	Врубель	110	408	Репин	408
employees																							
id	last_name	mgr_id																					
110	Ворещагин	408																					
118	Малевиц	408																					
234	Кандинский	118																					
305	Врубель	110																					
408	Репин	408																					
<table><tr><th colspan="3">CTE</th></tr><tr><th>last_name</th><th>mgr_id</th><th>step</th></tr><tr><td>Кандинский</td><td>118</td><td>0</td></tr><tr><td>Малевиц</td><td>408</td><td>1</td></tr><tr><td>Репин</td><td>408</td><td>2</td></tr><tr><td>Репин</td><td>408</td><td>3</td></tr><tr><td>Репин</td><td>408</td><td>4</td></tr></table>			CTE			last_name	mgr_id	step	Кандинский	118	0	Малевиц	408	1	Репин	408	2	Репин	408	3	Репин	408	4
CTE																							
last_name	mgr_id	step																					
Кандинский	118	0																					
Малевиц	408	1																					
Репин	408	2																					
Репин	408	3																					
Репин	408	4																					

Подведем резюме. Под запрос CTE не хранится как объект базы данных и действует только в пределах основного запроса, CTE может ссылаться сам на себя, то есть быть рекурсивным, подзапросы удобно использовать в следующих случаях: для аккуратного оформления кода в большом запросе, для создания рекурсивных запросов, для многократного обращения к одной и той же таблицы внутри одного запроса, для создания аналога представления или временной таблицы без создания объекта и, соответственно, без необходимости обновлять метаданные. Все это позволяет сделать код удобным для чтения и обеспечить максимально быструю обработку запроса. Обратите внимание, если вы считаете, что результат выполнения запроса

подзапроса понадобится вам еще много раз, то это уже кандидат для создания полноценного представления.

Лабораторная работа № 17. Оконные функции

Цель работы: познакомиться с технологией работы с оконными функциями

Оконные и аналитические функции - это расширение языка SQL. Они позволяют разбивать строки данных на группы или окна, упорядочивать строки внутри окна и применять к данным набор функций. Эти функции позволяют реализовывать внутри СУБД сложную логику обработки данных, упрощают некоторые запросы, позволяют избавиться от лишних соединений и группировок. Они часто применяются, когда требуется сравнить показатель со следующим или предыдущим значением в рамках определенного диапазона строк, вычислить агрегатную функцию по части записей и для других подобных задач.

ОКОННЫЕ ФУНКЦИИ

```
ИМЯ_ФУНКЦИИ ([ИМЯ_КОЛОНКИ]) OVER (  
  [PARTITION BY колонка1] [ORDER BY колонка2]  
  [ROWS BETWEEN n FOLLOWING | PRECEDING | CURRENT ROW  
    AND m FOLLOWING | PRECEDING | CURRENT ROW]  
)
```

- Реализуют сложную логику обработки запросов
- Упрощают некоторые запросы
- Избавляют от лишних join и группировок

Конструкция начинается с указания конкретной функции, которая будет применена к данным. В таблице приведены часто встречающиеся оконные и аналитические функции. Среди них есть функции ранжирования данных: Rank, Dense_rank, Row_number. Функции получения конкретных элементов набора данных: Lag, Lead, First_value, Last_Value. Функции вычисления агрегатов: Sum, Count, Avg, Min, Max.

ОПРЕДЕЛЕНИЕ ФУНКЦИИ

```

имя_функции ([имя_колонки]) OVER (
  [PARTITION BY колонка1] [ORDER BY колонка2]
  [ROWS BETWEEN n FOLLOWING | PRECEDING | CURRENT ROW
   AND m FOLLOWING | PRECEDING | CURRENT ROW]
)
  
```

Название функций	Описание
Rank/Dense_rank	Ранг значения
Row_number	Номер строки
Lag,Lead	Возвращает значение поля в записи, отстоящей на N от текущей назад или вперед
First_value,Last_Value	Первое значение в окне, последнее значение в окне
Sum	Сумма элементов окна
Count	Количество элементов в окне
Avg	Среднее элементов окна
Min,Max	Минимальный и максимальный элемент в окне

Описание окна состоит из нескольких элементов: PARTITION BY для указания ключа группы, ORDER BY для указания ключа сортировки данных внутри группы, ROWS для описания размера окна. Все элементы описания окна являются опциональными.

ОПРЕДЕЛЕНИЕ ОКНА

```

имя_функции ([имя_колонки]) OVER (
  [PARTITION BY колонка1] [ORDER BY колонка2]
  [ROWS BETWEEN n FOLLOWING | PRECEDING | CURRENT ROW
   AND m FOLLOWING | PRECEDING | CURRENT ROW]
)
  
```

PARTITION BY ключ группы

ORDER BY ключ сортировки внутри группы

ROWS размер окна

Рассмотрим пример задания окна для набора тестовых данных. Пусть мы описали окно выражением OVER для групп с одинаковым номером отдела и сортировка внутри группы по фамилии, по возрастанию. Это означает, что во время обработки все данные таблицы будут разбиты на группы с одинаковым Department_Number. Внутри каждой группы записи будут упорядочены по значению показателя Last_Name.

ОПРЕДЕЛЕНИЕ ОКНА

```

OVER (PARTITION BY department_no ORDER BY last_name ASC)
  
```

Идентификатор группы – department_no
Сортировка внутри группы по none last_name

employee_id	department_no	first_name	last_name
1	0	Алексей	Иванов
2	0	Марина	Петрова
5	1	Елена	Воронина
6	1	Елена	Кузнецова
4	1	Алексей	Сергеев
3	1	Юрий	Сидоров
7	2	Нина	Михайлова
9	2	Мамед	Салим-Оглы
8	2	Оксана	Сидорина
10	3	Александр	Коняров

Для определения размера окна используется конструкция Rows, которая задает верхнюю и нижнюю границу окна. Ключевые слова, которыми можно задать границу, приведены в таблице.

ОПРЕДЕЛЕНИЕ РАЗМЕРА ОКНА	
ROWS BETWEEN <граница1> AND <граница2>	
Граница	Описание
Unbounded preceding	С первой записи окна
N preceding	Начиная с/до N-ой записи до текущей
current row	Текущая запись
N following	Начиная с/до N-ой записи после текущей
Unbounded following	До последней записи окна
Окно по умолчанию: ROWS BETWEEN unbounded preceding AND unbounded following	

По умолчанию, если граница не задана явно, окно — это записи от начала до конца группы.

В определении размера окна может применяться ключевое слово CURRENT ROW, для каждой из обрабатываемых записей - это она сама. Если мы зададим определение окна, как ROWS BETWEEN, CURRENT ROW AND CURRENT ROW, то при обработке каждой записи будут учитываться только значения колонок самой этой записи. При задании определения окна, как ROWS BETWEEN, CURRENT ROW AND 2 FOLLOWING, мы можем применять функции к атрибутам текущей записи и двум следующим за ней записям. При задании окна выражением ROWS BETWEEN unbounded preceding AND current row функции будут применяться к значениям колонок записей от начала окна и до текущей записи. Например, задав так окно, можно вычислить нарастающий итог.

ОПРЕДЕЛЕНИЕ РАЗМЕРА ОКНА. ПРИМЕРЫ	
PARTITION BY partition,key ORDER BY order,key ROWS BETWEEN unbounded preceding AND current row	
Ключ группы	Ключ сортировки
n	1
n	2
n	3
...	4
...	...
n	k-1
n	k
n	k+1
n	k+2
n	...
n	m

← unbounded preceding AND

← current row

Рассмотрим пример использования агрегатных оконных функций. Пусть у нас есть таблица с данными о зарплатах сотрудников какой-то организации. Сами сотрудники и идентификатор отдела, в котором они

работают, находится в таблице EMPLOYEE. Данные об изменении их зарплат в таблице SALARY. Пускай мы поставили себе задачу вычислить суммарную зарплату всех сотрудников в рамках каждого отдела на данный момент среднюю зарплату в рамках отдела и вывести для каждого сотрудника долю его зарплаты в суммарной зарплате его отдела. Это можно посчитать приведенным на рисунке запросом, он использует агрегатные оконные функции: Сумма и Средняя, результаты выделены рамкой.

АГРЕГАТНЫЕ ОКОННЫЕ ФУНКЦИИ

```

SELECT e.*,
       Sum(s.salary_amt)
       OVER (PARTITION BY e.department_no)
       AS department_total, /* Сумма по окну */
       Avg(s.salary_amt)
       OVER (PARTITION BY e.department_no)
       AS department_avg, /* Среднее по окну */
       s.salary_amt / department_total as salary_part
FROM employee e INNER JOIN salary s
ON s.employee_id = e.employee_id
WHERE Current_Date BETWEEN s.salary_start_dt
AND s.salary_end_dt

```

employee_id	department_no	salary_amt	department_total	department_avg	salary_part
2	0	95 000,00	170 000,00	85 000,00	0,56
1	0	75 000,00	170 000,00	85 000,00	0,44
5	1	120 000,00	470 000,00	117 500,00	0,26
3	1	120 000,00	470 000,00	117 500,00	0,26
6	1	130 000,00	470 000,00	117 500,00	0,28
4	1	100 000,00	470 000,00	117 500,00	0,21
8	2	90 000,00	220 000,00	73 333,33	0,27
7	2	70 000,00	220 000,00	73 333,33	0,32
9	2	60 000,00	220 000,00	73 333,33	0,41
10	3	150 000,00	150 000,00	150 000,00	1

Можно отметить, что для вычисления не понадобилось добавлять в запрос агрегацию или подзапросы. Функции Ранжирования применяются для вычисления ранга или места строки в окне при сортировке по какому-либо показателю. Допустим, что требуется отранжировать сотрудников по убыванию зарплаты в каждом из отделов организации. Запрос для выполнения этого расчета приведен на слайде. В нем используются функции RANK и DENSE_RANK. Как видно на примерах у функции RANK и DENSE_RANK нет параметров, так как ее поведение полностью определяется предложением OVER. Обратите внимание, что у двух сотрудников отдела №1 одинаковая зарплата.

ФУНКЦИИ РАНЖИРОВАНИЯ

```

SELECT e.*,
       Rank()
       OVER (PARTITION BY e.department_no
            ORDER BY s.salary_amt DESC)
       AS _rank,
       Dense_Rank()
       OVER (PARTITION BY e.department_no
            ORDER BY s.salary_amt DESC)
       AS _dense_rank
FROM employee e
INNER JOIN salary s
ON e.employee_id = s.employee_id
WHERE Current_Date BETWEEN s.salary_start_dt
AND s.salary_end_dt

```

employee_id	department_no	salary_amt	_rank	_dense_rank
2	0	95 000,00	1	1
1	0	75 000,00	2	2
6	1	130 000,00	1	1
5	1	120 000,00	2	2
3	1	120 000,00	2	2
4	1	100 000,00	4	3
9	2	90 000,00	1	1
7	2	70 000,00	2	2
8	2	60 000,00	3	3
10	3	150 000,00	1	1

Обе функции поставили их на второе место, но в значениях функции RANK есть пропуски, а в значениях функции DENSE_RANK их нет. Рассмотрим пример с использованием нестандартного задания окна.

Допустим, что в таблице с историей изменения зарплаты SALARY могут быть разрывы в истории. То есть если для одного из сотрудников есть несколько записей о зарплате. Дата начала действия о какой-то записи не равна дате окончания действия предыдущей записи + один день. Размер разницы между SALARY_START_DT текущей записи и SALARY_END_DT предыдущей, может быть вычислен запросом на слайде.

РАСШИРЕННЫЕ ФУНКЦИИ ЗАДАНИЯ ОКНА. ПРИМЕР

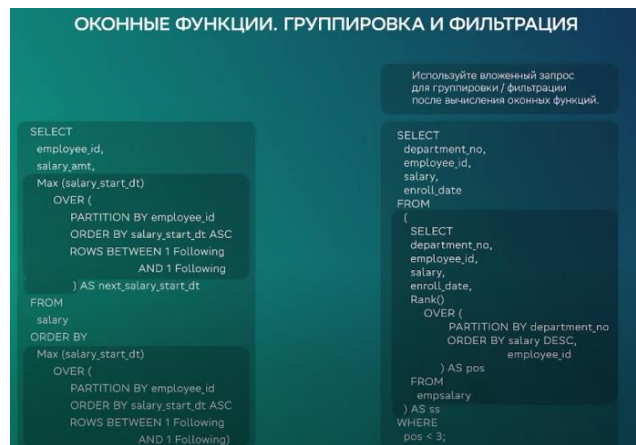
```

SELECT
  employee_id,
  salary_start_dt,
  salary_end_dt,
  next_salary_start_dt - salary_end_dt AS gap_size,
  salary_smt
FROM
(
  SELECT
    employee_id,
    salary_start_dt,
    salary_end_dt,
    Max (salary_start_dt) Over (
      PARTITION BY employee_id
      ORDER BY
        salary_start_dt ASC
        ROWS BETWEEN 1 Following
        AND 1 Following
    ) AS next_salary_start_dt,
    salary_smt
  FROM salary
)
ORDER BY
  employee_id,
  salary_start_dt

```

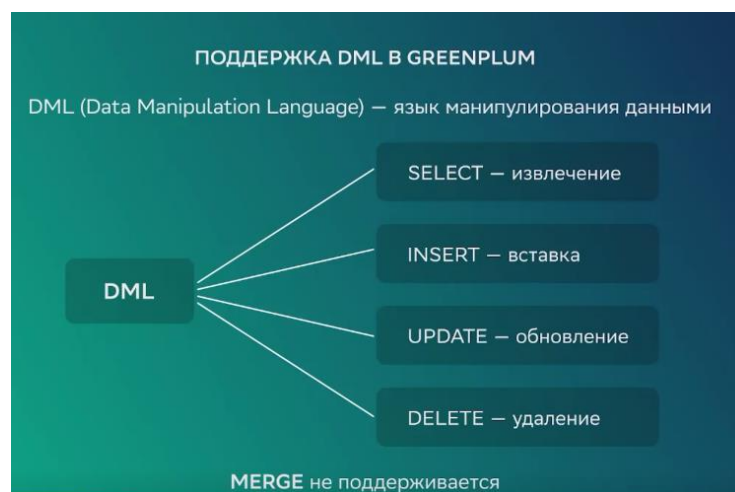
employee_id	salary_start_dt	salary_end_dt	next_salary_start_dt	gap_size
1	03.01.2004	31.12.2008	01.01.2009	1
1	01.01.2009	31.12.2012	01.01.2013	1
1	01.01.2013	31.12.9999	null	null
2	15.09.2014	31.12.2016	01.01.2017	1
2	01.01.2017	31.12.9999	null	null
3	01.10.2012	30.06.2016	01.07.2016	1
3	01.07.2016	31.12.9999	null	null
4	01.10.2012	12.01.2018	16.01.2018	4
4	16.01.2018	31.12.9999	null	null
5	16.05.2018	31.12.9999	null	null
6	12.11.2019	31.12.9999	null	null
7	14.12.2009	31.12.2010	01.01.2011	1
7	01.01.2011	01.05.2013	02.05.2013	1
7	02.05.2013	31.12.9999	null	null
8	27.08.2010	31.12.9999	null	null

В этом запросе используется агрегирующая функция MAX, размер окна для которой - это записи, начиная с первой после и до первой после для окна упорядоченного по возрастанию SALARY_START_DT. То есть функция MAX применяется к значению одной из следующей записи. В этом примере для сотрудника №4 есть разрыв в три дня, когда значение зарплаты не определено, это даты с 13 по 15 января 2018 года. Подобного же результата можно достичь, используя функцию LT. Оконные функции разрешается использовать в запросе только в списке SELECT и в предложении ORDER BY. Во всех остальных предложениях, включая группы BY, HAVING и WHERE, они запрещены. Это объясняется тем, что логически они выполняются после выполнения этих операторов, а также после не оконных агрегатных функций. Это значит, что агрегатную функцию можно вызывать в аргументах оконной, но не наоборот. Если вам нужно отфильтровать или сгруппировать строки после вычисления оконных функций вы можете использовать вложенный запрос. Пример использования оконной функции в подзапросе показан на рисунке. Данный запрос покажет только те строки внутреннего запроса, у которых RANK порядковый номер меньше 3.

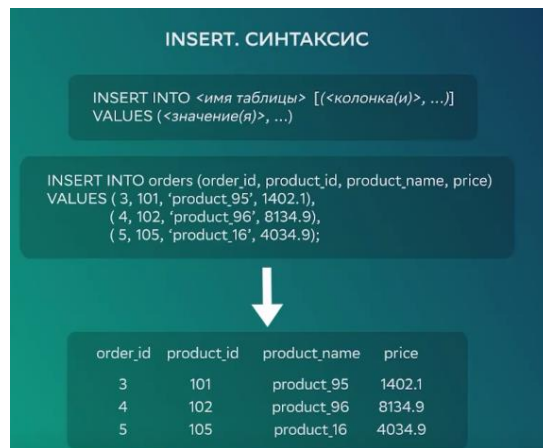


Лабораторная работа № 18 DML операции. Транзакции

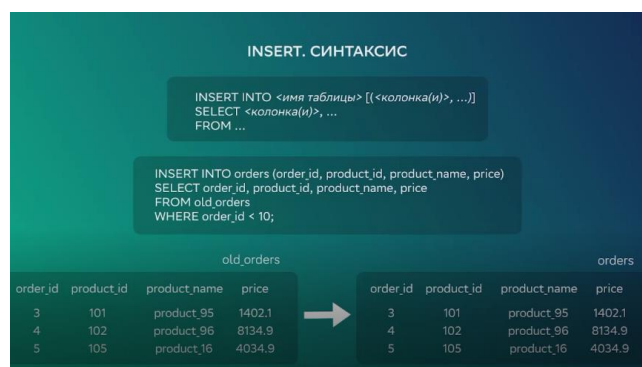
В этой теме мы поговорим об основных командах для манипулирования данными DML. DML это язык манипулирования данными, который используется для извлечения вставки, удаления и обновления данных. Greenplum поддерживает большинство функций стандарта SQL 2003, в том числе такие DML-команды, как Select-извлечение, INSERT-вставка, UPDATE-обновление и DELETE-удаление данных. Команда MERGE на данный момент в Greenplum не поддерживается.



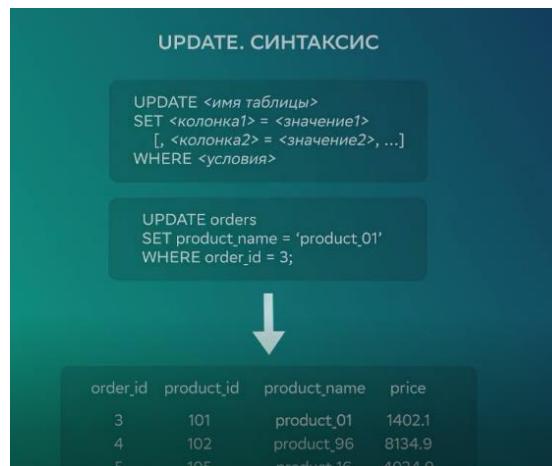
Команда SELECT подробно рассмотрена в отдельном видео. Давайте кратко разберем особенности использования остальных команд в Greenplum. Команда INSERT используется для вставки данных в таблицу. В Greenplum поддерживаются два варианта вставки данных: Первый с помощью конструкции INSERT INTO VALUES в этом случае данные для вставки указываются в тексте команды.



Например, следующая команда вставляет в таблицу ORDERS данные заказа №3 в соответствующие колонки. Также после слова VALUES можно указывать несколько строк для вставки следующим способом: Второй способ вставки данных выполняется с помощью конструкции INSERT INTO SELECT FROM. Здесь данные для вставки формируются на основе запроса. Пример скрипт вставки данных на основе запроса показаны на рисунке.



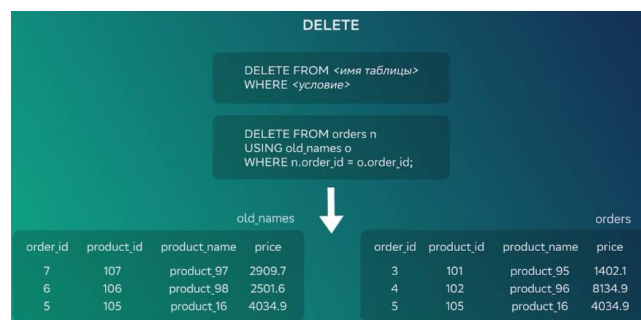
Команда Update используется для обновления существующих данных в таблице. Конструкция выглядит следующим образом: здесь указывается имя таблицы одна или несколько колонок и опциональное условие WHERE на строке обновления. Например, следующий скрипт обновит имя продукта для заказа №3 назначение Product_01. Также в команде Update можно использовать запрос с соединением. Для того, чтобы обновлять соединившиеся строки.



Команда Delete, используется для удаления строк из таблиц. В команде необходимо указать имя таблицы и условия, по которому должны быть удалены строки. Примеры удаление строки с идентификатором продукта 6 изображен на рисунке.

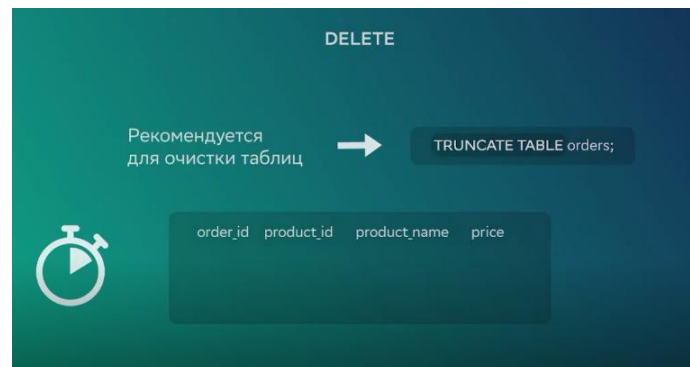


Также в Greenplum возможен вариант удаления строк по условию соединения с другой таблицей. Для этого используется ключевое слово USING, как в данном примере.

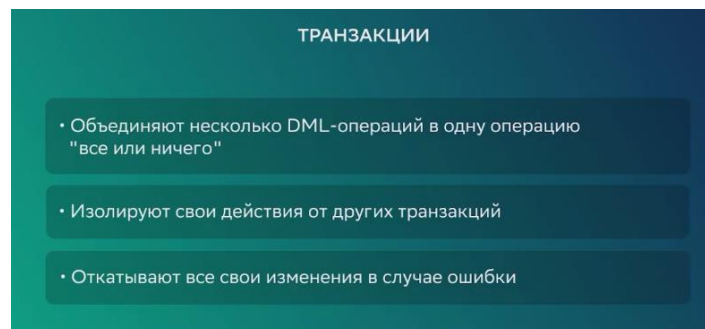


После выполнения такой команды будут удалены только те заказы, идентификаторы которых присутствует в таблице old_names. Выполнение команды Delete без условия WHERE, удалит все данные таблицы. Следует

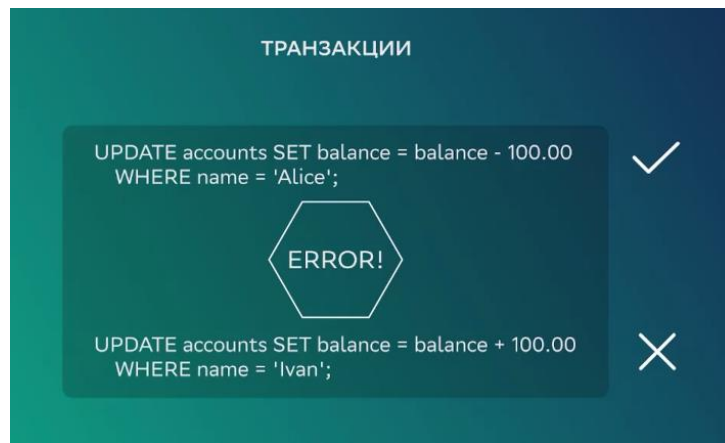
учесть, что для очистки всей таблицы рекомендуется использовать команду TRUNCATE TABLE. Данная команда выполняется быстрее и освобождает в базе данных занимаемое таблицей место.



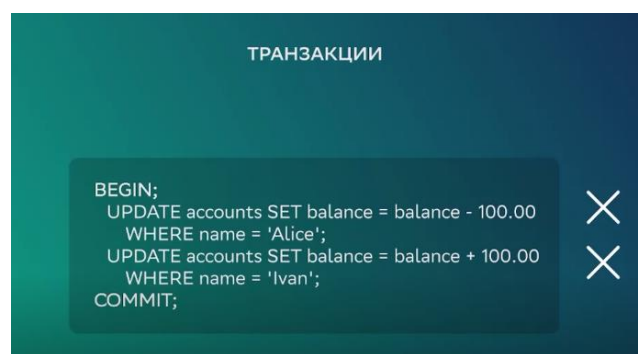
Итак, вы вспомнили операции, используемые для манипулирования данными. Теперь мы поговорим о таком фундаментальном для всех СУБД понятии, как транзакции. Суть транзакции заключается в том, что она объединяет последовательность DML-операций в одну операцию, которая выполняется по принципу все или ничего. При этом промежуточное состояние внутри этой последовательности не видны другим транзакциями. И если что-то мешает успешно завершить транзакцию, ни один из результатов этих DML-операций не сохранится в базе данных.



Для понимания работы транзакции рассмотрим пример типичной операции в СУБД: Перевод денег с одного счета на другой. Предположим, что мы хотим перевести 100 Р со счета Алисы на счет Ивана. В нашем примере первая операция списывает 100 Р со счета Алисы, а вторая операция зачисляет 100 Р на счет Ивана. Давайте теперь представим, что в середине что-то пошло не так. И из баланса Алисы было списано 100 Р, но затем произошел сбой, и на счет Ивана ничего не зачислилось, 100 Р как будто повисли в воздухе.

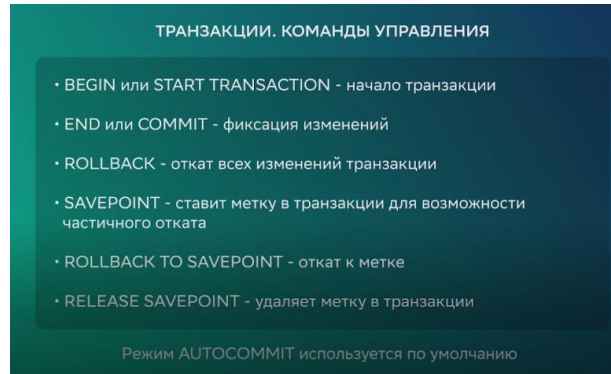


Для исключения подобных ситуаций в СУБД существуют транзакции. Если нам известно, что операции должны выполняться совместно, и при этом они не могут выполняться по отдельности, то мы объединяем эти операции в блок, который начинается со слова BEGIN и заканчивается словом COMMIT, фиксирующим изменения данных. Таким образом, мы определяем, что две операции находятся в рамках одной транзакции. Теперь, если транзакция была начата и сумма баланса Алисы уменьшилась на 100 Р, но увеличить баланс Иванова на 100 Р не удалось, то и изменение баланса Алиса не будут сохранены. Другими словами, транзакция - это группа последовательных операций, представляющая собой логическую единицу работы с данными и служащая для перевода базы данных из одного не противоречивого состояния в другое.

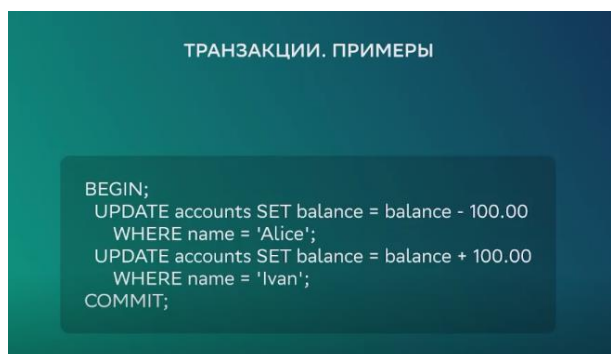


В Greenplum для управления транзакции используются следующие команды BEGIN или START TRANSACTION объявляют начало транзакции END или COMMIT- фиксирует изменения транзакции, ROLLBACK откладывает изменение транзакции, SAVEPOINT ставит отметку в транзакции для возможности частичного отката. Вы можете откатить

изменения, сделанные после этой метки, не затрагивая изменения, сделанные до этой метки. ROLLBACK TO SAVEPOINT откат к ранее установленной метки. RELEASE SAVEPOINT удаляет метку. В GreenPlum по умолчанию включен режим AUTOCOMMIT.



В этом режиме любая выполненная DML-операция, которая не содержит ошибок, будет автоматически зафиксирована в базе данных без необходимости явно прописывать COMMIT. Для того, чтобы отключить режим AUTOCOMMIT необходимо явно объявлять начало и конец транзакции, например, с помощью команд BEGIN и END. Еще одной особенностью GreenPlum является то, что транзакции не разрешены внутри хранимых процедур, то есть вся хранимая процедура всегда выполняется в рамках одной транзакции по принципу все или ничего.



Давайте теперь вернемся к примеру с переводом денег и немного усложнили логику изменения данных. Предположим, что мы списываем 100 \$ со счета Алисы, добавляем их на счет Ивана. И вдруг оказывается, что деньги нужно было перевести Марии. В данном случае мы можем применить точки сохранения. Как мы видим на примере, после первой команды создается метка my_savepoint и после второй команды выполняется откат к

этой точке сохранения с помощью команды ROLLBACK TO, после чего выполняется пополнение счета Марии.

ТРАНЗАКЦИИ. ПРИМЕРЫ

```
BEGIN;  
UPDATE accounts SET balance = balance - 100.00  
  WHERE name = 'Alice';  
SAVEPOINT my_savepoint;  
UPDATE accounts SET balance = balance + 100.00  
  WHERE name = 'Ivan';  
-- ошибочное действие... откатить его и использовать счёт Марии  
ROLLBACK TO my_savepoint;  
UPDATE accounts SET balance = balance + 100.00  
  WHERE name = 'Maria';  
COMMIT;
```

Каждая транзакция в СУБД должна соответствовать определенным требованиям, которые описаны в терминах ACID.

ТРАНЗАКЦИИ. ТРЕБОВАНИЯ ACID	
• Фундаментальные свойства обработки транзакций • Гарантируют стабильную и надежную работу системы	
Atomicity – Атомарность	Выполняются либо все операции в рамках транзакции, либо ни одной ("все либо ничего").
Consistency – Согласованность	БД сохраняет согласованность при успешно завершенной транзакции (т.е. фиксации результатов операций).
Isolation – Изолированность	При выполнении транзакции другие параллельные транзакции с теми же данными не оказывают влияние на ее результат. (Полная изолированность – "дорогое" требование, поэтому существуют разные режимы неполной изоляции транзакций)
Durability – Стойкость	Даже при сбоях оборудования изменения, сделанные успешно завершенной транзакцией, сохраняются.

ACID расшифровывается следующим образом: Atomicity или Атомарность. Атомарность гарантирует, что никакая транзакция не будет зафиксирована в системе частично, будут либо выполнены все ее операции, либо не выполнена ни одной. Consistency или согласованность - транзакция, достигающая своего успешного завершения и тем самым фиксирующая свои результаты, сохраняет согласованность базы данных. Isolation или изолированность - во время выполнения транзакции другие параллельные транзакции не должны оказывать влияние на ее результат. Далее мы рассмотрим, каким образом это обеспечивается в GreenPlum. Durability или стойкость - независимо от проблем на аппаратном уровне, к примеру, обесценивание системы или сбои в оборудовании изменения, сделанные успешно завершенной транзакцией, должны остаться сохраненными после возвращения системы в работу.

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



МЕТОДИЧЕСКИЕ УКАЗАНИЯ

для выполнения самостоятельной работы
по дисциплине «**Инжиниринг данных**»
для студентов направления 09.03.04 Программная инженерия
(направленность(профиль) «Разработка и сопровождение
программного обеспечения»)

СОДЕРЖАНИЕ

<u>Введение</u>	173
<u>1. Цель и задачи освоения дисциплины</u>	173
<u>2. Место дисциплины в структуре ОП ВО бакалавриата</u>	173
<u>3. Связь с предшествующими дисциплинами</u>	173
<u>4. Связь с последующими дисциплинами</u>	Ошибка! Закладка не определена.
<u>5. Организационно-методические рекомендации по освоению дисциплины</u>	174
<u>6. Методические указания по выполнению самостоятельной работы студентов</u>	175
<u>7. Содержание самостоятельной работы</u>	175
<u>8. Примерная тематика заданий для самостоятельной работы студентов</u>	176
<u>9. Организация контроля знаний студентов</u>	176
<u>9.1 Формы контроля знаний студентов</u>	177
<u>9.2. Рекомендации по подготовке к экзамену</u>	177
<u>10 Перечень рекомендуемой литературы</u>	177
<u>10.1. Основная литература</u>	177
<u>10.3. Методическая литература</u>	178
<u>10.4. Интернет-ресурсы</u>	178
<u>11. Материально-техническое обеспечение дисциплины</u>	178
<u>12.1. Программное обеспечение:</u>	178
<u>12.2. Материально-техническое обеспечение дисциплины (модуля)</u>	179

Введение

1. Цель и задачи освоения дисциплины

Целью изучения дисциплины «Инжиниринг данных» является формирование у студентов теоретических и практических знаний о методах сбора, извлечения и обработки данных, методах построения систем хранения данных, обучении технологии функционирования платформ распределенной обработки больших наборов данных, подготовка данных к аналитике.

Задачи дисциплины:

1. формирование знаний и навыков работы с данными, представленными в различных форматах;
2. обучение методам подготовки данных для аналитики и систем машинного обучения;
3. формирование знаний в области систем хранения данных.

2. Место дисциплины в структуре ОП ВО бакалавриата

Дисциплина «Инжиниринг данных» относится к дисциплинам части, формируемая участниками образовательных отношений и изучается в 6 семестре. Форма контроля - дифференцированный зачет.

3. Перечень планируемых результатов обучения по дисциплине (модулю), соотнесённых с планируемыми результатами освоения образовательной программы

Код, формулировка компетенции	Код, формулировка индикатора	Планируемые результаты обучения по дисциплине (модулю), характеризующие этапы формирования компетенций, индикаторов
УК-2 Способен определять круг задач в рамках поставленной цели и выбирать оптимальные способы их решения, исходя из действующих правовых норм, имеющихся ресурсов и ограничений	ИД-1 _{УК-2} формулирует цель проекта, определяет совокупность взаимосвязанных задач, обеспечивающих ее достижение и определяет ожидаемые результаты решения задач; ИД-2 _{УК-2} разрабатывает план действий для решения задач проекта, выбирая оптимальный способ их решения, исходя из действующих правовых норм и имеющихся ресурсов и ограничений ИД-3 _{УК-2} обеспечивает выполнение проекта в соответствии с установленными целями, сроками и затратами, исходя из действующих правовых норм, имеющихся ресурсов и ограничений, в том числе с использованием цифровых инструментов.	Формулирует задачу для решения методами дата инженера, проводит ее декомпозицию, выделяет проблемы. Разрабатывает план действия извлечения, обработки и загрузки данных исходя из программно-аппаратных ресурсов хранилища Формирует запрос в соответствии со сроками предоставления данных для аналитики
ОПК-2 Способен понимать принципы работы современных информационных	ИД-1 _{ОПК-2} Понимает основные принципы работы современных информационных технологий и программных средств	Знает принципы работы программных решений и специальных инструментов для представления разнородной информации в

технологий и программных средств, в том числе отечественного производства, и использовать их при решении задач профессиональной деятельности;	ИД-2 _{ОПК-2} Использует современные информационные технологии и программные средства, в том числе отечественного производства, при решении задач профессиональной деятельности.	едином виде, принцип работы аналитических баз данных Владеет современными транзакционными и аналитическими СУБД
ОПК-6 Способен разрабатывать алгоритмы и программы, пригодные для практического использования, применять основы информатики и программирования к проектированию, конструированию и тестированию программных продуктов;	ИД-1ОПК-6 Применяет основы языков программирования и работы с базами данных, операционные системы и оболочки, современные программные среды разработки интеллектуальных систем и технологий. ИД-2ОПК-6 Применяет языки программирования и работы с базами данных, современные программные среды разработки информационных систем и технологий для автоматизации бизнес-процессов, решения прикладных задач различных классов, ведения баз данных и информационных хранилищ.	Использует SQL запросы при решении задач дата инжиниринга, очищает данные: убирает дубли и устраняет ошибки Структурирует данные так, чтобы они стали пригодны для аналитики. Обновляет используемые инструменты и ищет способы улучшения и автоматизации существующих процессов
ОПК-8 Способен осуществлять поиск, хранение, обработку и анализ информации из различных источников и баз данных, представлять ее в требуемом формате с использованием информационных, компьютерных и сетевых технологий.	ИД-1 _{ОПК-8} Демонстрирует теоретические основы поиска, хранения, и анализа информации. ИД-2 _{ОПК-8} Применяет методы поиска, хранения и обработки информации из различных источников и баз данных. ИД-3 _{ОПК-8} Осуществляет поиск, хранение и анализ информации с использованием современных информационных, компьютерных и сетевых технологий.	Знает как собрать данные из различных источников Использует методы для сбора, обработки и загрузки данных из различных источников Использует программные решения и специальные инструменты для представления разнородной информации в едином виде, принцип работы аналитических баз данных

5. Организационно-методические рекомендации по освоению дисциплины

Самостоятельная работа студентов является важнейшим условием формирования научного способа познания. Она проводится накануне каждого лабораторного занятия и включает изучение необходимого для выполнения лабораторной работы теоретического материала, а также подготовку отчета по выполненным на лабораторном занятии заданиям.

Подготовленный материал оформляется в виде отчета.

Самостоятельные занятия (СЗ) являются одной из активных форм обучения.

Самостоятельные занятия по дисциплине «Web-технологии» имеют целью:

- закрепить и углубить знания, полученные студентами на лекциях и в процессе лабораторных занятий;
- привить практические навыки при разработке, тестировании Web-приложений.

Самостоятельные занятия проводятся в специализированных аудиториях, оборудованных СБТ и возможностью пользования Интернет-ресурсами, в библиотеке, а также дома.

Предлагаемые методические рекомендации содержат информацию для студентов, необходимую при подготовке и проведении лабораторных занятий по дисциплине «Web-технологии».

6. Методические указания по выполнению самостоятельной работы студентов

Методические указания включают следующие разделы:

- цель работы;
- задание, которое должно быть выполнено студентом в результате проведения самостоятельной работы;
- варианты индивидуальных заданий;
- основные теоретические положения, необходимые для выполнения задания, они должны быть краткими и содержать ссылки на литературу, в которой эти положения изложены в объеме, достаточном для выполнения самостоятельной работы;
- этапы выполнения задания с указанием конкретных сроков выполнения каждого из этапов и всего задания в целом;
- требования к оформлению графической и текстовой части самостоятельной работы;
- пример выполнения одного из вариантов задания и оформления отчета;
- библиографический список использованных источников

7. Содержание самостоятельной работы

Наименование разделов и тем дисциплины, их краткое содержание; вид самостоятельной работы представлено в таблице 2.

Таблица 2 – Наименование разделов и тем дисциплины, их краткое содержание; вид самостоятельной работы

№	Наименование разделов и тем дисциплины, их краткое содержание; вид самостоятельной работы	Форма контроля
	6 семестр	
1	подготовка к лабораторным занятиям	защита
2	Самостоятельное решение задач по темам: NULL, CASE, функции, подзапросы Агрегация и группировка, join Работа с форматами хранения в DWH	Собеседование
3	Самостоятельное изучение тем: Технологии хранилищ данных Инкрементальная загрузка в хранилищах данных	Опрос

Наименование тем дисциплины, цель, форма контроля, задания, требования к оформлению, перечень литературных источников представлен в таблице 3.

Таблица 3 – Наименование тем дисциплины, цель, форма контроля, задания и требования к оформлению самостоятельной работы

Название темы	Цель	Форма контроля	Задания для СРС
Подготовка к	изучить основные	индивидуальное	Сформулировать

лекциям и лабораторным занятиям	технологии запросов, обработки и хранения данных	собеседование	ответы на контрольные вопросы
Разработка хранилища данных	Получить навыки разработки хранилищ данных		сформулировать и ответить на вопросы для контроля владения компетенциями данного раздела программы

8. Примерная тематика заданий для самостоятельной работы студентов

Примерные варианты заданий по дисциплине «Инжиниринг данных»:

Задание 1.

1. Создание модели хранилища данных с помощью специализированного CASE-средства.
2. Импорт описания метаданных.
3. Настройка внешних источников.
4. Загрузка данных.
5. Хранение данных.
6. Предоставление данных для анализа

Задание 2.

1. Описать бизнес-кейс, включая краткую характеристику деятельности компании; описание проблем бизнеса, требующих разработки аналитической системы на основе хранилища данных; характеристику действующих в компании OLTP-систем.

2. Разработать витрины данных для аналитической системы, включая описание предметной области для витрины данных; формулировку основного вопроса, для ответа на который разрабатывается аналитическая система; список вариантов вопросов, которые могут сформулировать лица, принимающие решения, работая с аналитической системой (не менее 5 вариантов вопросов); примеры структур аналитических отчетов, которые могут быть сгенерированы аналитической системой для лиц, принимающих решения (не менее двух структур); список фактов для таблицы фактов с определенными для каждого факта источником данных; список измерений с ключами и атрибутами с определенными для каждой колонки источником данных; разработать схему витрины данных в CA ERwin Data Modeler, используя нотацию Dimensional Modeling.

2. Шаблон отчета по разработанной модели витрины данных, созданный с помощью ERwin Report Designer, а также сгенерированный на основе этого шаблона отчет по модели.

3. Прямую генерацию каталога витрины данных на MS SQL Server, включая скриншоты фрагмента сгенерированного каталога и соответствующей диаграммы из MS SQL Server Management Studio.

4. Обратную генерацию каталога витрины данных из MS SQL Server в модель физического уровня в CA ERwin Data Modeler, включая скриншот модели данных.

5. Полное сравнение (Complete Compare) каталога витрины данных из MS SQL Server и модели физического уровня в CA ERwin Data Modeler (после внесения изменений в одну из сравниваемых сторон), включая скриншот сгенерированной ERwin таблицы результатов сравнения. Отчет оформить как документ Word. В названии документа указать номер группы и фамилию. Документ сохранить в личной папке. К отчету приложить файлы всех разработанных моделей, шаблонов, отчетов

9. Организация контроля знаний студентов

9.1 Формы контроля знаний студентов

Контроль и оценка знаний, умений и навыков студентов осуществляется на лабораторных занятиях, консультациях, получении зачета с оценкой. В ходе контроля знаний преподаватель оценивает понимание студентом содержания дисциплины «Web-технологии», его способность разрабатывать и тестировать веб-приложение, применять веб-технологии в разработке.

Контроль знаний студентов может осуществляться в следующих формах:

- текущий контроль знаний;
- итоговый контроль знаний.

Текущий контроль знаний студентов имеет целью: дать оценку работы каждого студента по усвоению им учебного материала, выявить недостатки в его подготовке и оказать практическую помощь в их устранении;

Основными формами текущего контроля знаний студентов являются:

- устный контрольный опрос;
- защита лабораторной работы.

Устный контрольный опрос студентов проводится на лекциях (и лабораторных занятиях). По его результатам преподаватель оценивает качество подготовки студента к занятию.

На лабораторных занятиях знания и практические навыки студентов оцениваются по 5-балльной системе.

9.2. Рекомендации по подготовке к зачету

Подготовка к зачету начинается с начала изучения дисциплины.

Необходимо посещать все лекции и лабораторные занятия.

Зачет проводится в 4 семестре после защиты всех лабораторных работ в объеме учебной программы и выполненной самостоятельной работы.

10 Перечень рекомендуемой литературы

1.1. Основная литература

1. Тарасов, С. В. СУБД для программиста. Базы данных изнутри / С. В. Тарасов. - СУБД для программиста. Базы данных изнутри, 2022-05-25. - Электрон. дан. (1 файл). - Москва : СОЛОН-Пресс, 2018. - 320 с. - электронный. - Книга находится в премиум-версии ЭБС IPR BOOKS. - ISBN 978-2-7466-7383-0, экземпляров неограничено

2. Бродовская, Е. В. Большие данные в исследовании политических процессов : учебное пособие / Е. В. Бродовская, А. Ю. Домбровская. - Большие данные в исследовании политических процессов, Весь срок охраны авторского права. - Электрон. дан. (1 файл). - Москва : Московский педагогический государственный университет, 2018. - 88 с. - электронный. - Книга находится в премиум-версии ЭБС IPR BOOKS. - ISBN 978-5-4263-0712-4, экземпляров неограничено

3. Воронов, В. И. Data Mining - технологии обработки больших данных Электронный ресурс : Учебное пособие / В. И. Воронов, Л. И. Воронова, В. А. Усачев. - Data Mining - технологии обработки больших данных, 2024-02-26. - Москва : Московский технический университет связи и информатики, 2018. - 47 с. - Книга находится в премиум-версии ЭБС IPR BOOKS. - ISBN 2227-8397, экземпляров неограничено

Дополнительная литература:

1. Самойлова, Е. М. Интегрированные системы проектирования и управления. Цифровое управление инженерными данными и жизненным циклом изделия : учебное пособие / Е. М. Самойлова. - Интегрированные системы проектирования и управления. Цифровое управление инженерными данными и жизненным циклом изделия, Весь срок охраны авторского права. - Электрон. дан. (1 файл). - Москва : Ай Пи Ар Медиа, 2020. - 283 с. - электронный. - Книга находится в премиум-версии ЭБС IPR BOOKS. - ISBN 978-5-4497-0640-9, экземпляров неограничено
2. Мочалов, А. И. Базы и банки данных на транспорте Электронный ресурс / Мочалов А. И. : учебное пособие. - Санкт-Петербург : СПбГУ ГА, 2020. - 147 с. - ISBN 978-5-907354-00-5, экземпляров неограничено
3. Наместников, А. М. Базы данных. Практический курс. В 2 частях. Ч.1. Объектно-реляционные базы данных на примере PostgreSQL 9.5 : учебное пособие / А. М. Наместников, А. А. Филиппов. - Базы данных. Практический курс. В 2 частях. Ч.1. Объектно-реляционные базы данных на примере PostgreSQL 9.5, 2025-10-27. - Электрон. дан. (1 файл). - Ульяновск : Ульяновский государственный технический университет, 2017. - 113 с. - электронный. - Книга находится в премиум-версии ЭБС IPR BOOKS. - ISBN 978-5-9795-1743-8, экземпляров неограничено

10.3. Методическая литература

10.4. Интернет-ресурсы

1	https://biblioclub.ru/
2	https://www.elibrary.ru/
3	Sql-ex.ru
4	Sql-tutorial.ru
5	Postgresql.org
6	Docs.oracle.com

11. Материально-техническое обеспечение дисциплины

- индивидуальное взаимодействие со студентами по электронной почте для предварительного ознакомления с их разработками при подготовке к аудиторным занятиям;
- использование на лекциях и лабораторных занятиях мультимедийного оборудования для демонстрации листингов программного кода, скриншотов выполненных запросов, работы программ и пр.;
- включение в лабораторные работы индивидуального поиска, систематизации и анализа информации через Интернет;
- авторские презентации к лекциям;
- лекционная аудитория должна быть оборудована мультимедиа проектором;
- Лабораторные работы рассчитаны на компьютеры, имеющие доступ к Internet.

12.1. Программное обеспечение:

1	PostgreSQL
2	DBEaver
3	ERwin Data Modeler
4	Master Data Management

12.2. Материально-техническое обеспечение дисциплины (модуля)

Мультимедиа аудитория: проектор, акустическая система, аудио-усилитель, ноутбук, Internet.

Компьютерный класс. Высокопроизводительная ЭВМ, локальная вычислительная сеть.