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

Ставрополь, 2026

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

Цель работы: познакомиться с особенностями применения SQL в инжиниринге данных

Теоретическое обоснование

Системные требования

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

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

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

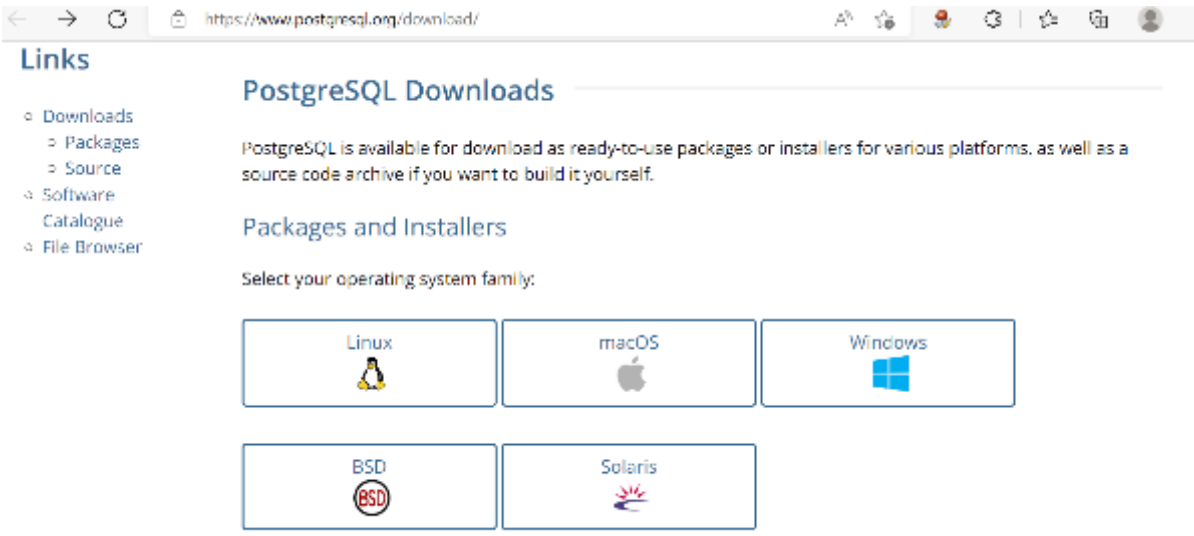
Кроме этого есть и другие требования:

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

Ссылка для скачивания

PostgreSQL: Downloads

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



На следующей странице выберите ссылку загрузки для инсталляции

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

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

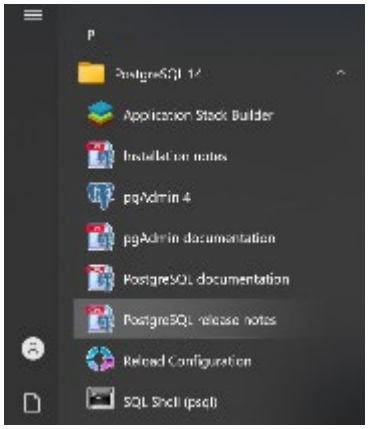
- Главный серверный процесс, управляющий файлами баз данных, принимающий подключения клиентских приложений и выполняющий различные запросы клиентов к базам данных. Эта программа сервера БД называется postgres.
- Клиентское приложение пользователя, желающее выполнять операции в базе данных. Клиентские приложения могут быть очень разнообразными: это может быть текстовая утилита, графическое приложение, веб-сервер, использующий базу данных для отображения веб-страниц, или специализированный инструмент для обслуживания БД. Некоторые клиентские приложения поставляются в составе дистрибутива PostgreSQL, однако большинство создают сторонние разработчики.

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

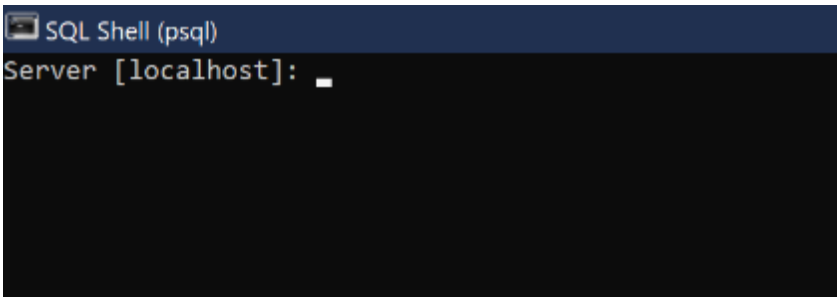
Сервер PostgreSQL может обслуживать одновременно несколько подключений клиентов. Для этого он запускает («порождает») отдельный процесс для каждого подключения. Можно сказать, что клиент и серверный процесс общаются, не затрагивая главный процесс postgres. Таким образом, главный серверный процесс всегда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

Методика выполнения

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

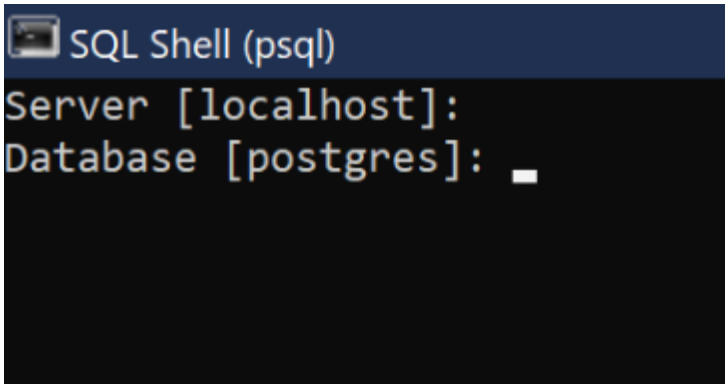


После запуска SQL Shell нас встречает надпись Server [local|host]



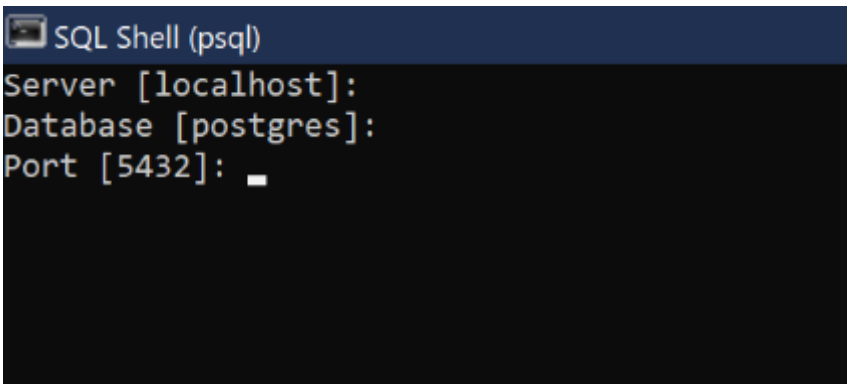
В квадратных скобках введено стандартное значение. Вводим Enter.

Появляется надпись Database [postgres]:



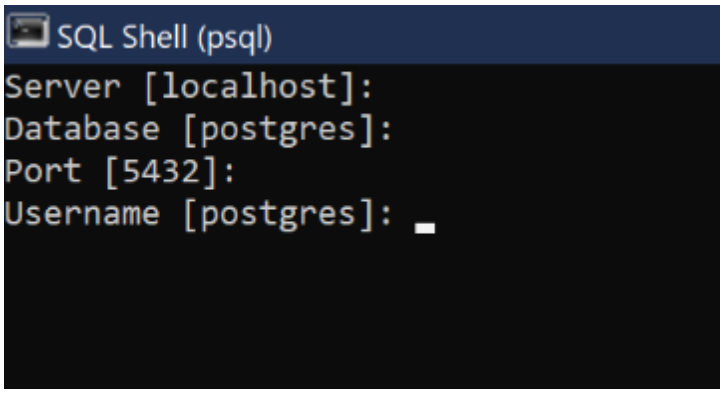
Нажимаем Enter.

Следующая строка Porf [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.



Нажимает Enter.

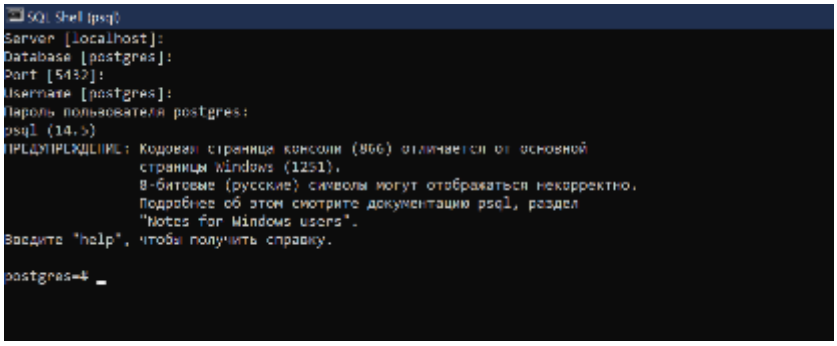
Появляется строка ввода логина и потом строка пароля. Пароль вводится тот, который был введен при установке.



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.



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 подробные сообщения 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
                                   Список баз данных
   Имя    | Владелец | Кодировка |  LC_COLLATE   |   LC_CTYPE    |   Права доступа
----------+----------+-----------+---------------+---------------+-----------------------
 postgres | postgres | UTF8      | Russian_Russia.1251 | Russian_Russia.1251 |
 template0 | postgres | UTF8      | Russian_Russia.1251 | Russian_Russia.1251 | =c/postgres          +
          |          |           |               |               | 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
                                   Список баз данных
   Имя    | Владелец | Кодировка |  LC_COLLATE   |   LC_CTYPE    |   Права доступа
----------+----------+-----------+---------------+---------------+-----------------------
 postgres | postgres | UTF8      | Russian_Russia.1251 | Russian_Russia.1251 |
 template0 | postgres | UTF8      | Russian_Russia.1251 | Russian_Russia.1251 | =c/postgres          +
          |          |           |               |               | 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
                                   Список баз данных
   Имя    | Владелец | Кодировка |  LC_COLLATE   |   LC_CTYPE    |   Права доступа
----------+----------+-----------+---------------+---------------+-----------------------
 postgres | postgres | UTF8      | Russian_Russia.1251 | Russian_Russia.1251 |
 starter  | postgres | UTF8      | Russian_Russia.1251 | Russian_Russia.1251 |
 template0 | postgres | UTF8      | Russian_Russia.1251 | Russian_Russia.1251 | =c/postgres          +
          |          |           |               |               | 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: ***C:\Program Files\PostgreSQL\13\scripts\runpsql.bat*** 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: ***C:\Program Files\PostgreSQL\13\scripts*** 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: DROP DATABASE starter;
\c	Команда **\c [имя_бд]** подключит вас к базе данных с названием **[имя_бд]**. Пример: \c starter

Лабораторная работа № 2. Начало работы с pgAdmin4

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

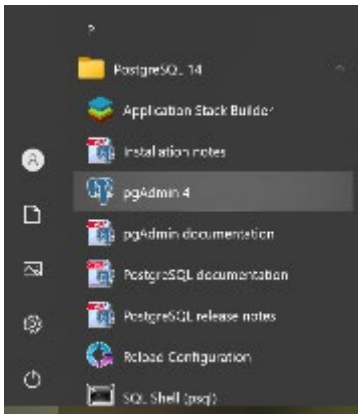
Теоретическое обоснование

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.

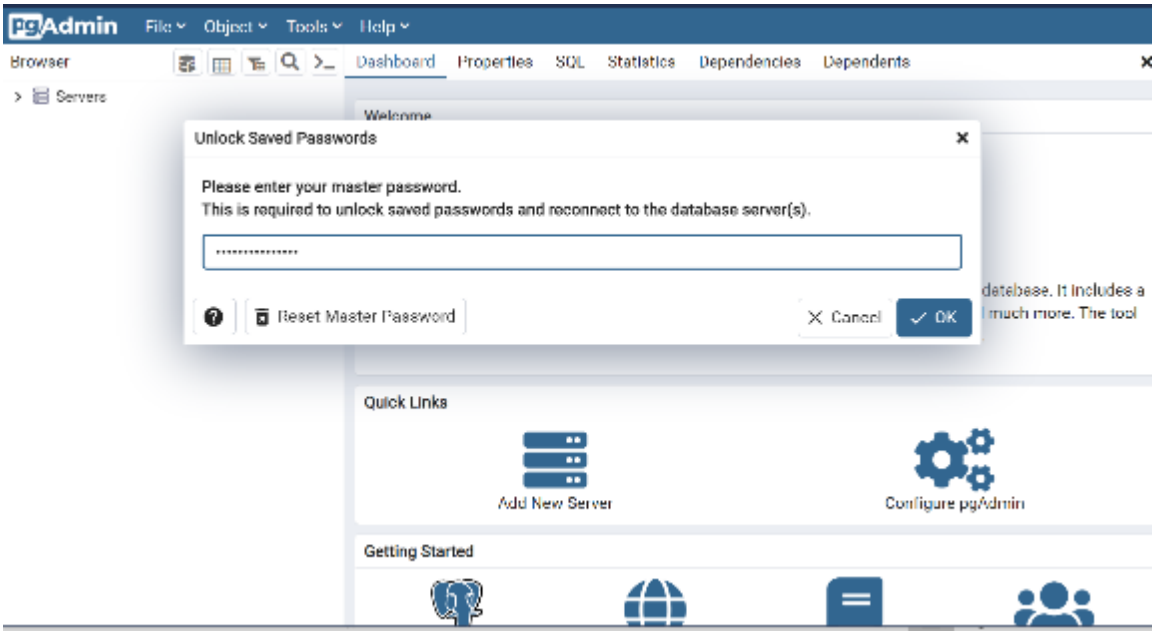
Этот инструмент был установлен в комплекте инструментов postgres. Давайте посмотрим на то, как мы начать работать с ним.

Методика выполнения

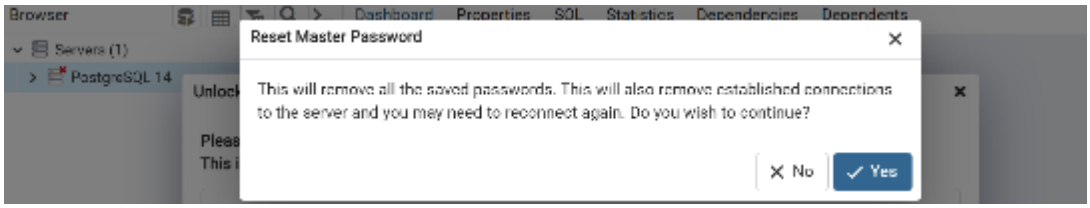
1. Запустить pgAdmin4 можно через меню пуск. Он находится на вкладке Пуск-PostgreSQL14-pgAdmin4.



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

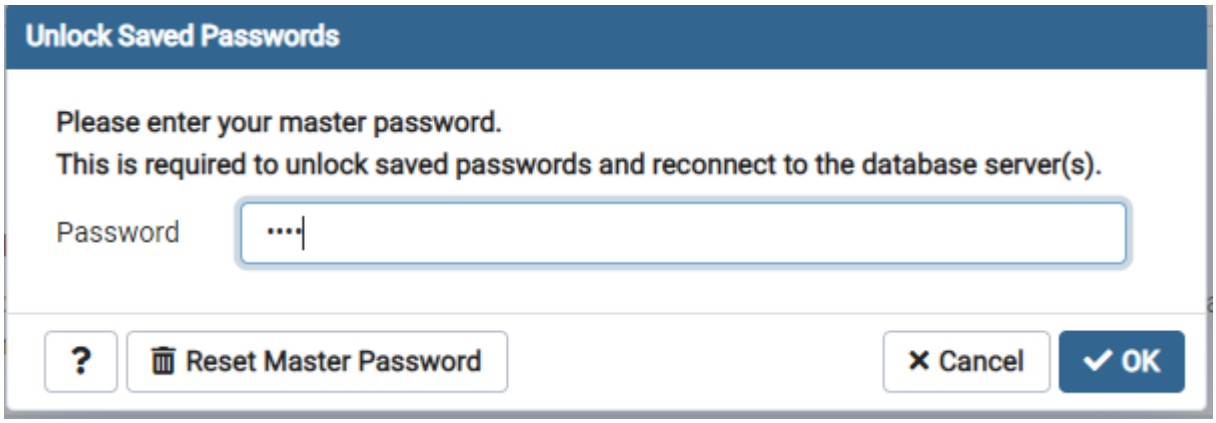


3. Если вы забыли пароль, то его можно поменять, нажав на кнопку Reset Master Password. В окне Reset Master Password вводим новый пароль и применяем изменения.

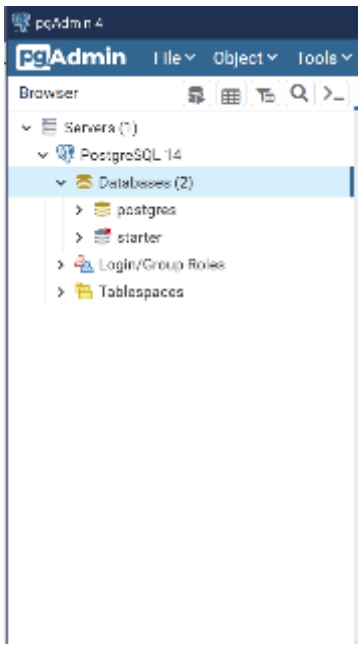


4. После входа, вас встретит пустой экран, без какой-либо информации. Мы не видим её так как не подключены ни к одному серверу. Нажмите на вкладку Servers в левой части экрана, а затем сделайте двойной щелчок по вкладке PostgreSQL14.

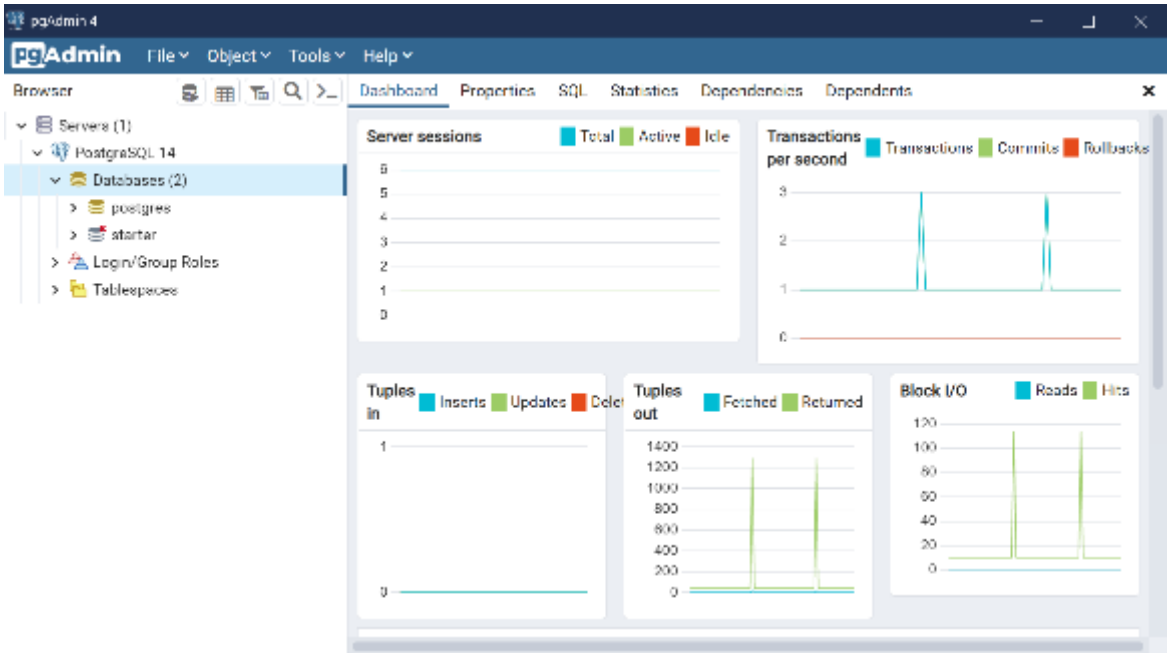
5. Нам предлагают повторно ввести пароль. Сюда мы тоже вводим то, что придумали при установке : “***”.



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 CREAT 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".
starter=# CREATE TABLE student(
starter(# id INT, name VARCHAR(50), surname VARCHAR(50), gender VARCHAR(50),
starter(# mobile VARCHAR(50), birthday DATE);
CREATE TABLE
starter=#
```

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

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

Введите эту команду

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

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


starter=#
```

Появится таблица со всеми нами созданными элементами таблицы и последовательностью.

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

Пример :

\d student;

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

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


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


starter-#
```

Для удаления таблицы используется команда

DROP TABLE

Синтаксис этой команды следующий:

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

Пример :

DROP TABLE student;

Выполните эту команду и удалите таблицу student.

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

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

Пример :

SELECT * FROM student;

Знак * означает выбор всех столбцов

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

starter-# _
```

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

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

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

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

Пример :

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

Обратите внимание на формат ввода даты дня рождения.

```
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', '2008 04 12');
INSERT 0 1
starter=# SELECT * FROM student;
 id |  name   | surname | gender |    mobile    |  birthday
----+---------+---------+--------+--------------+------------
    | Michael | Scott   | male   | +79884561237 | 2008-04-12
(1 строка)


starter=#
```

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

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

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

Для загрузки файла в базу в консоле введите команду \i

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

Пример :

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

Обратите внимание на формат ввода пути к файлу.

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


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

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   | 798-150-3796 | 2020-12-01
  4 | Kerna        | Hosby           | Genderqueer  | 399-757-2358 | 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    | Bahnke          | Bigender     | 558-165-8190 | 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       | Dennes          | Male         | 586-183-6597 | 2021-08-19
 14 | Jacqueline   | Micah           | Female       | 765-924-7203 | 2021-05-03
 15 | Othelia      | Kernley         | Genderfluid  | 258-358-3598 | 2021-03-28
 16 | Hedwiga      | Leile           | Bigender     | 742-197-7722 | 2020-12-15
 17 | Miguelita    | Dawid           | Female       | 588 295 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          | Kerrott         | Genderfluid  | 228-178-8717 | 2021-04-12
 21 | Aurelia      | Dansuna         | Polygender   | 378-864-9864 | 2021-05-21
 22 | Delcine      | Pendered        | Female       | 988-699-8718 | 2021-04-20
 23 | Lewiss       | Yanyshev        | Female       | 356-598-3839 | 2021-05-18
 24 | Collen       | Strang          | Agender      | 777-628-6934 | 2021-07-14
 25 | Xymenes      | Oliferd         | Agender      | 742-618-8426 | 2021-03-17
 26 | Orelia       | Baurert         | Non-binary   | 256-837-1008 | 2021-08-03
 27 | Victoir      | Saile           | Agender      | 965-743-7953 | 2020-09-07
 28 | Mort         | Nerchinov       | Female       | 822 216 7261 | 2021 03 28
 29 | Rosabelle    | Fenich          | Agender      | 829 323 5779 | 2020 10 31
 30 | Analine      | Lardinar        | 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)

```
startar=# 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   | 2028-12-15
 17 | Miguelita    | Dewid         | Female       | 580-295-1834   | 2021-05-04
 18 | Retha        | Tidgewell     | Genderfluid  | 447 135 5218   | 2028-10-16
 19 | Balduin      | Cordall       | Polygender   | 314-694-9253   | 2021-05-22
 20 | Lia          | Kermott       | Genderfluid  | 220-178-8717   | 2021-04-12
 21 | Aurilia      | Densumba      | Polygender   | 378-864-9864   | 2021-05-21
 22 | Delcine      | Pendered      | Female       | 988-699-8710   | 2021-04-20
 23 | Lewiss       | Vanyshev      | Female       | 356-598-3839   | 2021-05-18
 24 | Collen       | Strang        | Agender      | 777-620-6934   | 2021-07-14
 25 | Xymenes      | Olifard       | Agender      | 742-610-8426   | 2021-03-17
 26 | Orelia       | Daumert       | Non-binary   | 256-037-1800   | 2021-08-03
 27 | Victoir      | Seile         | Agender      | 965-743-7563   | 2028-09-07
 28 | Mort         | Nemchinov     | Female       | 822-216-7261   | 2021-03-28
 29 | Rosabella    | Fenich        | Agender      | 829-323-5779   | 2028-10-31
 30 | Ameline      | Lardiner      | Polygender   | 598 751 2283   | 2021 05 19
 31 | Enlyn        | Ciobutaru     | Male         | 801 774 4694   | 2021 08 13
 32 | Wally        | Madge         | Agender      | 740-188-8291   | 2021-02-19
 33 | Cayla        | Spata         | Agender      | 148-609-6202   | 2021-08-03
 34 | Dacey        | Lovel         | Polygender   | 164-975-2195   | 2021-03-28
 35 | Liane        | Lomansey      | Bigender     | 132-737-9648   | 2028-09-02
 36 | Janesy       | Abbots        | Female       | 573-635-9892   | 2021-06-07
 37 | Cy           | Wagnerin      | Female       | 988-387-2252   | 2021-03-16
 38 | Pet          | Bloxham       | Genderqueer  | 103-828-3158   | 2021-07-23
 39 | Charlton     | Janous        | Genderqueer  | 248 974 1770   | 2021-02-28
 40 | Morti        | Fenlon        | Agender      | 923-167-7875   | 2021-02-24
 41 | Othelia      | Deaton        | Genderfluid  | 268-123-4960   | 2028-10-08
```

SQL Shell (psql)

```
 29 | Rosabella    | Fenich        | Agender      | 829-323-5779   | 2028-10-31
 30 | Ameline      | Lardiner      | Polygender   | 598-751-2283   | 2021-05-19
 31 | Enlyn        | Ciobutaru     | Male         | 801-774-4694   | 2021-08-13
 32 | Wally        | Madge         | Agender      | 740-188-8291   | 2021-02-19
 33 | Cayla        | Spata         | Agender      | 148-609-6202   | 2021-08-03
 34 | Dacey        | Lovel         | Polygender   | 164-975-2195   | 2021-03-28
 35 | Liane        | Lomansey      | Bigender     | 132-737-9648   | 2028-09-02
 36 | Janesy       | Abbots        | Female       | 573-635-9892   | 2021-06-07
 37 | Cy           | Wagnerin      | Female       | 988-387-2252   | 2021-03-16
 38 | Pet          | Bloxham       | Genderqueer  | 103-828-3158   | 2021-07-23
 39 | Charlton     | Janous        | Genderqueer  | 248 974 1770   | 2021 02 28
 40 | Morti        | Fenlon        | Agender      | 923 167 7875   | 2021 02 24
 41 | Othelia      | Deaton        | Genderfluid  | 268-123-4960   | 2028-10-08
 42 | Rodrigo      | Dumbrall      | Bigender     | 201-571-6159   | 2021-05-21
 43 | Jillane      | Lampard       | Bigender     | 544-231-9712   | 2021-01-07
 44 | Ansele       | Escalante     | Polygender   | 280-109-6404   | 2021-02-05
 45 | Myrwyn       | Prewle        | Non-binary   | 737-653-2605   | 2021-03-16
 46 | Dorie        | Prebble       | Agender      | 554-970-2127   | 2021-07-25
 47 | Ginny        | Dramney       | Polygender   | 134-048-7360   | 2021-06-13
 48 | Ulberto      | Nelmes        | Female       | 277-658-0811   | 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        | Kleinhout     | Male         | 123-393-7442   | 2028-09-10
 52 | Cullen       | Wittering     | Polygender   | 705 230 3984   | 2021 08 04
 53 | Hirsch       | Hulance       | Polygender   | 565 506 8634   | 2021 03 26
 54 | Bartholomeus | Klimrek       | 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  | 299-693-8554   | 2021-02-24
 58 | Delaney      | MacCague      | Agender      | 700-988-8161   | 2021-07-18
 59 | Pren         | Posen         | Polygender   | 451-905-2126   | 2021-08-29
 60 | Karoly       | Murny         | Non-binary   | 960-785-3984   | 2021-06-25
(50 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     | Cogin      | Female      | 372-739-5886 | 2020-12-26
 103 | Felike    | Catanheira | Genderqueer | 478-812-6605 | 2020-10-27
 104 | Rosy      | Challen    | Bigender    | 877-212-6652 | 2020-10-24
 105 | Gabriel   | Sokell     | Genderqueer | 482-147-0643 | 2021-01-17
 106 | Lita      | MacLise    | Non-binary  | 869-737-3336 | 2020-10-07
 107 | Ab        | Treadwell  | Genderqueer | 183-613-1424 | 2020-11-21
 108 | Livvyy    | Troup      | Male        | 520-592-5512 | 2020-11-21
 109 | Almeta    | Haistwell  | Agender     | 334-967-9255 | 2021-06-28
 110 | Giulietta | Agett      | Agender     | 782-465-0262 | 2021-01-20
(10 строк)


starter=#
```

Как мы видим, полученный результат идентичен результату команды LIMIT.

Не однократно мы видели результаты команды SELECT со «*»в виде копии таблицы, которая хранится на сервере базы данных. Но что, ели 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
---------------+-----------------
 Cormey        | Numberstone
 Rina          | Leaves
 Leyla         | Culpen
 Karna         | Hosby
 Berkie        | Yule
 Avrit         | Dahl
 Andrai        | Cockney
 Gates         | Doohie
 Ulrich        | Bridge
 Georgiana     | Behnke
 Lynn          | Cokly
 Chicky        | Schonfelder
 Rodina        | Dennes
 Jacqueline    | Micah
 Othelia       | Kornley
 Hedwiga       | Leile
 Migualita     | Dawid
 Ratha         | Tidgewell
 Balduin       | Cordall
 Lia           | Kermott
 Aurilia       | Densunbe
 Delcine       | Pendered
 Lewiss        | Yanyshev
 Collen        | Strang
 Xynenes       | Olifard
 Orelia        | Bouwert
 Victoir       | Seile
 Mort          | Nemchinov
 Rosabella     | Fenich
```

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

Сделаем еще один запрос по полю birthday

SELECT birthday FROM student_auto;

```
SQL Shell (psql)
startar=# 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)
startar=# SELECT * FROM student_auto ORDER BY birthday DESC;
 id  |     name     |     surname     |    gender    |     mobile     |  birthday
-----+--------------+-----------------+--------------+----------------+------------
 418 | Blondie      | Barltrop        | Genderfluid  | 369-255-9129   | 2021-08-29
 418 | Blondie      | Barltrop        | Genderfluid  | 369-255-9129   | 2021-08-29
 418 | Blondie      | Barltrop        | Genderfluid  | 369-255-9129   | 2021-08-29
 399 | Seka         | Harly           | Male         | 853-826-0288   | 2021-08-28
 399 | Seka         | Harly           | Male         | 853-826-0288   | 2021-08-28
 480 | Elihu        | Stent           | Female       | 458-777-2977   | 2021-08-28
 480 | Elihu        | Stent           | Female       | 458-777-2977   | 2021-08-28
 399 | Seka         | Harly           | Male         | 853-826-0288   | 2021-08-28
 480 | Elihu        | Stent           | Female       | 458-777-2977   | 2021-08-28
 296 | Morse        | Albery          | Female       | 621-496-6036   | 2021-08-27
 296 | Morse        | Albery          | Female       | 621-496-6036   | 2021-08-27
  74 | Llywellyn    | 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  | 304-653-9778   | 2021-08-25
 313 | Andras       | Cutill          | Genderqueer  | 304-653-9778   | 2021-08-25
 313 | Andras       | Cutill          | Genderqueer  | 304-653-9778   | 2021-08-25
 412 | Shea         | Lapworth        | Male         | 914-307-1809   | 2021-08-24
 177 | Chad         | 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        | Willis          | Bigender     | 957-734-3893   | 2021-08-20
```

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

SELECT name FROM student_auto ORDER BY birthday ASC;

```
SQL Shell (psql)
startar=# SELECT name FROM student_auto ORDER BY birthday ASC;
     name
--------------
 Babette
 Giraldo
 Babette
 Giraldo
 Babette
 Giraldo
 Gypsy
 Gypsy
 Gypsy
 Liana
 Liana
 Bethany
 Bethany
 Liana
 Bethany
 Alvina
 Trevor
 Trevor
 Trevor
 Alvina
 Alvina
 Lynnell
 Victoir
 Lynnell
 Lautitia
 Victoir
 Victoir
 Lautitia
 Lynnell
 Lautitia
 Trezer
```

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

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

SELECT DISTINCT birthday FROM student_auto ORDER BY birthday ASC;

```
SQL Shell (psql)
                                                    ^
startar=# 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  | David           | Female | 588-295-1034   | 2021-06-04
  22 | Delcine    | Pendered        | Female | 980-699-8718   | 2021-04-28
  23 | Lewiss     | Yanyshev        | Female | 356-590-1839   | 2021-05-10
  28 | Mort       | Nemchinov       | Female | 822-216-7261   | 2021-03-20
  36 | Janesy     | Abbots          | Female | 573-635-9092   | 2021-06-07
  37 | Cy         | Wagnerin        | Female | 980-387-2252   | 2021-03-16
  48 | Ulberto    | Nelves          | Female | 277-658-8011   | 2021-02-28
  62 | Loincy     | Swaffield       | Female | 555 558 4213   | 2020 10 28
  70 | Alisander  | Burdin          | Female | 518 741 9227   | 2021 07 14
  81 | Fred       | Pinre           | Female | 758-803-0513   | 2021-03-24
  87 | Luise      | Coleshill       | Female | 508-191-4154   | 2020-11-20
  93 | Katrinka   | Casay           | Female | 584-649-8258   | 2021-04-22
  98 | Beryle     | Pedrick         | Female | 528-348-0038   | 2020-09-14
 102 | Canny      | 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 | 979-954-7803   | 2020-09-20
 158 | Viloria    | Hael            | Female | 633-657-1558   | 2021-04-03
 160 | Chico      | Seviour         | Female | 873-409-9343   | 2021-02-08
 169 | Dill       | Jendrys         | Female | 922-351-0984   | 2021-01-15
 173 | Bendicty   | Mizen           | Female | 404-849-6163   | 2021-04-01
 179 | Janss      | Mazillius       | Female | 182-141-9569   | 2021-01-31
 183 | Justinn    | Rycroft         | Female | 218 252 5928   | 2021 04 24
 188 | Kinny      | Le Gallo        | Female | 478 596 8113   | 2020 09 28
 193 | Solaich    | English         | Female | 305-951-9427   | 2021-06-28
```

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

Для составления логических выражений используются, следующие знаки:

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

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

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
```

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             | cdutch@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 | Kerwin        | Lannas            | klannas6@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               | crea@godaddy.com                    | Male   | China
 12 | Ladonna       | Comfort           | lcomfortb@plala.or.jp               | Male   | Uganda
 13 | Ranique       | Kulver            | rkulverc@weebly.com                 | Male   | Brazil
 14 | Mallory       | Warhan            | mwarhand@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 | Brennon       | Cottisford        | bcottisfordh@yellowpages.com        | Female | China
 19 | Gale          | Di Nisco          | gdinisscoi@umich.edu                | Male   | United Kingdom
 20 | Salomo        | Dyka              | sdykaj@java.com                     | Male   | Greece
 21 | Yasreen       | Tanser            | ytanserk@shareasale.com             | Male   | Finland
 22 | Rodie         | Garrity           | rgarritylb@pcworld.com              | Male   | Greece
 23 | Maggy         | Wiggett           | mwiggettm@wix.com                   | Male   | Peru
 24 | Jody          | Rollin            | jrollinn@xblog.jp                   | Female | Brazil
 25 | Boris         | Maisey            | bmaiseyo@wix.com                    | Male   | China
```

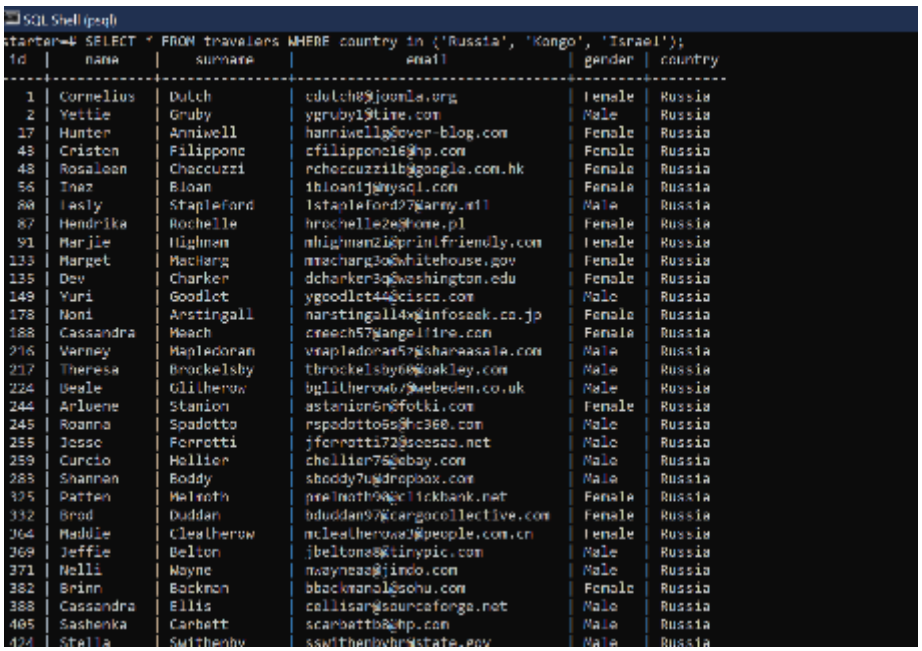
Как результат, мы видим точную копию таблицы, которая хранится на сервере.

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 хоим 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.



```
SQL Shell(psql)
starter=# SELECT * FROM travelers WHERE country in ('Russia', 'Kongo', 'Israel');
 id |    name    |    surname    |              email              | gender | country
----+------------+---------------+---------------------------------+--------+---------
  1 | Cornelius  | Dutch         | cdutch0@jounle.org              | Female | Russia
  2 | Yettie     | Gruby         | ygruby1@tire.com                | Male   | Russia
 17 | Hunter     | Anniwell      | hanniwell@over-blog.com         | Female | Russia
 43 | Cristen    | Filippone     | cfilippone16@hp.com             | Female | Russia
 48 | Rosaleen   | Checcuzzi     | rcheccuzzi1b@google.com.hk      | Female | Russia
 56 | Inez       | Bloan         | ibloan1j@mysql.com              | Female | Russia
 80 | Tasly      | Stapleford    | lstapleford27@army.mil          | Male   | Russia
 87 | Hendrika   | Rochelle      | hrochelle2e@home.pl             | Female | Russia
 91 | Marjie     | Highmen       | mhighmen2i@printfriendly.com    | Female | Russia
133 | Marget     | Machary       | mmachary3o@whitehouse.gov       | Female | Russia
135 | Dev        | Charker       | dcharker3q@washington.edu       | Female | Russia
149 | Yuri       | Goodlet       | ygoodlet4g@cisco.com            | Male   | Russia
178 | Noni       | Aestingall    | naestingall4x@infoseek.co.jp    | Female | Russia
188 | Cassandra  | Meach         | cmeach57@angelfire.com          | Female | Russia
216 | Varney     | Mapledoran    | vmapledoran5r@shareasale.com    | Male   | Russia
217 | Theresa    | Brockelsby    | tbrockelsby6@oakley.com         | Male   | Russia
224 | Dewle      | Gilthorw      | bgilthorw6r@webeden.co.uk       | Male   | Russia
244 | Arluene    | Stanion       | astanion6n@fotki.com            | Female | Russia
245 | Roanna     | Spadotto      | rspadotto6s@hc360.com           | Male   | Russia
255 | Jesse      | Ferrotti      | jferrotti7j@sesso.net           | Male   | Russia
259 | Curcio     | Hellier       | chellier7o@ebay.com             | Male   | Russia
283 | Shannan    | Boddy         | sboddy7u@dropbox.com            | Male   | Russia
325 | Patten     | Melnoth       | pmelnoth9@clickbank.net         | Female | Russia
332 | Brod       | Duddan        | bduddan97@cargocollective.com   | Female | Russia
364 | Maddie     | Cleatherow    | mcleatherow3j@people.com.cn     | Female | Russia
369 | Jeffie     | Belton        | jbelton8@tinypic.com            | Male   | Russia
371 | Nelli      | Wayne         | nwayne8j@jimdo.com              | Male   | Russia
382 | Brinn      | Backman       | bbackman8l@sohu.com             | Female | Russia
388 | Cassandra  | Ellis         | cellisar@sourceforge.net        | Male   | Russia
405 | Sashenka   | Carbett       | scarbett8@hp.com                | Male   | Russia
424 | Stella     | Smithenbyhr   | ssmithenbyhr@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 | Delli      | Chapelhow   | dchapelhow5@godaddy.com             | Female | China
 10 | Jedidiah   | Cayser      | jcayser9@wordpress.com              | Female | China
 11 | Clary      | Rea         | crea@godaddy.com                    | Male   | China
 13 | Ronique    | 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      | bnajarao@wix.com                    | Male   | China
 32 | Trudie     | Westwater   | twestwaterv@rambler.ru              | Male   | China
 42 | Margaretha | Deuckham    | mdeuckham1e@mapquest.com            | Female | Colombia
 45 | Marc       | Sant        | msant1h@devhub.com                  | Male   | China
 52 | Lemmond    | Shefton     | lshefton1f@1688.com                 | Female | China
 58 | Brigit     | Swatten     | bswatten1k@unicef.org               | Male   | Brazil
 60 | Hodric     | Chipps      | hchipps1m@nydailynews.com           | Male   | China
 62 | Court      | Klambt      | cklambt1p@ifeng.com                 | Female | China
 66 | Sherwin    | Ivakin      | sivakin1t@blinklist.com             | Male   | Canada
 69 | Queenie    | Brodest     | qbrodest1w@roncast.net              | Female | Czech Republic
 71 | Horatius   | Farghar     | hfarghar1y@vimeo.com                | Male   | Colombia
 78 | Toni       | Exall       | texall25@instagram.com              | Male   | China
 79 | Sheffy     | Radclyffe   | sradclyffe26@usa.gov                | Male   | China
 81 | Heather    | Colbourne   | hcolbourne2@hud.gov                 | Male   | China
 89 | Ardelle    | McGurgan    | amcgurgan2g@marriott.com            | Female | China
 90 | Ellie      | Broske      | ebroske2h@imgur.com                 | Female | Colombia
 95 | Saree      | Torrett     | storrett2m@google.co.jp             | Female | China
 96 | Lorianna   | Cicccitti   | lciccciitti2n@prinearticles.com     | Female | China
 98 | Giraud     | Kiddle      | gkiddle2p@zinbio.com                | Female | China
101 | Sammy      | Devanish    | sdevanish2q@nbcnews.com             | Male   | Brazil
105 | Florie     | Puckinghorne | fpuckinghorne2u@blog.com           | Female | Brazil
111 | Clovis     | Bilan       | cbilan32@google.ru                  | Male   | China
```

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

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

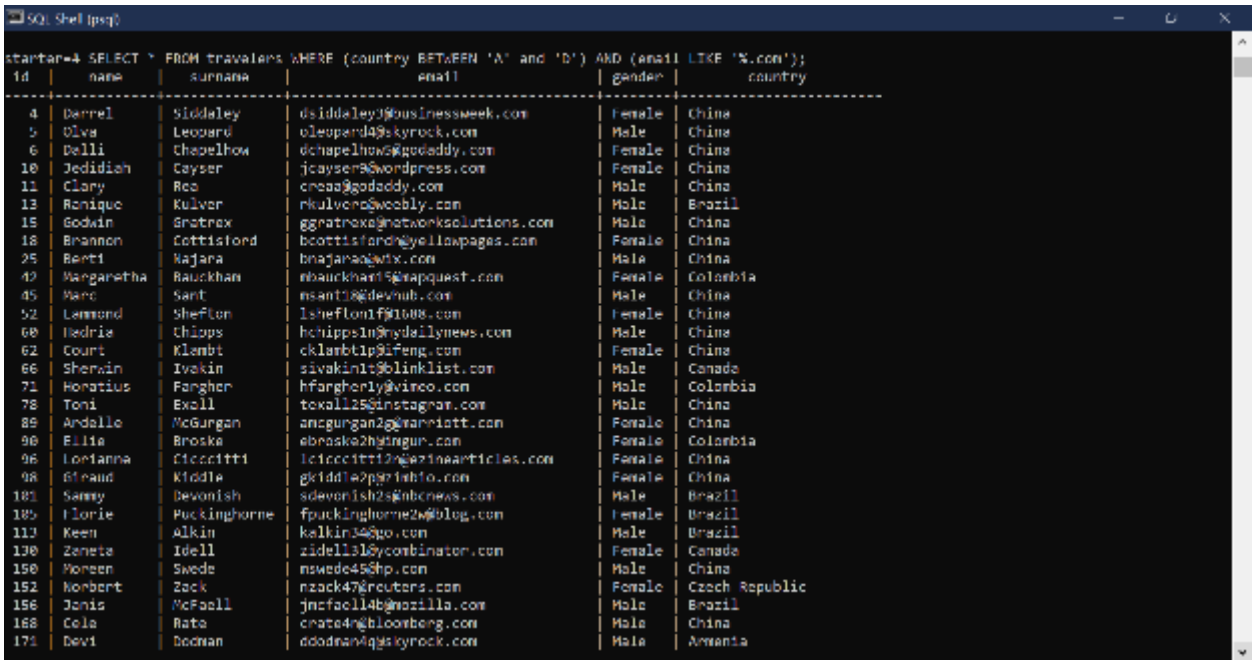
Используя команду LIKE мы оставим только те строки, в которых почта имеет символы внутри одинарных кавычек. Знак % - это специальный символ. Он заменяет весь текст, который следует до искомой нами комбинации.

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 Risco    | gdiriscoi@ucl.edu       | Male   | United Kingdom
 26 | Sebastien  | Vines       | svines@columbia.edu     | Male   | Philippines
 39 | Sargent    | Lipman      | slipman12@mit.edu       | Female | Uruguay
 46 | Rosie      | Arnold      | rarnold19@msu.edu       | Male   | Egypt
 116 | Spenser    | Pollicott   | spollicott37@illinois.edu | Male   | Lithuania
 135 | Dev        | Charker     | dcharker3o@washington.edu | Female | Russia
 151 | Diandra    | Alcixo      | dalcixo46@cornell.edu   | Female | Palau
 158 | Walsh      | Brockhurst  | wbrockhurst4d@tamu.edu  | Male   | Azerbaijan
 172 | Jana       | Cutriss     | jcutriss4n@berkeley.edu | Female | Bulgaria
 191 | Rosana     | Ratnaga     | rratnaga5a@princeton.edu | Male   | Czech Republic
 214 | Nada       | Litchmore   | nlitchmore5x@tamu.edu   | Female | China
 215 | Charita    | Shadfourth  | cshadfourth5y@upenn.edu | Female | Vietnam
 233 | Terdinande | Stockhill   | fstockhill6g@washington.edu | Female | Thailand
 241 | Larry      | Cliburn     | lcliburn6o@si.edu       | Female | Hungary
 252 | Britte     | Barents     | bbarents6z@washington.edu | Male   | China
 295 | Sigfrid    | Banbridge   | sbanbridge86@ucla.edu   | Male   | China
 315 | Torgos     | Boschmann   | tboschmann8q@washington.edu | Male   | Democratic Republic of the Congo
 330 | Kathi      | Baptie      | kbaptie95@unc.edu       | Female | Nicaragua
 345 | Jarinah    | Chanson     | jchanson9k@utexas.edu   | Male   | Brazil
 367 | Joby       | Inglesant   | jinglesant6@harvard.edu | Male   | Malaysia
 374 | Fern       | Irnis       | firnisad@ucsd.edu       | Female | Greece
 378 | Harriett   | Zecchetti   | hzecchetti1ah@washington.edu | Female | Canada
 410 | Cortney    | Daudray     | cdaudraybc@cmu.edu      | Female | Croatia
 452 | Philippe   | Madgewick   | pmadgewickcj@ucsd.edu   | Female | Philippines
 462 | Hurbert    | Parell      | hparellct@mit.edu       | Female | Canada
 463 | Gavan      | Halladay    | ghalladaycu@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');



```
SQL Shell (psql)
startar=# SELECT * FROM travelers WHERE (country BETWEEN 'A' and 'D') AND (email LIKE '%.com');
 id  |    name    |   surname   |                email                | gender |        country
-----+------------+-------------+-------------------------------------+--------+-----------------------
   4 | Darrel     | Siddaley    | dsiddaley3@businessweek.com         | Female | China
   5 | Olva       | Leopard     | oleopard4@skyrock.com               | Male   | China
   6 | Delli      | Chapelhow   | dchapelhow5@godaddy.com             | Female | China
  10 | Jedidiah   | Cayser      | jcayser9@wordpress.com              | Female | China
  11 | Clary      | Rce         | crece@godaddy.com                   | Male   | China
  13 | Ronique    | Kulver      | rkulver0@weebly.com                 | Male   | Brazil
  15 | Godwin     | Gratrex     | ggratrex@networksolutions.com       | Male   | China
  18 | Brannon    | Cottisford  | bcottisford@yellowpages.com         | Female | China
  25 | Berti      | Najara      | bnajara@wix.com                     | Male   | China
  42 | Margaretha | Bauckham    | mbauckham15@mapquest.com            | Female | Colombia
  45 | Marc       | Sant        | msant1@devnub.com                   | Male   | China
  52 | Lemmond    | Sheflon     | lsheflon1f@1688.com                 | Female | China
  60 | Hedrie     | Thipps      | hthipps1n@mydailynews.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 | Ardalle    | McGurgan    | amcgurgan2g@marriott.com            | Female | China
  90 | Ellie      | Broske      | ebroske2h@imgur.com                 | Female | Colombia
  96 | Lorianne   | Ficcitti    | lficcitti2n@ezinearticles.com       | Female | China
  98 | Giraud     | Kiddle      | gkiddle2p@rediff.com                | Female | China
 181 | Sammy      | Devonish    | sdevonish2s@nbcnews.com             | Male   | Brazil
 185 | Florie     | Puckinghorne | fpuckinghorne2w@blog.com            | Female | Brazil
 113 | Keen       | Alkin       | kalkin34@go.com                     | Male   | Brazil
 130 | Zenete     | Idell       | zidell3l@ycombinator.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 | Cole       | 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;
```

```
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;

```
starter-# SELECT name, COUNT(*) FROM travelers GROUP BY name;
     name      | count
---------------+-------
 Carly         |     2
 Ewan          |     1
 Key           |     1
 Bebe          |     1
 Romain        |     1
 Peggie        |     1
 Eton          |     1
 Brnaba        |     1
 Launce        |     2
 Conrado       |     1
 Gay           |     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       |   104
 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;

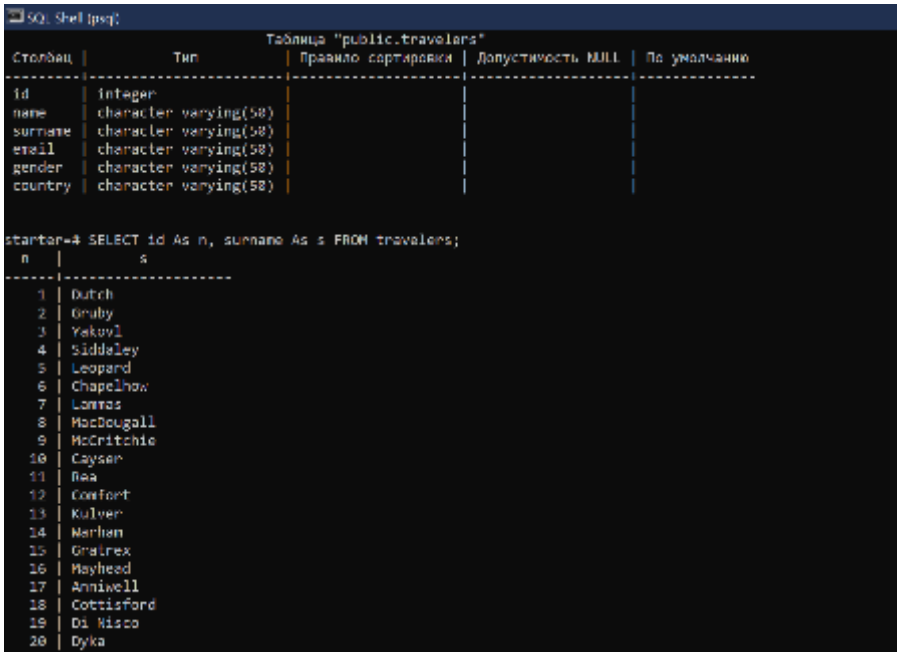
```
starter=# SELECT country, COUNT(*) FROM travelers GROUP BY country HAVING COUNT(*)>25 ORDER BY COUNT(*) DESC;
   country   | count
-------------+-------
 China       |   104
 Indonesia   |   104
 Russia      |    65
 Philippines |    63
 France      |    40
 Brazil      |    38
 Portugal    |    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 | Yakuvl
    4 | Siddaley
    5 | Leopard
    6 | Chapelhow
    7 | Lamas
    8 | MacDougall
    9 | McCritchie
   10 | Cayser
   11 | Rea
   12 | Comfort
   13 | Kulver
   14 | Marhan
   15 | Gralrex
   16 | Mayhead
   17 | Anniwell
   18 | Cottisford
   19 | Di Risco
   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                         |   980
  7 | Gerl        | Crickmoor         | Tajikistan             | Dominican Republic            |   189
  8 | Florinda    | Japp              | Brazil                 | Argentina                     |   464
  9 | Valerye     | Vasipov           | Poland                 | China                         |    75
 10 | Jeniffer    | Goosey            | Mexico                 | Thailand                      |   447
 11 | Phedra      | Moralas           | Bangladesh             | Macedonia                     |   124
 12 | Nikolas     | Parnham           | Honduras               | China                         |   374
 13 | Errell      | Iacabucci         | Serbia                 | Russia                        |   469
 14 | Tonlin      | Lghan             | China                  | Myanmar                       |   849
 15 | Fairfax     | Dark              | China                  | Russia                        |   364
 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 | Gilenelle   | Waggall           | 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 арифмет
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;

```
SQL Shell (psql)
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                         | 282
 South Africa            | China                         | 343
 Czech Republic          | Saint Barthelemy              | 587
 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;

```
SQL Shell (psql)
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
```

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 ровна 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 необходим 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        | Numberstone      | 1 year 6 mons 1 day 19:05:12.286245
 Rina          | Leaves           | 1 year 10 mons 20 days 19:05:12.286245
 Leyla         | Culpen           | 1 year 11 mons 6 days 19:05:12.286245
 Karna         | Hosby            | 1 year 9 mons 19:05:12.286245
 Berkie        | Yule             | 1 year 5 mons 8 days 19:05:12.286245
 Avrit         | Dahl             | 1 year 3 mons 19:05:12.286245
 Andrei        | Cockney          | 1 year 4 mons 27 days 19:05:12.286245
 Gates         | Dobbie           | 1 year 7 mons 19 days 19:05:12.286245
 Ulrich        | Bridge           | 1 year 9 mons 23 days 19:05:12.286245
 Georgiana     | Rehnke           | 1 year 8 mons 9 days 19:05:12.286245
 Fena          | Cokly            | 1 year 3 mons 26 days 19:05:12.286245
 Chicky        | Schonfelder      | 1 year 3 mons 19:05:12.286245
 Rodina        | Dennes           | 1 year 2 mons 19 days 19:05:12.286245
 Jacqueline    | Micah            | 1 year 6 mons 4 days 19:05:12.286245
 Othelia       | Kernley          | 1 year 7 mons 10 days 19:05:12.286245
 Hedwiga       | Leile            | 1 year 10 mons 23 days 19:05:12.286245
 Miguelita     | Dawid            | 1 year 5 mons 3 days 19:05:12.286245
 Retha         | Tidgewell        | 2 years 22 days 19:05:12.286245
 Balduin       | Cordall          | 1 year 5 mons 16 days 19:05:12.286245
 Lia           | Kermott          | 1 year 6 mons 25 days 19:05:12.286245
 Aurilia       | Densumbe         | 1 year 5 mons 17 days 19:05:12.286245
 Dalcina       | Pendared         | 1 year 6 mons 17 days 19:05:12.286245
 Lewiss        | Yarwshey         | 1 year 5 mons 20 days 19:05:12.286245
```

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 первичны 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

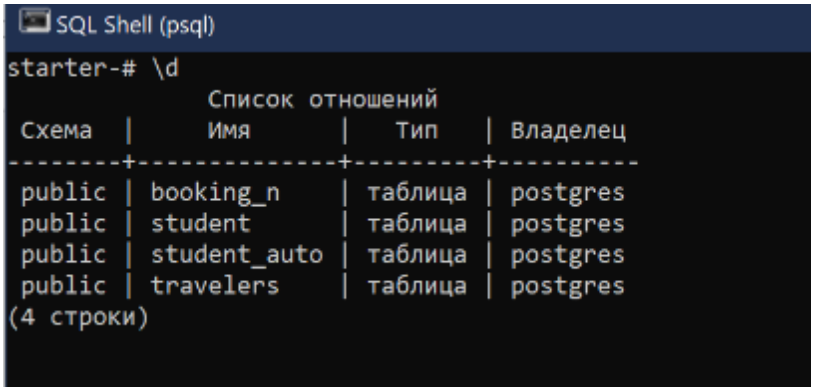


Рисунок 1

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
 Roncin        |     1
 Peggie        |     1
 Etan          |     1
 Brnaba        |     1
 Launce        |     2
 Conrado       |     1
 Say           |     1
 Aurea         |     1
 Suzanne       |     1
 Tuck          |     1
 Anstice       |     1
 Karla         |     1
 Hirsch        |     1
 Sylvester     |     1
 Washington    |     1
 Wayne         |     1
 Annaliese     |     1
 Barbabas      |     1
 Dalt          |     1
 Clary         |     1
 Pancho        |     1
 Keene         |     1
 Sheryl        |     1
```

Рисунок 2

Как видим из запроса, повторяются имена не часто.

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

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

```
starter=# select name, count(*) from travelers group by name having count(*)>1;
    name     | count
             +
 Carly       |     2
 Launce      |     2
 Pattan      |     3
 Shelly      |     2
 Darya       |     3
 Ellynn      |     2
 Merry       |     2
 Salomo      |     2
 Spenser     |     2
 Rafaelita   |     2
 Connie      |     2
 Suzann      |     2
 Arsta       |     2
 Heriberto   |     2
 Theresa     |     2
 Marc        |     2
 Fina        |     2
 Mile        |     2
 Christabel  |     2
 Marline     |     2
 Cesare      |     2
 Gino        |     2
 Delcine     |     2
 Cassandra   |     2
 Kylie       |     2
 Berti       |     2
 Kameko      |     2
 Ferdinanda  |     2
 Charlot     |     2
 Hugh        |     2
 Velvet      |     2
```

Рисунок 3

Создадим ограничения.

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

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

и вставим эту строку в консоль postgresql

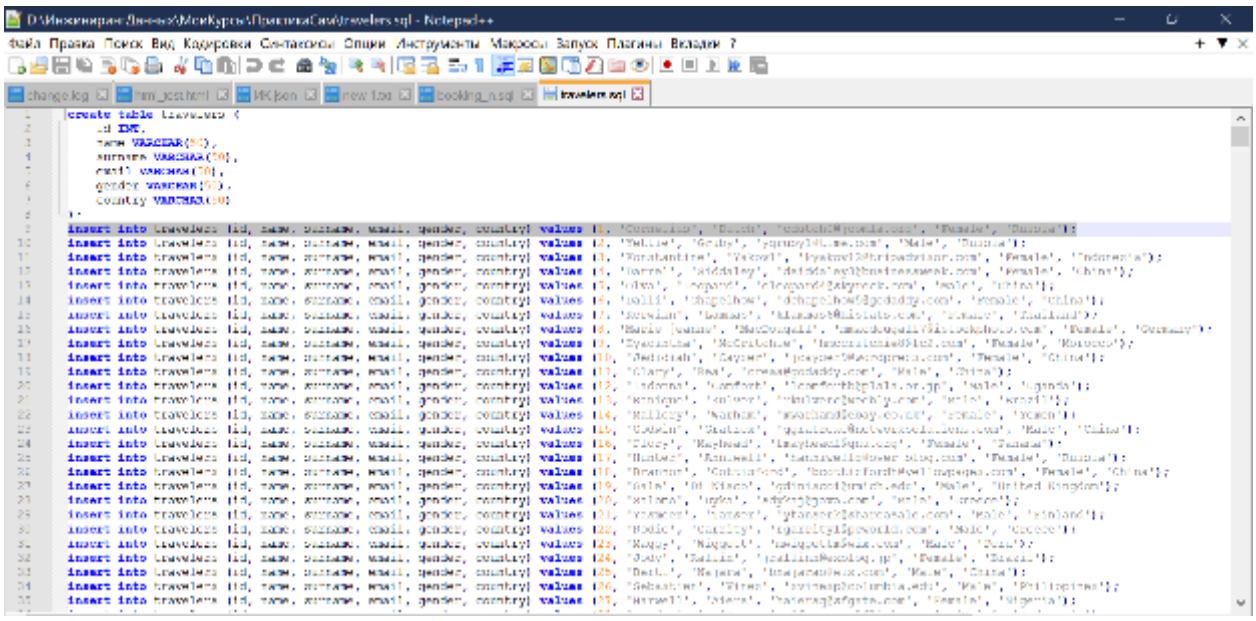


Рисунок 4

Как результат, успешная вставка в таблицу (рисунок 5)

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

Рисунок 10

Как видим на рисунке 10, внизу появилась надпись о ключевом полем 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@jocela.org', 'Female', 'Rus
sia');
ОШИБКА:  повторяющееся значение ключа нарушает ограничение уникальности "unique_id"
ПОДРОБНОСТИ:  Ключ "(id)=(1)" уже существует.
starter=#
```

Рисунок 11

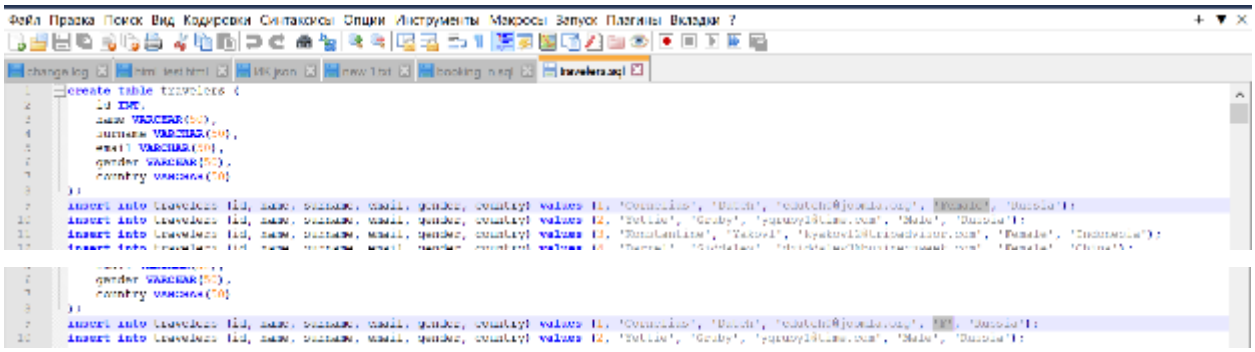
Как видим, появляется сообщение об ошибке о внесении повторяющихся значений.

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

select * from travelers;

Рисунок 12

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

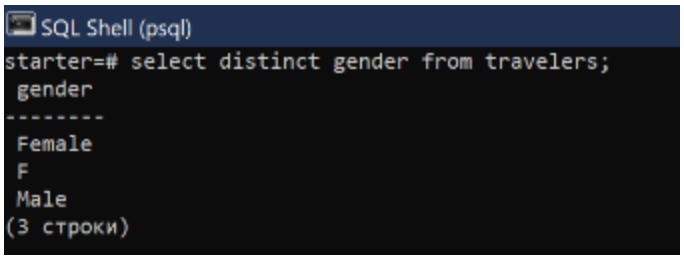


Рисунок 14

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');

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

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

delete from travelers where gender='F';

```
starter=# delete from travelers where gender='F';
DELETE 1
starter=#
```

Теперь снова применим запрос

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

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

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;

```
SQL Shell (psql)
startar=# select * from booking_n;
 id |    name    |     surname     |     departure_from     |        arriving_to        | price
----+------------+-----------------+------------------------+---------------------------+-------
  1 | Reider     | MacBarron       | Brazil                 | Portugal                  |   414
  2 | Alison     | Rothschild      | France                 | Russia                    |   169
  3 | Darr       | Le Fleming      | Argentina              | Philippines               |   255
  4 | Tanny      | Jobbins         | Indonesia              | Indonesia                 |   781
  5 | Rurik      | Semeniuk        | Japan                  | Peru                      |   392
  6 | Briana     | Yare            | China                  | Japan                     |   908
  7 | Gert       | Crickmoor       | Tajikistan             | Dominican Republic        |   188
  8 | Florinda   | Japp            | Brazil                 | Argentina                 |   464
  9 | Valerya    | Yastpov         | Poland                 | China                     |    75
 10 | Jeniffer   | Gaskay          | Mexico                 | Thailand                  |   447
 11 | Phedra     | Moralas         | Bangladesh             | Macedonia                 |   124
 12 | Nikolas    | Parnham         | Honduras               | China                     |   374
 13 | Errell     | Jacobucci       | Serbia                 | Russia                    |   469
 14 | Torlin     | Lghan           | China                  | Myanmar                   |   849
 15 | Fairfax    | Dark            | China                  | Russia                    |   394
 16 | Gaby       | Yesson          | Cuba                   | Colombia                  |   748
 17 | Rylcy      | Petrasek        | Philippines            | Mexico                    |   991
 18 | Kelly      | Elvy            | China                  | Russia                    |    96
 19 | Alisun     | Posnatte        | China                  | Japan                     |   537
 20 | Gilenatte  | Waggatt         | Indonesia              | China                     |   545
 21 | Jienia     | Broomer         | France                 | China                     |   147
 22 | Josias     | Tirte           | France                 | Palestinian Territory     |   532
 23 | Janine     | Rens            | Lithuania              | Czech Republic            |   211
 24 | Renala     | Danielkiewicz   | France                 | China                     |   712
 25 | Augustus   | Spellwood       | 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                    |   388
 30 | Christopha | Grena           | 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

```
 4 | Tanny      | Jobbins         | Indonesia
 5 | Rurik      | Semeniuk        | Japan
```

Необходимо изменить это имя на настоящее.

Сделать мы это можем при помощи команды UPSERT. Во время выполнения этой команды мы можем уточнить где конкретно происходить изменение с помощью той же логики, что мы использовали для выведения 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.

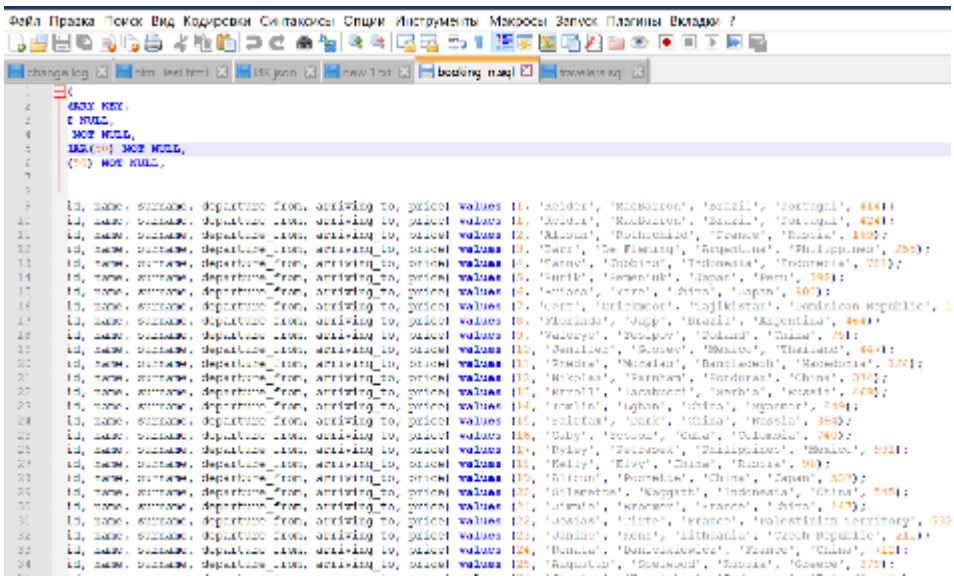
Допустим были перепутаны места вылета и прибытия

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 попытки вставить этой строки мы получим сообщение о попытки 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.

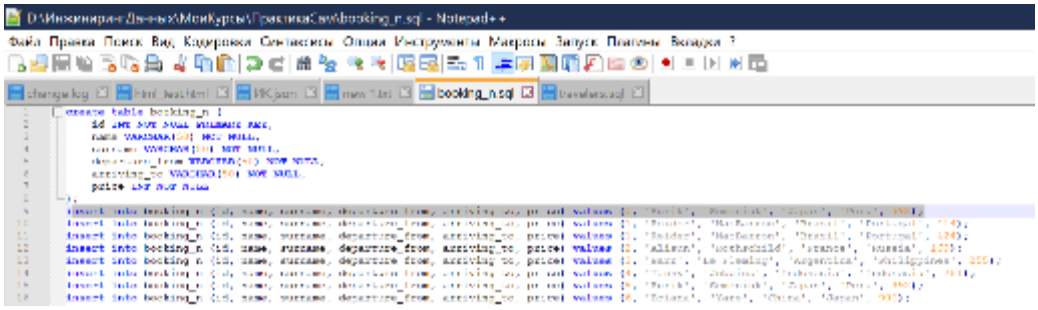
```
starter=# insert into booking_n (id, name, surname, departure_from, arriving_to, price) values (5, 'Rurik', 'Semeniuk',
'Japan', 'Peru', 392) on conflict (id) do nothing;
INSERT 0 0
starter=# _
```

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


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

Команду **on conflict** нельзя применить к столбцам не связанным ни к каким правилами. На дынный 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.



```
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 | Rurik | Semeniuk | 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', 'Semeniuk',
'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', 'Semeniuk', 'Japan', 'Peru', 392) on conflict (**surname**) do nothing;

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

RE id = 0;
om | arriving_to | price
---+-------------+-------
   | Peru        |   392


name, surname, departure_from, arriving_to, price) values (0, 'Rurik', 'Semeniuk', 'Spain', 'Peru', 392);
 нарушает ограничение уникальности "booking_n_pkey"
ествует.
name, surname, departure_from, arriving_to, price) values (0, 'Rurik', 'Semeniuk', '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
                                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) |                    |                   |
 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 ссылки из вне.

```
SQL Shell (psql)
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=# _
```

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   | kyakovl2@tripadvisor.com    | Female | Indonesia |
  4 | Darrel      | Siddaley | dsiddaley3@businessweek.com | Female | China     |
  5 | Olva        | Leopard  | oleopard4@skyrock.com       | Male   | China     |
(3 строки)
```

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

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

Лабораторная работа № 7. Соединение таблиц. Экспорт в CSV

Цель работы: освоить технологию соединения таблиц и проводить экспорт данных

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

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

Соединить таблицы можно с помощью команды JOIN ON/

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

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

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

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

starter=#
```

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

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

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

id	name	surname	email	gender	country	address_id	id	street	house
3	Konstantine	Yakovl	kyakovl2@tripadvisor.com	Female	Indonesia				
4	Darrell	Siddaley	dsiddaley3@businessweek.com	Female	China				
5	Olya	Leopard	oleopard4@nytimes.com	Male	China				
6	Delli	Chapelhow	dchapelhow5@godaddy.com	Female	China				
7	Kerwin	Tamas	ktamas6@hostgator.com	Female	Thailand				
8	Marie-jeanne	Macdougall	mmacdougall7@istockphoto.com	Female	Germany				
9	Hyacintha	McCritchie	hmccritchie8@hc2.com	Female	Morocco				
10	Jedidiah	Cayser	jcayser9@wordpress.com	Female	China				
11	Clary	Koa	ckoa2@godaddy.com	Male	China				
12	Ladonna	Comfort	lcomfortb@plala.or.jp	Male	Uganda				
13	Manique	Salyer	rkalyerc@weebly.com	Male	Brazil				
14	Mallory	Warham	mwarhamd@ebay.co.uk	Female	Yemen				
15	Godwin	Gratrex	ggratrexe@networksolutions.com	Male	China				
16	Flury	Mayhead	fmayheadf@gnu.org	Female	Panama				
17	Hunter	Annivell	hannivellg@over-blog.com	Female	Russia				
18	Brannon	Cottrafford	bcottrafordh@yellowpages.com	Female	China				
19	Gale	Di Nicro	gdinicroni@march.edu	Male	United Kingdom				
20	Salomo	Dyka	sdykaj@java.com	Male	Greece				
21	Yasmeen	Tanner	ytannerk@chronicle.com	Male	Finland				
22	Rodie	Garrity	rgarrityl@powerd.com	Male	Greece				
23	Maggy	Wiggett	mwiggettm@vix.com	Male	Peru				
24	Jody	Kallin	jkallinn@exblog.jp	Female	Brazil				
25	Berti	Najara	bnajaraoo@vix.com	Male	China				

Как мы видим, таблица путешественников показалась полностью, создав три столбца из таблицы адресов. Данные столбцы пусти 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 получм 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)
```

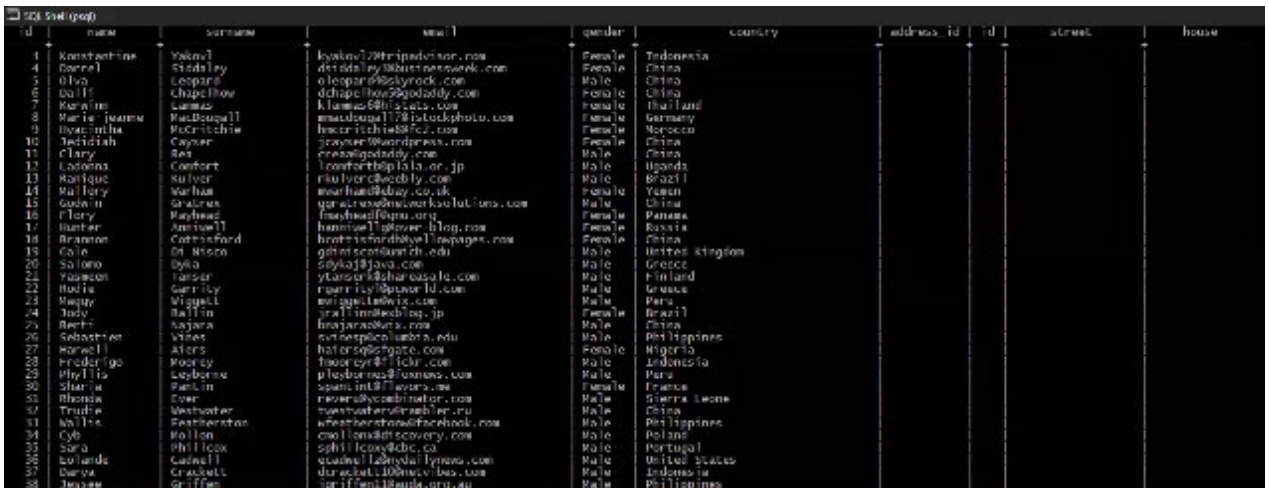
Команда right join полностью покажет ту таблицу, которую присоединяют

```
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.



RIGHT JOIN

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

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

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

LEFT JOIN

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

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

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

FULL JOIN

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

Пример :

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

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

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

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

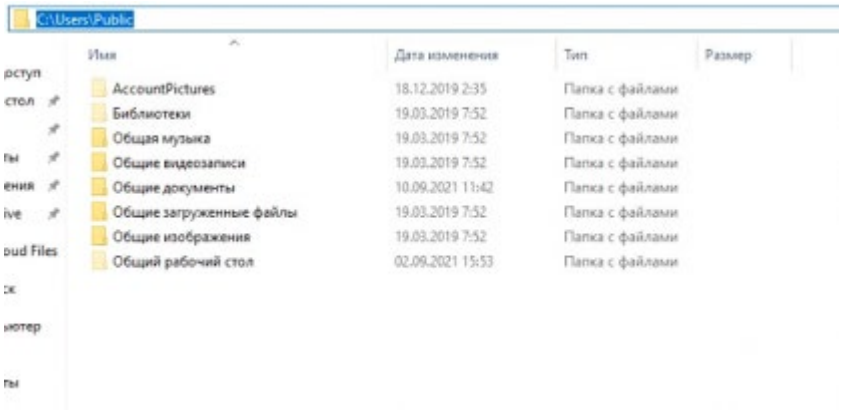
1. Сделаем выборку из таблицы travelers, например, первые 50 строк, с помощью команды **SELECT * FROM travelers WHERE id < 50**. Посмотрим на её общий внешний вид и запомним его.

```
starter=# SELECT * FROM travelers WHERE id < 50;
 id |     name     |   surname    |              email              | gender |    country    | address_id
----+--------------+--------------+---------------------------------+--------+---------------+------------
  1 | Cornelius    | Dutch        | cdutch0@joomla.org              | Female | Russia        |
  2 | Yettie       | Gruby        | ygruby1@time.com                | Male   | Russia        |          1
  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     | 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  | wfeatherstonw@facebook.com      | Male   | Philippines   |
 34 | Cyb          | Mollon       | cmollonx@discovery.com          | Male   | Poland        |
 35 | Sara         | Phillcox     | sphillcoxy@cbc.ca               | Male   | Portugal      |
 36 | Eolande      | Cadwell      | ecadwellz@mydailynews.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        | Milroy       | kmilroy13@yolasite.com          | Female | Indonesia     |
 41 | Bartholemy   | Milliam      | bmilliam14@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     | Checcuzzi    | rcheccuzzi1b@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.



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

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

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

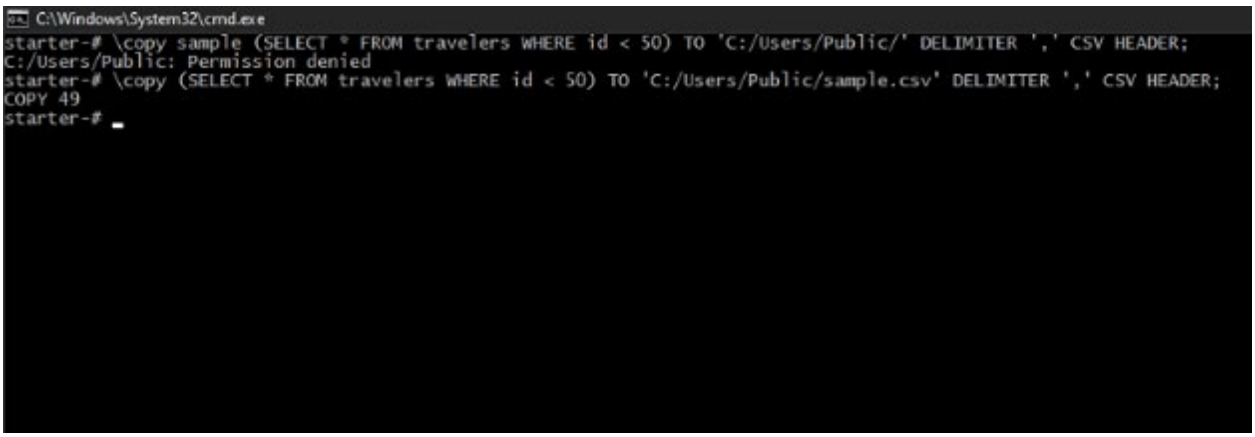
```
C:\Windows\System32\cmd.exe
starter=# \copy (SELECT * FROM travelers WHERE id < 50) TO 'C:/Users/Public/' DELIMITER ',' CSV HEADER;
```

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

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

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

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




```
\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

```
id,name,surname,email,gender,country,address_id
1,Cornelius,Dutch,cdutch0@joomla.org,Female,Russia,
2,Yettie,Gruby,ygruby1@time.com,Male,Russia,1
3,Konstantine,Yakov1,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@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,
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,wfeatherstonw@facebook.com,Male,Philippines,
34,Cyb,Mollon,cmollonx@discovery.com,Male,Poland,
35,Sara,Phillcox,sphillcoxy@cbc.ca,Male,Portugal,
36,Eolande,Cadwell,ecadwellz@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,Milroy,kmilroy13@yolasite.com,Female,Indonesia,
41,Bartholemy,Milliam,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,Checcuzzi,rcheccuzzi1b@google.com.hk,Female,Russia,
49,Lib,Bilson,lbilson1c@de.vu,Female,Greece,
```

Лабораторная работа № 8-9.

Знакомство с базой данных «Авиаперевозки»

Простые запросы

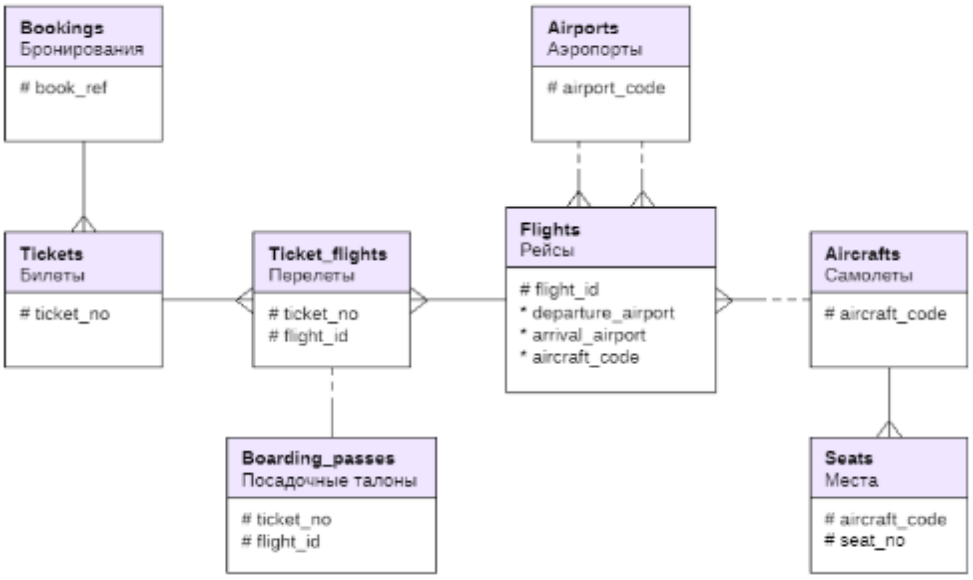
Цель работы: изучить схему данных БД «Авиаперевозки». Познакомится с платформой DBeaver.

Теоретическое обоснование

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

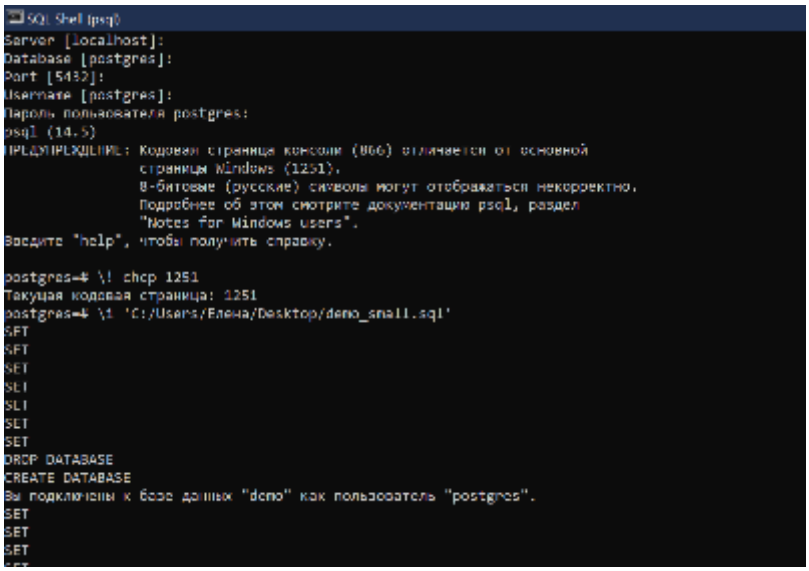
Мультплатформенный универсальный (работает с большим числом СУБД, как SQL, так и NoSQL) менеджер баз данных DBeaver как раз и является инструментом, который в значительной степени облегчает проектирование, реализацию, модификацию и работу с различными СУБД.

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

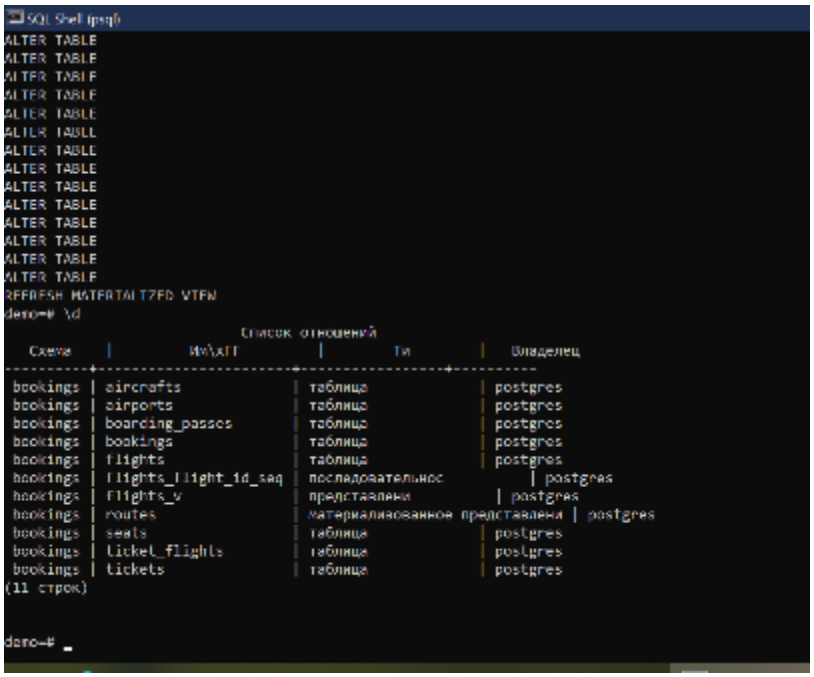


Методика выполнения

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.



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



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.

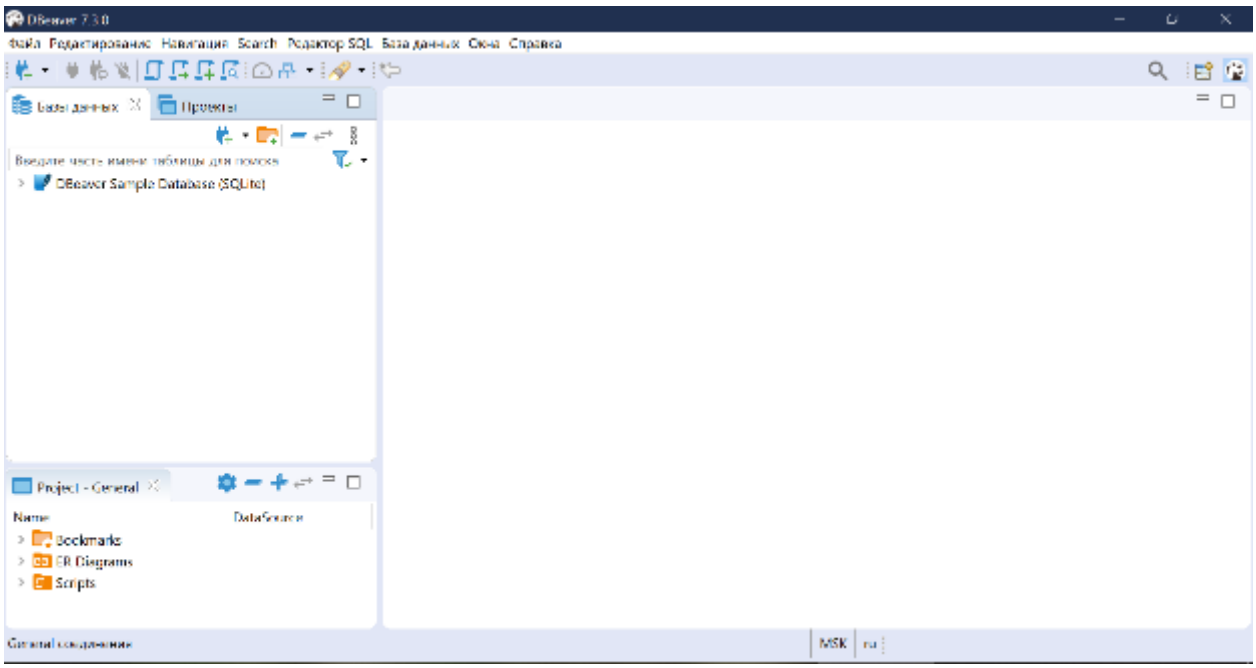


Рисунок 1 - Окно интерфейса DBeaver

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

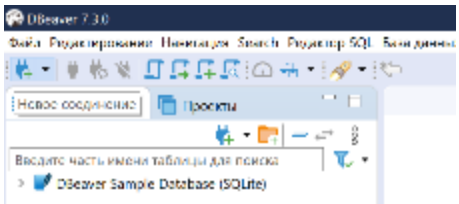


Рисунок 2 – Подключение БД

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

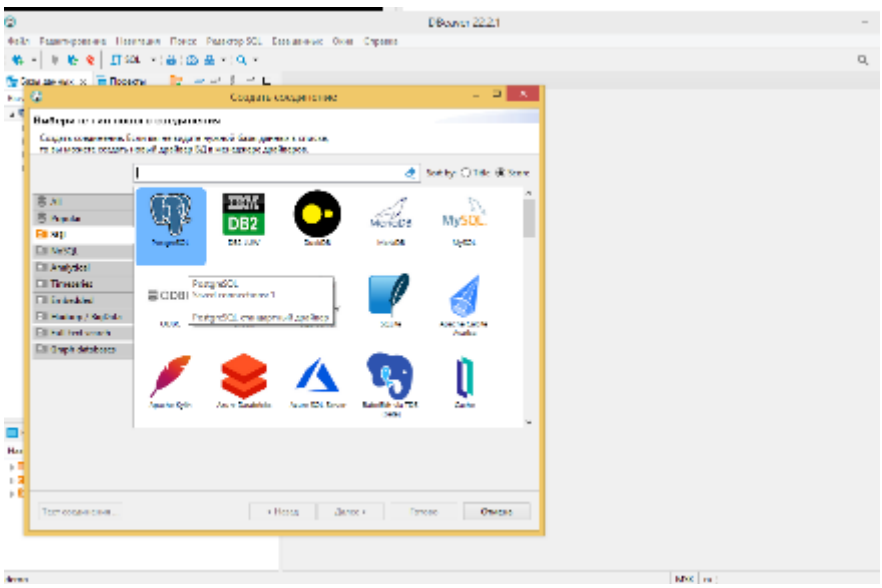


Рисунок 3 – Окно выбора нового соединения

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

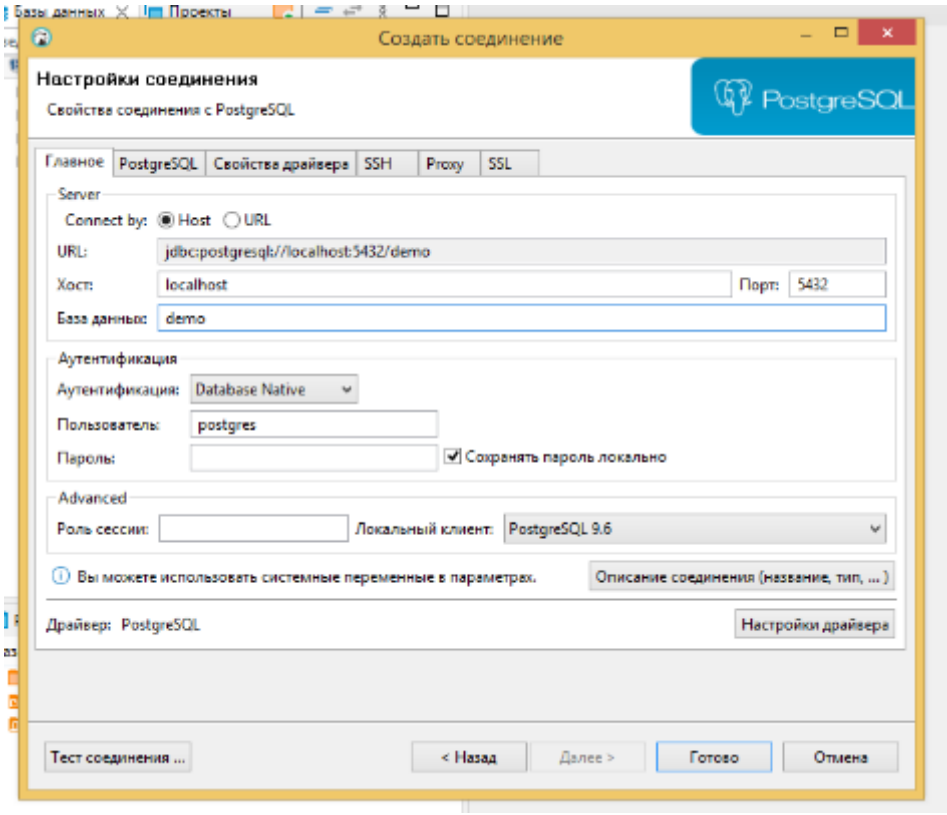


Рисунок 4 – Настройка соединения

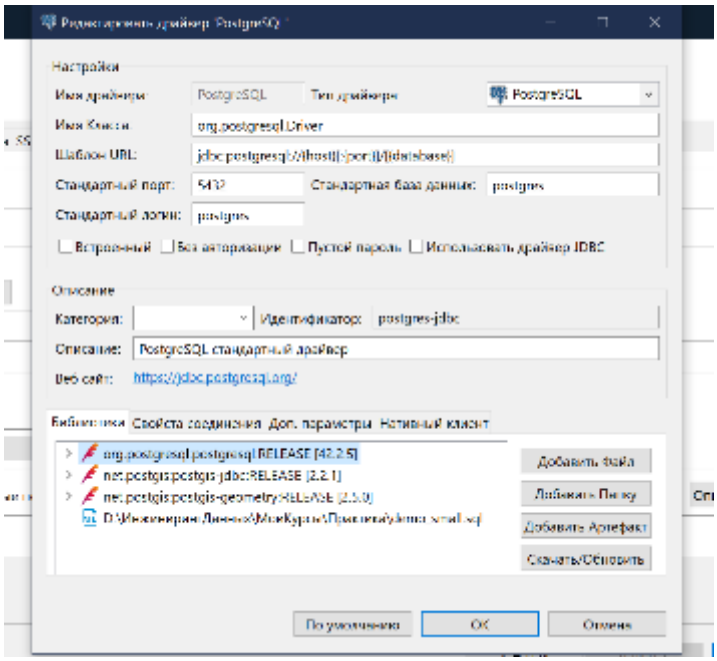


Рисунок 5 – Выбор драйвера

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 появиться БД demo (рисунок 6).

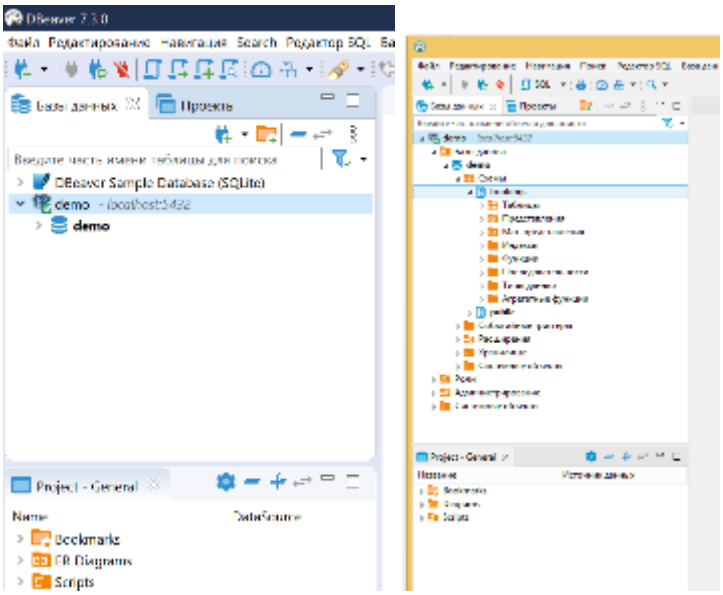


Рисунок 6 – Подключение БД

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).

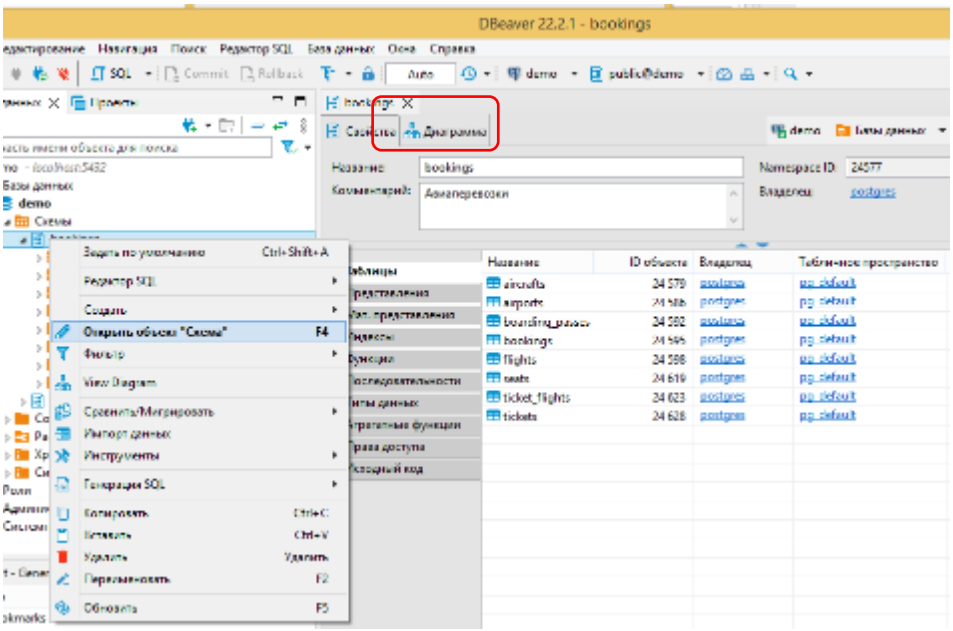


Рисунок 7 – Открытие Схемы данных

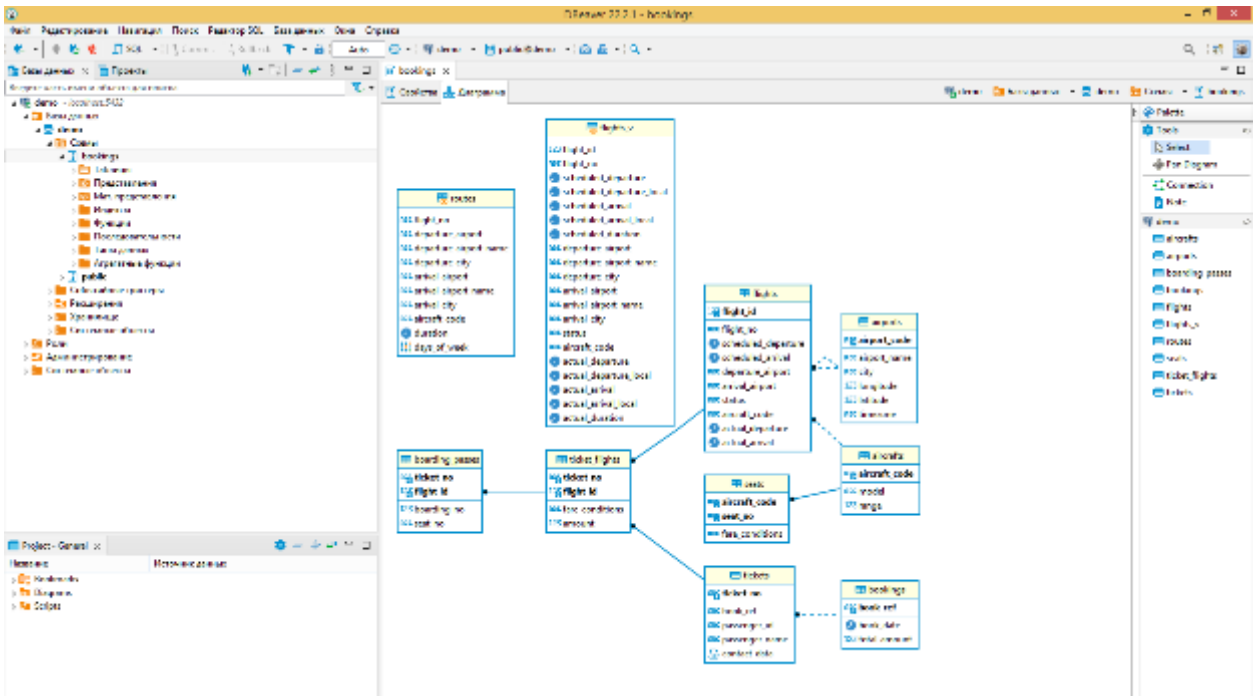


Рисунок 8 – Схема данных БД

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).

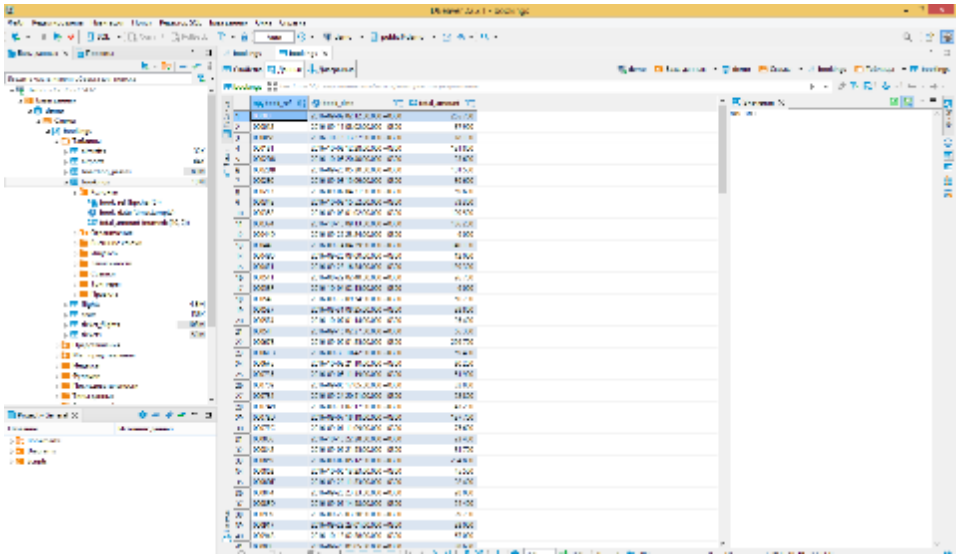


Рисунок 9 – Таблица Bookings (данные таблицы)

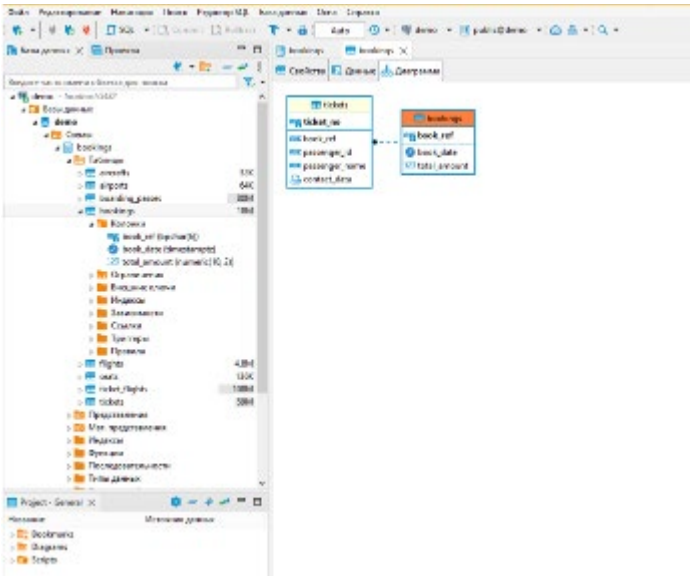


Рисунок 10 – Диаграмма таблицы Bookings

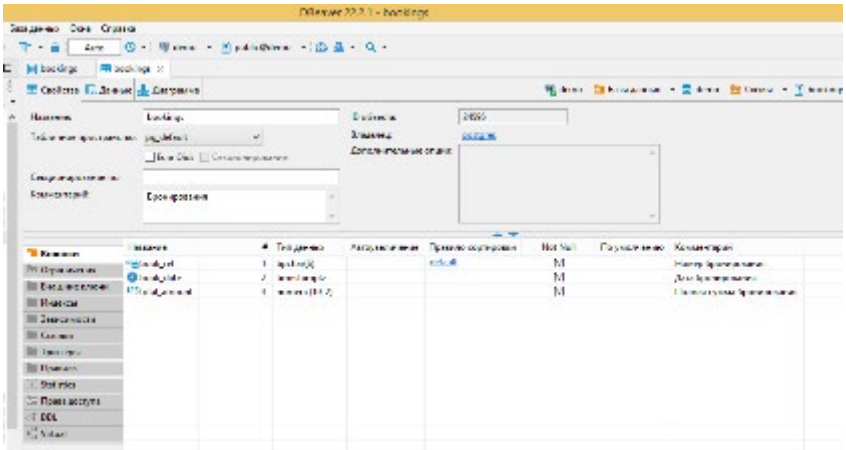


Рисунок 11 – Поля таблицы Bookings

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).

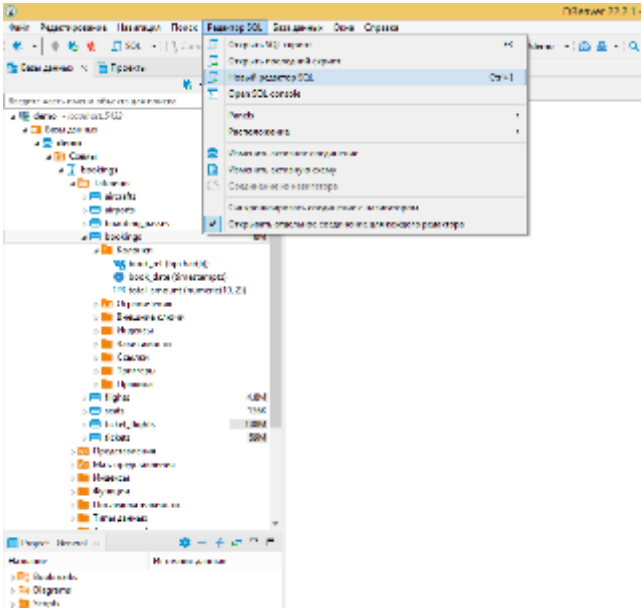


Рисунок 12 – Открытие редактора SQL

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

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

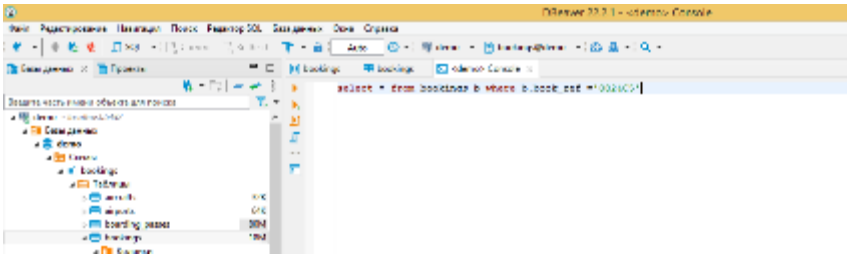


Рисунок 13 – Заброс о бронировании

Мы видим дату бронирования и общую сумму.

Рисунок 14 – Результат запроса бронирования

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

```
select now();
```

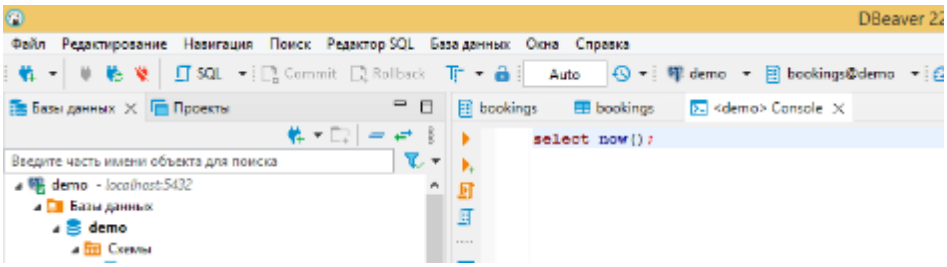


Рисунок 15 – Запрос текущей даты

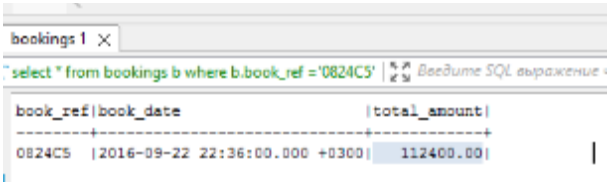


Рисунок 16 – Результат запроса текущей даты

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

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

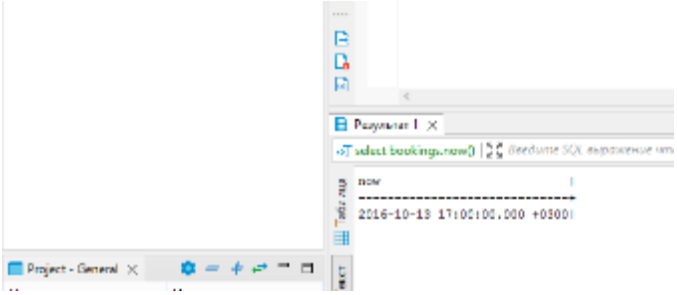


Рисунок 17 – Результат запроса текущей даты БД

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).

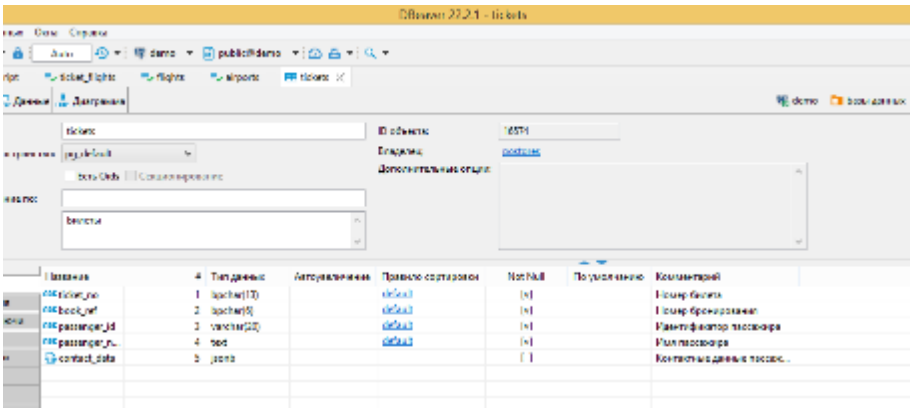


Рисунок 18 – Таблица Билеты (Tickets).

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

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

Посмотрим, какие билеты включены в выбранное бронирование.

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

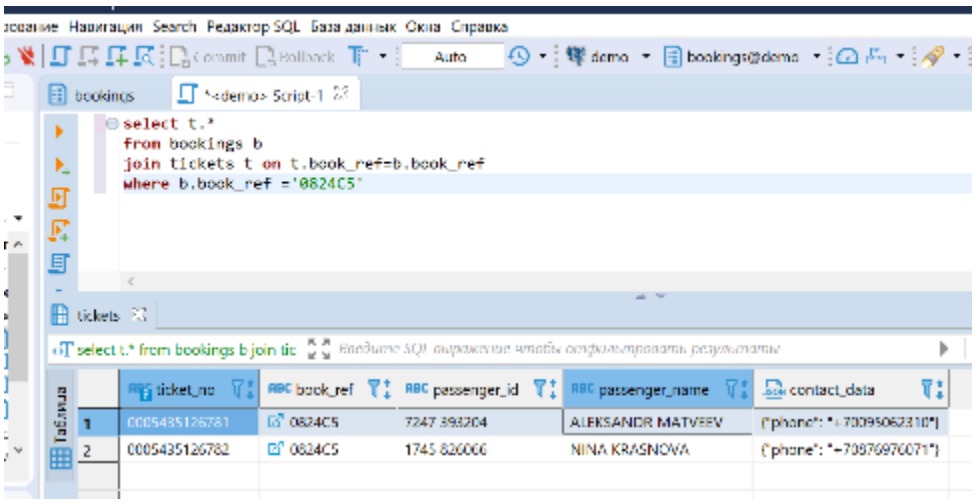


Рисунок 18 – Запрос о билетах бронирования

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

Следующая таблица Перелеты (Ticket_flights).

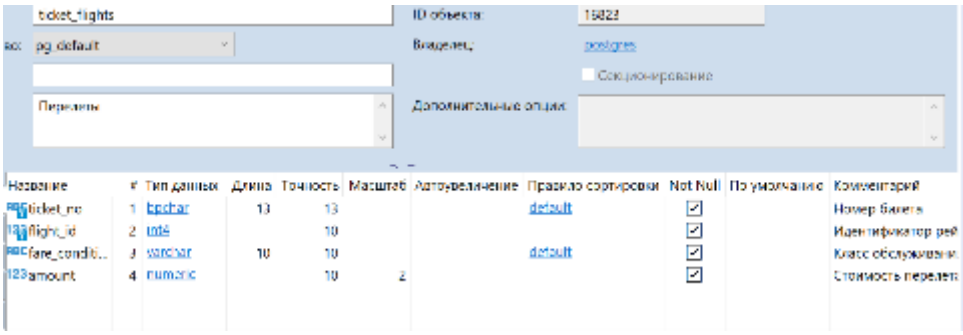


Рисунок 19 – Структура таблицы Перелеты (Ticket_flights)

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

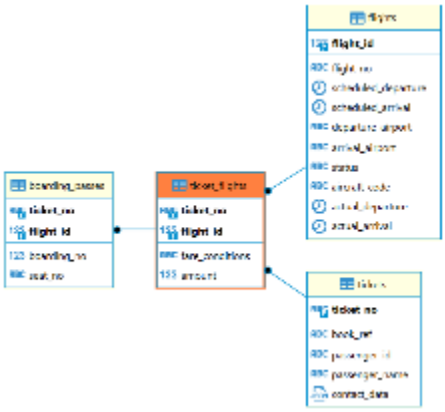


Рисунок 20 – Схема данных Перелеты (Ticket_flights)

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';
```

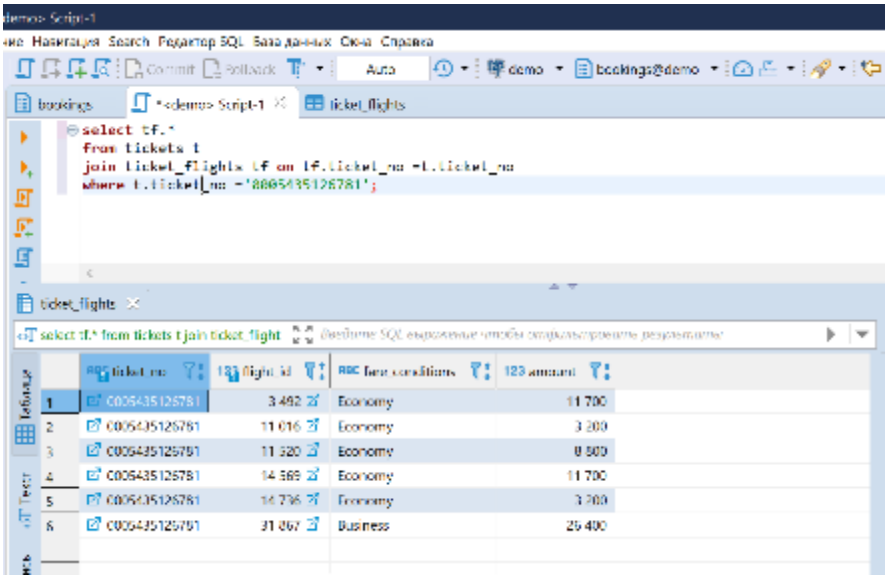


Рисунок 21 – Запрос по перелетам

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.

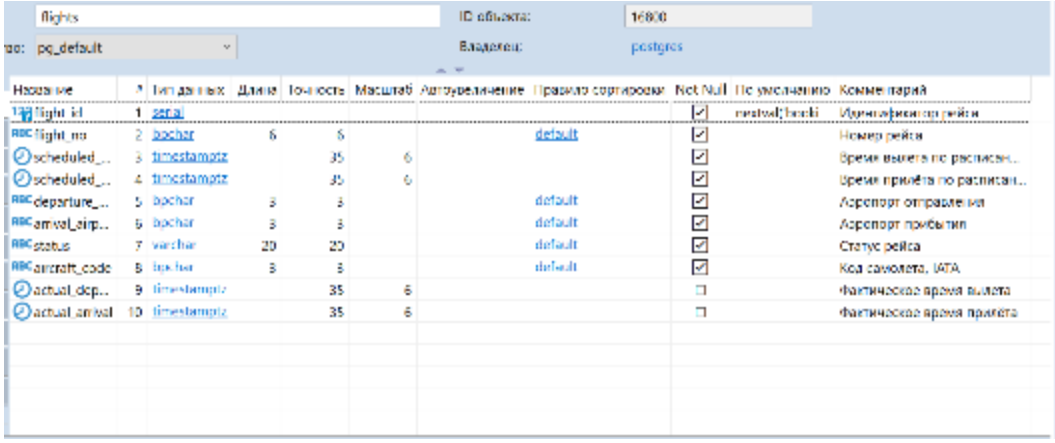


Рисунок 22 – Структура таблицы Рейсы (Flights)

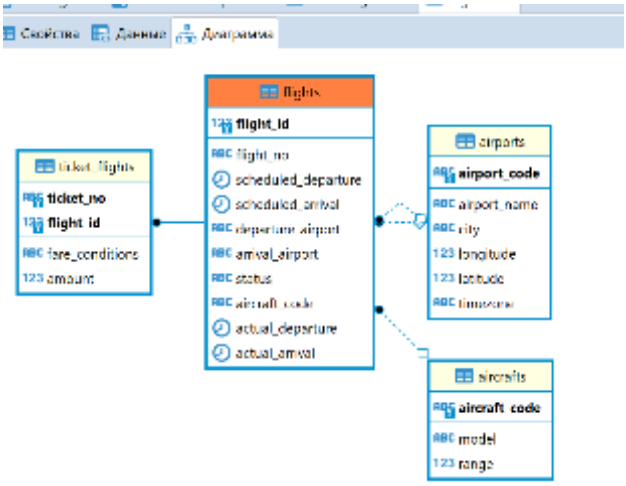


Рисунок 23 – Схема данных таблицы Рейсы (Flights)

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 ;
```

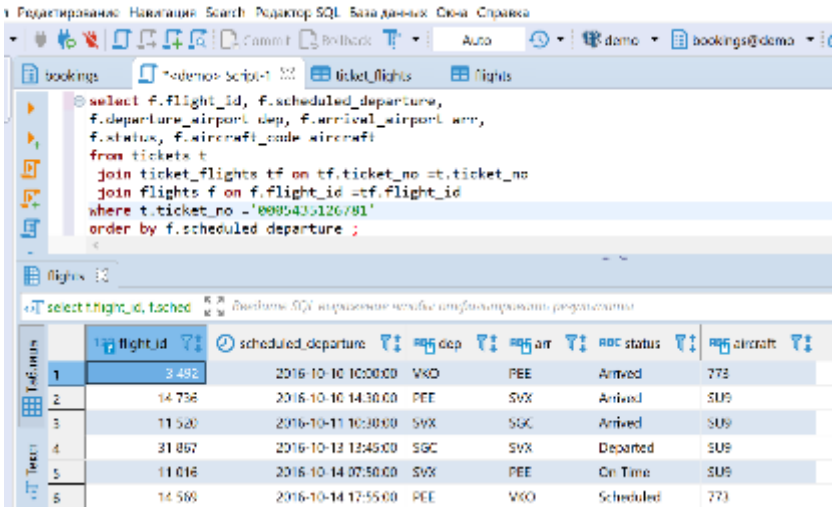


Рисунок 24 – Результат запроса таблицы Рейсы (Flights)

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.

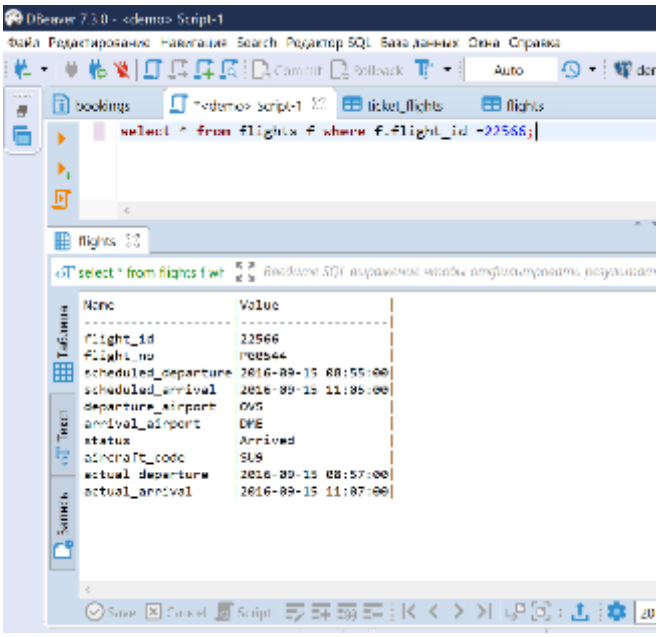


Рисунок 25

Реальное время может отличатся 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;
```

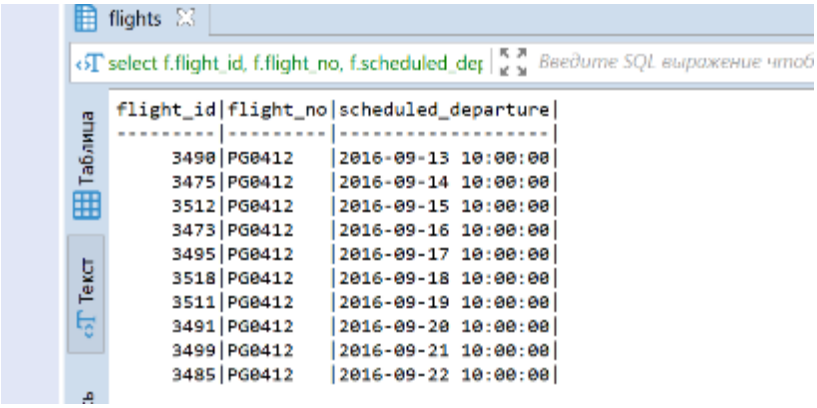


Рисунок 26

Следующая таблица «Аэропорты» (Airports)

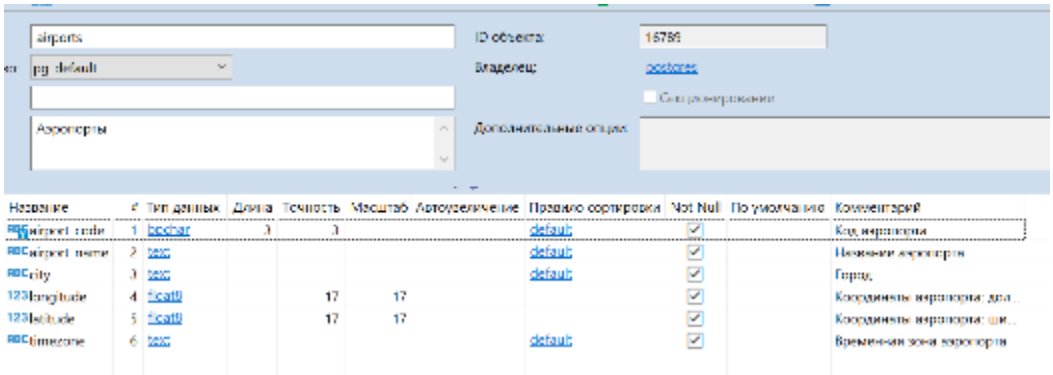


Рисунок 27

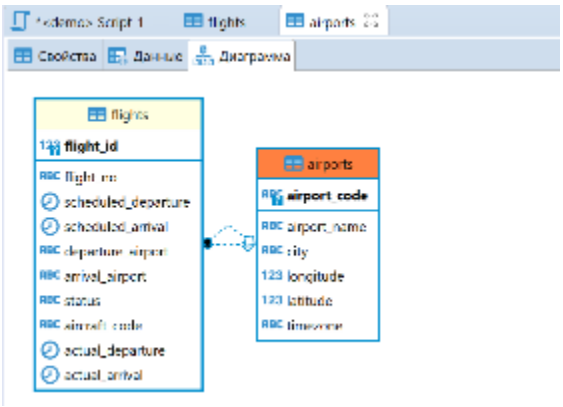


Рисунок 28 – Схема данных таблицы «Аэропорты» (Airports)

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 || ')' arrivel
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;
```

flights

select f.scheduled_departu | Введите SQL выражение чтобы отфильтровать результаты

Таблица | Текст

scheduled_departure	departure	arrivel
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;
```

flights_v 1

SELECT * FROM flights_v f WHERE f.f | Введите SQL выр

Таблица | Текст | Запись

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

Рисунок 30

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

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

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

routes 1

select * from routes r where r.flight_r

Таблица

Текст

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.

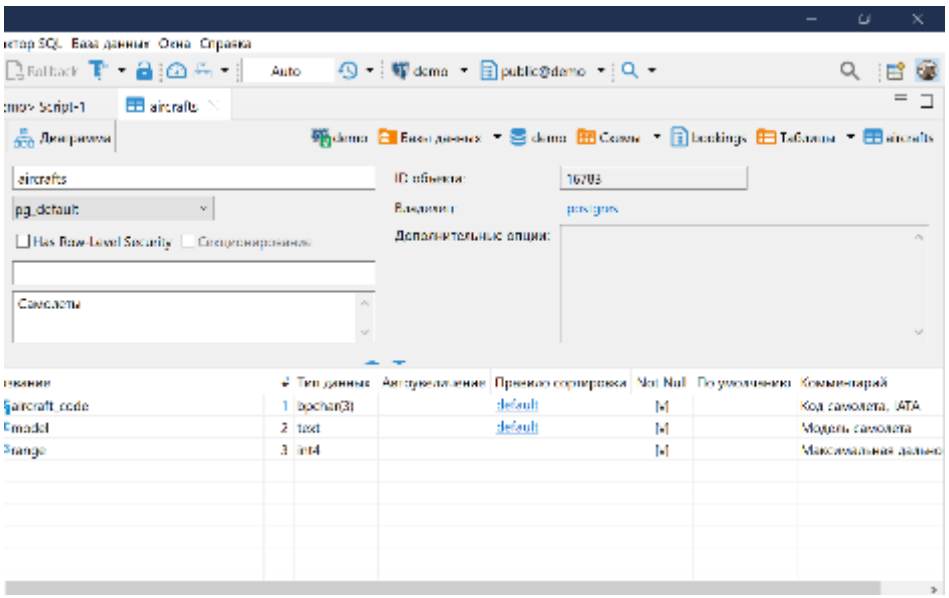


Рисунок 32

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;
```

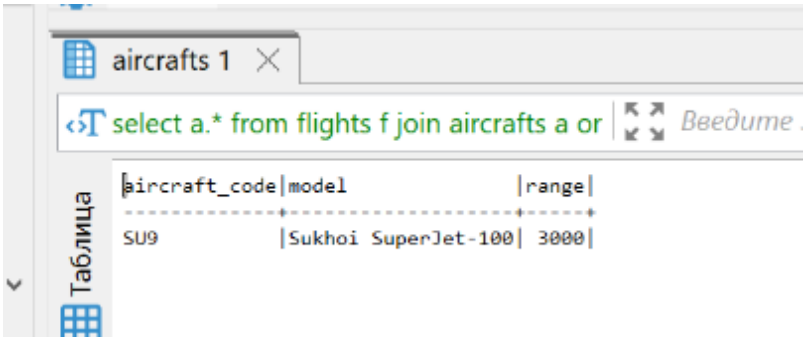


Рисунок 33

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.

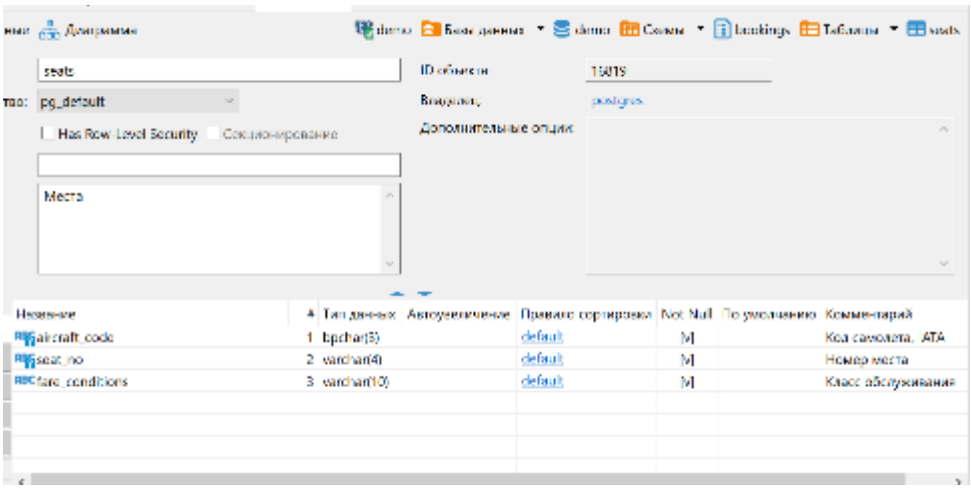


Рисунок 34

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.$';
```

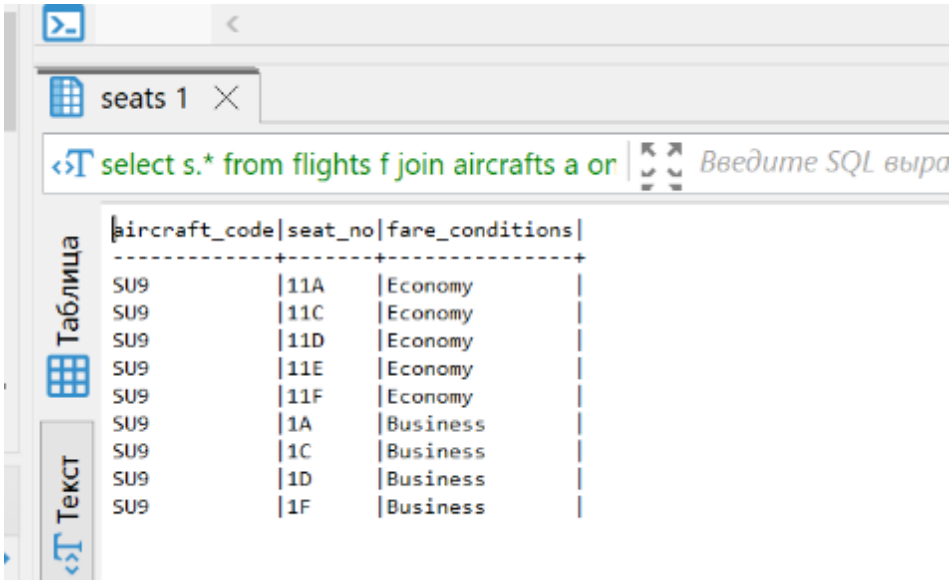


Рисунок 35

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 ;
```

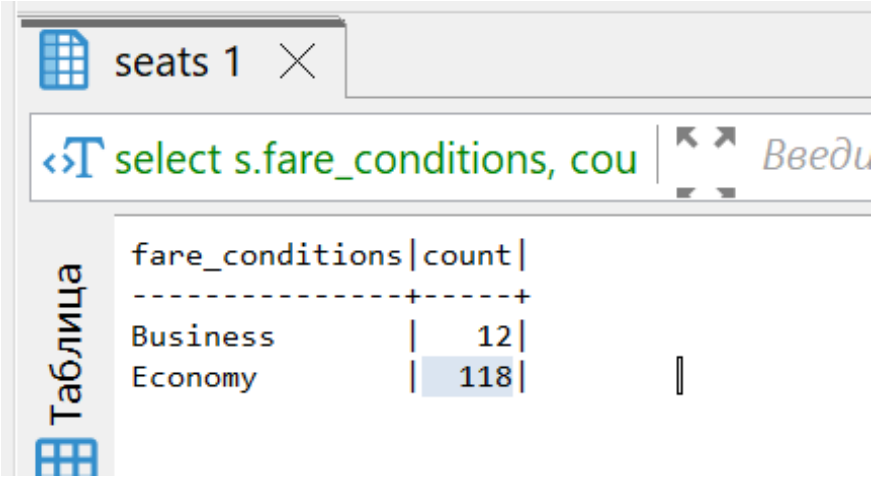


Рисунок 36

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).

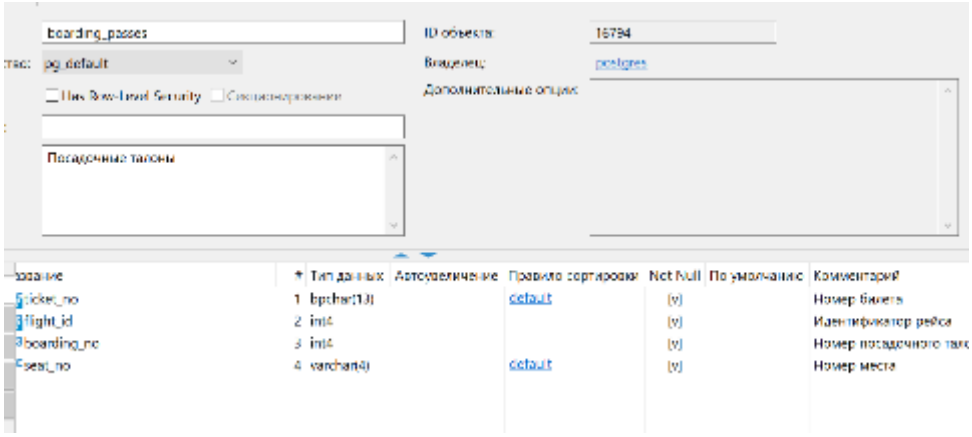


Рисунок 37

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

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

flights(+) 1

select f.status, bp.* from tickets t join tic Введите SQL выражение

Таблица

Текст

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.

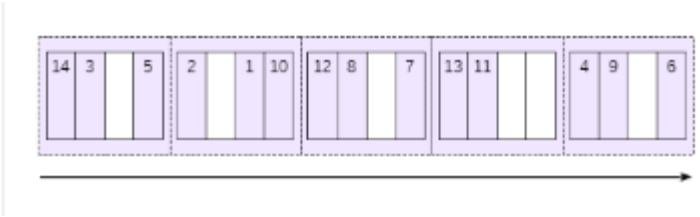


Рисунок 1 – Схема последовательного сканирования

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)

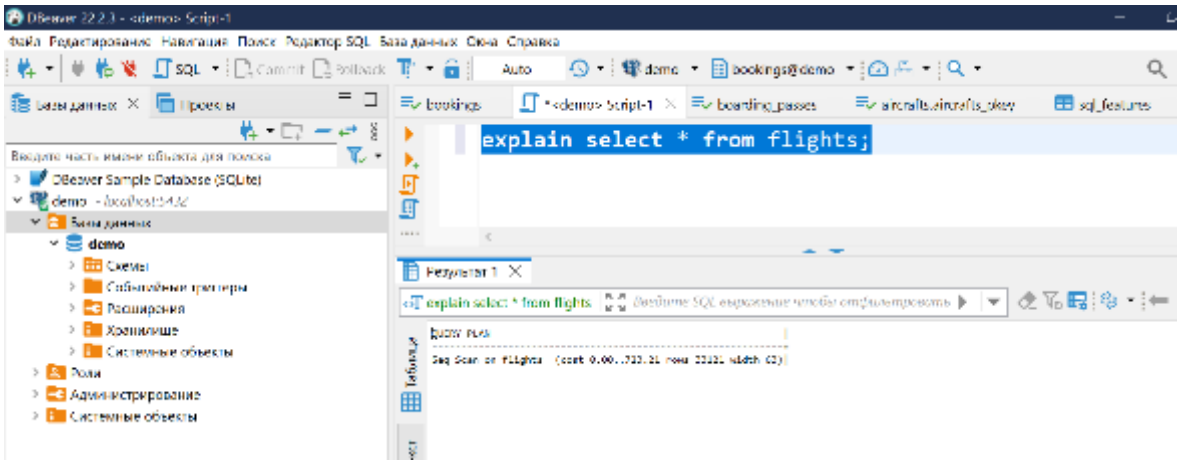


Рисунок 2 – Результат запроса последовательного сканирования

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';
```

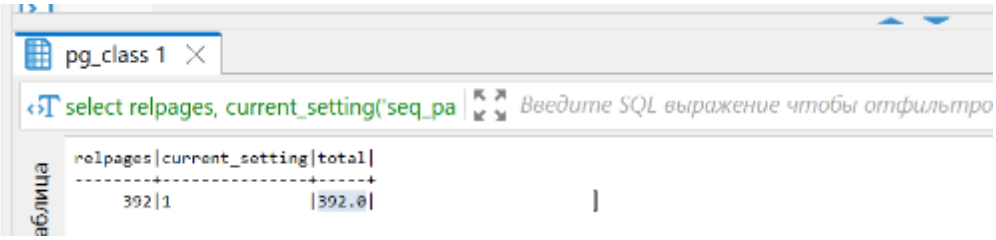


Рисунок 3 – Результат вычисления составляющей ввода-вывода

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';
```

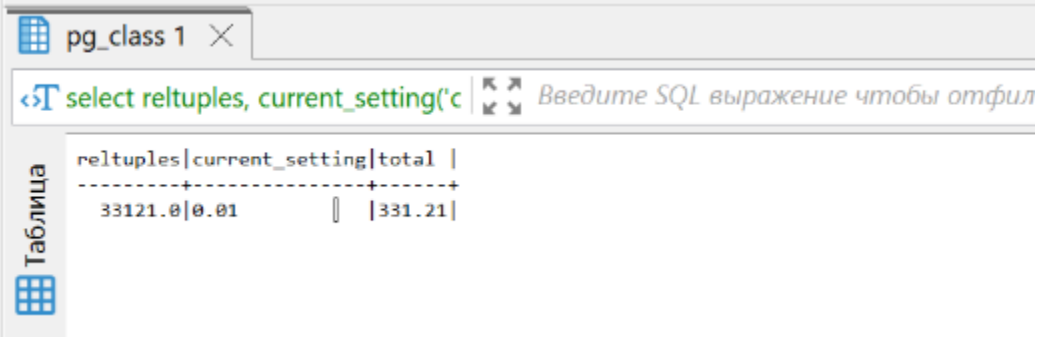


Рисунок 4 - Результат вычисления составляющей процессора

Агрегация

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

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

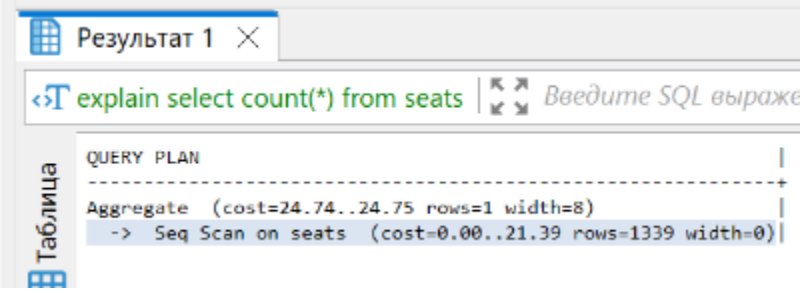


Рисунок 5 - Сложный запрос с агрегатной функцией

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';
```

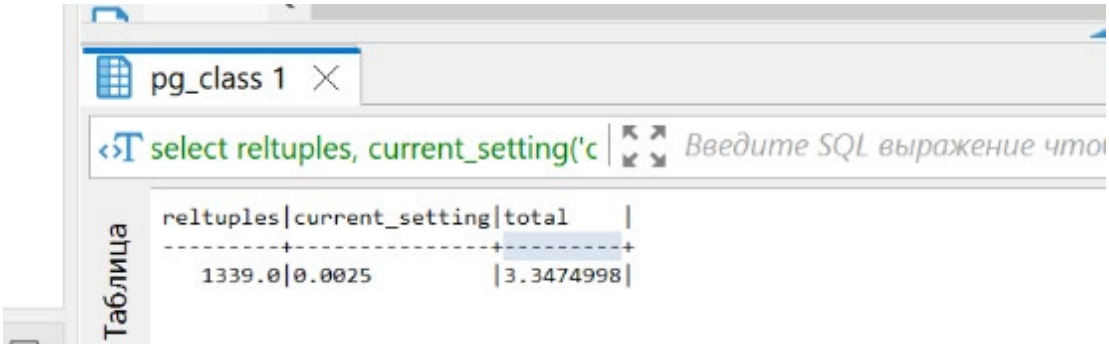


Рисунок 6 –

Параллельные планы

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 ;
```

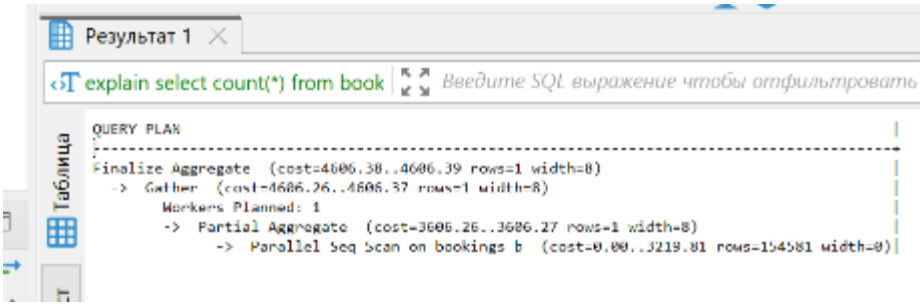


Рисунок 9

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';
```

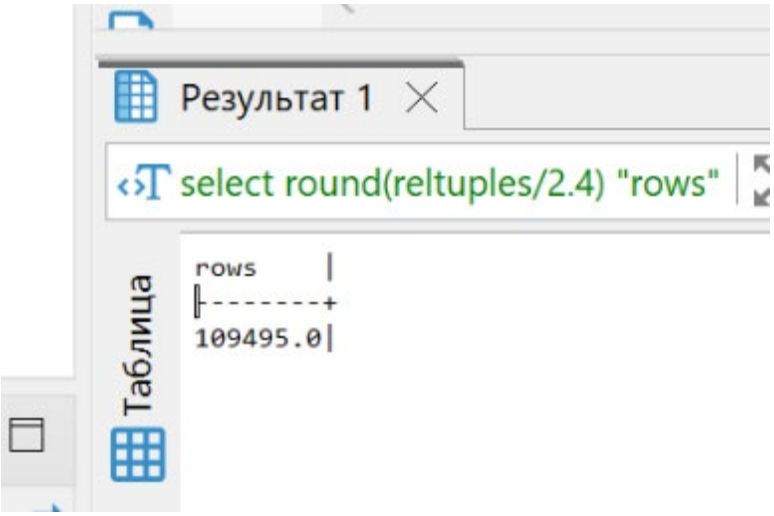


Рисунок 10 –

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

```
show parallel_leader_participation;
```

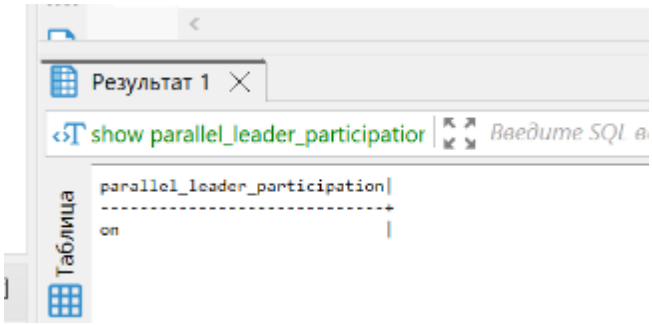


Рисунок 11

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

```
set parallel_leader_participation=off;
```

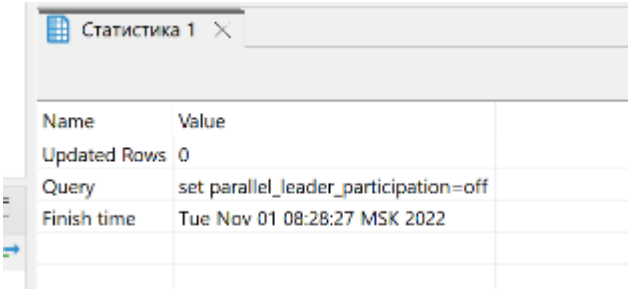


Рисунок 12

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

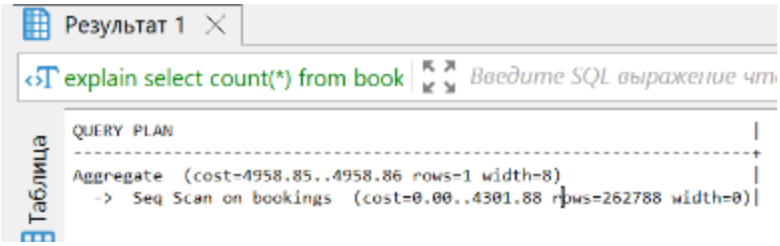


Рисунок 13

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

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

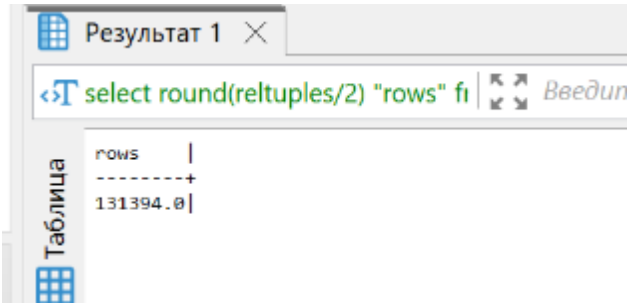


Рисунок 14

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

```
reset parallel_leader_participation;
```

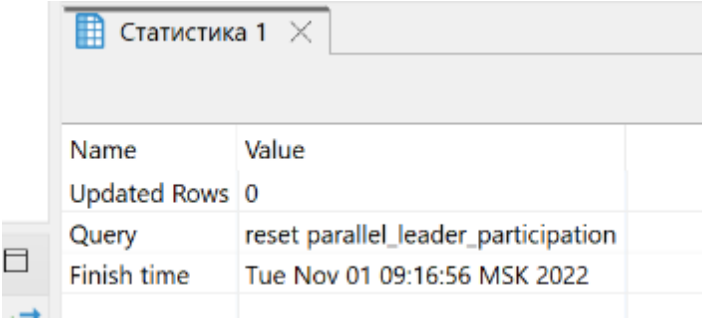


Рисунок 15

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.

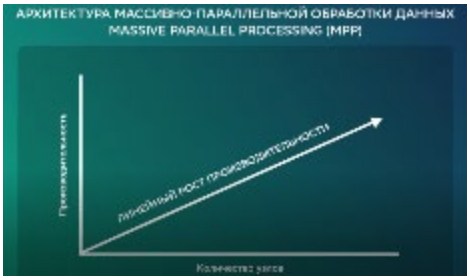
Системы обладают рядом преимуществ, таких как высокая скорость обмена данными между процессорами за счет наличия общей памяти, простота, универсальность обслуживания и относительно невысокая цена.

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.

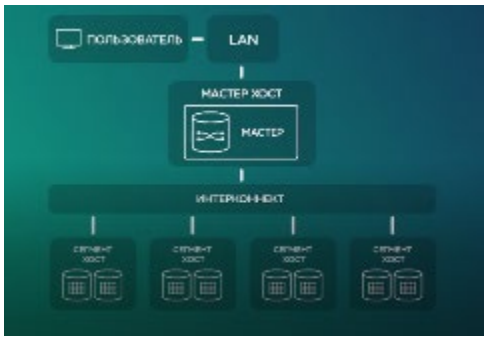
Как правило, данная операция является дорогостоящей, а в некоторых случаях и вовсе невозможно из-за физических ограничений в конфигурации сервера. Для решения проблемы масштабируемости в системах аналитической обработки данных стали применять архитектуру MPP или архитектуру массивно-параллельной обработки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

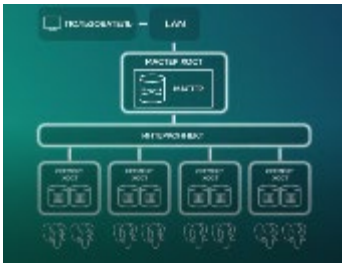
В MPP системах вычислительная мощность и объем хранения данных увеличиваются за счет добавления в систему дополнительных вычислительных узлов. Данный подход позволяет достигнуть линейный рост производительности в зависимости от количества узлов в системе.



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 PostgresSQL. 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.

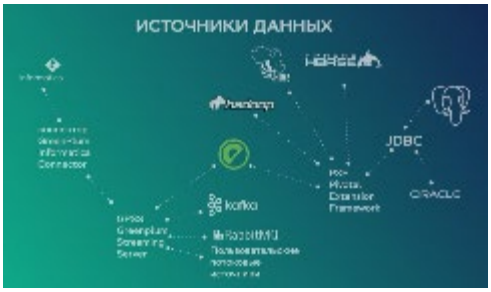
Позже, в 2015 году компания, разработчик данной СУБД Pivotal, выпустила исходный код ядра GreenPlum под свободные лицензии и выложила его на публичный репозиторий GitHub.

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.

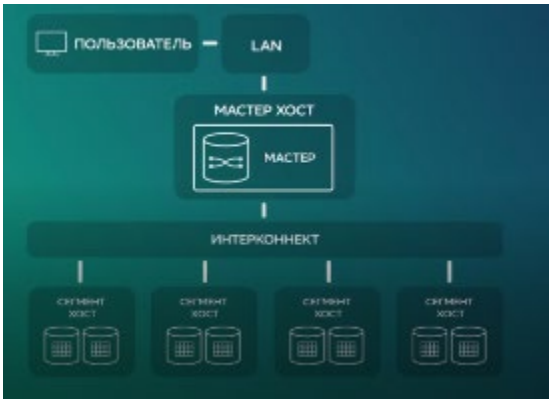
GreenPlum предоставляет широкий выбор инструментов для решения аналитических задач. Например, поддержка различных сложных типов данных, библиотек для задач Data Science, статистики, поддерживает географические и геометрические типы данных, а также текстовые данные в виде документов.

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.

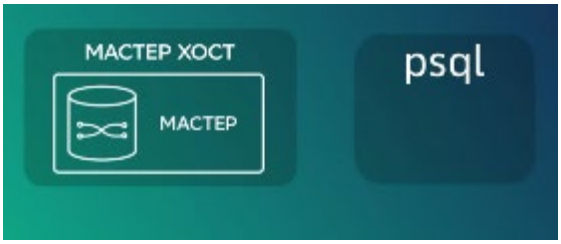


Также в GreenPlum можно разрабатывать процедуры не только на встроенным процедурным языке pl/pgSQL, но и на многих других популярных языках программирования, которые будут компилироваться и выполняться внутри базы данных. Также есть возможность разрабатывать процедуры, которые будут работать в контейнерах.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

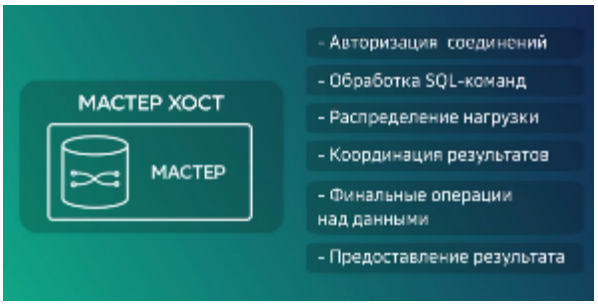
СУБД Greenplum реализована на массивной параллельной архитектуре без совместного использования ресурсов.



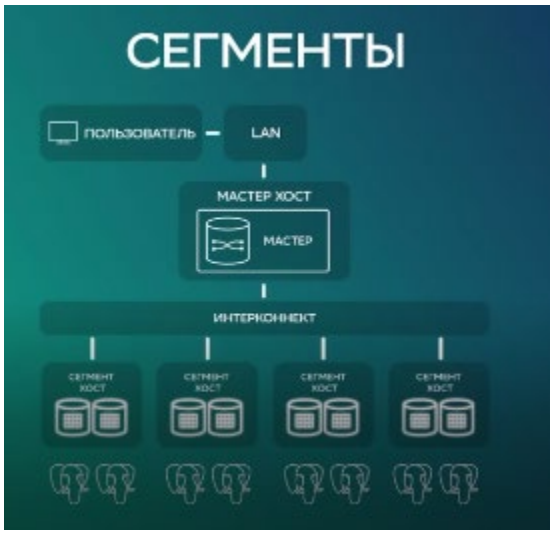
Также мы узнали, что Greenplum представляет из себя множество модифицированных баз данных PostgreSQL, которые расположены на нескольких серверах и взаимодействуют друг с другом, образуя единую цельную базу данных. Логически архитектур Greenplum выглядит следующим образом: мастер. Мастер — это входная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



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

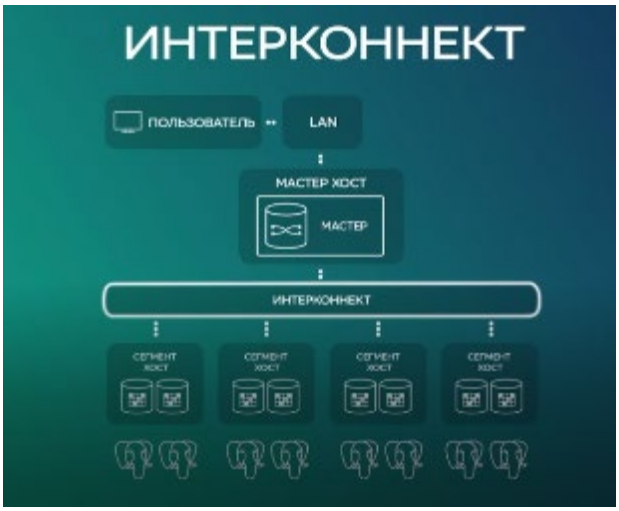


Сегменты - это отдельные экземпляры базы данных Postgres, которые хранят определенную порцию пользовательских данных и выполняют большую часть обработки запросов.



Когда пользователь подключается к базе данных через мастера и запускает запрос, на каждом сегменте создаются процессы для обработки этого запроса, затем каждый сегмент параллельно обрабатывает свою порцию данных и возвращает финальный результат своей работы на мастер. Сегменты работают на серверах, которые называются сегмент хосты или ноды, на каждом ноде обычно располагается от 2 до 8 сегментов Greenplum. Количество сегментов фигурирует при первоначальной установке системы и напрямую зависит от профиля рабочих нагрузок системы и технических характеристик сервера, таких как количество и производительность ядер процессора, объема оперативной и постоянной памяти и сетевых интерфейсов. Предполагается, что все ноды системы будут настроены идентично. Ключом к достижению максимальной производительности от базы данных Greenplum является равномерное распределение данных и рабочих нагрузок по большому количеству сегментов с одинаковой производительностью. Это необходимо для того, чтобы все сегменты одновременно начинали работать над задачей и одновременно завершали свою работу. В этом случае ни один из узлов не станет узким горлышко всей системы.

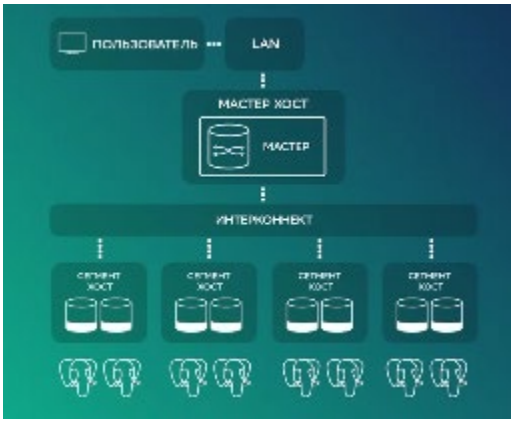
Интерконнект - это сетевой слой архитектуры Greenplum и интерконнект обеспечивает взаимосвязь между мастером, сегментами и сетевой инфраструктурой, в которой развернут кластер.



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

2. Дистрибьюция

Одним из важнейших факторов 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.

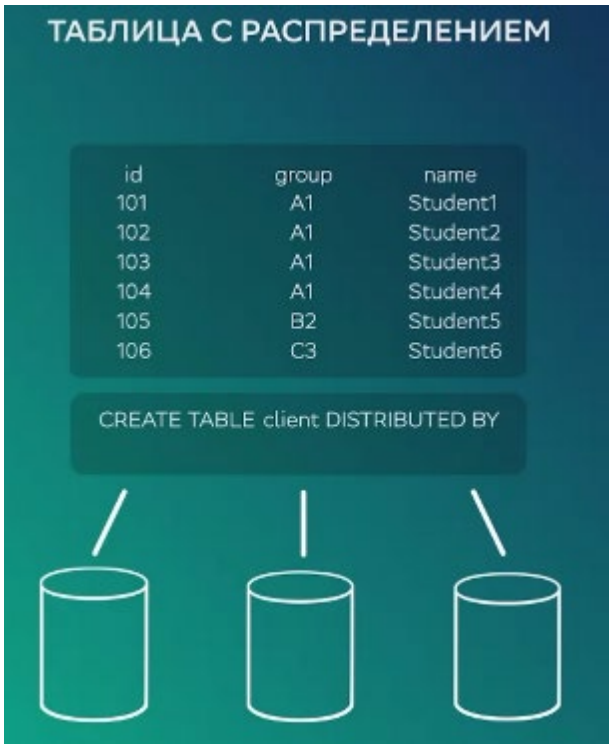


Если данные распределены неравномерно, сегменты с большим количеством данных будут выполнять работу дольше, что снижает общую

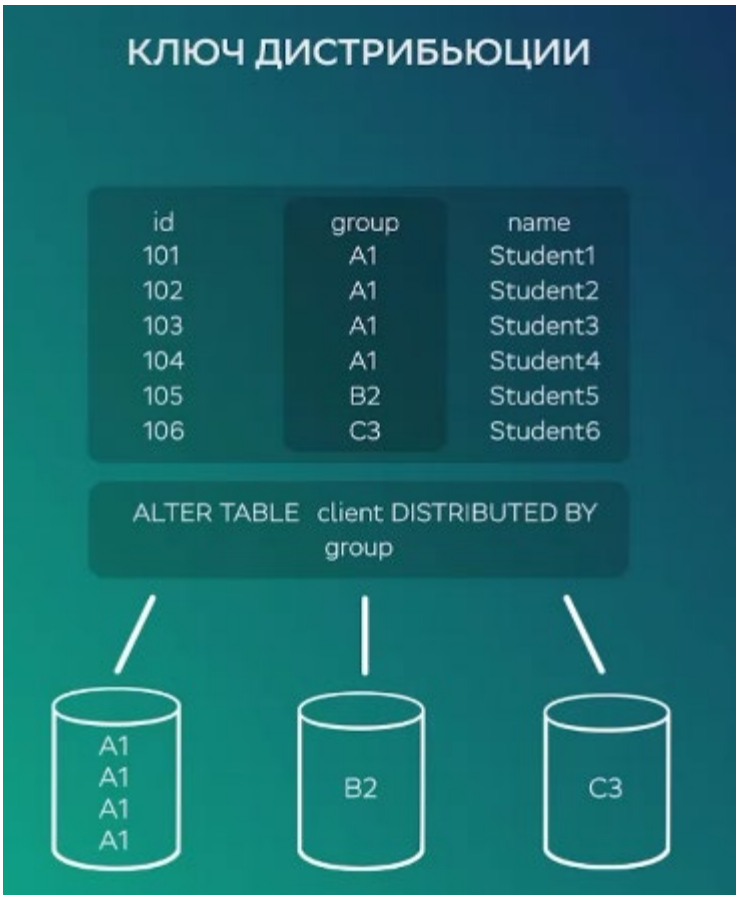
производительность. Таким образом, для повышения производительности необходимо стремиться к равномерному распределению данных по всем сегментам системы.

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

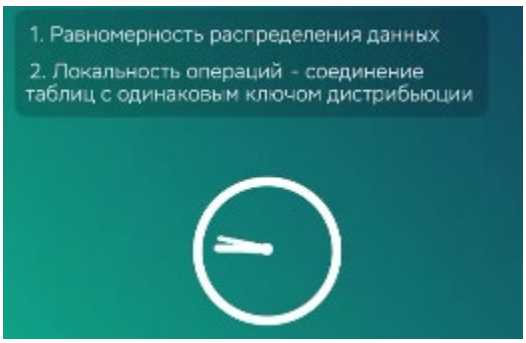
Даже одна таблица Greenplum хранится не на одном, а на нескольких сегментах. Каждый сегмент хранит уникальную порцию данных. Для каждой таблицы задаётся своя политика дистрибьюции. Способ, согласно которому строки распределяются по сегментам, он задаётся при создании таблицы в команде CREATE TABLE, а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



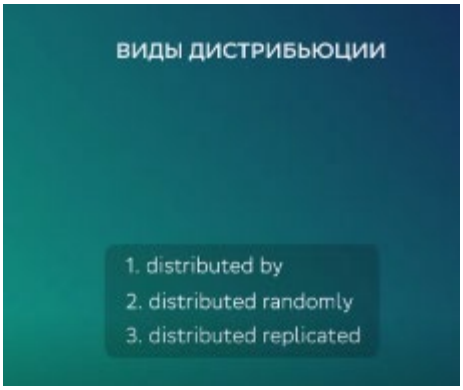
Ключ дистрибьюции это поле или набор полей, по значениям которых определяется, на каком сегменте будет храниться существующая строка таблицы.



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

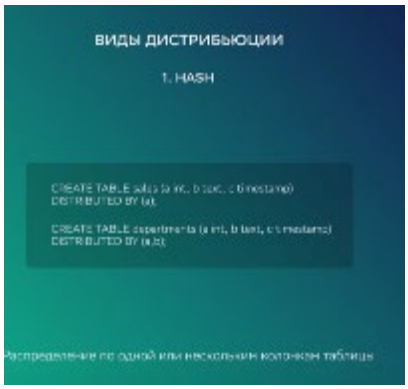


В Greenplum существует три вида дистрибьюции. Это distributed by, distributed randomly, distributed replicated.



Давайте рассмотрим каждый из них более подробно.

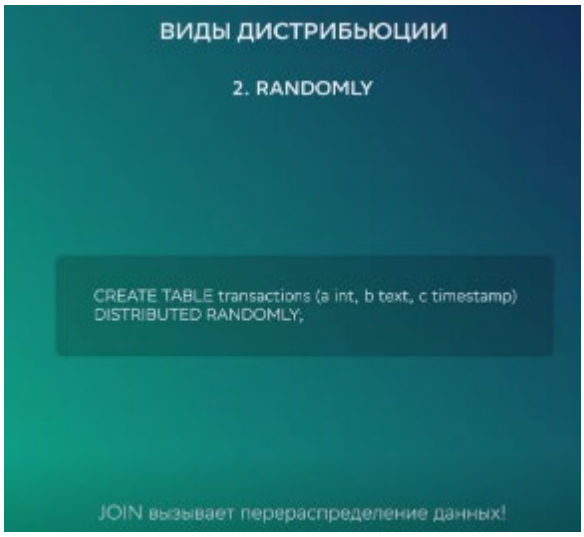
Distributed by - Это дистрибуция по HASH указанных полей. В этом случае и значение колонки или нескольких колонок вычисляется хэш-значение, по которому в дальнейшем определяется, на какой сегмент попадает запись.



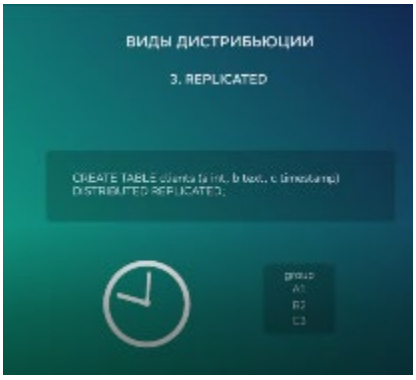
В приведенных примерах вычисления хэш-значения осуществляется по значению одной или двух колонок.

Distributed randomly - Это метод, при котором планировщики сам равномерно размещает данные по всем сегментам. Данный метод лучше всего подходит для таблиц, в которых отсутствует или сложно определить ключ,

который обеспечит равномерное распределение данных. Важно учесть, что при таком алгоритме соединение с другими таблицами будет всегда вызывать перераспределение данных, так как строки распределены случайным образом.



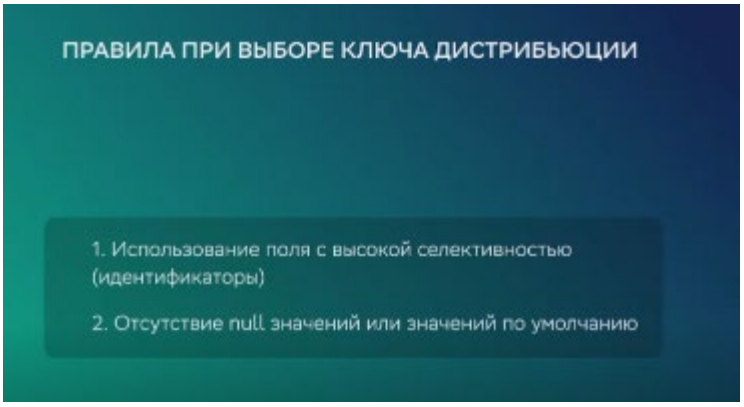
Distributed replicated - В этом случае на каждом сегменте хранится полная копия таблицы. Такой метод подходит для небольших таблиц, например, справочников, которые регулярно участвуют в соединениях с другими таблицами.



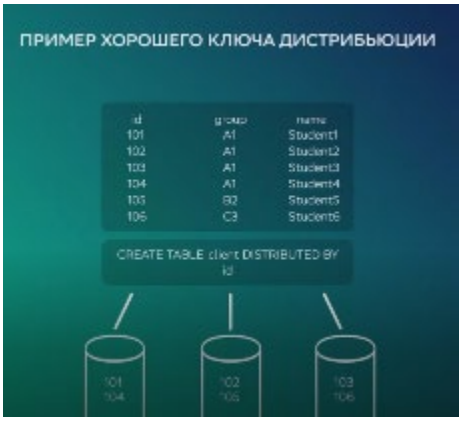
Хранения полной копии таблицы на каждом сегменте позволяет сэкономить время, которое в противном случае было бы потрачено на перераспределение во время выполнения запроса.

При выборе ключа дистрибьюции необходимо руководствоваться следующими правилами: в качестве ключа дистрибьюции необходимо выбирать полис с большой селективностью, то есть поля с большим количеством уникальных значений. Как правило, под этот критерий подходят поля с идентификаторами. Если данные распределены по ключу с низкой

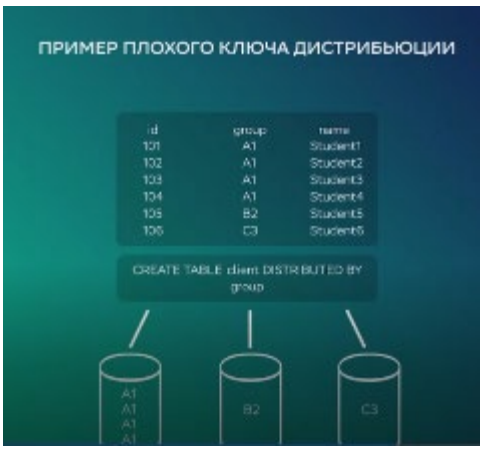
селективностью, то велика вероятность неравномерного распределения данных, так как одинаковое значение будут располагаться на одном сегменте. В ключах не должно быть назначений или значений по умолчанию. Иначе все NULL значения или другие значения по умолчанию будут попадать на один сегмент, что приведет к перекосу в данных.



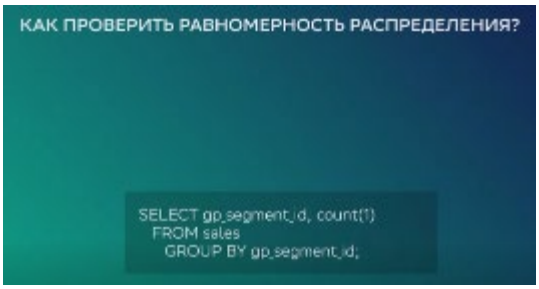
Рассмотрим примеры. В данном примере в качестве ключа дистрибьюции выбрано поле ID. В результате распределение на каждый сегмент падает ровно по две строки. Распределение получается равномерное, а значит, ключ дистрибьюции хороший.



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



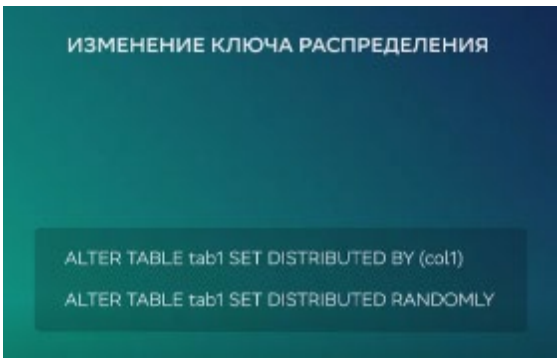
Делаем вывод о том, что ключ выбран не оптимально проверить равномерное распределение строк между сегментами можно, используя служебный столбец gp_segment_id, который присутствует в каждой таблице.



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

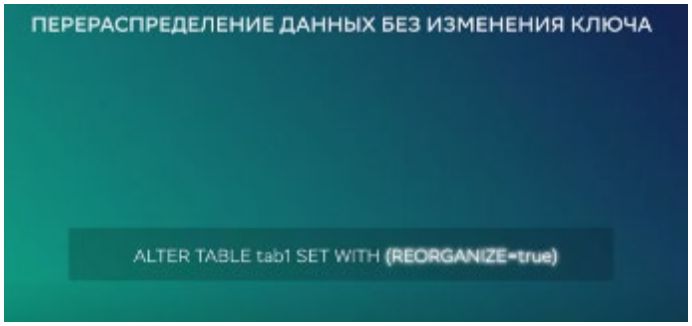
По результату запроса, видно, что распределение данных таблицы сейчас практически равномерное.

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

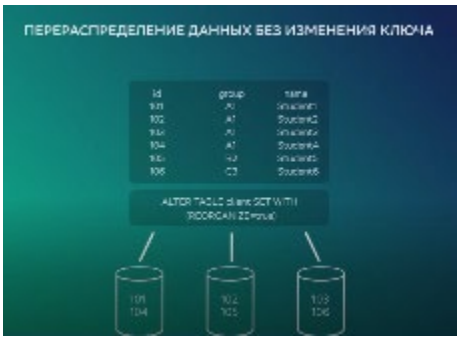


Примеры использования приведены на экране.

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

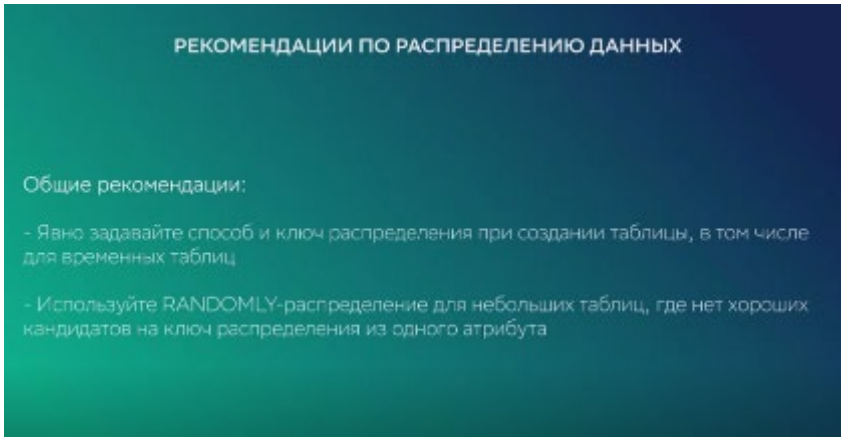


Эта опция будет полезна в двух случаях: при добавлении нового сегмента и для избежания раздутая таблиц. Про раздутие таблицы мы поговорим позже. А при добавлении нового сегмента в кластер возникает необходимость заново перераспределить данные, чтобы избежать перекоса в распределении данных, когда новый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

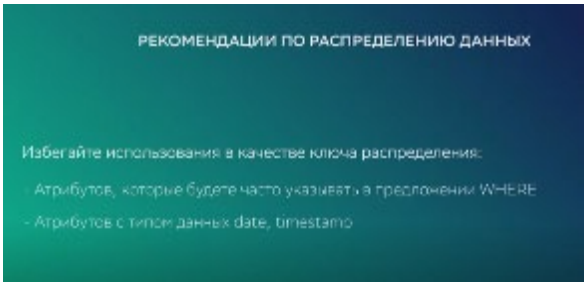


Вот некоторые рекомендации, которые помогут вам правильно распределять данные: явно задавайте способ и ключ распределения при создании таблицы, в том числе для временных таблиц; для небольших таблиц,

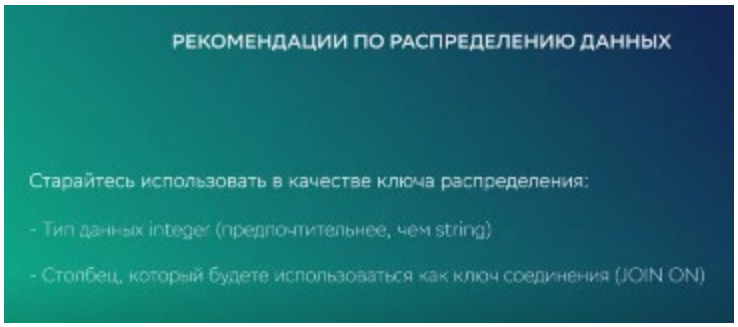
где нет хороших кандидатов на ключи распределения из одного атрибута используйте RANDOMLY для распределения.



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

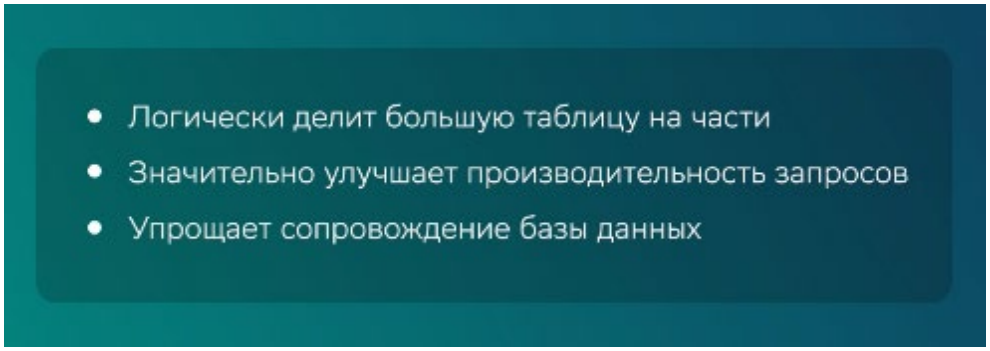


В качестве ключа распределения, предпочтительно использовать столбцы с типом данных Integer, а также столбцы, которые используются в качестве ключа соединения JOIN ON.



Ещё одна особенность greenplum, улучшающая производительность. Это партиционирование. Партиционирование или секционирование представляет собой логическое разделение таблицы на части по определенному критерию, применение партиционирование на больших

таблицах значительно улучшает производительность запросов, а также упрощает сопровождение базы данных благодаря возможности разделять, перемещать и компактно хранить данные в зависимости от их востребованности.



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

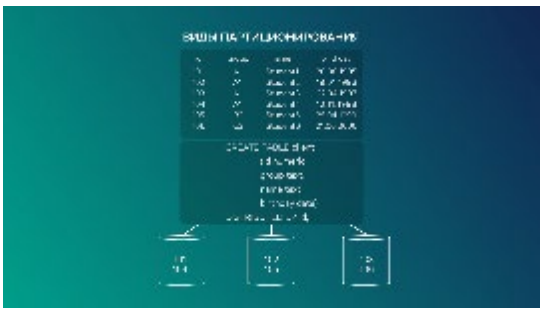
Дистрибьюция - это распределения данных по сегментам с целью равномерно распределить нагрузку по всем узлам кластера.

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

Партиционирование распространяется на все сегменты и выполняется на каждом сегменте одинаково. Например, если таблица при создании поделено 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 распределенными и имеют ключ дистрибьюции.

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

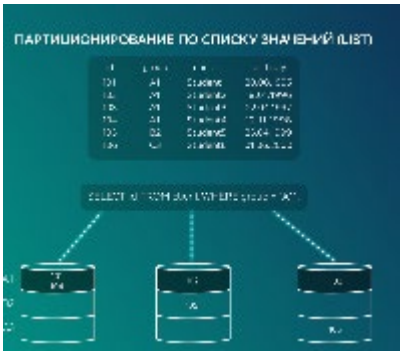
1) по диапазону значений RANGE;
2) по списку значений;
3) лист;
4) многоуровневая партиционирование, которое представляет из себя комбинацию, предыдущих типов.



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 RANCH позиционирование по году рождения. Вид позиционирование задается при создании таблицы с помощью выражения PARTITION BY RANCH.

Диапазон значений задан выражением старт и and, а шаг каждой партиции выражением EVERY INTERVAL интервал после создания таблицы на каждом сегменте будут созданы по шесть партиций, 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.



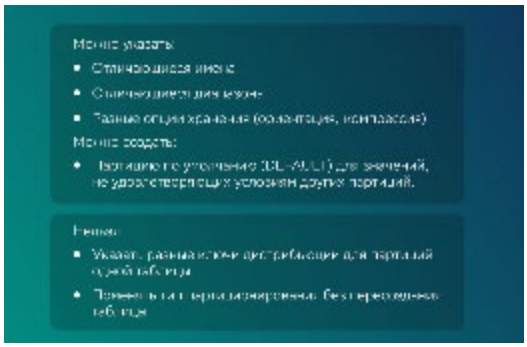
Например, мы можем применить к той же таблице CLIENT партиционирование по полю группа студентов. Разделение партиций по значениям выполняется с помощью выражения PARTITIONAL VALUE. На каждом сегменте создается по три партиции акции для групп А1, Б-2 и С3.

Внутри каждой партиции есть студенты с разными годами рождения. Например, запись о студенте четыре хранится в сегменте один в партици А-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.

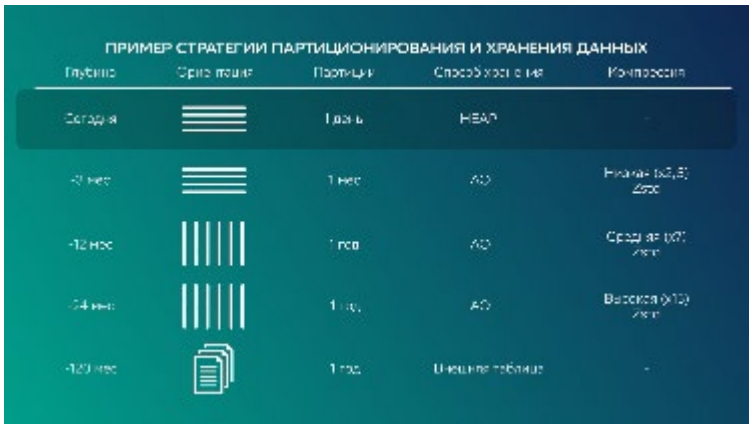
Имейте в виду, что использование многоуровневого партиционирования может привести к созданию избыточного количества партиций, что, в свою очередь, может замедлить работу всей базы данных.

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.

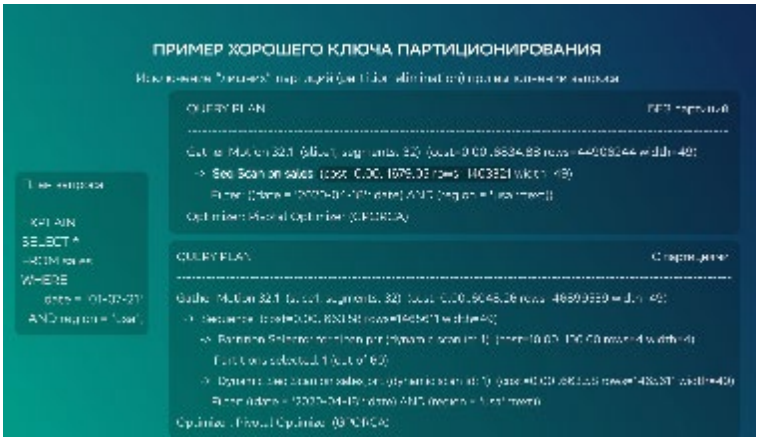


Рассмотрим пример стратегией партиционирования с применением разных опций хранения данных.



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 EXPLAY. Подробное описание запроса будет сказано в заключительной части этого курса.

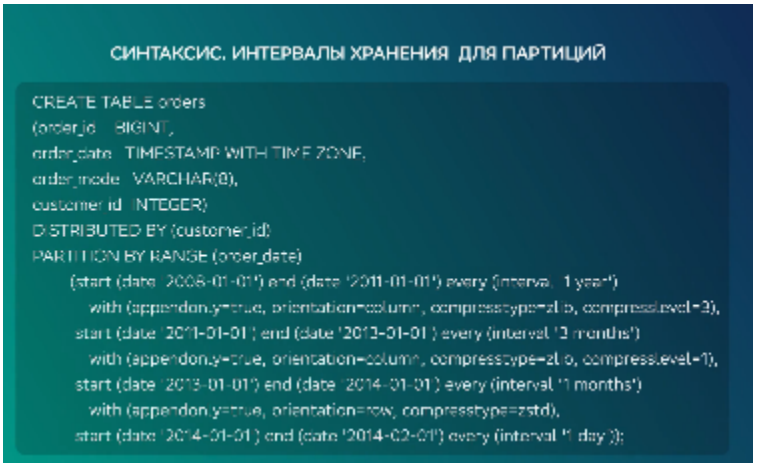
А сейчас давайте посмотрим на план выполнения запроса в случае таблицы с партиционированием или без него.



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.

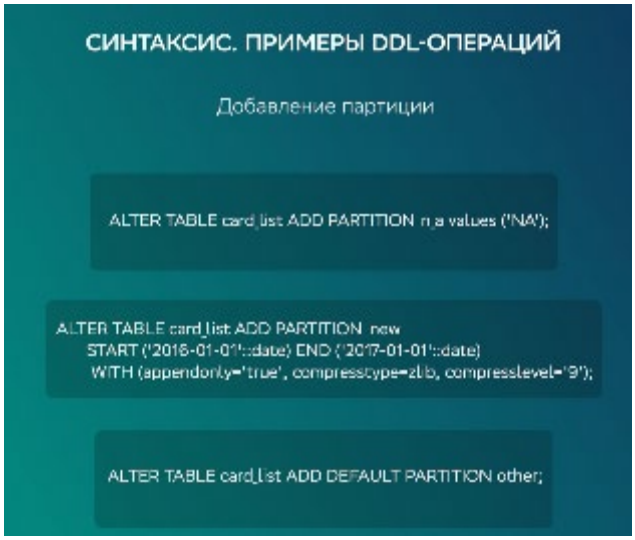
Рассмотрим примеры синтаксиса.

Для разных партиций можно указывать разные интервалы и опции хранения.



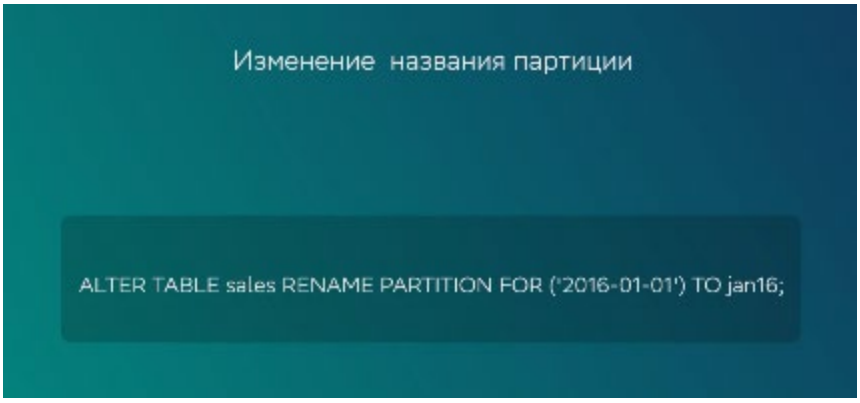
Для партиции с интервалом год или три месяца задана колоночная ориентация хранения и тип компрессии THE LEAP с разными коэффициентами сжатия. Для партиции с интервалом один месяц применяется ориентация по строкам и другой тип компрессии без явного указания коэффициента сжатия. Для партиции с интервалом 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 партици.

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 TRANKEP PARITION. На рисунке показано, как очистить партицию с названием EP 2019 или партицию, содержащую дату 28 апреля 2019 года.

Очистка партиции

```
ALTER TABLE table_prt_mixed TRUNCATE PARTITION year2019;
```

```
ALTER TABLE table_prt_mixed TRUNCATE PARTITION FOR ('2019-04-28'::date);
```

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

Удаление партиции

```
ALTER TABLE card_list DROP PARTITION n_a;
```

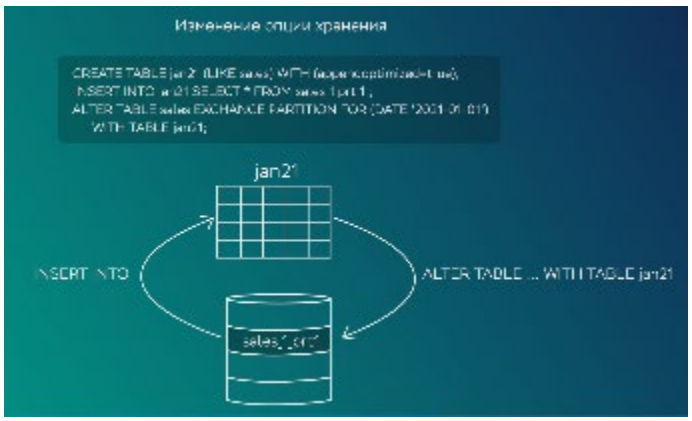
```
ALTER TABLE card_list DROP DEFAULT PARTITION;
```

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.

```
ALTER TABLE card_list DROP PARTITION n_a IF EXIST;
```

```
ALTER TABLE card_list DROP DEFAULT PARTITION IF EXIST;
```

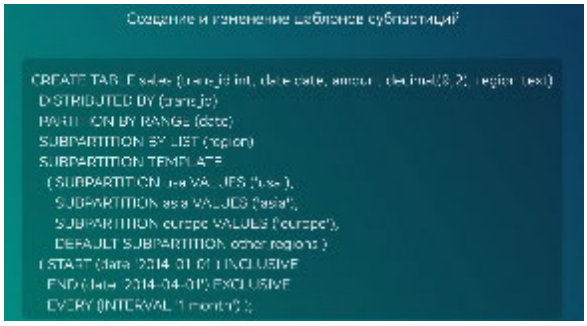
Для изменения опций и хранения в отдельные партиции можно создать таблицу с нужной опцией хранения, вставить в нее данные из этой партици и затем заменить всю прежнюю партию на только что созданную таблицу с помощью команды ALTER TABLE EXCHANGE PARTITION.



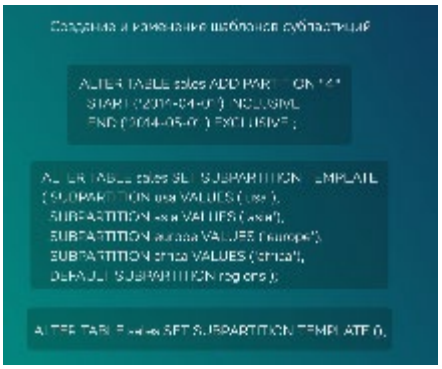
Для разделения одной партиции на две части используется команда ALTER TABLE SPLEET PARTITION.

В первом примере на рисунке выполняется разделение партиции, которые хранятся данные за январь на две партиц. 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 партиция, то добавить новую партицию в такой таблице можно только путем разделения дефолт партиции, как это показано во втором примере. В этом случае в предложении in to в качестве второй партиции необходимо указать дефолт партицию.

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

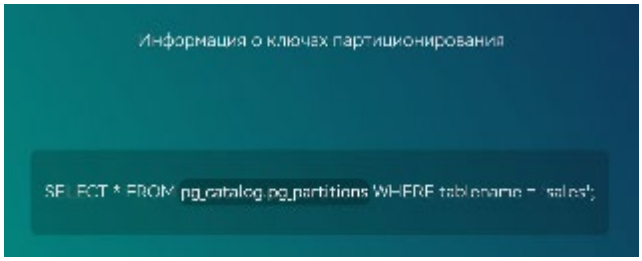


Например, в следующем скрипт создания шаблон определяется с помощью предложения SUP-PARTITION TEMPLATE, в котором мы определяем четыре суб-партиции для разных регионов. Теперь, если мы добавляем в таблицу партицию следующем с скприптом, то эта позиция по шаблону автоматически разделится на четыре суб-партиции 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 партиции из просмотра, или когда нужно обеспечить так называемое окно данных.

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

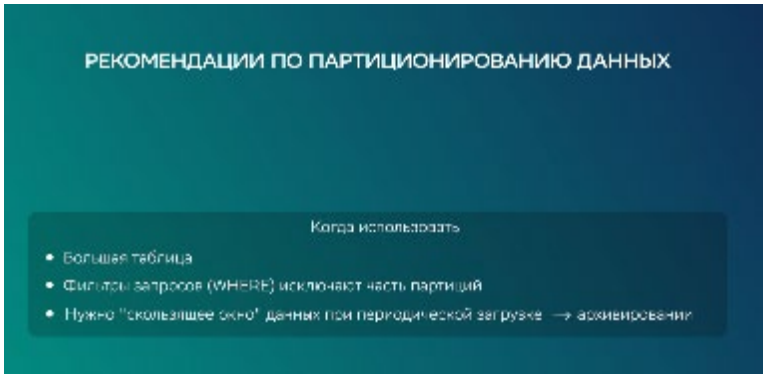
Старайтесь выбирать партиционирование по диапазону, а не по списку значений, выполняйте партиционирование на основе часто используемого столбца.

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

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

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

Избегайте создания большого количества партиций. Greenplum эффективно обрабатывает операции чтения FULL SKAN, что позволяет уменьшать количество партиций, при этом увеличивать объем данных в каждой партиции, не используйте дефолт партицию, так как она всегда просматривается при выполнении любого запроса.



Не используйте многоуровневое партиционирование, так как это обычно приводит к созданию чрезмерно большого количества партиций. Никогда не используйте партиции по столбцу, который является ключом дистрибьюции.

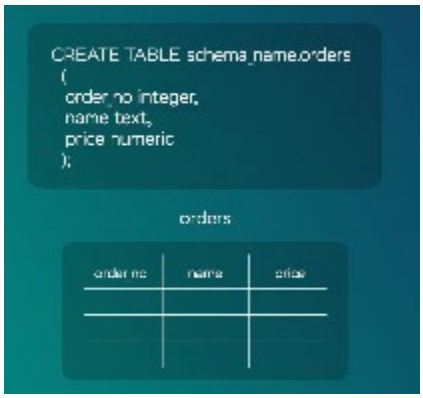
Не создавайте слишком много партиций при колоночной ориентации таблицы, так как это приведет к слишком большому количеству физических файлов на сегменте. Ведь количество файлов — это произведение количества сегментов, столбцов и партиций.

Лабораторная работа № 14. Работа с таблицами в GreenPlum.

Цель работы: освоить технологию работы с таблицами в GreenPlum

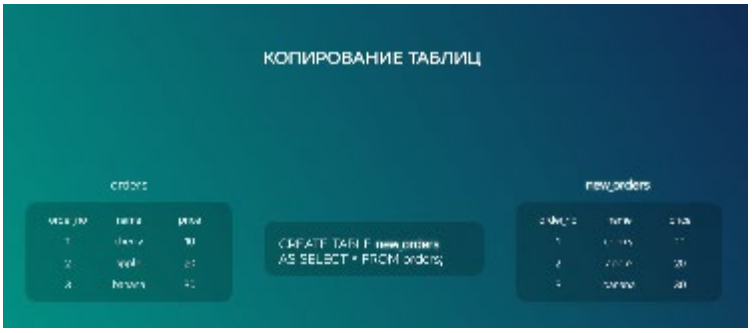
Рассмотрим, как создать таблицу, выясним какие методы хранения данных и способы ориентации для таблиц существуют, а также в каких случаях следует их использовать.

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.

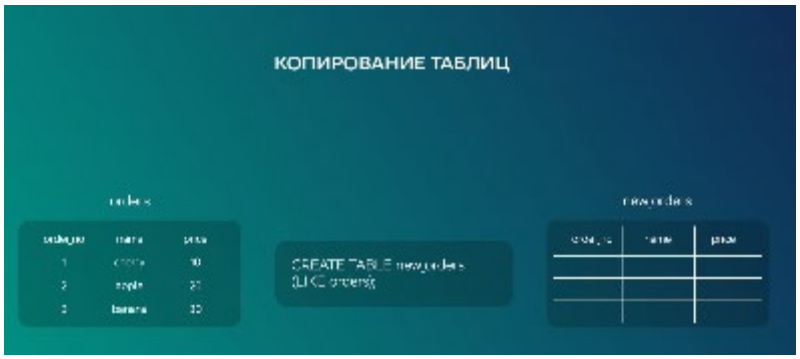


Таблицы по умолчанию создаются в схеме Public, при этом есть возможность явно задать схему создания, указав ее перед именем таблицы. Имя таблицы должно быть уникальным в текущей схеме не должно сосуществовать других таблиц, внутренних или внешних индексов или представлений с тем же именем. Предположим, 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.



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



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

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.

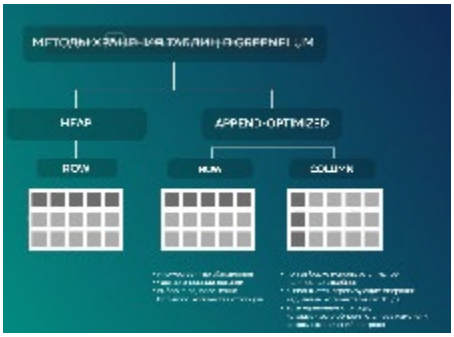
Второй тип это Appent Optimized таблицы, данный тип встречается только в GreenPlum, таблицы данного типа обладают особой структурой, являются компактными и обладают рядом специальных возможностей. Для Appent Optimized таблиц мы можем выбирать ориентацию строковую или колоночную, в отличии от Heap таблицы Appent Optimized таблицы сжимать, применяя различные алгоритмы компресии, также Appent Optimized таблицы менее ресурсоемкий в обслуживании, например, сбор статистики может быть выполнен инкрементально и таким образом быстрее, вместе с этим у Appent 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 данных 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 Appent Optimized?

Хранения типа Appent 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 Appent 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 Apent 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)
);
```

Heap-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);
```

AO-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)
```

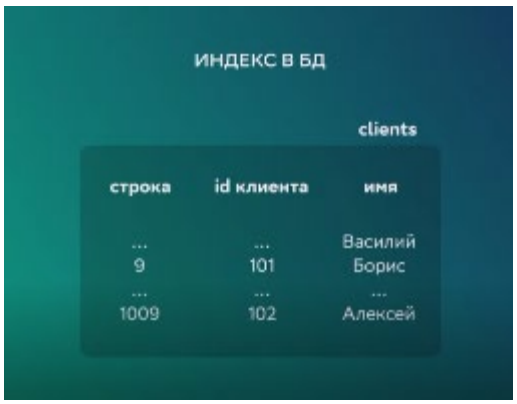
```
DISTRIBUTED BY (a);

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.

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 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.

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

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

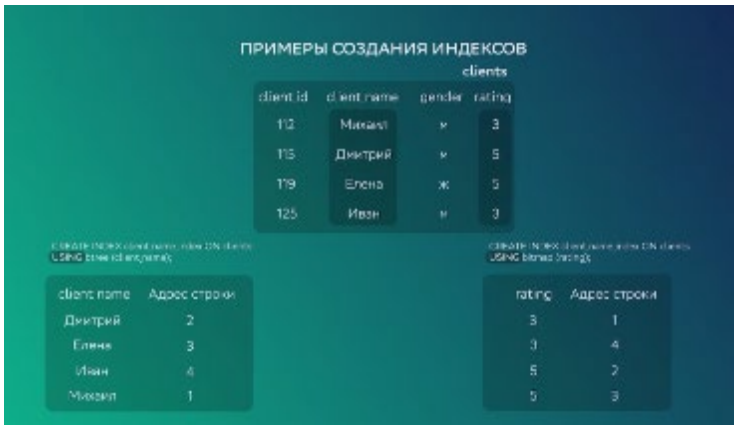
Индекс BTREE используется по умолчанию, его рекомендуется использовать, если к конденсированному полю применяются такие операции, как сравнение, больше, меньше, равно, between и is null, is not null, like.

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

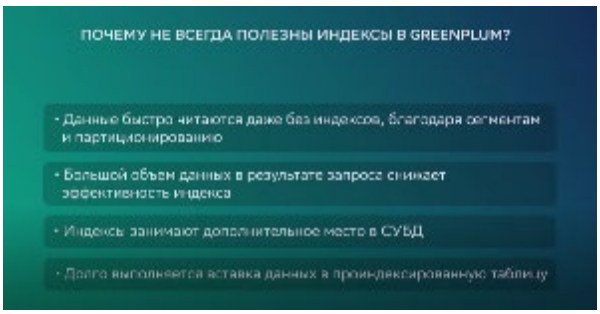
Индекс BITMAP в следующих случаях, когда колонка содержит относительно небольшое количество уникальных значений, обычно от 100 до 100000, и при этом много повторений. То есть имеет низкую кардинальность, когда чтение намного больше, чем update, то есть тип загрузки не OLTP, когда в запросах к таблице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

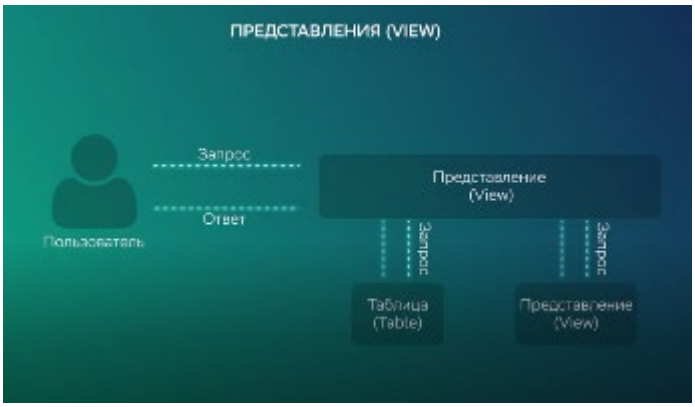


Используйте выражение Using, чтобы задать способ индексирования и поле для индексации. В Greenplum индексы нужно использовать с осторожностью по нескольким причинам: Во-первых, Greenplum умеет быстро считывать данные в связи с тем, что данные распределены по сегментам, и каждый сегмент считывает только 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.

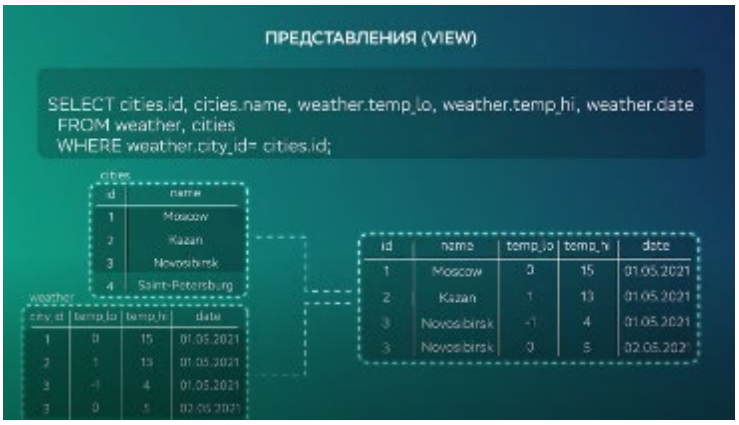


Придерживайтесь следующих правил при работе с индексами в Greenplum. В первую очередь используйте другие способы оптимизации, применения индексов для оптимизации в хранилищах данных необходимо выполнять в последнюю очередь, индексы подходят для близкой OLTP- нагрузки, частый выбор небольшого числа строк 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 загрузки 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.

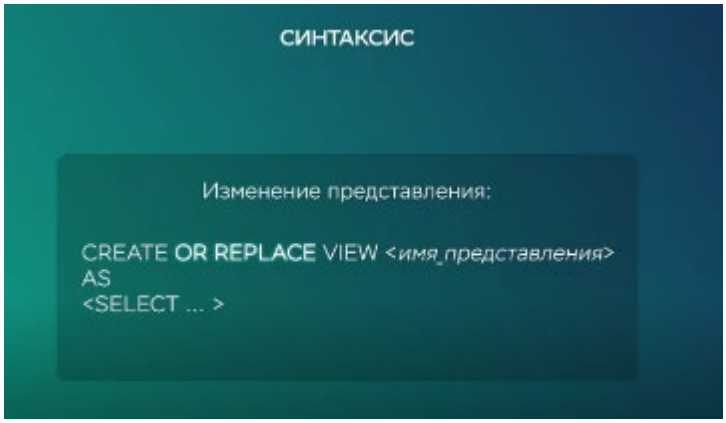


VIEW - Это объект базы данных, который, по сути, представляет из себя сохраненный запрос, в момент обращения к которому пользователю возвращается результат данного запроса. Представление хранят не сами данные, а только ссылки на них. Представление может ссылаться на одну или несколько таблиц, либо на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



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.

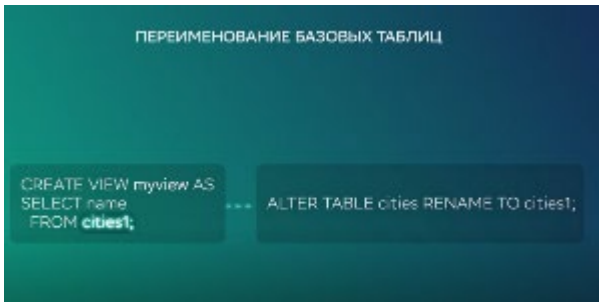
Рассмотрим синтаксис создания изменения и удаления представлений. Для создания представления предназначен оператор CREATE VIEW. Для изменения существующего представления необходимо добавить после оператора CREATE предложение OR REPLACE.



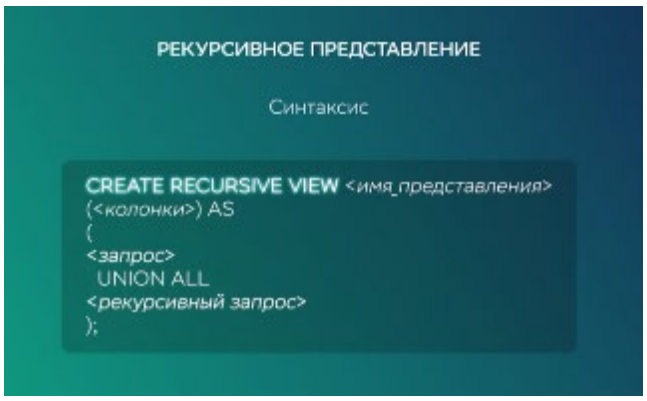
Для удаления представлении используется оператор DROP VIEW. Следует учесть, что в случаях, когда необходимо удалить базовые таблицы, то есть таблицы, участвующие в запросе представления, удаление необходимо выполнять с дополнительной опцией CASCADE.

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.

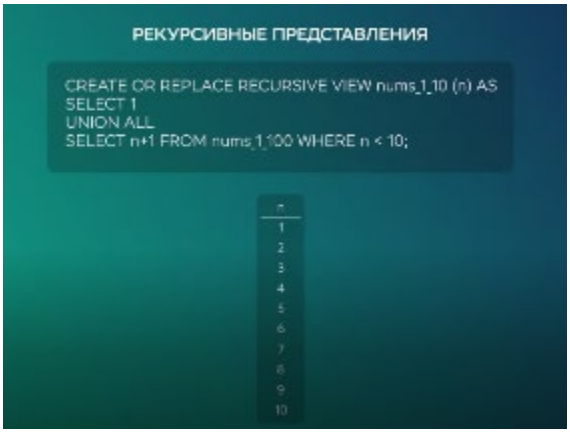


Рекурсивным называется представление, в тексте запроса которого используется ссылка на это же представление. Это способ вызова цикла прямо из SQL-выражения внутри определения представления. Обратите внимание на синтаксис определения рекурсивного представления на рисуноке.



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.

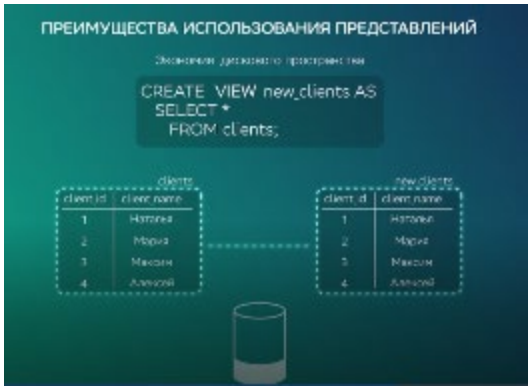
Простой пример репрессивного запроса вы видите на рисунке.



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 условии 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.

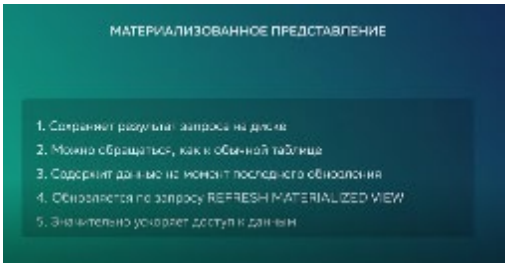
С помощью представления можно скрыть сложность запросов и сложность структур базовых таблиц. Например, можно скрыть всевозможные вычисления соединения, а также исключить избыточные поля из внутреннего определения представления. Эта особенность добавляет модульность и лаконичность при дальнейшем использовании 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 дисковые пространство. Они не хранят каких-либо данных, а извлекают их по запросу из базовых таблиц.

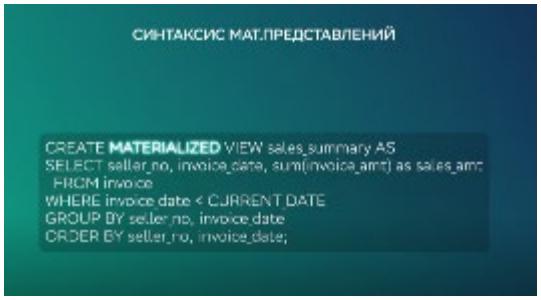


Теперь давайте рассмотрим такой вид представлений, как материализованные представления. Главной особенностью

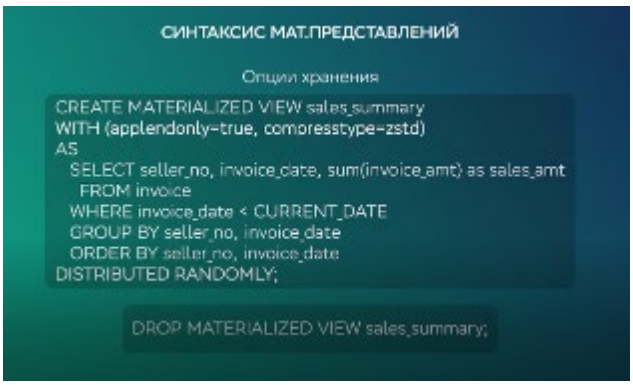
материализованных представлений является то, что в отличие от обычных представлений, они сохраняют результат запроса на диске и в дальнейшем позволяют обращаться к ним как к обычной таблице. При этом данные актуальны на момент последнего обновления материализованного представления. Данные в мат представлении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



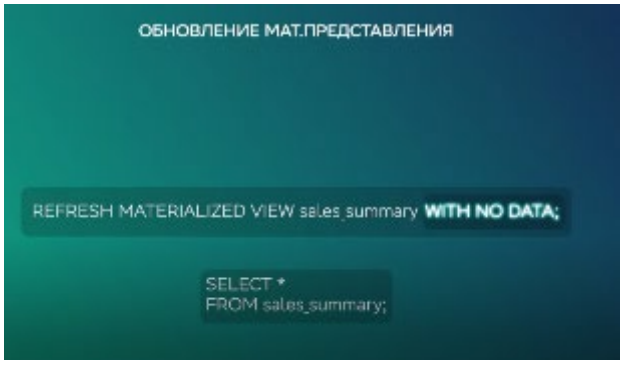
Давайте рассмотрим синтаксис создания мат представлений. Как видно на примере, в отличие от обычных представлений, после слова CREATE добавляется слово MATERIALIZED.



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 OPTIMAZED 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.

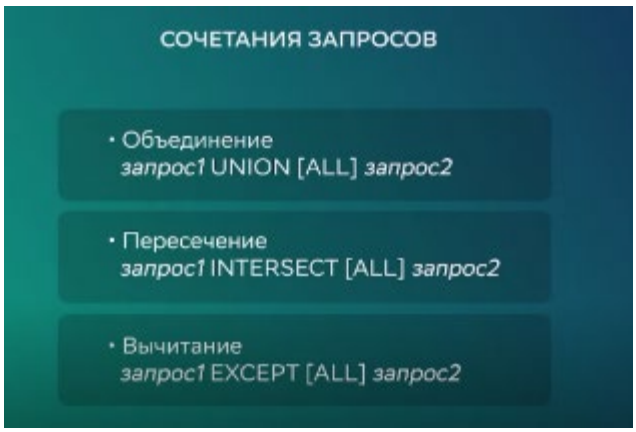


Способ с опцией WITH NO DATA удалит данные в мат представлении и приведет его в состояние UNSCANNABLE запросы к нему будут возвращать ошибку до тех пор, пока представление не будет обновлено стандартным способом. Это дает возможность временно ограничить доступ пользователей к неактуальным данным.

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.

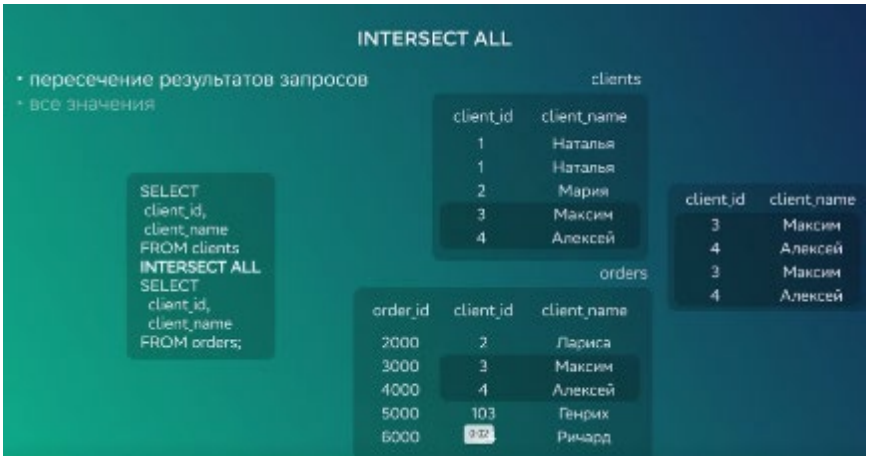


Общий синтаксис сочетания запросов показан на рисунке. Здесь запрос1 и запрос2 это запросы, в которых могут использоваться все возможности, рассмотренные до этого. При этом оба запроса должны содержать одинаковое количество столбца, а также соответствующие столбцы должны иметь одинаковые либо совместимые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

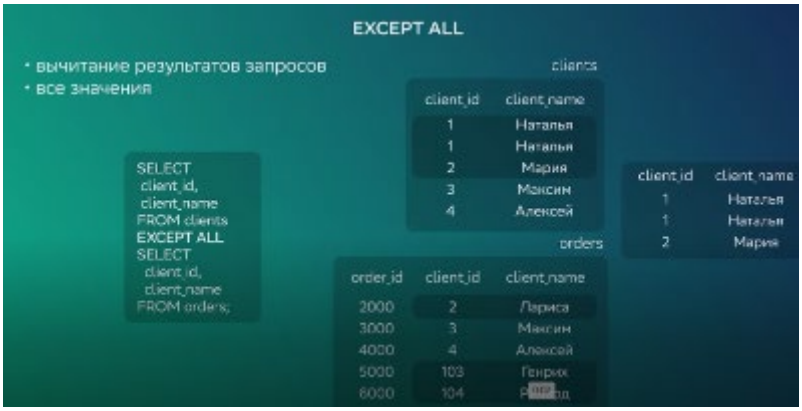
Давайте рассмотрим каждую операцию по отдельности.

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.

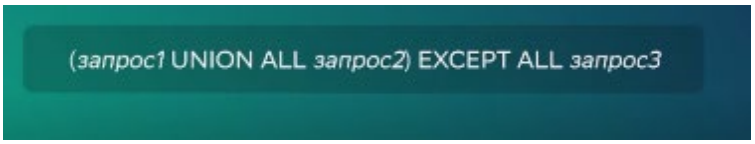


И, наконец, команда EXCEPT выполняет операцию вычитания результатов запросов. Приведенный пример команды возвращает все строки, которые есть в результате первого запроса, но отсутствуют в результате второго.

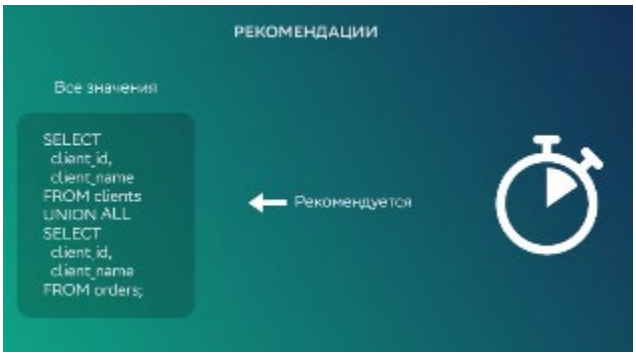


Здесь, как и в предыдущих командах EXCEPT с опцией ALL оставляет повторяющиеся строки, а без такой опции отфильтровает дубликаты. Синтаксически сочетание запросов используется так же, как обычный запрос Select.

Например, показанное на рисунке выражение позволяет удалить из таблицы CLIETNS 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.

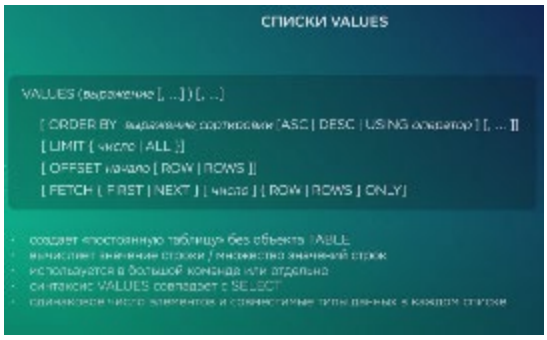


Поскольку поиск и удаление дублирующихся строк требует дополнительных вычислительных ресурсов базы данных, рекомендуется использовать сочетания с ключевым словом ALL, используйте выражения без ALL, только если требуется выводить уникальные значения.



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

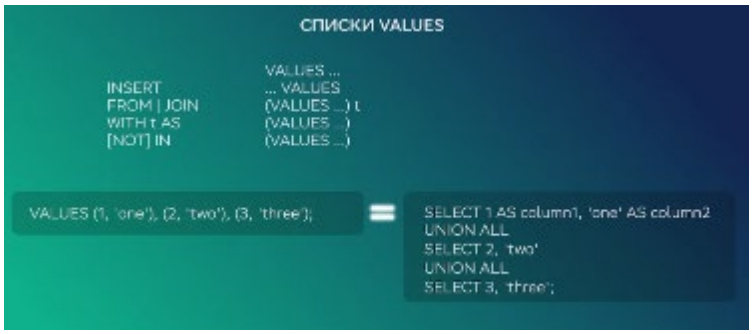
запросе, не создавая и не наполняя таблицу в базе данных. VALUES вычисляет значение строки или множество значений строк, заданное выражениями. Чаще всего эта команда используется для формирования таблицы констант в большой команде, но ее можно использовать и отдельно. В составе других команд синтаксис 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.



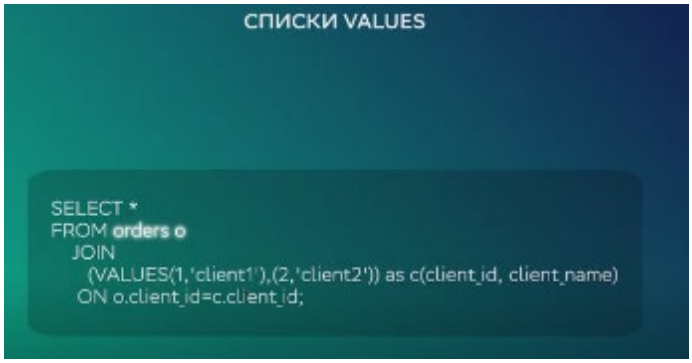
Например, следующие выражения Values, состоящие из трех элементов, вернет таблицу из двух столбца и трех строк. Такое выражение равносильно следующему запросу с применением конструкции UNION ALL. По умолчанию Greenplum назначает столбцам таблицы Values имена Column1, Column2 и так далее.

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.

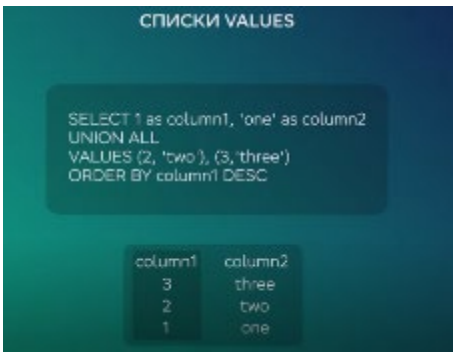
Синтаксически список VALUES с набором выражений равнозначен выражению SELECT FROM. Например, данное выражение VALUES эквивалентно следующему выражению SELECT.



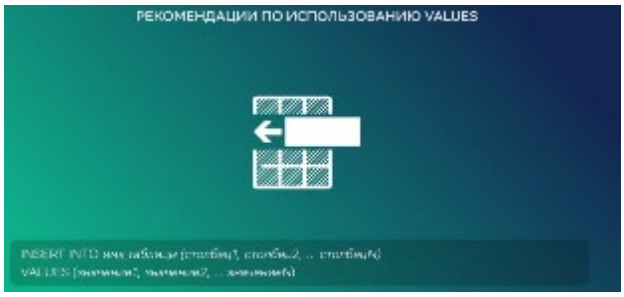
Поэтому применение списка VALUES допускается везде, где допустим SELECT. Например, вы можете использовать его в составе запроса, соединив с другой таблицей следующим образом. Здесь список клиентов, сформированный при помощи ключевого слова VALUES соединяется с таблицей Orders по полю ClientID.



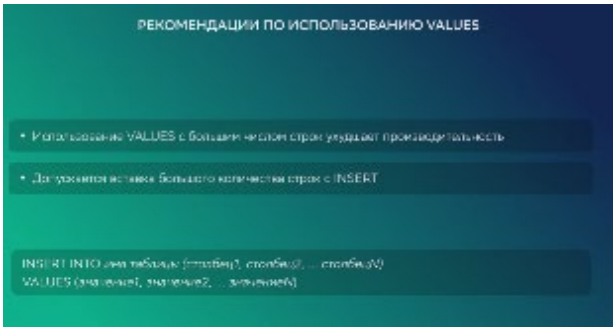
Либо, к примеру, вы можете использовать список VALUES в составе UNION, как в следующем запросе с применением сортировки ORDER BY.



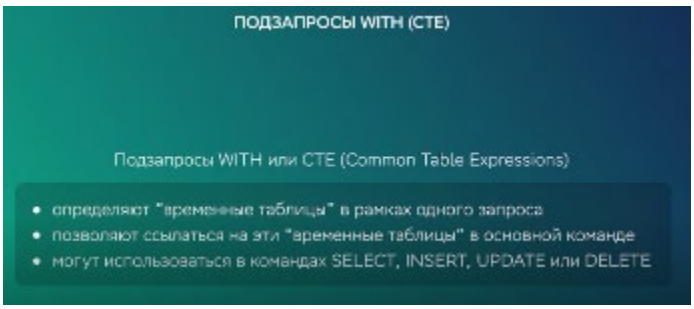
VALUES чаще всего используется как источник данных для команды INSERT, а также как подзапрос.



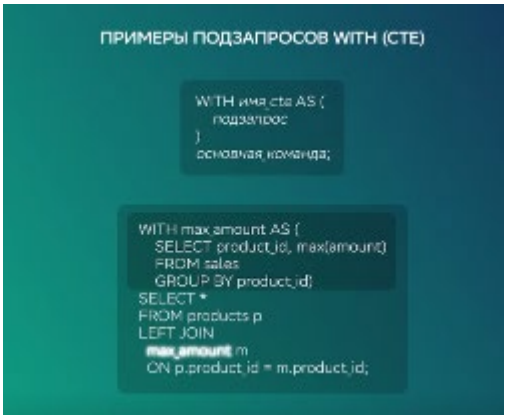
Следует избегать использования VALUES для операции со значительным количеством строк. Единственное исключение из этого правила команда INSERT, она позволяет вставить таким образом много строк, так как тип данных в колонках наследуется из уже существующей таблицы.



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 под запрос 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.

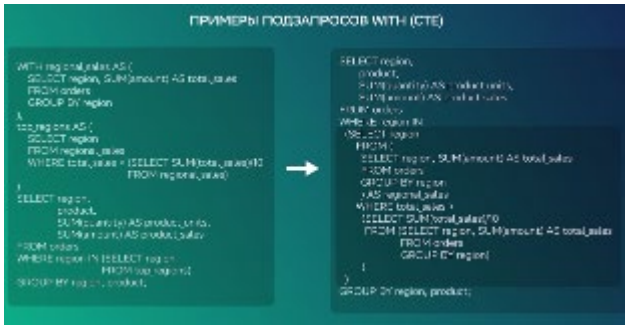
ПРИМЕРЫ ПОДЗАПРОСОВ WITH (CTE)

```
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 ссылаться на запросы CTE объявленные выше. Такой запрос можно переписать и без конструкции улиц 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.

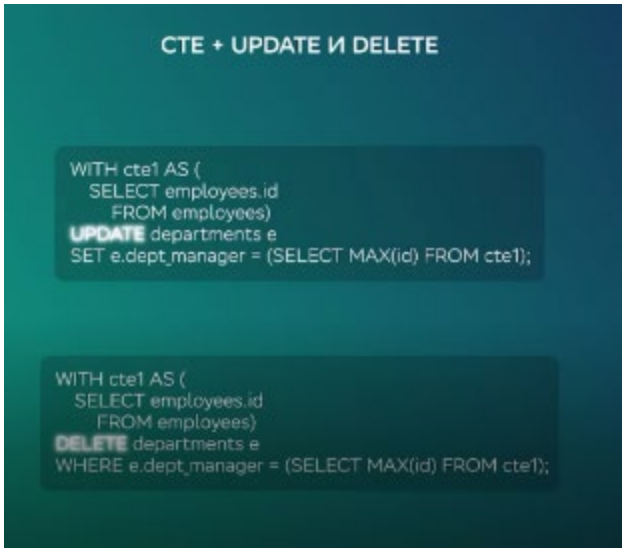


Например, в нашем запросе мы можем добавить еще одно соединение с под запросом TOP_REGIONS. В этом случае, несмотря на то, что TOP_REGIONS упоминается в основном запросе два раза, он будет вычислен только один раз для всего запроса. Таким образом, сложные вычисления,

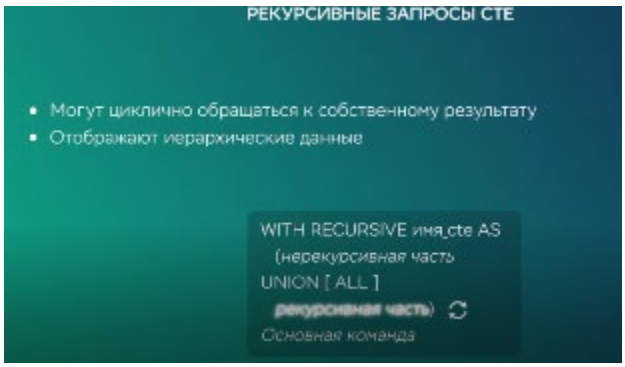
результаты которых нужны в нескольких местах, можно вносить в запросы WITH в целях оптимизации.

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

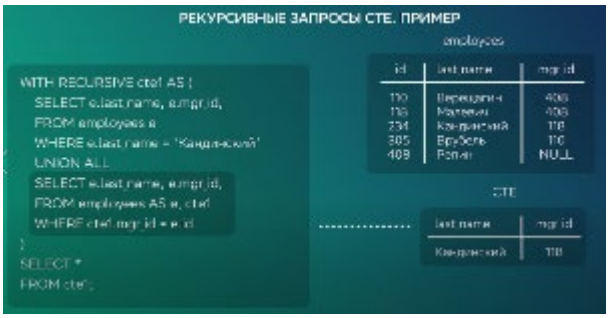
Аналогично CTE можно использовать в таких командах, как Update и Delete. Примеры конструкции вы видите на рисунке.



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.

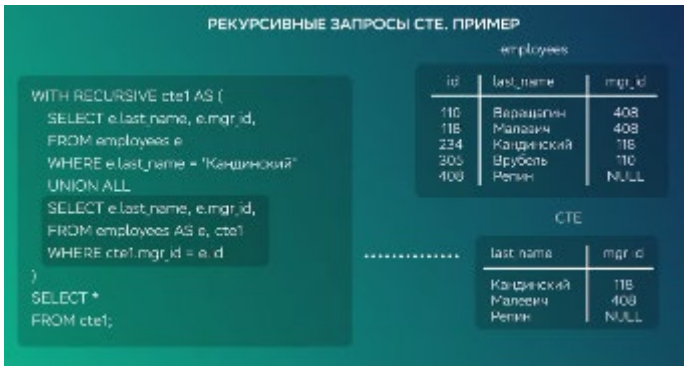


Приведем пример. Для начала рассмотрим структуру таблицы, которая будет использоваться в запросе. Это таблица сотрудников с колонками, идентификатор сотрудника, фамилия сотрудника и идентификатор менеджера сотрудника. Целью написания рекурсивного запроса будет вывести всех менеджеров Кандинского, то есть его менеджера, менеджера его менеджера и так далее. Перейдем теперь непосредственно к структуре рекурсивного CTE. Здесь после слова WITH, добавляется новая 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 Кандинским. 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.



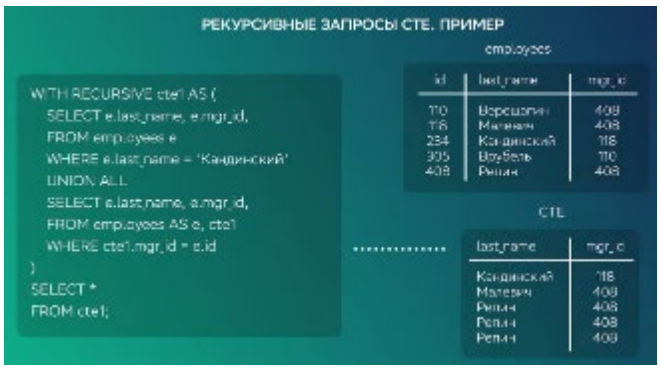
Итак, в рекурсивной части запроса будут найдены сотрудники, которые являются менеджерами сотрудников, полученных на предыдущих шагах рекурсии. В запросе это соединение между таблицей Employees и CTE1 через равенство Id MANAGER из CTE1 и ID сотрудника из Employees. Так на первом шаге рекурсии будет взята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

рекурсивной части (Информация О Кандинским) 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.

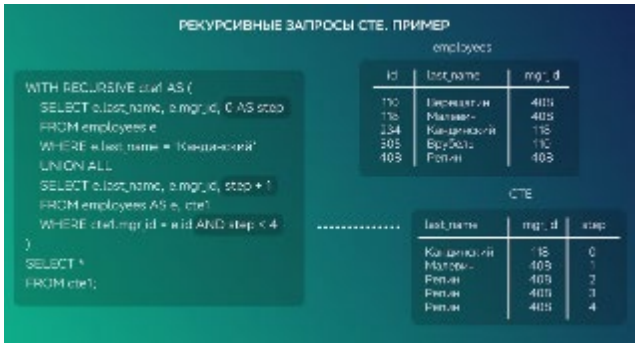


На последнем шаге запроса собирается информация из CTE, пользователю возвращается все содержимое CTE, то есть информация о Кандинским, его менеджере Малевиче и менеджере его менеджера Репине. Так происходит выполнение рекурсивных запросов в GreenPlum. Также важно отметить, что связь между рекурсивной 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 СТЕЮ, 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.



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.

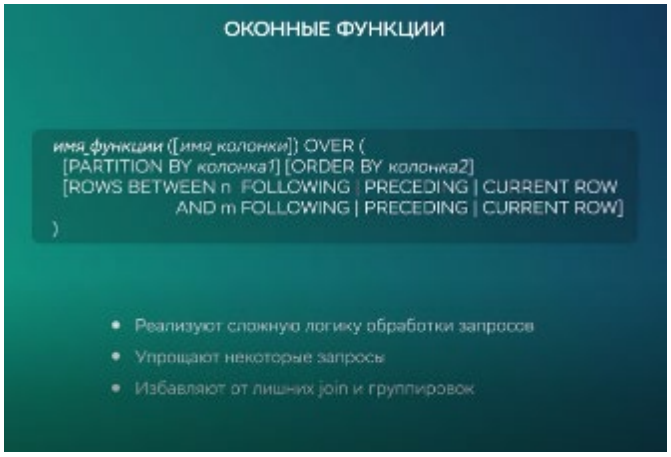


Подведем резюме. Под запрос CTE не хранится как объект базы данных и действует только в пределах основного запроса, CTE может ссылаться сам на себя, то есть быть рекурсивным, подзапросы удобно использовать в следующих случаях: для аккуратного оформления кода в большом запросе, для создания рекурсивных 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.

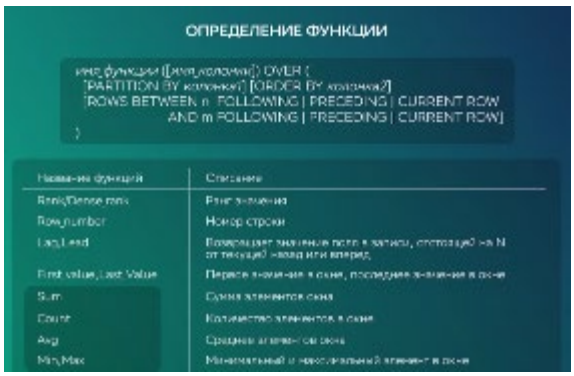
Лабораторная работа № 17. Оконные функции

Цель работы: познакомиться с технологией работы с оконными функциями

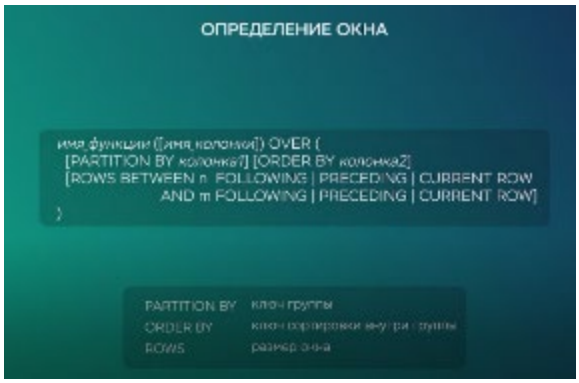
Оконные и аналитические функции - это расширение языка SQL. Они позволяют разбивать строки данных на группы или окна, упорядочивать строки внутри окна и применять к данным набор функций. Эти функции позволяют реализовывать внутри СУБД сложную логику обработки данных, упрощают некоторые запросы, позволяют 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.



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 вычисление агрегатов: Sum, Count, Avg, Min, Max.

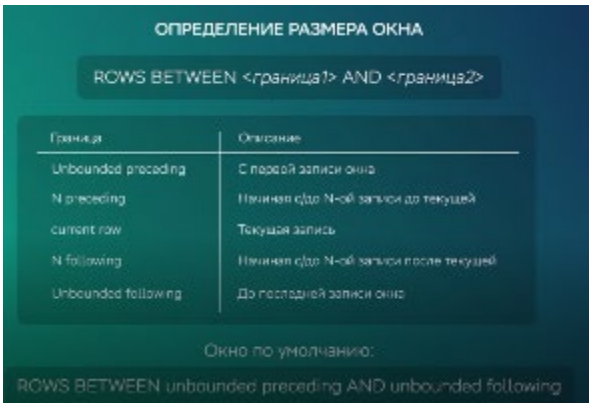


Описание окна состоит из нескольких элементов: PARTITION BY для указания ключа группы, ORDER BY для указания ключа сортировки данных внутри группы, ROWS для описания размера окна. Все элементы описания окна являются опциональными.



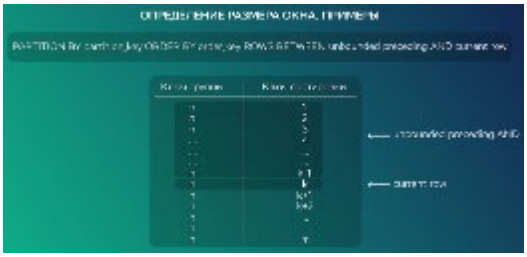
Рассмотрим пример задания окна для набора тестовых данных. Пусть мы описали окно выражением OVER для групп с одинаковым номером отдела и сортировка внутри группы по фамилии, по возрастанию. Это означает, что во время обработки все данные таблицы будут разбиты на группы с одинаковым Department_Number.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

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



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.



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

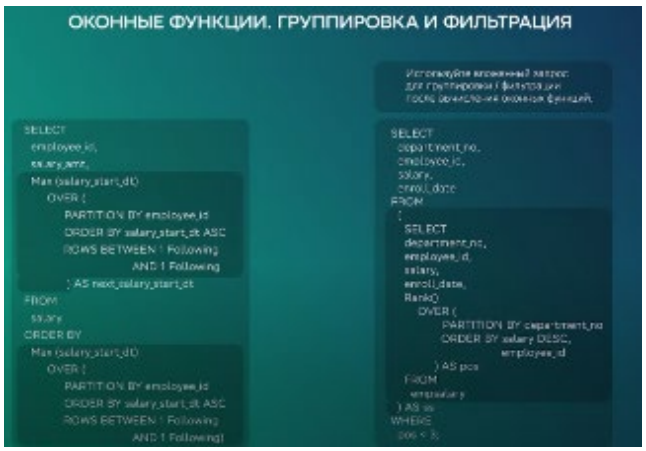
работают, находится в таблице EMPLOYEE. Данные об изменении их зарплат в таблице SALARY. Пускай мы поставили себе задачу вычислить суммарную зарплату всех сотрудников в рамках каждого отдела на данный момент среднюю зарплату в рамках отдела и вывести для каждого сотрудника долю его зарплаты в суммарной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

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.

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.

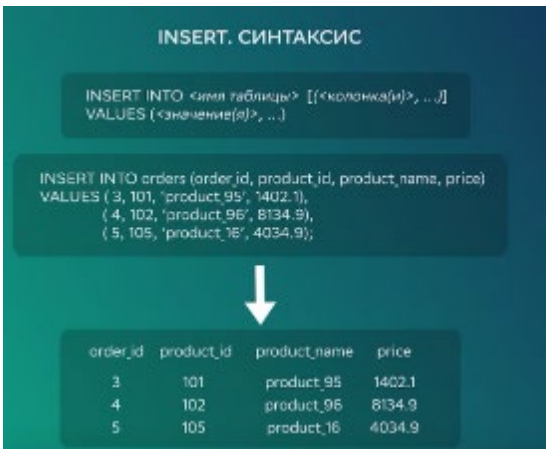
В этом запросе используется агрегирующая функция MAX, размер окна для которой - это записи, начиная с первой после и до первой после для окна упорядоченного по возрастанию SALARY_START_DT. То есть функция MAX применяется к значению одной из следующей записи. В этом примере для сотрудника №4 есть разрыв 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, HEAVING 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.



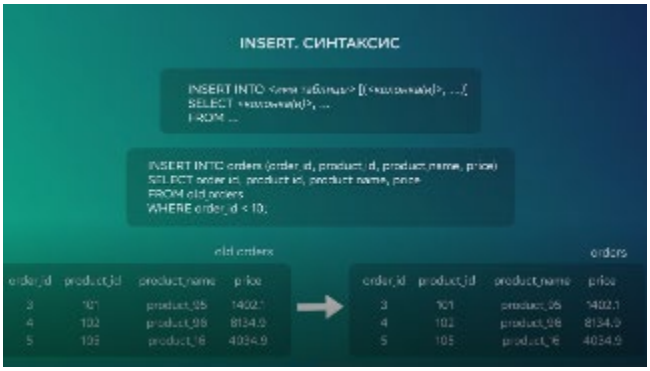
Лабораторная работа № 18 DML операции. Транзакции

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.



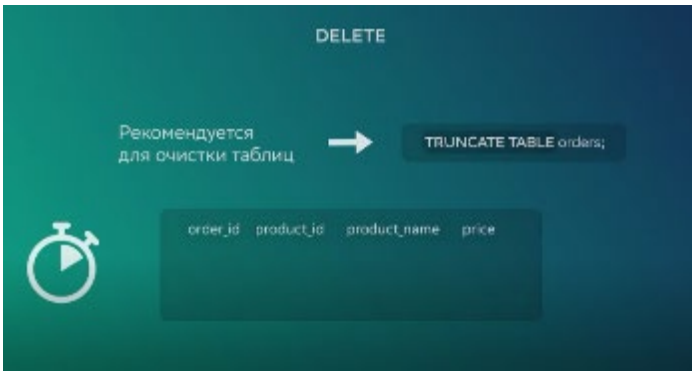
Команда Update используется для обновления существующих данных в таблице. Конструкция выглядит следующим образом: здесь указывается имя таблицы одна или несколько колонок и опциональное условие WHERE на строке обновления. Например, следующий скрипт обновит имя продукта для заказа №3 назначение Product_01. 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.

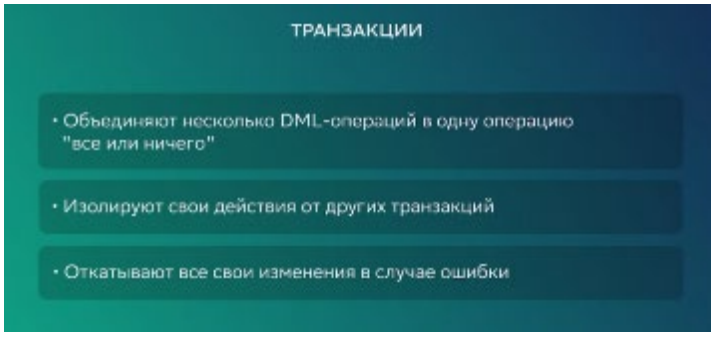
Также в Greenplum возможен вариант удаления строк по условию соединения с другой таблицей. Для этого используется ключевое слово USING, как в данном примере.

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

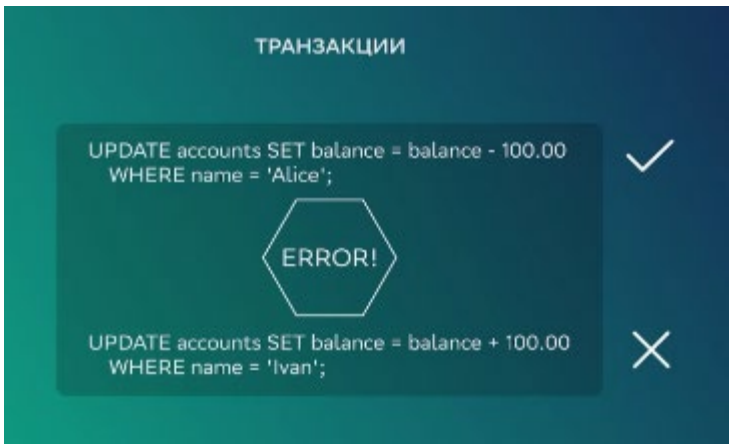
учесть, что для очистки всей таблицы рекомендуется использовать команду TRUNCATE TABLE. Данная команда выполняется быстрее и освобождает в базе данных занимаемое таблицей место.



Итак, вы вспомнили операции, используемые для манипулирования данными. Теперь мы поговорим о таком фундаментальном для всех СУБД понятий, 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 помешает успешно завершить транзакцию, ни один из результатов этих DML-операций не сохранится в базе данных.



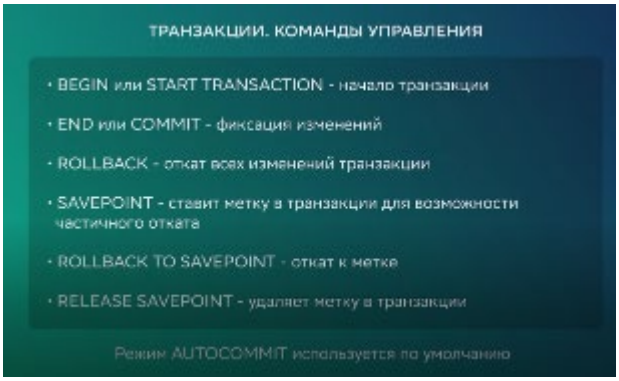
Для понимания работы транзакции рассмотрим пример типичной операции в СУБД: Перевод денег с одного счета на другой. Предположим, что мы хотим перевести 100 ₽ со счета Алисы на счет Ивана. В нашем примере первая операция списывает 100 ₽ со счета Алисы, а вторая операция зачисляют 100 ₽ на счет Ивана.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



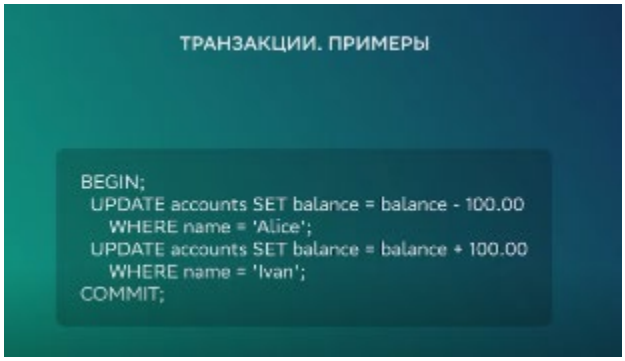
Для исключения подобных ситуаций в СУБД существуют транзакции. Если нам известно, что операции должны выполняться совместно, и при этом они не могут выполняться по отдельности, то мы объединяем эти операции в блок, который начинается со слова BEGIN и заканчивается словом COMMIT, фиксирующим изменения 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 ₽, но увеличить баланс Иванова на 100 ₽ 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.

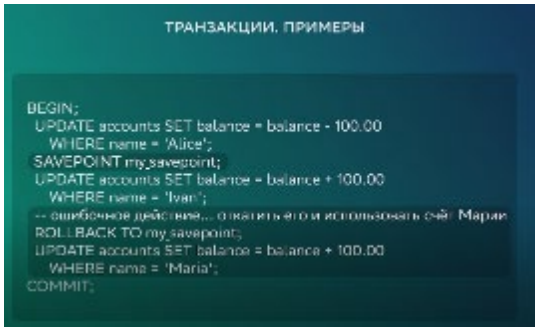


В этом режиме любая выполненная DML-операция, которая не содержит ошибок, будет автоматически зафиксирована в базе данных без необходимости явно прописывать COMMIT. Для того, чтобы отключить режим AUTOCOMMIT необходимо явно объявлять начало и конец транзакции, например, с помощью команд BEGIN и END. 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.

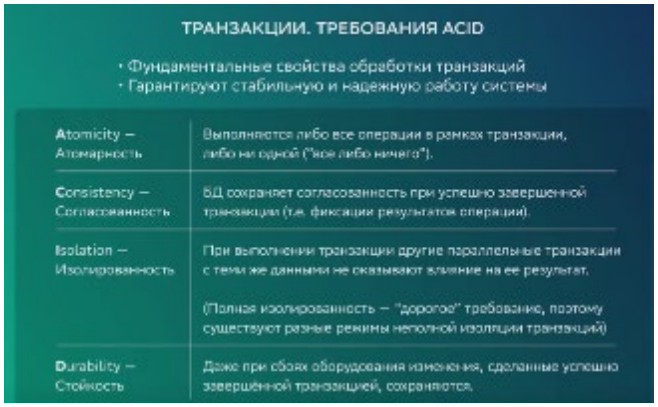


Давайте теперь вернемся к примеру с переводом денег и немного усложнили логику изменения данных. Предположим, что мы списываем 100 $ со счета Алисы, добавляем их на счет Ивана. И вдруг оказывается, что деньги нужно было перевести Марии. В данном случае мы можем применить точки сохранения. Как мы видим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



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



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 оборудования изменения, сделанные успешно завершенной транзакции, 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

по выполнению самостоятельной работы
по дисциплине «**Инжиниринг данных**»

для студентов направления 09.03.02 Информационные системы и технологии
направленность (профиль) «Прикладное программирование в интеллектуальных информационных системах»

2026

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
ПК-2 Способен разрабатывать системы управления данными и знаниями	ПК-2$_{ИД-2.1}$ – Демонстрирует знания в области проектирования и разработки баз данных и приложений на основе баз данных	Знает методы проектирования и разработки баз данных
	ПК-2$_{ИД-2.2}$ – Разрабатывает подсистемы обработки данных на основе СУБД различного типа	Разрабатывает подсистемы различных баз данных
	ПК-2$_{ИД-2.3}$ – Применяет методы поиска, очистки и предварительной обработки данных различного формата	Использует SQL запросы при решении задач дата инжиниринга, очищает данные: убирает дубли и устраняет ошибки
	ПК-2$_{ИД-2.4}$ – Демонстрирует знания методов обработки и инструментария больших данных	Знает принципы работы программных решений и специальных инструментов для представления разнородной информации в едином виде, принцип работы аналитических баз данных. Использует программные решения и специальные инструменты для представления разнородной информации в едином виде, принцип работы аналитических баз данных
	ПК-2$_{ИД-2.5}$ – Разрабатывает интеллектуальные системы, в том числе распределённые, обучающиеся на больших данных	Разрабатывает план действия извлечения, обработки и загрузки данных исходя из программно-аппаратных

		ресурсов хранилища
ПК-6 Способен проектировать информационные подсистемы обработки данных	ПК-6$_{ИД-6.1}$ – Демонстрирует знания в области архитектуры систем управления базами данных, типологии баз данных, методов оптимизации баз данных	Знает как собрать, обработать и хранить данные из различных источников.
	ПК-6$_{ИД-6.2}$ – Применяет инструментальные средства для проектирования баз данных различного типа, в том числе распределенных	Владеет современными транзакционными и аналитическими СУБД
	ПК-6$_{ИД-6.3}$ – Владеет методикой оптимизации систем хранения данных на основе интерпретируемых критериев эффективности	Структурирует данные так, чтобы они стали пригодны для аналитики. Обновляет используемые инструменты и ищет способы оптимизации и автоматизации существующих процессов. Использует методы оптимизации для обработки и загрузки данных из различных источников
	ПК-6$_{ИД-6.4}$ – Анализирует предметную область в рамках решения профессиональных задач и проектирует архитектуру систем обработки данных на основании типов данных, конфигурации потоков данных	Формулирует задачу для решения методами дата инженера, проводит ее декомпозицию, выделяет проблемы.

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 СВТ и возможностью пользования Интерент-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	Самостоятельная изучение тем: Технологии хранилищ данных Инкриментальная загрузка в хранилищах данных	Опрос

Наименование тем дисциплины, цель, форма контроля, задания, требования к оформлению , перечень литературных источников представлен в таблице3.

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 лекциям и лабораторным занятиям	изучить основные технологии запросов, обработки и хранения данных	индивидуальное собеседование	Сформулировать 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,2026-02-26. - 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 9.5,2026-10-27. - 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.