**Урок 1**

**Создание базы данных**

CREATE DATABASE testdb

WITH

OWNER = mihaildb

ENCODING = 'UTF8'

LC\_COLLATE = 'Russian\_Russia.1251'

LC\_CTYPE = 'Russian\_Russia.1251'

TABLESPACE = pg\_default

CONNECTION LIMIT = -1

COMMENT ON DATABASE testdb

IS 'Учебная база данных`;

**Назначение прав**

CREATE SCHEMA public

AUTHORIZATION mihaildb;

COMMENT ON SCHEMA public

IS 'standard public schema';

GRANT ALL ON SCHEMA public TO PUBLIC;

GRANT ALL ON SCHEMA public TO mihaildb;

**Удаление базы данных**

*Отключение базы данных*

SELECT pg\_terminate\_backend(pg\_stat\_activity.pid)

FROM pg\_stat\_activity

WHERE pg\_stat\_activity.datname = 'testdb'

AND pid <> pg\_backend\_pid() ;

*Непосредственно удаление*

DROP DATABASE testdb ;

**Создание таблицы**

CREATE TABLE IF NOT EXISTS public.book

(

book\_id integer NOT NULL,

title text NOT NULL,

isbn character varying(32) NOT NULL,

CONSTRAINT pk\_book\_id PRIMARY KEY (book\_id)

);

ALTER TABLE public.book

OWNER to mihaildb;

**Вставка данных**

INSERT INTO publisher

VALUES

(1, 'Everymans''s library', 'NY'),

(2, 'Oxford university press', 'NY');

Здесь Everymans's library – поэтому используется дополнительная кавычка

**Урок 2**

**Подсчёт числа уникальных названий стран в таблице customers**

COUNT(DISTINCT country)

FROM customers;

**Конструкция, выводящая пустую таблицу**

SELECT \*

FROM orders

WHERE ship\_region = NULL;

**Правильный вариант**

SELECT \*

FROM orders

WHERE ship\_region IS NULL;

**Выбор данных из двух таблиц и объединение их в одном столбце без повторений**

SELECT country

FROM customers

UNION

SELECT country

FROM employees ; -- 21 аналог ∪

**Выбор данных из двух таблиц и объединение их в одном столбце с повторениями**

SELECT country

FROM customers

UNION ALL

SELECT country

FROM employees ; -- 100

**Оператор ALL добавляет столько повторов, сколько их содержится в той таблице, в которой их меньше, чем в другой.**

SELECT country, count(COUNTRY)

FROM customers

WHERE country = 'UK' OR country = 'USA'

GROUP BY country ; -- 7 and 13

SELECT country, count(COUNTRY)

FROM employees

WHERE country = 'UK' OR country = 'USA'

GROUP BY country ; -- 4 and 5

SELECT country

FROM customers

INTERSECT

SELECT country

FROM employees ; -- 1 and 1

SELECT country

FROM customers

INTERSECT ALL

SELECT country

FROM employees ; -- 4 and 5

SELECT DISTINCT country

FROM customers

INTERSECT ALL

SELECT country

FROM employees ; -- 1 and 1

SELECT country

FROM customers

INTERSECT ALL

SELECT DISTINCT country

FROM employees ; -- 1 and 1

**Урок 3**

**Самосоединение – для построения иерархии.**

CREATE TABLE IF NOT EXISTS employee (

employee\_id INT PRIMARY KEY,

first\_name VARCHAR (255) NOT NULL,

last\_name VARCHAR (255) NOT NULL,

manager\_id INT, -- внешний ключ, ссылающийся на первичный этой же таблицы

FOREIGN KEY (manager\_id) REFERENCES employee(employee\_id)

);

INSERT INTO employee (

employee\_id,

first\_name,

last\_name,

manager\_id

)

VALUES

(1, 'Windy', 'Hays', NULL),

(2, 'Ava', 'Christensen', 1),

(3, 'Hassan', 'Conner', 1),

(4, 'Anna', 'Reeves', 2),

(5, 'Sau', 'Norman', 2),

(6, 'Kelsie', 'Hays', 3),

(7, 'Tory', 'Goff', 3),

(8, 'Salley', 'Lester', 3) ;

**Вывод имён работника и его менеджера:**

SELECT

ee.first\_name || ' ' || ee.last\_name AS employee,

mm.first\_name || ' ' || mm.last\_name AS manager

FROM employee AS ee

LEFT JOIN employee AS mm ON ee.manager\_id = mm.employee\_id

ORDER BY manager;

**Не рекомендовано к использованию естественное соединение**

Оно автоматически находит одинаково называющийся столбец и по нему соединяет как внутреннее соединение. Но если в таблицах существует ещё по одному столбцу с одним и тем же названием, то возникнет ошибка.

SELECT order\_id, customer\_id, first\_name, last\_name, title

FROM orders

NATURAL JOIN employees ;

НЕ ИСПОЛЬЗУЙТЕ ТАКОЙ СИНТАКСИС

**Работа с postgres**

- для перехода в серверный процесс postgres в командной строке набирается вызов:

> psql -U postgres postgres (+ пароль от пользователя postgres)

- вывод версии PostgreSQL:

postgres=# SELECT version();

- получить список всех возможных расширений

postgres=> SELECT \* FROM pg\_available\_extensions;

- получить список установленных расширений (extensions) на сервере:

postgres=> SELECT \* FROM pg\_available\_extensions WHERE installed\_version IS NOT NULL;

postgres=# CREATE EXTENSION file\_fdw; - установить расширение file\_fdw.

postgres=# CREATE EXTENSION "uuid-ossp"; - установить расширение uuid-ossp

ПАКЕТЫ

postgresql-contrib – содержит расширения

БИБЛИОТЕКИ

Libpq – поддерживает переменные окружения (PGHOST, PGDATABASE, PGUSER).

Задаёт псевдоним activity1 для запроса в кавычках.

postgres=# \set activity1 'SELECT pid, query, backend\_type, state FROM pg\_stat\_activity';

После этого запрос можно запускать как

postgres=# :activity1;

pg\_hba.conf – конфигурационный файл аутентификации пользователей

вход под другим пользователем:

psql name\_database mihaildb

psql -U mihaildb name\_database

<user>=<privileges>/granted by – представление пользователю разрешений для доступа к объектам PostgreSQL: БД, таблицы, представления, последовательности. Если пользователь не указан, то считают, что разрешения предоставляют роли PUBLIC. По умолчанию эта роль имеет право подключаться к ШБД template1.

Для базы данных определены следующие разрешения:

-с – создание новых схем в БД;

-С – подключение к БД;

-Т – создание временных таблиц.

Атрибуты БД (находятся в таблице pg\_database):

Обслуживание – **datfrozenxid** – позволяет узнать, пора ли очищать БД;

Управление хранением – **dattablespace** – определяет табличное пространство БД;

Параллелизм – **datconnlimit** – задаёт число допустимых подключений (-1 означает, что ограничения отсутствуют);

Защита – **datallowconn** – запрещает подключение к БД. (обычно применяется к template0).

oid – идентификатор объекта;

datname – имя БД;

datdba –

encoding –

datcollate –

datctype –

datistemplate – определяет, является ли БД шаблонной или нет.

datlastsysoid –

datminmxid –

datacl – показывает выданные разрешения БД.

Просмотр и изменение максимального числа подключений к БД с -1 на 1

postgres=# SELECT datconnlimit FROM pg\_database WHERE datname= 'postgres';

postgres=# ALTER DATABASE postgres CONNECTION LIMIT 1;

Фронтальные команды:

psql

clasterdb эквивалента CLUSTER – физически переупорядочивает таблицу в соответствии с индексом

createdb эквивалентна CREATE DATABASE

dropdb эквивалентна DROP DATABASE [IF EXISTS]

createuser

dropuser

reindexdb эквивалентна REINDEX

vacuumdb

pg\_basebackup – физическое резервное копирование и восстановление (в том числе репликация)

pg\_dump – логическое копирование одной базы данных, схемы, отношения. Он не включает команду CREATE DATABASE и не копирует особые привилегия.

pg\_dumpall – логическое копирование всего кластера. Вызывает pg\_dump для всех объектов

pg\_restore – восстанавливает данные из файлов, созданных с помощью pg\_dump или pg\_dumpall

pg\_isready

pg\_recieveWAL

pg\_recvlogical

pg\_catalog – схема, хранящая таблицы.

Таблицы и представления:

pg\_database – список таблиц в кластере

pg\_pltemplate – сведения о шаблонных процедурных языках

pg\_settings – список параметров

postgres=# SELECT \* FROM pg\_settings; (329 параметров на 28.08.2021):

ENABLE SEQSCAN – булев тип;

shared\_buffers – целочисленный тип с указанием единиц измерения;

ERROR, WARNING – перечисления (предопределённые значения);

cpu\_operator\_cost – вещественный тип с плавающей запятой;

строковый.

Контекст:

Внутренний требует перекомпиляции кода, либо новой инициализации кластера;

Серверный – требует перезагрузки сервера (параметры хранятся в postgresql.conf);

Sighup – изменяется файл postgresql.conf и посылается сигнал sighup серверному процессу postgres. Более безопасный вариант – использование функции pg\_reload\_conf(), а также скрипт init. [ psql -U postgres -c “SELECT pg\_reload\_conf()” ];

Заднего плана – устанавливается для сеанса;

Суперпаользовательский – задаётся суперпользователем в postgresql.conf или командой SET (просмотр командой SHOW);

Пользовательский – аналогичен суперпользовательскому, обычно задаётся на уровне сеанса.

Установка права **только чтение** для некоторых клиентов в текущем сеансе:

postgres=# SET default\_trunsaction\_read\_only to on;

Но если параметр default\_trunsaction\_read\_only задать в файле postgresql.conf, то настройка станет глобальной.

Просмотр таблицы в каталоге можно проводить двумя способами:

1. SELECT \* FROM pg\_catalog.pg\_database;
2. TABLE pg\_catalog.pg\_database;

Показ пути поиска перечисления схем, в которых сервер ищет имена объектов:

postgres=# SHOW search\_path;

Если существует схема с таким же именем, как у текущего пользователя, то объекты ищутся сначала в ней и в ней же будут они создаваться.

Сообщать полную информацию о сообщаемых ошибках

postgres=# \set VERBOSITY verbose

Дополнительные команды

\setenv

\с – открыть новое соединение с БД и закрыть текущее, с указанием роли и пароля.

\ef - генерирует шаблон:

CREATE FUNCTION()

RETURNS

LANGUAGE

AS $function$

\d+ [pattern] – вывод всей информации об отношении (таблица, представление, последовательность, индекс). Например:

car\_portal=> \d account\_information

\df+ [pattern] – описывает функцию

\dE[S+] [ *######* ]

\di[S+] [ *######* ]

\dm[S+] [ *######* ]

\ds[S+] [ *######* ]

\dt[S+] [ *######* ]

\dv[S+] [ *######* ]

В этой группе команд буквы E, i, m, s, t и v обозначают соответственно: внешнюю таблицу, индекс, материализованное представление, последовательность, таблицу и представление. Можно указывать все или часть этих букв, в произвольном порядке, чтобы получить список объектов этих типов. Например, \dit выводит список таблиц и индексов. При добавлении + к имени команды для каждого объекта дополнительно будут выводиться состояние хранения (постоянный, временный, нежурналируемый), физический размер на диске и описание, при его наличии. Если указан *######*, выводятся только объекты, имена которых соответствуют ему. По умолчанию показываются только объекты, созданные пользователями. Для включения системных объектов нужно задать шаблон или добавить модификатор S.

\z [pattern] – выводит права доступа к отношению

\l[+] или \list[+] [ *######* ] - выводит список баз данных на сервере и показывает их имена, владельцев, кодировку набора символов и права доступа. Если указан *######*, выводятся только базы данных, имена которых соответствуют ему. При добавлении + к команде также отображаются: размер базы данных, табличное пространство по умолчанию и описание. (Информация о размере доступна только для баз данных, к которым текущий пользователь может подключиться.)

-l или –list - выводит список всех доступных баз данных и завершает работу. Другие параметры, не связанные с соединением, игнорируются. Это похоже на метакоманду \list. Когда используется этот аргумент, psql будет подключаться к базе данных postgres, если только в командной строке не задана другая база данных (в параметре -d или не через параметры, а, например, через запись службы, но не через переменную окружения).

-q – не выводить приветственное или информационное сообщение

-t – выводить кортежи без заголовков

-X – игнорировать конфигурационный файл ~/.psqlrc

-o *###\_#####* или--output=*###\_#####* - записывает вывод результатов всех запросов в файл *###\_#####*. Эквивалентно команде \o. Если файл не указан, вывод производится на экран (стандартный вывод).

\o или \out [ *###\_#####* ] или \o или \out [ |*#######* ] - результаты запросов будут сохраняться в файле *###\_#####* или перенаправляться команде оболочки (заданной аргументом *#######*). Если аргумент не указан, результаты запросов перенаправляются на стандартный вывод. Если аргумент начинается с |, весь остаток строки воспринимается как *#######*, подлежащая выполнению, в которой не производится ни подстановка переменных, ни раскрытие обратных кавычек. Это продолжение строки просто передаётся оболочке в буквальном виде. «Результаты запросов» включают в себя все таблицы, ответы команд, уведомления, полученные от сервера баз данных, а также вывод от метакоманд, обращающихся к базе (таких как \d), но не сообщения об ошибках. Чтобы вставить текст между результатами запросов, используйте \qecho. Для отмены действия вывода в файл необходимо ввести \o без аргументов.

Пример:

postgres=# \o I:/pg\_database\_my\_Postgre.txt

-F – задаёт разделитель полей. Генерирует CSV-файл

\conninfo - выводит информацию о текущем подключении к базе данных.

\encoding [ ######### ] - устанавливает кодировку набора символов на клиенте. Без аргумента команда показывает текущую кодировку.

\pset format [aligned, asciidoc, csv, html, latex, latex-longtable, troff-ms, unaligned, wrapped] – устанавливает формат вывода. По умолчанию aligned.

-H, --html - переключает в режим вывода HTML. Равнозначно команде \pset format html или \H.

\H или \html - включает вывод запросов в формате HTML. Если формат HTML уже включён, происходит переключение обратно на выровненный формат.

-A – не выравнивать вывод. Равнозначно команде \pset format unaligned.

\i или \include ###\_##### - читает ввод из файла ###\_##### и выполняет его, как будто он был набран на клавиатуре. Если ###\_##### задано как - (минус), читается стандартный ввод до признака конца файла или до метакоманды \q. Это может быть полезно для совмещения интерактивного ввода со вводом команд из файлов. Заметьте, что при этом поведение Readline будет применяться, только если оно активно на внешнем уровне. Если вы хотите видеть строки файла на экране по мере их чтения, необходимо установить для переменной ECHO значение all.

PGOPTIONS – позволяет задать параметры, передаваемые серверу на этапе выполнения и определяющие его поведение

PROMT1, PROMT2, PROMT3 – переменные для задания пользовательских предпочтений

\set PROMT1 ‘(%n@%M:%>) [%/]%R%#%x > ‘ (postgres@[local]:5432) [postgres]=# > BEGIN;

%M – полное имя узла

%> - номер порта PostgreSQL

%n – имя текущего пользователя базы данных

%/ - имя текущей базы данных

%R – заменяется знаком =, но если сеанс прерван, то знаком !

%# - отличает суперпользователя (#) от обычного (>)

%x – состояние транзакции: \* - команда внутри блока транзакции, ! – в транзакции ошибка.

Поведение транзакций определяется параметрами:

ON\_ERROR\_ROLLBACK – по умолчанию параметр выключен (= off) и тогда при возникновении ошибки вся транзакция откатывается. Если = on, то ошибка игнорируется и транзакция выполняется. Если ON\_ERROR\_ROLLBACK=interactive, то ошибки игнорируются в интерактивных сеансах, но не при чтении скрипта из файла.

ON\_ERROR\_STOP – по умолчанию psql продолжает выполнение команд после возникновения. Ошибки. При разработке приложения этот параметр нужно включить: ON\_ERROR\_STOP = ON. Чаще используется с командами -f, \i, \ir.

AUTOCOMMIT – SQL-команды, не включённые явно в транзакционный блок, фиксируются автоматически. Включать не рекомендуется, тогда остаётся возможность выполнить rollback и отменить нежелательные изменения.

\timing – показывает время выполнения и используется для грубой оценки производительности.

log\_destination = ‘csvlog’

logging\_collector = on

log\_filename = ‘PostgreSQL.log’

log\_statement = ‘all’

service PostgreSQL restart – перезапуск PostgreSQL

DDL-скрипт создания объектов, являющихся частью схемы

DDL- скрипт создания объектов, не являющихся частью схемы (представления, функции)

DML-скрипт заполнения таблиц статическими данными

DCL-скрипт безопасности

Flywaydb – миграция с интеграцией с Git.

По умолчанию новая база данных создаётся по шаблону template1. Шаблонная база включает часть системной схемы pg\_catalog, а именно: таблицы, представления и функции.

Шаблонная БД template0 используется для ремонта шаблона template1, для восстановления резервной копии. Она не содержит данных, зависящих от кодировки и локали.

**РОЛИ**

Роли имеют атрибуты:

superuser – обходит все проверки, кроме login;

login – разрешено подключаться к БД;

created – разрешено создавать БД;

createrole – разрешено создавать, изменять и удалять другие роли;

replication – разрешено проводить потоковую репликацию;

password – аутентификация роли методом md5;

connection limit – задаёт максимальное число подключений;

inherit – наследует разрешения ролей, членом которых она является;

bypassrls – обходит механихм безопасности на уровне строк (RLS).

CREATE USER = CREATE ROLE с атрибутом LOGIN

CREATE GROUP = CREATE ROLE с атрибутом NOLOGIN

Пример создания роли car\_portal\_role и БД car\_portal, у которой кодировка не совпадает с исходной. Установление владельцем данной БД роль car\_portal\_role.

CREATE ROLE car\_portal\_role LOGIN;

CREATE DATABASE car\_portal

ENCODING `UTF-8`

LC\_COLLATE `en\_US.UTF-8`

LC\_TYPE `en\_US.UTF-8`

TEMPLATE template0

OWNER car\_portal\_role;

CREATE SCHEMA car\_portal\_app AUTORIZATION car\_portal\_role;

Если название схемы не указано, то оно совпадает с именем владельца.

В многопользовательской среде с несколькими БД необходимо отозвать у всех пользователей разрешение создавать объекты в схеме public (во вновь созданной базе или ы ШБД template1):

REVOKE CREATE ON SCHEMA public FROM PUBLIC;

При работе с порядковыми типами необходимо выдать разрешения на доступ к последовательностям.

Если задаётся принадлежность последовательности столбцу, то при удалении столбца удаляется и последовательность:

ALTER SEQUENCE name\_sequence OWNED BY name\_table.name\_column;

**ТИПЫ** **ДАННЫХ**

name\_var varchar(n) эквивалентен

name\_var text,

CONSTRAINT name\_constraint CHECK(length(name\_var) <= n) ;

Задание формата отображения типов timestamp и timestamptz и их интерпретация при помощи параметра datestyle.

Представления

pg\_timezone\_names – выдаёт полный список полных названий часовых поясов,

pg\_timezone\_abbrevs – выдаёт полный список сокращённых названий часовых поясов.

SELECT \* FROM pg\_timezone\_names;

Установка часового пояса для региона:

postgres=# SET timezone TIO `name\_region`;

вывод текущей даты и времени

postgres=# SELECT now();

вывод часового пояса:

postgres=# SHOW timezone;

преобразование временной метки с часовым поясом или без в указанный часовой пояс при помощи команды AT TIME ZONE, поведение которой зависит от преобразуемого типа:

postgres=# SELECT now() AT TIME ZONE 'CST';

Здесь происходит приведение к центральному стандартному времени (CST), которое смещено на -6 часов от универсального координированного времени (UTC)

При приведении типа timestamptz к timestamp, например, now()::timestamp смещение часового пояса отбрасывается.

Преобразование y типа timestamp в часовой пояс сеанса из часового пояса x:

`y`:: timestamp WITHOUT TIMEZONE AT TIME ZONE `X`

Преобразование y типа timestamptz в значение типа timestamp часового пояса x:

`y`:: timestamp WITH TIMEZONE AT TIME ZONE `X`

**ПРЕДСТАВЛЕНИЯ**

Чем больше в представлении агрегированных таблиц, тем ниже производительность. Сложную программную логику лучше перенести на уровень приложения, а не реализовывать её в базе данных с помощью представлений и хранимых процедур.

Невозможно удалить или структурно изменить представление, от которого зависят другие представления.

Категории представлений:

- временные представления

- рекурсивные представления, включая рекурсивные общие табличные выражения (CTE)

- обновляемые представления

- материализованные представления

**Создание представления:**

CREATE [OR REPLACE] [TEMP | TEMPORARY] [RECURSIVE] VIEW name\_view

[ (column\_name1, …) ]

[WITH (view\_option\_name [=view\_option\_value] [,…] ) ] AS

Тело запроса

[WITH [CASCADED | LOCAL] CHECK OPTION];

Простое представление является автообновляемым, т.е. если:

- представление построена над одной таблицей или над одним обновляемым представлением;

- определение представления не содержит на верхнем уровне: DISTINCT, WITH, GROUP BY, OFFSET, HAVING, LIMIT, UNION, EXCEPT, INTERSECT;

- в списке SELECT находятся только столбцы базовой таблицы без использования функций и выражений. Столбцы не повторяются;

- не должно быть установлено свойство security\_barrier.

Проверка, является ли представление автообновляемым:

car\_portal=> SELECT table\_name, is\_insertable\_into FROM information\_schema.tables WHERE table\_name = 'user\_account';

WITH CHECK OPTION – управляет поведением автообновляемых представлений. Если этой фразы нет, то команды UPDATE и INSERT успешно выполнятся, даже если строка не видна в представлении, что не безопасно.

**Создание материализованного представления:**

CREATE MATERIALIZED VIEW [IF NOT EXISS] name\_mview

[ (column\_name1, …) ]

[WITH (storage\_parameter [=value] [,…] ) ]

[TABLESPACE tablespace\_name] AS

Тело запроса

[WITH [NO] DATA];

Если при создании материализованного представления оно было пустым, то заполнить его данными можно так:

REFRESH MATERIALIZED VIEW [CONCURRENTLY] name\_mview [WITH [NO] DATA];

Обращение к представлению:

TABLE name\_mview;

При обновлении представления другие команды SELECT блокируются. Чтобы этого не происходило, над представлением должен быть построен уникальный индекс, тогда обновление будет происходить в параллельном режиме.

**Индексы**

Физические объекты БД, необходимые для оптимизации производительности и контроля ограничений.

Создание индекса

CREATE [UNIQUE] INDEX [CONCURRENTLY] [ [IF NOT EXISTS] name ]

ON table\_name [USING method]

( { column\_name | (expression) }

[COLLATE collation] [opclass] [ASC | DESC] [NULLS {FIRST | LAST} ] [, …] )

[ WITH ( storage\_parameter = value [, …] ) ]

[TABLESPACE tablespace\_name]

[WHERE predicate]

Автоматически создаются для первичных и уникальных ключей.

Индекс может строиться по столбцам или выражениям.

Типы индексов:

**B-дерево** – сбалансированное дерево (по умолчанию). Эффективен для поиска по условию равенства, принадлежности диапазону и совпадения с null. Строится для любых типов данных.

**Хеш-индекс** – на основе хеш-таблиц. Полезен для поиска по условию равенства.

**Обобщённый обратный индекс (GIN)** предназначен при отображении нескольких значений на одну строку. Используется со сложными структурами данных (массивы) и для полнотекстового поиска.

**Обобщённое дерево поиска (GiST)** – обобщённые сбалансированные древовидные структуры. Эффективен для индексации геометрических типов данных и для полнотекстового поиска.

Использование GIST-индекса для запрета перекрывающихся диапазонов дат:

CREATE TABLE no\_data\_overlap

( date\_range daterange,

EXCLUDE USING GIST (date\_range WITH &&)

) ;

**Обобщённое дерево поиска с двоичным разбиением пространства (SP-GiST)** поддерживает деревья поиска по двоичному разбиению пространства. Для сложных пользовательских данных.

**Блочно-диапазонный индекс (BRIN)** полезен для очень больших таблиц при ограниченном месте на диске. Работает медленнее сбалансированного дерева, но занимает меньше места на диске.

Категории индексов:

**Частичный** индекс – индексирует подмножество таблицы по заданному предикату (WHERE), что уменьшает размер индекса и увеличивает скорость выборки.

**Уникальный** индекс – для уникального ключа.

**Индекс по нескольким столбцам** – для поддержки запросов определённого вида:

SELECT \* FROM table\_name WHERE c1=v1 AND c2=v2 AND … cn=vn ;

Где n < 33.

По нескольким столбцам можно строить индексы типов СБ-дерева, ООИ, ОДП и БДИ. При создании многостолбцового индекса важен порядок столбцов. Может быть не эффективным.

**Индекс по выражению**, включая вызовы функции. Используется, если выражение во фразе WHERE в точности совпадает с выражением, по которому строился индекс. Также используется для фильтрации строк после приведения к другому типу данных (поиск по дате в столбце с датой и временем).

Если таблица мала, а также если имя недостаточно уникально при фильтрации, то индексы не используются (при их наличии). Планировщик просматривает таблицу целиком.

Наличие индексов снижает скорость вставки.

Каталожные таблицы и функции:

pg\_stat\_all\_indexes – таблица хранения статистики использования индексов.

Проверяйте наличие индекса, так как не возбраняется создавать несколько индексов по одному и тому же столбцу.

Если индекс разросся, то его следует перестроить при помощи команды REINDEX. Она является блокирующей, поэтому чтобы перестройка выполнялась в параллельном режиме необходимо создать ещё один точно такой же индекс, а первый перестроить:

REINDEX index name\_shema.name\_index

Второй способ параллельного создания индекса:

CREATE UNIQUE INDEX CONCURRENTLY ON name\_schema.name\_table(c1, c2, c3) ;

Третий способ – удалить индекс. Если он был уникальным, то сначала удаляется ограничение в таблице, а затем сам индекс:

DROP INDEX

После чего восстановить ограничение в таблице и добавить на него уникальный индекс.

**Ключевые слова и функции**

ALTER TABLE {OWNER}

ANALYZE

AND

AS

ASC

AVG(name\_column)

BETWEEN xxx AND yyy – как эквивалент для >= AND <=

CHECK

CLUSTER

COLLATE

COMMENT ON [DATABASE | SCHEMA] nameDB | nameS

CONCAT

CONCURRENTLY

COUNT

CREATE DATABASE {WITH OWNER ENCODING LC\_COLLATE LC\_CTYPE TABLESPACE (CONNECTION LIMIT)}

CREATE EXTENSION – загрузка расширения

CREATE FOREIGN TABLE name\_log (…) SERVER name\_fileserver OPTIONS (…) – создание FDW-таблицы, связанной с файлом журнала, находящегося в папке кластерного каталога PostgreSQL

CREATE [OR REPLACE] FUNCTION name\_function(p1, p2, …) RETURNS type AS $$  
BEGIN …. END; $$ LANGUAGE PLPGSQL;

CREATE LANGUAGE

CREATE OPERATOR symbol (…) – создание оператора

CREATE SCHEMA name\_schema AUTHORIZATION имя пользователя

CREATE SERVER name\_fileserver FOREIGN DATA WRAPPER name\_fdw – создание FDW-сервера

CREATE TABLE {IF NOT EXISTS} ADD

CREATE TABLESPACE

CREATE TYPE name\_type AS (…)– создание типа данных

CUBE

DESC

DISTINCT

DROP DATABASE

EXCEPT {ALL} аналог А\В {с дубликатами} для данных в столбце

EXPLAIN

FIRST

FROM

FUNCTION

GRANT {ALL} ON [DATABASE | SCHEMA] nameDB | nameS TO имя пользователя

GROUP BY

GROUPING SETS

HAVING

ILIKE

IN {NOT IN}

INDEX

INSERT INTO name\_table VALUES (), () …

INTERSECT аналог ∩ для данных в столбце

JOIN { INNER JOIN аналог ∩ для таблиц

| NATURAL JOIN заменяет INNER c USING – объединение по одинаково называющимся столбцам

| CROSS JOIN аналог × без ON !

| FULL OUTER JOIN аналог ∪

| LEFT OUTER JOIN первое множество

| RIGHT OUTER JOIN второе множество } name\_tbl ON t1.c1.=t2.c1

LAST

LEFT

LIKE

LIMIT

MAX(name\_column)

MIN(name\_column)

NOT

NULL {IS {NOT} NULL)

NULLS

OFSET

OR

ORDER BY name\_column1, column2, … {DESC, ASC)

OWNED BY

PROCEDURE

REVOKE

RIGHT

ROLLUP

SEQUENCE

SELECT {COUNT({{DISTINCT}\*/name\_column)}

SELECT {DISTINCT {\*/name\_column1, name\_column2, … } FROM {WHERE}} {GROUP BY} {HAVING} {ORDER BY}

SET

SHOW

SUM(name\_column)

TABLE

TABLESPACE

UNION {ALL} аналог ∪ {с повторениями} для данных в столбце

USING

VACUUM

WHERE

WITH

|/ квадратный корень

% - заменяет несколько символов

\_ - заменяет один символ

integer

varchar, varchar(n)

char, char(n)

text

data

year

Получить список установленных расширений (extensions) в базе:

testdb=# select \* from pg\_available\_extensions where installed\_version is not null;

Оконные функции:

row\_number()

Синтаксис

**SELECT**

[ WITH [ RECURSIVE ] *######\_WITH* [, ...] ]

SELECT [ ALL | DISTINCT [ ON ( *#########* [, ...] ) ] ]

[ \* | *#########* [ [ AS ] *###\_##########* ] [, ...] ]

[ FROM *#######\_FROM* [, ...] ]

[ WHERE *#######* ]

[ GROUP BY *#######\_#############* [, ...] ]

[ HAVING *#######* ]

[ WINDOW *###\_####* AS ( *###########\_####* ) [, ...] ]

[ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] *#######* ]

[ ORDER BY *#########* [ ASC | DESC | USING *########* ] [ NULLS { FIRST | LAST } ]

[, ...] ]

[ LIMIT { *#####* | ALL } ]

[ OFFSET *######* [ ROW | ROWS ] ]

[ FETCH { FIRST | NEXT } [ *#####* ] { ROW | ROWS } { ONLY | WITH TIES } ]

[ FOR { UPDATE | NO KEY UPDATE | SHARE | KEY SHARE } [ OF *###\_#######* [, ...] ]

[ NOWAIT | SKIP LOCKED ] [...] ]

1. Сначала выполняются запросы в списке WITH, которые формируют временные таблицы или представления. Предложение WITH позволяет задать один или несколько подзапросов, к которым затем можно обратиться по имени в основном запросе. При использовании в WITH оператора, изменяющего данные, (INSERT, UPDATE или DELETE) обычно добавляется предложение RETURNING. Заметьте, что именно результат RETURNING, а *не* нижележащая таблица, изменяемая запросом, формирует временную таблицу, которую затем читает основной запрос. Если RETURNING опущено, оператор, тем не менее, выполняется, но не выдаёт никакого результата, так что на него нельзя сослаться как на таблицу в основном запросе. Действие оператора, изменяющего данные в WITH, не будут видеть другие части запроса, кроме как прочитав его вывод RETURNING.

# *######\_WITH*:

*###\_#######\_WITH* [ ( *###\_#######* [, ...] ) ] AS [ [ NOT ] MATERIALIZED ] ( *#######*

| *values* | *insert* | *update* | *delete* )

SELECT

TABLE [ ONLY ] *###\_#######* [ \* ]

Если указано RECURSIVE, подзапрос SELECT может ссылаться сам на себя по имени. Такой подзапрос должен иметь форму

*#############\_#####* UNION [ ALL | DISTINCT ] *###########\_#####*

где рекурсивная ссылка на сам запрос может находиться только справа от UNION. Для одного запроса допускается только одна рекурсивная ссылка на него же. Операторы, изменяющие данные, не могут быть рекурсивными, но результат рекурсивного запроса SELECT в таких операторах можно использовать.

Ещё одна особенность RECURSIVE в том, что запросы WITH могут быть неупорядоченными: запрос может ссылаться на другой, идущий в списке после него. Без RECURSIVE запрос в WITH может ссылаться только на запросы того же уровня в WITH, предшествующие ему в списке WITH.

Ключевое свойство запросов WITH состоит в том, что они обычно вычисляются один раз для всего основного запроса, даже если в основном запросе содержатся несколько ссылок на них. Однако от этой гарантии можно отказаться, добавив для запроса WITH пометку NOT MATERIALIZED. В этом случае запрос WITH может быть свёрнут в основной запрос, как если бы это был простой SELECT внутри предложения FROM основного запроса. В результате запрос WITH может вычисляться неоднократно, если основной запрос обращается к нему несколько раз. Но если при каждом таком обращении требуются лишь отдельные строки из всего результата запроса WITH, указание NOT MATERIALIZED, позволяющее оптимизировать запросы совместно, в целом может оказаться выгодным. Указание NOT MATERIALIZED игнорируется в запросе WITH, который имеет рекурсивный характер или не свободен от побочных эффектов (то есть, когда это не простой SELECT без изменчивых функций). По умолчанию запрос WITH, свободный от побочных эффектов, заворачивается в основной запрос, если он используется в его предложении FROM ровно один раз. Это позволяет совместно оптимизировать два уровня запросов в ситуациях, когда семантика запроса в целом сохраняется. Но это заворачивание можно предотвратить, пометив запрос WITH как MATERIALIZED. Это может быть полезно, например когда предложение WITH применяется как преграда для оптимизатора, не позволяющая ему выбрать неудачный план.

1. Затем вычисляются все элементы в списке FROM, исключаются кортежи, неудовлетворяющие условию WHERE.

При наличии группировки GROUP BY вывод разделяется на групповые строки, для каждой из которых рассчитываются агрегатные функции. После этого производится отсеивание групп по условию HAVING.

SELECT ALL – является по умолчанию и выводит все строки, включая дубликаты

SELECT DISTINCT – исключает дубликаты строк

SELECT DISTINCT ON (имя-столбца1, имя\_столбца2, …) – исключает дубликаты строк по указанным столбцам

Пример:

**SELECT** **DISTINCT** **ON** (a2."DETECTION") a2."DETECTION"

**FROM** "APPDETECTION" a2

Операторы UNION, INTERSECT и EXCEPT объединяют вывод нескольких команд SELECT в один результирующий набор.

Оператор UNION возвращает все строки, представленные в одном, либо обоих наборах результатов.

Оператор INTERSECT возвращает все строки, представленные строго в обоих наборах. Оператор EXCEPT возвращает все строки, представленные в первом наборе, но не во втором. Во всех трёх случаях повторяющиеся строки исключаются из результата (по умолчанию стоит DISTINCT), если явно не указано ALL.

Предложение ORDER BY возвращаемые строки сортирует в указанном порядке. В отсутствие ORDER BY строки возвращаются в том порядке, в каком системе будет проще их выдать.

Если указано предложение LIMIT (или FETCH FIRST) и/либо OFFSET, оператор SELECT возвращает только подмножество строк результата.

Если указано FOR UPDATE, FOR NO KEY UPDATE, FOR SHARE или FOR KEY SHARE, оператор SELECT блокирует выбранные строки, защищая их от одновременных изменений. Применение блокировок FOR NO KEY UPDATE, FOR UPDATE, FOR SHARE или FOR KEY SHARE требует прав SELECT и UPDATE (как минимум для одного столбца в каждой выбранной для блокировки таблице).

Для всех столбцов, задействованных в команде SELECT, необходимо иметь право SELECT.

**Создание базы данных для сайта по продаже автомобилей**

**Создать роль:**

postgres=# CREATE ROLE car\_portal\_role LOGIN;

Создать базу давнных:

CREATE DATABASE car\_portal

WITH

OWNER = car\_portal\_role

ENCODING = 'UTF8'

LC\_COLLATE = 'Russian\_Russia.1251'

LC\_CTYPE = 'Russian\_Russia.1251'

TEMPLATE template0

TABLESPACE = pg\_default

CONNECTION LIMIT = -1 ;

COMMENT ON DATABASE car\_portal

IS 'Учебная база данных для сайта по продаже автомобилей';

ALTER ROLE car\_portal\_role PASSWORD '12345'; -- назначение пароля

Войти в базу данных:

C:\> psql -U car\_portal\_role car\_portal

**Создание схемы:**

CREATE SCHEMA car\_portal\_app

AUTHORIZATION car\_portal\_role;

COMMENT ON SCHEMA car\_portal\_app

IS 'основной сайт';

**Назначение прав:**

-- GRANT ALL ON SCHEMA car\_portal\_app TO PUBLIC;

GRANT ALL ON SCHEMA car\_portal\_app TO car\_portal\_role;

GRANT ALL ON SCHEMA car\_portal\_app TO mihaildb;

**Установка созданной схемы в качестве основной** (при обращении к таблицам без указания названия схемы в их имени поиск этих таблиц будут вестись по этой схеме)

car\_portal=> set search\_path=car\_portal\_app;

**Создание таблиц:**

CREATE TABLE car\_portal\_app.account (

account\_id SERIAL PRIMARY KEY,

first\_name TEXT NOT NULL,

last\_name TEXT NOT NULL,

email TEXT NOT NULL UNIQUE,

password TEXT NOT NULL,

CHECK(email ~\* '^\w+@\w+[.]\w+$'),

CHECK(char\_length(password)>=9)

) ;

CREATE TABLE car\_portal\_app.seller\_account (

seller\_account\_id SERIAL PRIMARY KEY,

account\_id INT UNIQUE NOT NULL REFERENCES account(account\_id),

number\_of\_advertizement INT DEFAULT 0,

user\_ranking FLOAT,

total\_rank FLOAT

) ;

CREATE TABLE account\_history (

account\_id INT NOT NULL REFERENCES account(account\_id),

search\_key text NOT NULL,

search\_date date NOT NULL,

CONSTRAINT account\_history\_unique UNIQUE (account\_id, search\_key, search\_date)

) ;

**Создание самоссылающейся таблицы:**

CREATE TABLE employee

( employee\_id INT PRIMARY KEY,

supervisor\_id INT,

CONSTRAINT supervisor\_id\_fkey FOREIGN KEY (supervisor\_id) REFERENCES employee(employee\_id)

) ;

Создание уникального индекса для ограничения на одну запись без начальника:

CREATE UNIQUE INDEX ON employee((1)) WHERE supervisor\_id IS NULL ;

**Создание представления:**

CREATE OR REPLACE VIEW account\_information

(account\_id, first\_name, last\_name, email) AS

SELECT account\_id, first\_name, last\_name, email

FROM account;

**Создание обновляемого представления:**

CREATE VIEW user\_account AS

SELECT account\_id, first\_name, last\_name, email, password

FROM account

WHERE account\_id NOT IN (

SELECT account\_id

from seller\_account

) ;

Одновременная вставка учётной записи и записи продавца:

WITH account\_info AS

( INSERT INTO user\_account

VALUES (default, ‘first\_name2’, ‘last\_name2’, ‘test2@email.com’, ‘password’)

RETURNING account\_id)

INSERT INTO seller\_account

(account\_id, street\_name, street\_number, zip\_ code, city)

SELECT account\_id, ‘street1’, ‘555’, ‘555’, ‘test\_city’

FROM account\_info;

Удалить данную вставку с помощью обновляемого представления не получится из-за наличия проверочных ограничений.