
PostgreSQL

Занятие первое
Темы:

- Вводная часть
- Способы подключения к PostgreeSQL
- Создание БД
- Схемы
- Констрейнты, ограничения, первичные и внешние ключи
- Создание, редактирование и удаление таблиц

Почему PostgreSQL?

- Free & Open Source
- Лучший выбор для изучения: проинсталлировал и «понеслась»!
- «Взрослая» СУБД, поддерживает транзакционность из коробки.
- Весьма развитый диалект SQL.
- В сравнении с MySQL есть свои плюсы и минусы
- До 90% возможностей диалекта PostgreSQL можно без изменений использовать в других СУБД.

SQL – Structured Query Language
ANSI SQL-92

- DDL – Data Definition Language (CREATE, ALTER, DROP)
- DML – Data Manipulation Language (SELECT, INSERT, UPDATE, DELETE)
- TCL – Transaction Control Language (BEGIN, COMMIT, ROLLBACK, SAVEPOINT)
- DCL – Data Control Language (GRANT, REVOKE, DENY)

Подключаемся к базе данных

- DBeaver Community https://dbeaver.io/download/
- pgAdmin https://www.pgadmin.org/download/
- Инструкции по настройке подключений тут: https://disk.yandex.ru/d/djGiu1dvaB4IHQ

Создание, редактирование и удаление баз данных

- Шаблоны
- Табличные пространства
- Кодировка символов
- Владелец
- Создание базы: *CREATE DATABASE test_db WITH OWNER = postgres ENCODING = 'UTF8' TABLESPACE = pg_default TEMPLATE = 'template0';*
- Переименование базы: *ALTER DATABASE test_db RENAME TO new_db;*
- Сменить владельца базы: *ALTER DATABASE test_db OWNER TO admin;*
- Сменить табличное пространство: *ALTER DATABASE test_db SET TABLESPACE new_ts;*
- Удалить базу данных: *DROP DATABASE test_db;*
- *https://postgrespro.ru/docs/postgrespro/15/sql-alterdatabase*

Правила наименования объектов в PostgreSQL

- PostgreSQL работает в нижнем регистре и все наши скрипты в него транслирует
- Для наименований использовать латинские буквы в нижнем регистре и цифры
- Для разделения слов использовать символ _
- Не начинать наименование с цифры
- Если необходимо обойти данные правила, заключайте наименование в двойные кавычки
- Не начинать имена с pg_ это внутренние объекты PostgeSQL

Численные типы данных PostgreSQL

	Наименование	Размер	Описание	Диапазон значений
Целые числа	smallint	2 Bytes	Маленькие целые числа	от -32768 до 32767
	Integer / int	4 Bytes	Целые числа	от -2147483648 до 2147483647
	bigint	8 Bytes	Большие целые числа	от -9223372036854775808 до 9223372036854775807
Числа с точкой	decimal / numeric	variable	Числа с точкой фикс. точности *	до 131072 цифр до десятичной точки и до 16383 — после
	real	4 Bytes	Вещественное число с переменной точностью	от 1E-37 до 1E+37 с точностью не меньше 6 десятичных цифр
	float / double precision	8 Bytes	Вещественное число с переменной точностью	от 1E-307 до 1E+308 и с точностью не меньше 15 десятичных цифр
Автоинкрементны	smallserial	2 Bytes	Маленькие целые числа	от 1 до 32767
	serial	4 Bytes	Целые числа	от 1 до 2147483647
	bigserial	8 Bytes	Большие целые числа	от 1 до 9223372036854775807

*decimal / numeric дают точный результат, но операции с ними выполняются медленнее, чем с real / float

https://postgrespro.ru/docs/postgrespro/15/datatype-numeric#DATATYPE-NUMERIC-DECIMAL

Текстовые типы данных PostgreSQL

Наименование	Размер	Описание	Примечание
char(n)	Variable	Строка фиксированной длины не более 10485760 символов	При записи значения < указанного дополняет строку пробелами
varchar(n)	variable	Строка произвольной длины не более 10485760 символов. Если не указать размерность = text	Размер на диске зависит от кодировки символов + значение длины строки
text	variable	Текст	Любой текст, который занимает на диске не более 1 Гб

https://postgrespro.ru/docs/postgrespro/15/datatype-character

Типы данных даты и времени в PostgreSQL

Наименование	Размер	Описание	Диапазон значений
date	4 Bytes	Только дата	От 4713 до НЭ до 5874897 НЭ
time	8 Bytes	Только время	От 00:00:00.000 до 24:00:00.000
timestamp	8 Bytes	Дата и время	От 4713 до НЭ до 294276 НЭ
timestamptz	8 Bytes	Дата и время + часовой пояс	От 4713 до НЭ до 294276 НЭ + tz
interval	16 Bytes	Интервал между двумя timestamp	От -178000000 лет до 178000000 лет

https://postgrespro.ru/docs/postgrespro/15/datatype-datetime

Прочие типы данных PostgreSQL

Наименование	Размер	Описание	Примечание
boolean	1 байт	Логический тип данных	true/yes/on/1 или false/no/off/0
bit(n)	variable	Хранение битовых масок. Длина значения типа bit должна в точности равняться *n*	Запись bit без указания длины равнозначна записи bit(1), тогда как bit varying без указания длины подразумевает строку неограниченной длины
UUID	128 бит	Универсальные уникальные идентификаторы	
XML	variable	Тип xml может сохранять «документы», в соответствии со стандартом XML	Как тип text, но проверяет вводимые значения на допустимость по правилам XML
JSON/JSONB	variable	Хранение данных в JSON. JSONB хранит информацию в бинарной форме, что экономит место на диске.	JSON как text, но проверяет вводимые значения на допустимость по правилам JSON. +Поддержка функций для работы с JSON.

https://postgrespro.ru/docs/postgrespro/15/datatype

Схемы

- Есть несколько возможных объяснений, для чего стоит применять схемы:
  - Чтобы одну базу данных могли использовать несколько пользователей, независимо друг от друга.
  - Чтобы объединить объекты базы данных в логические группы для облегчения управления ими.
  - Чтобы в одной базе сосуществовали разные приложения, и при этом не возникало конфликтов имён.
- Одно и то же имя объекта можно свободно использовать в разных схемах.
- В отличие от баз данных, схемы не ограничивают доступ к данным: пользователи могут обращаться к объектам в любой схеме текущей базы данных, если им назначены соответствующие права.

Операции со схемами

- Создать схему CREATE SCHEMA test;
- Удалить пустую схему DROP SCHEMA test;
- Удалить схему со всеми объектами DROP SCHEMA test CASCADE;
- Создать схему с владельцем CREATE SCHEMA test AUTHORIZATION test_user;
- Переименовать схему ALTER SCHEMA test RENAME TO new_test;
- Сменить владельца схемы ALTER SCHEMA test OWNER TO test_user;
- https://postgrespro.ru/docs/postgrespro/15/ddl-schemas

Таблицы: основные понятия

- Число и порядок столбцов фиксированы, а каждый столбец имеет имя
- Число строк переменно — оно отражает текущее количество находящихся в ней данных.
- Каждому столбцу сопоставлен тип данных. Тип данных ограничивает набор допустимых значений.
- Число столбцов в таблице ограничивается максимумом в пределах от 250 до 1600, в зависимости от типов столбцов.
- Столбцу можно назначить значение по умолчанию.
- Если значение по умолчанию не объявлено явно, им считается значение NULL.
- Ограничения дают вам возможность управлять данными в таблицах так, как вы захотите. Если пользователь попытается сохранить в столбце значение, нарушающее ограничения, возникнет ошибка. Ограничения будут действовать, даже если это значение по умолчанию.
- Ограничения уникальности гарантируют, что данные в определённом столбце или группе столбцов уникальны среди всех строк таблицы.
- Ограничение NOT NULL просто указывает, что столбцу нельзя присваивать значение NULL.

Первичные ключи таблиц

- Ограничение первичного ключа означает, что образующий его столбец или группа столбцов может быть уникальным идентификатором строк в таблице. Для этого требуется, чтобы значения были одновременно уникальными и отличными от NULL.
- При добавлении первичного ключа автоматически создаётся уникальный индекс-B-дерево для столбца или группы столбцов, перечисленных в первичном ключе, и данные столбцы помечаются как NOT NULL
- Таблица может иметь максимум один первичный ключ.
- Первичные ключи полезны и для документирования, и для клиентских приложений. Например, графическому приложению с возможностями редактирования содержимого таблицы, вероятно, потребуется знать первичный ключ таблицы, чтобы однозначно идентифицировать её строки. Первичные ключи находят и другое применение в СУБД; в частности, первичный ключ в таблице определяет целевые столбцы по умолчанию для сторонних ключей, ссылающихся на эту таблицу.
- https://postgrespro.ru/docs/postgrespro/15/ddl-constraints#DDL-CONSTRAINTS-PRIMARY-KEYS

Внешние ключи таблиц

- Ограничение внешнего ключа указывает, что значения столбца (или группы столбцов) должны соответствовать значениям в некоторой строке другой таблицы. Это называется *ссылочной целостностью* двух связанных таблиц.
- Внешний ключ также может ссылаться на группу столбцов. В этом случае его нужно записать в виде обычного ограничения таблицы.
- Иногда имеет смысл задать в ограничении внешнего ключа в качестве «другой таблицы» ту же таблицу; такой внешний ключ называется *ссылающимся на себя*.
- Таблица может содержать несколько ограничений внешнего ключа. Это полезно для связи таблиц в отношении многие-ко-многим.
- Внешний ключ поможет защитить связанные записи от удаления либо удалить их каскадно, в зависимости от того, как мы это опишем при создании ключа.

Создание и удаление таблиц

- Создание таблицы: *CREATE TABLE mytbl (id INTEGER PRIMARY KEY, name TEXT);*
- Удаление таблицы: *DROP TABLE mytbl;*
- Удаление таблицы, на которую есть ссылка по внешнему ключу: *DROP TABLE mytbl CASCADE;*
- *https://postgrespro.ru/docs/postgrespro/15/ddl-basics*

Изменение таблиц

- Добавление столбца: *ALTER TABLE mytbl ADD COLUMN new_col VARCHAR(100);*
- Удаление столбца: *ALTER TABLE mytbl DROP COLUMN new_col;*
- Добавление ограничения: *ALTER TABLE mytbl ADD CONSTRAINT new_const (new_col);*
- Удаление ограничения: *ALTER TABLE mytbl DROP CONSTRAINT new_const;*
- Добавление внешнего ключа: *ALTER TABLE mytbl ADD FOREIGN KEY (unit_id) REFERENCES prodmag.units(unit_id);*
- Удаление внешнего ключа: *ALTER TABLE mytbl DROP FOREIGN KEY (unit_id);*
- Изменение значения по умолчанию: *ALTER TABLE mytbl ALTER COLUMN new_col DEFAULT 'ttt';*
- Удаление значения по умолчанию: *ALTER TABLE mytbl ALTER COLUMN new_col DROP DEFAULT;*
- Изменение типа столбца: *ALTER TABLE mytbl ALTER COLUMN new_col TYPE text;*
- Переименование столбца: *ALTER TABLE mytbl RENAME COLUMN new_col TO column_new1*
- Переименование таблицы: *ALTER TABLE mytbl RENAME TO new_my_tbl;*
- https://postgrespro.ru/docs/postgrespro/15/ddl-alter

Анализ исходного xlsx-файла

- Анализируем файл «Продукты питания.xlsx»
- Определяем, на сколько таблиц нужно разбить данную структуру
- Определяемся с типами полей для таблиц
- Создадим таблицы (см. пример № 1)
- Отредактируем таблицы (исправим недочеты)

Домашнее задание №1

- Проанализировать данные в исходном файле «HomeWork_1.xlsx»
- Создать в своей тестовой базе данных схему «tag_data»
- Создать в схеме «tag_data» структуру таблиц для хранения данных, представленных в файле «HomeWork_1.xlsx»
- Предоставить для проверки скрипты, которыми было выполнено задание.
- Подсказка: у вас должно получиться 5 таблиц – 4 справочных и одна со значениями по тегам.
- Подсказка: не забываем выбирать правильные типы данных для колонок и создавать первичные и внешние ключи в таблицах.

Занятие второе
Темы:

- Основные операторы DML: Select, Insert, Update и Delete
- Условия выборки Where
- Сортировка результатов запроса Order By
- Ограничение результатов Limit и Offset
- Псевдонимы колонок и таблиц
- Подзапросы
- Команда COPY

Добавление данных (INSERT)

- Вставка одной строки: *INSERT INTO mytbl VALUES (1, ‘txt-1’);*
- Вставка нескольких строк: *INSERT INTO mytbl VALUES (1, ‘txt-1’), (2, ‘txt-2’), (3, ‘txt-3’);*
- Вставка значений по умолчанию: *INSERT INTO mytbl VALUES (DEFAULT, ‘txt-4’);*
- Вставка результата запроса: *INSERT INTO tbl1 SELECT * FROM tbl2;*
- Вставим данные в наши таблицы (см. пример № 2)
- https://postgrespro.ru/docs/postgrespro/15/dml-insert

Чтение данных (SELECT)

- Простой запрос: *SELECT * FROM mytbl;*
- Псевдонимы имен таблиц и полей: *SELECT t.id, (t.price * t.qty) AS cost FROM myscheme.mytbl AS t;*
- Запрос с условиями: *SELECT * FROM mytbl WHERE id > 2;*
- Использование *AND, OR, IN, BETWEEN* в условиях выборки;
- Выборка уникальных значений: *SELECT DISTINCT name FROM mytbl;*
- Сортировка данных в выборке: *SELECT * FROM mytbl ORDER BY name ASC (DESC);*
- Ограничение количества строк в выборке: *SELECT * FROM tbl ORDER BY name LIMIT 1 OFFSET 20;*
- Подзапросы
- Почитаем данные из наших таблиц (см. пример № 3)
- https://postgrespro.ru/docs/postgrespro/15/queries

Изменение данных (UPDATE)

- Изменение одного поля в строке: *UPDATE mytbl SET name = ‘new_txt’ WHERE id = 1;*
- Изменение множества строк: *UPDATE mytbl SET price = price * 1.1 WHERE price > 0;*
- Изменение из подзапроса: *UPDATE tbl_1 SET name = s.name FROM (SELECT id, name FROM tbl_2) AS s WHERE id = s.id;*
- Изменим данные в наших таблицах (см. пример № 4)
- https://postgrespro.ru/docs/postgrespro/15/dml-update

Удаление данных (DELETE и TRUNCATE)

- Удаление одной строки: *DELETE FROM mytbl WHERE id = 1;*
- Удаление всех строк: *DELETE FROM mytbl ;*
- Если таблица большая, команда DELETE может выполняться очень долго, в этом случае полную очистку таблицы необходимо выполнять командой TRUNCATE: *TRUNCATE TABLE mytbl;*
- Удаление из подзапроса: *DELETE FROM tbl_1 AS t1 USING tbl_2 AS t2 WHERE t1.id = t2.id;*
- Попробуем удалять в подопытных таблицах (см. пример № 5)
- https://postgrespro.ru/docs/postgrespro/15/dml-delete

Копирование данных (COPY)

- Копирование из таблицы в файл: *COPY(SELECT * FROM tbl_1) TO 'C:/Data/tbl_data.csv' CSV DELIMITER ';' HEADER ENCODING ‘UTF8’;*
- Копирование из файла в таблицу: *COPY tbl_1 FROM 'C:/Data/tbl_data.csv’ WITH (FORMAT CSV, DELIMITER ';', ENCODING ' UTF8 ', HEADER);*
- Пробуем копировать в подопытных таблицах (см. пример № 6)
- https://postgrespro.ru/docs/postgrespro/15/sql-copy

Домашнее задание №2

- В первом задании вы создали набор таблиц на основе файла «HomeWork_1.xlsx»
- На сервере в папке C:/DataImportExport находится файл tag_data.csv с данными. С его содержимым вы можете ознакомиться в папке домашнего задания.
- Необходимо загрузить из файла данные в ранее подготовленные таблицы.
- Написать запрос, который вернет из ваших таблиц данные в исходном виде (как в файле)
- Предоставить для проверки скрипты, которыми было выполнено задание.
- Подсказка: необходимо временно создать таблицу со структурой, повторяющей структуру в файле, заполнить временную таблицу данными из файла, а затем из временной таблицы заполнять целевые. После заполнения целевых таблиц, временную удалите.
- Подсказка: данные в файле в кодировке UTF8.

Занятие третье
Темы:

- Табличные пространства
- Способы приведения типов данных Cast
- Работа с представлениями View
- Объединение запросов Union
- Соединение запросов Join

Табличные пространства

- Назначение табличного пространства:
  - 1. Нехватка места в разделе, на котором был инициализирован кластер и невозможность его расширения. Табличное пространство можно создать в другом разделе.
  - 2. Позволяет оптимизировать производительность. Например, часто используемый индекс можно разместить на очень быстром и надёжном, но дорогом SSD-диске. А таблицу с архивными данными, которые редко используются и скорость доступа к ним не важна, можно разместить в более дешёвом и медленном хранилище.
- Создание табличного пространства: *CREATE TABLESPACE ts_archive LOCATION 'E:/postgresql/data';*
- Создание базы данных в новом табличном пространстве: *CREATE DATABASE archive_db WITH OWNER = postgres ENCODING = 'UTF8' TABLESPACE = ts_archive;*
- Создание таблицы в новом табличном пространстве: *CREATE TABLE archice_data (id INT, value FLOAT, date TIMESTAMP) TABLESPACE = ts_archive;*
- Тестируем работу с табличным пространством (см. пример № 7)
- https://postgrespro.ru/docs/postgrespro/15/manage-ag-tablespaces

Приведение типов (CAST)

- Функция приведения типов: *SELECT CAST (‘22’ AS INTEGER);*
- Аналогичного результата можно добиться используя конструкцию «::»: *SELECT ’22’::INTEGER;*
- https://postgrespro.ru/docs/postgrespro/15/sql-createcast

Представления (VIEW)

- Создание представления: *CREATE VIEW my_view AS SELECT * FROM tbl_1 WHERE type='2';*
- Переопределить запрос внутри представления можно так: *CREATE OR REPLACE VIEW my_view AS SELECT * FROM tbl_1 WHERE type='5';* Причем количество и типы возвращаемых полей меняться не должны.
- Переименовать представление можно так: *ALTER VIEW my_view RENAME TO new_view;*
- Переименовать поле, возвращаемое представлением можно так: *ALTER VIEW my_view RENAME COLUMN id TO new_id;*
- Переместить представление в другую схему: *ALTER VIEW my_view SET SCHEMA new_schema;*
- Удалить представление: *DROP VIEW my_view;*
- Чтение данных из схемы аналогично чтению таблиц: *SELECT * FROM my_view;*
- Тестируем работу с представлениями (см. пример № 8)
- https://postgrespro.ru/docs/postgrespro/15/sql-createview

Сочетание запросов (UNION, INTERSECT, EXCEPT)

- Объединение запросов UNION. Добавляет результаты второго запроса к результатам первого : *SELECT * FROM tbl_1 UNION (ALL) SELECT * FROM tbl_2;*
- Пересечение запросов INTERSECT. Возвращает все строки, содержащиеся в результате и первого, и второго запроса:

  *SELECT * FROM tbl_1 INTERSECT SELECT * FROM tbl_2;*
- Вычитание запросов EXCEPT. Возвращает все строки, которые есть в результате первого запроса, но отсутствуют в результате второго:

  *SELECT * FROM tbl_1 EXCEPT SELECT * FROM tbl_2;*
- Пробуем сочетать запросы (см. пример № 9)
- https://postgrespro.ru/docs/postgrespro/15/queries-union

Соединение запросов (JOIN)

- Внутреннее соединение INNER JOIN или просто JOIN. Такое соединение, при котором выбираются записи присутствующие как в левой, так и в правой таблице: *SELECT a.id, a.name, b.code FROM tbl_1 AS a JOIN tbl_2 AS b ON a.id = b.id;*
- Левое внешнее соединение LEFT OUTER JOIN или LEFT JOIN. Возвращает все строки, содержащиеся в левой таблице, строки отсутствующие в правой таблице заменяются значением NULL:

  SELECT a.id, a.name, b.code FROM tbl_1 AS a JOIN tbl_2 AS b ON a.id = b.id;
- Правое внешнее соединение RIGHT OUTER JOIN или RIGHT JOIN. Аналогично LEFT JOIN, но таблицы поменялись местами – ведущая справа, отсутствующие значения слева значением NULL.
- FULL JOIN Возвращает все соединенные строки обоих таблиц, плюс несоединенные строки левой таблицы и несоединенные строки правой таблицы дополненные значениями NULL для отсутствующих значений.
- Пробуем сочетать запросы (см. пример № 10)
- https://postgrespro.ru/docs/postgrespro/15/sql-select

Домашнее задание №3

- Изучить структуру таблиц в схеме «bookings» в своей тестовой базе данных. Подробно о взаимосвязи таблиц и описание их полей можно узнать из файла «Bookings.html».
- Создать представление с использованием JOIN, которое вернет следующие поля:
  - flights.flight_no – номер рейса
  - flights.scheduled_departure – дата и время вылета
  - flights.departure_airport – код аэропорта вылета
  - airports.city – город вылета
  - airports.airport_name - аэропорт вылета
  - flights.scheduled_arrival - дата и время прилета
  - flights.arrival_airport – код аэропорта прилета
  - airports.city – город прилета
  - airports.airport_name - аэропорт прилета
  - aircraft_code.model – модель самолета
- Предоставить для проверки скрипты, которыми было выполнено задание.

Занятие четвертое
Темы:

- Последовательности Sequence
- Группировка и агрегатные функции Group By
- Оконные функции
- Табличные выражения With (CTE и Recursive)

Последовательности (SEQUENCE)

- Создание последовательности для поля id в таблице my_tbl:

  CREATE SEQUENCE my_seq INCREMENT 1 MINVALUE 1 MAXVALUE 2147483647 START 1 OWNED BY my_tbl.id;

- Получить следующее значение последовательности:

  SELECT nextval (‘my_seq’);

- Использовать последовательность во вставке данных:

  INSERT INTO tbl VALUES (nextval('my_seq'), ‘txt’);

- Установить последовательность в качестве значения по умолчанию:

  ALTER TABLE my_tbl ALTER COLUMN id SET DEFAULT nextval(‘my_seq’);

- Удалить последовательность: *DROP SEQUENCE my_seq CASCADE;*
- Протестируем работу с последовательностью (см. пример № 11)

Группировка (GROUP BY)

- Выражение GROUP BY собирает в одну строку все строки, имеющие одинаковые значения в столбце (столбцах) группировки см. файл *Группировка.xlsx*
- Необязательное выражение HAVING фильтрует из результирующего набора групп строки не удовлетворяющих условию, описанному в HAVING. Работает аналогично WHERE, но WHERE применяется до группировки, а HAVING к уже сгруппированному набору строк.
- К столбцам, не указанным в выражении GROUP BY, необходимо применить одну из агрегатных функций, чтобы из значений множества строк получить одно агрегированное значение.
- Протестируем работу группировки (см. пример № 12)

Агрегатные функции

- ARRAY_AGG() собирает значения в массив.
- AVG() вычисляет среднее арифметическое.
- COUNT() возвращает количество строк.
- JSON_AGG() собирает значения в массив JSON.
- JSON_OBJECT_AGG(key, value) собирает значения в объект JSON.
- MAX() выдает максимальное значение.
- MIN() выдает минимальное значение.
- STRING_AGG(value, delimiter) собирает значения в строку через разделитель.
- SUM() вычисляет сумму значений.
- https://postgrespro.ru/docs/postgresql/15/functions-aggregate

Оконные функции

- ROW_NUMBER() Создает нумерацию строк [по группам].
- FIRST_VALUE() Возвращает первое значение из рамки.
- LAST_VALUE() Возвращает последнее значение из рамки.
- LAG(col_name,offset,default) Возвращает значение из строки сдвинутой на offset к началу рамки. Если такой строки нет, возвращает значение default.
- LEAD(col_name,offset,default) Возвращает значение из строки сдвинутой на offset к концу рамки. Если такой строки нет, возвращает значение default.
- NTH_VALUE(col_name,row_num) Возвращает значение из строки номер row_num от начала рамки. Если такой строки нет, возвращает значение NULL.
- Протестируем работу оконных функций (см. пример № 13)
- https://postgrespro.ru/docs/postgresql/15/functions-window
- https://postgrespro.ru/docs/postgrespro/15/sql-expressions#SYNTAX-AGGREGATES

Табличные выражения (WITH)

- Оператор WITH предоставляет возможность использовать подзапрос, как временную таблицу, существующую в рамках текущего запроса. Данная структура называется CTE (Common Table Expressions).
- SELECT в WITH CTE помогает разбить сложные запросы на простые части и впоследствии обращаться к ним, как к обычным таблицам.
- SELECT внутри WITH выполняется один раз и хранится в памяти, что позволяет ускорить запросы (иногда скорость выполнения запроса повышается в несколько раз)
- Конструкция WITH RECURCIVE позволяет обращаться WITH к собственному результату, что дает нам рекурсивный запрос.
- WITH RECURCIVE следует применять с осторожностью т.к. не верно описанный выход из рекурсии может привести к зависанию сервера из за исчерпания оперативной памяти.
- Протестируем работу табличных выражений (см. пример № 14)
- https://postgrespro.ru/docs/postgrespro/15/queries-with

Домашнее задание №4

- Выполнить в своей тестовой базе данных скрипт script.sql.
- Используя рекурсивный запрос WITH RECURSIVE, посчитать сколько у каждого человека сотрудников в подчинении. Для рядовых сотрудников вывести 0.
- Задача со * посчитать сколько у каждого человека сотрудников в подчинении, и непосредственных, и косвенных. Т.е. у гендира в непосредственном подчинении четыре зама, а в косвенном подчинении все сотрудники компании, кроме него самого.
- Подсказка: сотрудники взаимосвязаны через parent_id.
- Предоставить для проверки скрипты, которыми было выполнено задание.