Бесплатная электронная книга

учусь

postgresql

Free unaffiliated eBook created from
Stack Overflow contributors.

#postgresql

Около

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: postgresql

It is an unofficial and free postgresql ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official postgresql.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

глава 1: Начало работы с postgresql

замечания

В этом разделе представлен обзор того, что такое postgresql, и почему разработчик может захотеть его использовать.

Следует также упомянуть о любых крупных предметах в postgresql и ссылаться на связанные темы. Поскольку документация для postgresql нова, вам может потребоваться создать начальные версии этих связанных тем.

Версии

Версия	Дата выхода	Дата EOL
9,6	2016-09-29	2021-09-01
9,5	2016-01-07	2021-01-01
9,4	2014-12-18	2019-12-01
9,3	2013-09-09	2018-09-01
9,2	2012-09-10	2017-09-01
9,1	2011-09-12	2016-09-01
9,0	2010-09-20	2015-09-01
8,4	2009-07-01	2014-07-01

Examples

Установка на GNU + Linux

В большинстве операционных систем GNU + Linux PostgreSQL можно легко установить с помощью диспетчера пакетов операционной системы.

Семейство Red Hat

Репозитории можно найти здесь: https://yum.postgresql.org/repopackages.php

Загрузите репозиторий на локальный компьютер с помощью команды

```
yum -y install https://download.postgresql.org/pub/repos/yum/X.X/redhat/rhel-7-x86_64/pgdg-redhatXX-X.X-X.noarch.rpm
```

Просмотр доступных пакетов:

```
yum list available | grep postgres*
```

Пакеты Neccesary: postgresqlXX postgresqlXX-server postgresqlXX-libs postgresqlXX-contrib

Они устанавливаются с помощью следующей команды: yum -y install postgresqlXX postgresqlXX-server postgresqlXX-libs postgresqlXX-contrib

После установки вам нужно будет запустить службу базы данных в качестве владельца сервиса (по умолчанию - postgres). Это делается с помощью команды pg_ctl.

```
sudo -su postgres
./usr/pgsql-X.X/bin/pg_ctl -D /var/lib/pgsql/X.X/data start
```

Для доступа к DB в CLI введите `psql`

Семейство Debian

В Debian и производных операционных системах введите:

```
sudo apt-get install postgresql
```

Это установит пакет сервера PostgreSQL в версии по умолчанию, предлагаемой репозиториями пакета операционной системы.

Если версия, установленная по умолчанию, не та, которая вам нужна, вы можете использовать диспетчер пакетов для поиска конкретных версий, которые могут одновременно предлагаться.

Вы также можете использовать репозиторий Yum, предоставленный проектом PostgreSQL (известный как PGDG), чтобы получить другую версию. Это может позволить версии, еще не предлагаемые репозиториями системных пакетов.

Как установить PostgreSQL через MacPorts в OSX

Чтобы установить PostgreSQL на OSX, вам нужно знать, какие версии в настоящее время поддерживаются.

Используйте эту команду, чтобы узнать, какие версии у вас есть.

```
sudo port list | grep "^postgresql[[:digit:]]\{2\}[[:space:]]"
```

Вы должны получить список, который выглядит примерно так:

```
postgresql80                    @8.0.26         databases/postgresql80
postgresql81                    @8.1.23         databases/postgresql81
postgresql82                    @8.2.23         databases/postgresql82
postgresql83                    @8.3.23         databases/postgresql83
postgresql84                    @8.4.22         databases/postgresql84
postgresql90                    @9.0.23         databases/postgresql90
postgresql91                    @9.1.22         databases/postgresql91
postgresql92                    @9.2.17         databases/postgresql92
postgresql93                    @9.3.13         databases/postgresql93
postgresql94                    @9.4.8          databases/postgresql94
postgresql95                    @9.5.3          databases/postgresql95
postgresql96                    @9.6beta2       databases/postgresql96
```

В этом примере самая последняя версия PostgreSQL, которая поддерживается в 9.6, поэтому мы ее установим.

```
sudo port install postgresql96-server postgresql96
```

Вы увидите журнал установки следующим образом:

```
--->  Computing dependencies for postgresql96-server
--->  Dependencies to be installed: postgresql96
--->  Fetching archive for postgresql96
--->  Attempting to fetch postgresql96-9.6beta2_0.darwin_15.x86_64.tbz2 from
https://packages.macports.org/postgresql96
--->  Attempting to fetch postgresql96-9.6beta2_0.darwin_15.x86_64.tbz2.rmd160 from
https://packages.macports.org/postgresql96
--->  Installing postgresql96 @9.6beta2_0
--->  Activating postgresql96 @9.6beta2_0

To use the postgresql server, install the postgresql96-server port

--->  Cleaning postgresql96
--->  Fetching archive for postgresql96-server
--->  Attempting to fetch postgresql96-server-9.6beta2_0.darwin_15.x86_64.tbz2 from
https://packages.macports.org/postgresql96-server
--->  Attempting to fetch postgresql96-server-9.6beta2_0.darwin_15.x86_64.tbz2.rmd160 from
https://packages.macports.org/postgresql96-server
--->  Installing postgresql96-server @9.6beta2_0
--->  Activating postgresql96-server @9.6beta2_0

To create a database instance, after install do
 sudo mkdir -p /opt/local/var/db/postgresql96/defaultdb
 sudo chown postgres:postgres /opt/local/var/db/postgresql96/defaultdb
 sudo su postgres -c '/opt/local/lib/postgresql96/bin/initdb -D
/opt/local/var/db/postgresql96/defaultdb'

--->  Cleaning postgresql96-server
--->  Computing dependencies for postgresql96
--->  Cleaning postgresql96
--->  Updating database of binaries
--->  Scanning binaries for linking errors
--->  No broken files found.
```

Журнал содержит инструкции по остальным этапам установки, поэтому мы делаем

следующее.

```
sudo mkdir -p /opt/local/var/db/postgresql96/defaultdb
sudo chown postgres:postgres /opt/local/var/db/postgresql96/defaultdb
sudo su postgres -c '/opt/local/lib/postgresql96/bin/initdb -D
/opt/local/var/db/postgresql96/defaultdb'
```

Теперь мы запускаем сервер:

```
sudo port load -w postgresql96-server
```

Убедитесь, что мы можем подключиться к серверу:

```
su postgres -c psql
```

Вы увидите приглашение от postgres:

```
psql (9.6.1)
Type "help" for help.

postgres=#
```

Здесь вы можете ввести запрос, чтобы увидеть, что сервер работает.

```
postgres=#SELECT setting FROM pg_settings WHERE name='data_directory';
```

И посмотрите ответ:

```
               setting
----------------------------------------
/opt/local/var/db/postgresql96/defaultdb
(1 row)
postgres=#
```

Введите \ q, чтобы выйти:

```
postgres=#\q
```

И вы вернетесь в окно командной строки.

Поздравляем! Теперь у вас есть исполняемый экземпляр PostgreSQL для OS / X.

Postgres.app для Mac OSX

Очень простой инструмент для установки PostgreSQL на Mac доступен, загружая Postgres.app .
Вы можете изменить настройки для запуска PostgreSQL в фоновом режиме или только при запуске приложения.

Установка PostgreSQL в Windows

Хотя хорошей практикой является использование операционной системы на базе Unix (например, Linux или BSD) в качестве производственного сервера, вы можете легко установить PostgreSQL в Windows (надеюсь, только в качестве сервера разработки).

Загрузите установочные файлы Windows из EnterpriseDB: http://www.enterprisedb.com/products-services-training/pgdownload. Это сторонняя компания, созданная основными разработчиками проекта PostgreSQL, которые оптимизировали двоичные файлы для Windows.

Выберите последнюю стабильную (не бета-версию) версию (9.5.3 на момент написания). Скорее всего, вы захотите получить пакет Win x86-64, но если вы используете 32-разрядную версию Windows, которая является обычной на старых компьютерах, вместо этого выберите Win x86-32.

Примечание. Переключение между версиями Beta и Stable будет включать сложные задачи, такие как свалка и восстановление. Обновление в бета-версии или стабильной версии требует перезапуска службы.

Вы можете проверить, равна ли ваша версия Windows 32 или 64 бит, выбрав «Панель управления -> Система и безопасность -> Система -> Тип системы, на которой будет указано« ## - bit Operating System ». Это путь для Windows 7, он может немного отличаться от других версий Windows.

В программе установки выберите пакеты, которые вы хотели бы использовать. Например:

- pgAdmin (https://www.pgadmin.org) - бесплатный графический интерфейс для управления вашей базой данных, и я настоятельно рекомендую его. В 9.6 это будет установлено по умолчанию.
- PostGIS (http://postgis.net) предоставляет функции геопространственного анализа в координатах GPS, дистанциях и т. Д., Которые очень популярны среди разработчиков ГИС.
- Языковой пакет предоставляет необходимые библиотеки для официально поддерживаемых процедурных языков PL / Python, PL / Perl и PL / Tcl.
- Другие пакеты, такие как pgAgent, pgBouncer и Slony, полезны для больших производственных серверов, проверяются только по мере необходимости.

Все эти дополнительные пакеты могут быть позже установлены через «Application Stack Builder».

Примечание. Существуют также другие языки, не поддерживаемые официально, такие как PL / V8 , PL / Lua PL / Java.

Откройте pgAdmin и подключитесь к своему серверу, дважды щелкнув его имя, например. «

PostgreSQL 9.5 (localhost: 5432).

С этого момента вы можете следить за руководствами, такими как отличная книга PostgreSQL: Up and Running, 2nd Edition (http://shop.oreilly.com/product/0636920032144.do).

Дополнительно: Тип запуска вручную

PostgreSQL работает как служба в фоновом режиме, которая немного отличается от большинства программ. Это характерно для баз данных и веб-серверов. Его Тип запуска по умолчанию является автоматическим, что означает, что он будет всегда запускаться без какого-либо ввода от вас.

Почему вы хотите вручную управлять службой PostgreSQL? Если вы иногда используете свой ПК в качестве сервера разработки и используете его для воспроизведения видеоигр, PostegreSQL может немного замедлить работу вашей системы.

Почему бы вам не ручное управление? Запуск и остановка службы может быть проблемой, если вы часто это делаете.

Если вы не заметили какой-либо разницы в скорости и предпочитаете избегать хлопот, оставьте его Тип запуска как Автоматический и игнорируйте остальную часть этого руководства. Иначе...

Перейдите в Панель управления -> Система и безопасность -> Администрирование.

Выберите «Службы» из списка, щелкните правой кнопкой мыши по его значку и выберите «Отправить -»> «Рабочий стол», чтобы создать значок рабочего стола для более удобного доступа.

Закройте окно «Администрирование», затем запустите «Службы» на рабочем столе, который вы только что создали.

Прокрутите вниз, пока не увидите службу с именем postgresql-x ## - 9. # (например, postgresql-x64-9.5 ").

Щелкните правой кнопкой мыши службу postgres, выберите «Свойства» -> «Тип запуска» -> «Вручную» -> «Применить» -> «ОК». Вы можете легко вернуть его в автоматическом режиме.

Если вы видите в списке другие связанные с PostgreSQL сервисы, такие как «pgbouncer» или «PostgreSQL Scheduling Agent - pgAgent», вы также можете изменить свой тип запуска на «Вручную», потому что они не очень полезны, если PostgreSQL не работает. Хотя это будет означать больше хлопот при каждом запуске и остановке, так что это зависит от вас. Они не используют столько ресурсов, сколько PostgreSQL, и могут не иметь заметного влияния на производительность вашей системы.

Если служба работает, ее статус будет указывать «Запущен», иначе он не будет запущен.

Чтобы запустить его, щелкните правой кнопкой мыши и выберите «Пуск». Появится запрос на загрузку и вскоре исчезнет. Если он даст вам ошибку, попробуйте второй раз. Если это не сработает, возникла некоторая проблема с установкой, возможно, из-за того, что вы изменили некоторые настройки в Windows, большинство людей не меняются, поэтому для поиска проблемы может потребоваться некоторая привязка.

Чтобы остановить postgres, щелкните правой кнопкой мыши службу и выберите «Стоп».

Если вы когда-нибудь получите сообщение об ошибке при попытке подключиться к своей базе данных, проверьте Службы, чтобы убедиться в ее работе.

Для других очень конкретных деталей об установке EDB PostgreSQL, например версии исполнения на языке python в официальном языковом пакете конкретной версии PostgreSQL, всегда обращайтесь к официальному руководству по установке EBD , изменяйте версию в ссылке на основную версию вашего установщика.

Установите postgresql с пивом на Mac

Homebrew называет себя « *отсутствующим менеджером пакетов для macOS* ». Его можно использовать для создания и установки приложений и библиотек. После установки вы можете использовать команду `brew` для установки PostgreSQL и ее зависимостей следующим образом:

```
brew update
brew install postgresql
```

Homebrew обычно устанавливает последнюю стабильную версию. Если вам нужен другой, то `brew search postgresql` перечислит доступные версии. Если вам нужен PostgreSQL, построенный с определенными параметрами, тогда `brew info postgresql` перечислит, какие параметры поддерживаются. Если вам нужна опция неподдерживаемой сборки, вам, возможно, придется самостоятельно создавать сборку, но она все равно может использовать Homebrew для установки общих зависимостей.

Запустите сервер:

```
brew services start postgresql
```

Откройте приглашение PostgreSQL

```
psql
```

Если psql жалуется, что для вашего пользователя нет соответствующей базы данных, запустите `createdb` .

Установка PostgreSQL из источника на Linux

зависимости:

- GNU Сделать версию> 3.80
- ISO / ANSI C-Compiler (например, gcc)
- экстрактор, такой как смола или gzip
- Zlib-разви
- readline-devel oder libedit-devel

Источники: ссылка на последний источник (9.6.3)

Теперь вы можете извлечь исходные файлы:

```
tar -xzvf postgresql-9.6.3.tar.gz
```

Существует множество различных параметров конфигурации PostgreSQL:

Полная ссылка на процедуру полной установки

Небольшой список доступных опций:

- `--prefix=PATH` путь `--prefix=PATH` для всех файлов
- `--exec-prefix=PATH` Путь `--exec-prefix=PATH` для файла зависимости от архитектуры
- `--bindir=PATH` путь `--bindir=PATH` для исполняемых программ
- `--sysconfdir=PATH` путь `--sysconfdir=PATH` для файлов конфигурации
- `--with-pgport=NUMBER` указывает порт для вашего сервера
- `--with-perl` добавить поддержку perl
- `--with-python` добавить поддержку python
- `--with-openssl` добавить поддержку openssl
- `--with-ldap` добавить поддержку ldap
- `--with-blocksize=BLOCKSIZE` установить размер `--with-blocksize=BLOCKSIZE` в КБ
    - `BLOCKSIZE` должен иметь мощность 2 и от 1 до 32
- `--with-wal-segsize=SEGSIZE` задает размер WAL-сегмента в MB
    - `SEGSIZE` должен иметь мощность 2 от 1 до 64

Перейдите в новую созданную папку и запустите скрипт cofigure с нужными параметрами:

```
./configure --exec=/usr/local/pgsql
```

Запустить `make` для создания объектных файлов

Запустить `make install` для установки PostgreSQL из встроенных файлов

Запустите `make clean` чтобы привести в порядок

Для расширения добавьте каталог `cd contrib` , запустите `make` и `make install`

Прочитайте Начало работы с postgresql онлайн: https://riptutorial.com/ru/postgresql/topic/885/

начало-работы-с-postgresql

глава 2: COALESCE

Вступление

Coalesce возвращает первый пустой аргумент из набора аргументов. Только первый ненулевой аргумент является возвратом, все последующие аргументы игнорируются. Функция будет оценивать значение null, если все аргументы равны нулю.

Examples

Единый непустой аргумент

```
PGSQL> SELECT COALESCE(NULL, NULL, 'HELLO WORLD');

coalesce
--------
'HELLO WORLD'
```

Множественные ненулевые аргументы

PGSQL> SELECT COALESCE (NULL, NULL, 'first non null', null, null, 'second non null');

```
coalesce
--------
'first non null'
```

Все нулевые аргументы

```
PGSQL> SELECT COALESCE(NULL, NULL, NULL);

coalesce
--------
```

Прочитайте COALESCE онлайн: https://riptutorial.com/ru/postgresql/topic/10576/coalesce

глава 3: EXTENSION dblink и postgres_fdw

Синтаксис

- dblink ('dbname = name_db_distance port = PortOfDB host = HostOfDB user = usernameDB password = passwordDB', 'MY QUESRY')
- dbname = имя базы данных
- port = Порт базы данных
- host = Host из базы данных
- user = имя пользователя базы данных
- пароль = пароль базы данных ',
- MY QUESRY = это может быть любая операция, которую я хочу сделать SELECT, INSERT, ...

Examples

Расширение dblink

dblink EXTENSION - это способ подключения другой базы данных и обеспечения работы этой базы данных, чтобы сделать это вам:

1-Создать расширение dblink:

```
CREATE EXTENSION dblink;
```

2-Сделайте свою операцию:

Например, выберите другой атрибут из другой таблицы в другой базе данных:

```
SELECT * FROM
dblink ('dbname = bd_distance port = 5432 host = 10.6.6.6 user = username
password = passw@rd', 'SELECT id, code FROM schema.table')
AS newTable(id INTEGER, code character varying);
```

Расширение FDW

FDW является импликации dblink, это более полезно, поэтому использовать его:

1-Создать расширение:

```
CREATE EXTENSION postgres_fdw;
```

2-Создать СЕРВЕР:

```
CREATE SERVER name_srv FOREIGN DATA WRAPPER postgres_fdw OPTIONS (host 'hostname',
dbname 'bd_name', port '5432');
```

3-Создание сопоставления пользователей для сервера postgres

```
CREATE USER MAPPING FOR postgres SERVER name_srv OPTIONS(user 'postgres', password
'password');
```

4-Создать внешнюю таблицу:

```
CREATE FOREIGN TABLE table_foreign (id INTEGER, code character varying)
SERVER name_srv OPTIONS(schema_name 'schema', table_name 'table');
```

5 - используйте эту внешнюю таблицу, как в вашей базе данных:

```
SELECT * FROM table_foreign;
```

Внешний обертка данных

Для доступа к полной схеме сервера db вместо отдельной таблицы. Выполните следующие шаги:

1. Создать EXTENSION:

```
CREATE EXTENSION postgres_fdw;
```

2. Создать СЕРВЕР:

```
CREATE SERVER server_name FOREIGN DATA WRAPPER postgres_fdw OPTIONS (host 'host_ip',
dbname 'db_name', port 'port_number');
```

3. Создание ПОЛЬЗОВАТЕЛЯ:

```
CREATE USER MAPPING FOR CURRENT_USER
SERVER server_name
OPTIONS (user 'user_name', password 'password');
```

4. Создайте новую схему для доступа к схеме серверной БД:

```
CREATE SCHEMA schema_name;
```

5. Импортировать схему сервера:

```
IMPORT FOREIGN SCHEMA schema_name_to_import_from_remote_db
FROM SERVER server_name
INTO schema_name;
```

6. Доступ к любой таблице схемы сервера:

```
SELECT * FROM schema_name.table_name;
```

Это можно использовать для доступа к множественной схеме удаленной БД.

Прочитайте EXTENSION dblink и postgres_fdw онлайн:
https://riptutorial.com/ru/postgresql/topic/6970/extension-dblink-и-postgres-fdw

глава 4: ВСТАВИТЬ

Examples

Основной INSERT

Допустим, у нас есть простая таблица, называемая человеком:

```
CREATE TABLE person (
    person_id BIGINT,
    name VARCHAR(255).
    age INT,
    city VARCHAR(255)
);
```

Самая основная вставка включает в себя вставку всех значений в таблицу:

```
INSERT INTO person VALUES (1, 'john doe', 25, 'new york');
```

Если вы хотите вставить только определенные столбцы, вам нужно явно указать, какие столбцы:

```
INSERT INTO person (name, age) VALUES ('john doe', 25);
```

Обратите внимание: если в таблице существуют какие-либо ограничения, например NOT NULL, вам необходимо будет включить эти столбцы в любом случае.

Вставка нескольких строк

Вы можете вставлять сразу несколько строк в базу данных:

```
INSERT INTO person (name, age) VALUES
  ('john doe', 25),
  ('jane doe', 20);
```

Вставить из выбранного

Вы можете вставлять данные в таблицу в результате оператора select:

```
INSERT INTO person SELECT * FROM tmp_person WHERE age < 30;
```

Обратите внимание, что проекция выбора должна соответствовать столбцам, необходимым для вставки. В этом случае таблица `tmp_person` имеет те же столбцы, что и `person` .

Вставить данные с помощью COPY

COPY - механизм вставки вставки PostgreSQL. Это удобный способ передачи данных между файлами и таблицами, но он также намного быстрее, чем `INSERT` при добавлении более нескольких тысяч строк за раз.

Начнем с создания образца файла данных.

```
cat > samplet_data.csv

1,Yogesh
2,Raunak
3,Varun
4,Kamal
5,Hari
6,Amit
```

И нам нужна таблица с двумя столбцами, в которую эти данные могут быть импортированы.

```
CREATE TABLE copy_test(id int, name varchar(8));
```

Теперь фактическая операция копирования, это создаст шесть записей в таблице.

```
COPY copy_test FROM '/path/to/file/sample_data.csv' DELIMITER ',';
```

Вместо использования файла на диске можно вставлять данные из `stdin`

```
COPY copy_test FROM stdin DELIMITER ',';
Enter data to be copied followed by a newline.
End with a backslash and a period on a line by itself.
>> 7,Amol
>> 8,Amar
>> \.
Time: 85254.306 ms

SELECT * FROM copy_test ;
 id |  name
----+--------
  1 | Yogesh
  3 | Varun
  5 | Hari
  7 | Amol
  2 | Raunak
  4 | Kamal
  6 | Amit
  8 | Amar
```

Также вы можете скопировать данные из таблицы в файл, как показано ниже:

```
COPY copy_test TO 'path/to/file/sample_data.csv'  DELIMITER ',';
```

Подробнее о КОПИИ вы можете узнать здесь.

Данные INSERT и значения RETURING

Если вы вставляете данные в таблицу с столбцом автоматического увеличения, и если вы хотите получить значение столбца автоматического увеличения.

Скажем, у вас есть таблица под названием `my_table` :

```
CREATE TABLE my_table
(
id serial NOT NULL, -- serial data type is auto incrementing four-byte integer
name character varying,
contact_number integer,
CONSTRAINT my_table_pkey PRIMARY KEY (id)
);
```

Если вы хотите вставить данные в `my_table` и получить идентификатор этой строки:

```
INSERT INTO my_table(name, contact_number) VALUES ( 'USER', 8542621) RETURNING id;
```

Над запросом будет возвращен идентификатор строки, в которую была вставлена новая запись.

ВЫБОР данных в файл.

Вы можете копировать таблицу и вставлять ее в файл.

```
postgres=# select * from my_table;
 c1 | c2 | c3
----+----+----
  1 |  1 |  1
  2 |  2 |  2
  3 |  3 |  3
  4 |  4 |  4
  5 |  5 |
(5 rows)

postgres=# copy my_table to '/home/postgres/my_table.txt' using delimiters '|' with null as
'null_string' csv header;
COPY 5
postgres=# \! cat my_table.txt
c1|c2|c3
1|1|1
2|2|2
3|3|3
4|4|4
5|5|null_string
```

UPSERT - INSERT ... ON CONFLICT DO UPDATE ...

поскольку версия 9.5 postgres предлагает функциональность `UPSERT` с `INSERT` .

Скажем, у вас есть таблица my_table, созданная в нескольких предыдущих примерах. Мы

вставляем строку, возвращающую значение PK вставленной строки:

```
b=# INSERT INTO my_table (name,contact_number) values ('one',333) RETURNING id;
 id
----
  2
(1 row)

INSERT 0 1
```

Теперь, если мы попытаемся вставить строку с существующим уникальным ключом, это вызовет исключение:

```
b=# INSERT INTO my_table values (2,'one',333);
ERROR:  duplicate key value violates unique constraint "my_table_pkey"
DETAIL:  Key (id)=(2) already exists.
```

Функция Upsert предлагает возможность вставить ее в любом случае, разрешая конфликт:

```
b=# INSERT INTO my_table values (2,'one',333) ON CONFLICT (id) DO UPDATE SET name =
my_table.name||' changed to: "two" at '||now() returning *;
 id |                       name                              | contact_number
----+-----------------------------------------------------------------------------------------
------------------+----------------
  2 | one changed to: "two" at 2016-11-23 08:32:17.105179+00 |            333
(1 row)

INSERT 0 1
```

Прочитайте ВСТАВИТЬ онлайн: https://riptutorial.com/ru/postgresql/topic/2561/вставить

глава 5: ВЫБРАТЬ

Examples

ВЫБРАТЬ, используя ГДЕ

В этой теме мы опишем эту таблицу пользователей:

```
CREATE TABLE sch_test.user_table
(
  id serial NOT NULL,
  username character varying,
  pass character varying,
  first_name character varying(30),
  last_name character varying(30),
  CONSTRAINT user_table_pkey PRIMARY KEY (id)
)

+----+------------+-----------+----------+------+
| id | first_name | last_name | username | pass |
+----+------------+-----------+----------+------+
| 1  | hello      | world     | hello    | word |
+----+------------+-----------+----------+------+
| 2  | root       | me        | root     | toor |
+----+------------+-----------+----------+------+
```

Синтаксис

Выберите все:

```
SELECT * FROM schema_name.table_name WHERE <condition>;
```

Выберите несколько полей:

```
SELECT field1, field2 FROM schema_name.table_name WHERE <condition>;
```

Примеры

```
-- SELECT every thing where id = 1
SELECT * FROM schema_name.table_name WHERE id = 1;

-- SELECT id where username = ? and pass = ?
SELECT id FROM schema_name.table_name WHERE username = 'root' AND pass = 'toor';

-- SELECT first_name where id not equal 1
SELECT first_name FROM schema_name.table_name WHERE id != 1;
```

Прочитайте ВЫБРАТЬ онлайн: https://riptutorial.com/ru/postgresql/topic/9528/выбрать

глава 6: Высокая доступность PostgreSQL

Examples

Репликация в PostgreSQL

- **Настройка первичного сервера**
  - **Требования:**
    - Пользователь репликации для операций репликации
    - Каталог для хранения архивов WAL
  - **Создать пользователя репликации**

    ```
    createuser -U postgres replication -P -c 5 --replication
    ```

    ```
    + option -P will prompt you for new password
    + option -c is for maximum connections. 5 connections are enough for replication
    + -replication will grant replication privileges to the user
    ```

  - **Создание каталога архива в каталоге данных**

    ```
    mkdir $PGDATA/archive
    ```

  - **Отредактируйте файл pg_hba.conf**

    Это файл проверки подлинности базы хоста, содержащий настройки для аутентификации клиента. Добавить запись ниже:

    ```
    #hosttype    database_name     user_name       hostname/IP      method
     host        replication       replication     <slave-IP>/32    md5
    ```

  - **Отредактируйте файл postgresql.conf**

    Это файл конфигурации PostgreSQL.

    ```
    wal_level = hot_standby
    ```

    Этот параметр определяет поведение подчиненного сервера.

    ```
    `hot_standby` logs what is required to accept read only queries on slave server.

    `streaming` logs what is required to just apply the WAL's on slave.

    `archive` which logs what is required for archiving.
    ```

    ```
    archive_mode=on
    ```

Эти параметры позволяют отправлять сегменты WAL для архивирования местоположения с `archive_command` параметра `archive_command` .

```
archive_command = 'test ! -f /path/to/archivedir/%f && cp %p /path/to/archivedir/%f'
```

В основном, что делает `archive_command` , она копирует сегменты WAL для архивирования каталога.

`wal_senders = 5` Это максимальное количество процессов отправителя WAL.

Теперь перезапустите основной сервер.

- **Резервное копирование первичного сервера на подчиненный сервер**

  Перед внесением изменений на сервере остановите основной сервер.

  Важно. Не запускайте службу еще раз, пока все шаги настройки и резервного копирования не будут завершены. Вы должны открыть резервный сервер в состоянии, когда оно будет готово к резервному серверу. Это означает, что все настройки конфигурации должны быть на месте, а базы данных должны быть уже синхронизированы. В противном случае потоковая репликация не запустится.

- **Теперь запустите утилиту pg_basebackup**

  Утилита `pg_basebackup` копирует данные из каталога данных первичного сервера в подчиненный каталог данных.

  ```
  $ pg_basebackup -h <primary IP> -D /var/lib/postgresql/<version>/main -U replication -v -P
  --xlog-method=stream
  ```

```
    -D: This is tells pg_basebackup where to the initial backup

    -h: Specifies the system where to look for the primary server

    -xlog-method=stream: This will force the pg_basebackup to open another connection and
stream enough xlog while backup is running.
                         It also ensures that fresh backup can be started without failing back
to using an archive.
```

- **Настройка резервного сервера**

  Чтобы настроить резервный сервер, вы отредактируете postgresql.conf и создайте новый файл конфигурации с именем recovery.conf.

  ```
  hot_standby = on
  ```

  Это указывает, разрешено ли вам запускать запросы при восстановлении

  - **Создание файла recovery.conf**

```
standby_mode = on
```

Установите строку подключения на основной сервер. Замените внешний IP-адрес основного сервера. Замените пароль для имени пользователя с именем replication

`primary_conninfo = 'host = port = 5432 user = replication password ='

(Необязательно) Задайте местоположение файла триггера:

```
trigger_file = '/tmp/postgresql.trigger.5432'
```

Указанный путь `trigger_file` файла - это место, где вы можете добавить файл, если вы хотите, чтобы система перешла на резервный сервер. Наличие файла «запускает» переход на другой ресурс. В качестве альтернативы вы можете использовать команду pg_ctl promotion, чтобы вызвать переход на другой ресурс.

- **Запуск резервного сервера**

  Теперь у вас есть все на своем месте и готовы открыть резервный сервер

приписывание

Эта статья, по существу, получена и объясняется тем, как настроить PostgreSQL для высокой доступности и репликации в режиме «Горячий резерв» , с незначительными изменениями в форматировании и примерах, а также удалением некоторого текста. Источник был опубликован под лицензией Creative Commons Public 3.0 , которая поддерживается здесь.

Прочитайте Высокая доступность PostgreSQL онлайн:
https://riptutorial.com/ru/postgresql/topic/5478/высокая-доступность-postgresql

глава 7: Даты, отметки времени и интервалы

Examples

Отметьте временную метку или интервал для строки

Вы можете преобразовать `to_char() timestamp` или значение `interval` в строку с помощью функции `to_char()` :

```
SELECT to_char('2016-08-12 16:40:32'::timestamp, 'DD Mon YYYY HH:MI:SSPM');
```

В этом выражении появится строка «12 августа 2016 года 04:40:32». Строка форматирования может быть изменена различными способами; полный список шаблонов шаблонов можно найти здесь .

Обратите внимание, что вы также можете вставить простой текст в строку форматирования, и вы можете использовать шаблоны шаблонов в любом порядке:

```
SELECT to_char('2016-08-12 16:40:32'::timestamp,
               '"Today is "FMDay", the "DDth" day of the month of "FMMonth" of "YYYY');
```

Это приведет к созданию строки «Сегодня суббота, 12-й день августа 2016 года». Однако вы должны иметь в виду, что любые шаблоны шаблонов - даже однобуквенные, такие как « I», «D», «W» - преобразуются, если только обычный текст не заключен в двойные кавычки. В качестве меры безопасности вы должны поместить весь текст в двойные кавычки, как это было сделано выше.

Вы можете локализовать строку на выбранном вами языке (имя дня и месяца), используя модификатор TM (режим перевода). Эта опция использует параметр локализации сервера, на котором запущен PostgreSQL или клиент, подключающийся к нему.

```
SELECT to_char('2016-08-12 16:40:32'::timestamp, 'TMDay, DD" de "TMMonth" del año "YYYY');
```

С установкой испанской локали это производит «Sábado, 12 de Agosto del año 2016».

ВЫБРАТЬ последний день месяца

Вы можете выбрать последний день месяца.

```
SELECT (date_trunc('MONTH', ('201608'||'01')::date) + INTERVAL '1 MONTH - 1 day')::DATE;
```

```
201608
```

переменной.

Подсчитайте количество записей в неделю

SELECT date_trunc ('week', <>) AS "Week", count (*) FROM <> GROUP BY 1 ORDER BY 1;

Прочитайте Даты, отметки времени и интервалы онлайн:
https://riptutorial.com/ru/postgresql/topic/4227/даты--отметки-времени-и-интервалы

глава 8: Доступ к данным программно

Examples

Доступ к Postgresql из .NET с помощью поставщика Npgsql

Одним из наиболее популярных поставщиков .NET для Postgresql является Npgsql , совместимый с ADO.NET и используемый почти идентично другим поставщикам баз данных .NET.

Типичный запрос выполняется путем создания команды, параметров привязки и последующего выполнения команды. В C #:

```
var connString = "Host=myserv;Username=myuser;Password=mypass;Database=mydb";
using (var conn = new NpgsqlConnection(connString))
{
    var querystring = "INSERT INTO data (some_field) VALUES (@content)";

    conn.Open();
    // Create a new command with CommandText and Connection constructor
    using (var cmd = new NpgsqlCommand(querystring, conn))
    {
        // Add a parameter and set its type with the NpgsqlDbType enum
        var contentString = "Hello World!";
        cmd.Parameters.Add("@content", NpgsqlDbType.Text).Value = contentString;

        // Execute a query that returns no results
        cmd.ExecuteNonQuery();


        /* It is possible to reuse a command object and open connection instead of creating
new ones */

        // Create a new query and set its parameters
        int keyId = 101;
        cmd.CommandText = "SELECT primary_key, some_field FROM data WHERE primary_key =
@keyId";
        cmd.Parameters.Clear();
        cmd.Parameters.Add("@keyId", NpgsqlDbType.Integer).Value = keyId;

        // Execute the command and read through the rows one by one
        using (NpgsqlDataReader reader = cmd.ExecuteReader())
        {
            while (reader.Read())   // Returns false for 0 rows, or after reading the last row
of the results
            {
                // read an integer value
                int primaryKey = reader.GetInt32(0);
                // or
                primaryKey = Convert.ToInt32(reader["primary_key"]);

                // read a text value
                string someFieldText = reader["some_field"].ToString();
            }
```

```
        }
    }
}    // the C# 'using' directive calls conn.Close() and conn.Dispose() for us
```

Доступ к PostgreSQL с помощью C-API

C-API - это самый мощный способ доступа к PostgreSQL, и это удивительно удобно.

Компиляция и связывание

Во время компиляции вам нужно добавить каталог включения PostgreSQL, который можно найти с помощью `pg_config --includedir` , в путь включения.
Вы должны связаться с общей библиотекой клиента PostgreSQL (`libpq.so` в UNIX, `libpq.dll` в Windows). Эта библиотека находится в каталоге библиотеки PostgreSQL, который можно найти с помощью `pg_config --libdir` .

Примечание. По исторической причине библиотека называется `libpq.so` а *не* `libpg.so` , которая является популярной ловушкой для новичков.

Учитывая, что приведенный ниже пример кода находится в файле `coltype.c` , компиляция и компоновка будут выполняться с помощью

```
gcc -Wall -I "$(pg_config --includedir)" -L "$(pg_config --libdir)" -o coltype coltype.c -lpq
```

с компилятором GNU C (рассмотрите добавление `-Wl,-rpath,"$(pg_config --libdir)"` чтобы добавить путь поиска библиотеки) или с помощью

```
cl /MT /W4 /I <include directory> coltype.c <path to libpq.lib>
```

в Windows с Microsoft Visual C.

Пример программы

```
/* necessary for all PostgreSQL client programs, should be first */
#include <libpq-fe.h>

#include <stdio.h>
#include <string.h>

#ifdef TRACE
#define TRACEFILE "trace.out"
#endif

int main(int argc, char **argv) {
#ifdef TRACE
    FILE *trc;
```

```
#endif
    PGconn *conn;
    PGresult *res;
    int rowcount, colcount, i, j, firstcol;
    /* parameter type should be guessed by PostgreSQL */
    const Oid paramTypes[1] = { 0 };
    /* parameter value */
    const char * const paramValues[1] = { "pg_database" };

    /*
     * Using an empty connectstring will use default values for everything.
     * If set, the environment variables PGHOST, PGDATABASE, PGPORT and
     * PGUSER will be used.
     */
    conn = PQconnectdb("");

    /*
     * This can only happen if there is not enough memory
     * to allocate the PGconn structure.
     */
    if (conn == NULL)
    {
        fprintf(stderr, "Out of memory connecting to PostgreSQL.\n");
        return 1;
    }

    /* check if the connection attempt worked */
    if (PQstatus(conn) != CONNECTION_OK)
    {
        fprintf(stderr, "%s\n", PQerrorMessage(conn));
        /*
         * Even if the connection failed, the PGconn structure has been
         * allocated and must be freed.
         */
        PQfinish(conn);
        return 1;
    }

#ifdef TRACE
    if (NULL == (trc = fopen(TRACEFILE, "w")))
    {
        fprintf(stderr, "Error opening trace file \"%s\"!\n", TRACEFILE);
        PQfinish(conn);
        return 1;
    }

    /* tracing for client-server communication */
    PQtrace(conn, trc);
#endif

    /* this program expects the database to return data in UTF-8 */
    PQsetClientEncoding(conn, "UTF8");

    /* perform a query with parameters */
    res = PQexecParams(
        conn,
        "SELECT column_name, data_type "
            "FROM information_schema.columns "
            "WHERE table_name = $1",
        1,            /* one parameter */
        paramTypes,
```

```
        paramValues,
        NULL,        /* parameter lengths are not required for strings */
        NULL,        /* all parameters are in text format */
        0            /* result shall be in text format */
    );

    /* out of memory or sever communication broken */
    if (NULL == res)
    {
        fprintf(stderr, "%s\n", PQerrorMessage(conn));
        PQfinish(conn);
#ifdef TRACE
        fclose(trc);
#endif
        return 1;
    }

    /* SQL statement should return results */
    if (PGRES_TUPLES_OK != PQresultStatus(res))
    {
        fprintf(stderr, "%s\n", PQerrorMessage(conn));
        PQfinish(conn);
#ifdef TRACE
        fclose(trc);
#endif
        return 1;
    }

    /* get count of result rows and columns */
    rowcount = PQntuples(res);
    colcount = PQnfields(res);

    /* print column headings */
    firstcol = 1;

    printf("Description of the table \"pg_database\"\n");

    for (j=0; j<colcount; ++j)
    {
        if (firstcol)
            firstcol = 0;
        else
            printf(": ");

        printf(PQfname(res, j));
    }

    printf("\n\n");

    /* loop through rosult rows */
    for (i=0; i<rowcount; ++i)
    {
        /* print all column data */
        firstcol = 1;

        for (j=0; j<colcount; ++j)
        {
            if (firstcol)
                firstcol = 0;
            else
                printf(": ");
```

```
            printf(PQgetvalue(res, i, j));
        }

        printf("\n");
    }

    /* this must be done after every statement to avoid memory leaks */
    PQclear(res);
    /* close the database connection and release memory */
    PQfinish(conn);
#ifdef TRACE
    fclose(trc);
#endif
    return 0;
}
```

Доступ к PostgreSQL из python с помощью psycopg2

Вы можете найти описание водителя здесь .

Быстрый пример:

```
import psycopg2

db_host = 'postgres.server.com'
db_port = '5432'
db_un = 'user'
db_pw = 'password'
db_name = 'testdb'

conn = psycopg2.connect("dbname={} host={} user={} password={}".format(
                        db_name,  db_host, db_un, db_pw),
                        cursor_factory=RealDictCursor)
cur = conn.cursor()
sql = 'select * from testtable where id > %s and id < %s'
args = (1, 4)
cur.execute(sql, args)

print(cur.fetchall())
```

Приведет к:

```
[{'id': 2, 'fruit': 'apple'}, {'id': 3, 'fruit': 'orange'}]
```

Доступ к PostgreSQL с PHP с помощью Pomm2

На плечах водителей низкого уровня есть pomm . Он предлагает модульный подход, преобразователи данных, поддержку прослушивания / уведомления, инспектор базы данных и многое другое.

Предполагая, что Pomm был установлен с использованием композитора, вот полный пример:

```
<?php
use PommProject\Foundation\Pomm;
$loader = require __DIR__ . '/vendor/autoload.php';
$pomm = new Pomm(['my_db' => ['dsn' => 'pgsql://user:pass@host:5432/db_name']]);

// TABLE comment (
// comment_id uuid PK, created_at timestamptz NN,
// is_moderated bool NN default false,
// content text NN CHECK (content !~ '^\s+$'), author_email text NN)
$sql = <<<SQL
SELECT
  comment_id,
  created_at,
  is_moderated,
  content,
  author_email
FROM comment
  INNER JOIN author USING (author_email)
WHERE
  age(now(), created_at) < $*::interval
ORDER BY created_at ASC
SQL;

// the argument will be converted as it is cast in the query above
$comments = $pomm['my_db']
    ->getQueryManager()
    ->query($sql, [DateInterval::createFromDateString('1 day')]);

if ($comments->isEmpty()) {
    printf("There are no new comments since yesterday.");
} else {
    foreach ($comments as $comment) {
        printf(
                "%s has posted at %s. %s\n",
                $comment['author_email'],
                $comment['created_at']->format("Y-m-d H:i:s"),
                $comment['is_moderated'] ? '[OK]' : '');
    }
}
```

Модуль диспетчера запросов Pomm экранирует аргументы запроса, чтобы предотвратить SQL-инъекцию. Когда аргументы передаются, он также преобразует их из представления PHP в допустимые значения Postgres. Результатом является итератор, он использует внутренний курсор. Каждая строка преобразуется «на лету», boolean до boolean, timestamps to \ DateTime и т. Д.

Прочитайте Доступ к данным программно онлайн:
https://riptutorial.com/ru/postgresql/topic/2014/доступ-к-данным-программно

глава 9: Комментарии в postgresql

Вступление

Основной целью ***COMMMENT*** является определение или изменение комментария к объекту базы данных.

Только один комментарий (строка) может быть задан для любого объекта базы данных. COMMENT поможет нам узнать, для чего определен конкретный объект базы данных, какова его фактическая цель.

Правило для ***COMMENT ON ROLE*** заключается в том, что вы должны быть суперпользователем, чтобы комментировать роль суперпользователя или иметь привилегию ***CREATEROLE,*** чтобы комментировать роли не-суперпользователя. Конечно, *суперпользователь может прокомментировать что-нибудь*

Синтаксис

- COMMENT ON database_object object_name IS 'Text';

замечания

Полный синтаксис см .: http://www.postgresql.org/docs/current/static/sql-comment.html.

Examples

КОММЕНТАРИЙ:

КОММЕНТАРИЙ НА ТАБЛИЦЕ table_name IS 'это таблица сведений о студентах';

Удалить комментарий

КОММЕНТАРИЙ по ТАБЛИЦЕ студент IS NULL;

Комментарий будет удален с выполнением вышеуказанного оператора.

Прочитайте Комментарии в postgresql онлайн: https://riptutorial.com/ru/postgresql/topic/8191/комментарии-в-postgresql

глава 10: Криптографические функции Postgres

Вступление

В Postgres криптографические функции могут быть разблокированы с помощью модуля pgcrypto. CREATE EXTENSION pgcrypto;

Examples

дайджест

Функции `DIGEST()` генерируют двоичный хэш данных. Это **можно** использовать для создания случайного хэша.

Использование: `digest(data text, type text) returns bytea`

Или: `digest(data bytea, type text) returns bytea`

Примеры:

- `SELECT DIGEST('1', 'sha1')`
- `SELECT DIGEST(CONCAT(CAST(current_timestamp AS TEXT), RANDOM()::TEXT), 'sha1')`

Прочитайте Криптографические функции Postgres онлайн: https://riptutorial.com/ru/postgresql/topic/9230/криптографические-функции-postgres

глава 11: Найти длину строки / длину символа

Вступление

Чтобы получить длину полей «изменение характера», «текст», используйте char_length () или character_length ().

Examples

Пример получения длины символа в другом поле

Пример 1, Запрос: `SELECT char_length('ABCDE')`

Результат:

```
5
```

Пример 2: Запрос: `SELECT character_length('ABCDE')`

Результат:

```
5
```

Прочитайте Найти длину строки / длину символа онлайн:
https://riptutorial.com/ru/postgresql/topic/9695/найти-длину-строки---длину-символа

глава 12: наследование

замечания

Объяснение того, почему вы хотите использовать наследование в PostgreSQL, можно найти здесь: http://stackoverflow.com/a/3075248/653378

Examples

Создание таблиц для детей

```
CREATE TABLE users (username text, email text);
CREATE TABLE simple_users () INHERITS (users);
CREATE TABLE users_with_password (password text) INHERITS (users);
```

Наши три таблицы выглядят так:

пользователи

колонка	Тип
имя пользователя	текст
Эл. адрес	текст

simple_users

колонка	Тип
имя пользователя	текст
Эл. адрес	текст

users_with_password

колонка	Тип
имя пользователя	текст
Эл. адрес	текст

колонка	Тип
пароль	текст

Изменение таблиц

Давайте создадим две простые таблицы:

```
CREATE TABLE users (username text, email text);
CREATE TABLE simple_users () INHERITS (users);
```

Добавление столбцов

```
ALTER TABLE simple_users ADD COLUMN password text;
```

simple_users

колонка	Тип
имя пользователя	текст
Эл. адрес	текст
пароль	текст

Добавление одного и того же столбца в родительскую таблицу объединит определение обоих столбцов:

```
ALTER TABLE users ADD COLUMN password text;
```

> УВЕДОМЛЕНИЕ: слияние определения столбца «пароль» для дочерних « simple_users»

Удаление столбцов

Используя наши измененные таблицы:

```
ALTER TABLE users DROP COLUMN password;
```

пользователи

колонка	Тип
имя пользователя	текст
Эл. адрес	текст

simple_users

колонка	Тип
имя пользователя	текст
Эл. адрес	текст
пароль	текст

Поскольку мы сначала добавили столбец в `simple_users` , PostgreSQL гарантирует, что этот столбец не будет `simple_users` .

Теперь, если бы у нас был другой дочерний стол, его `password` столбец, конечно, был бы удален.

Прочитайте наследование онлайн: https://riptutorial.com/ru/postgresql/topic/5429/наследование

глава 13: ОБНОВИТЬ

Examples

Обновление всех строк в таблице

Вы обновляете все строки в таблице, просто предоставляя значение `column_name = value` :

```
UPDATE person SET planet = 'Earth';
```

Обновить все строки, удовлетворяющие условию

```
UPDATE person SET state = 'NY' WHERE city = 'New York';
```

Обновление нескольких столбцов в таблице

Вы можете обновлять несколько столбцов в таблице в том же самом выражении, разделяя пары `col=val` запятыми:

```
UPDATE person
   SET country = 'USA',
       state = 'NY'
WHERE city = 'New York';
```

Обновление таблицы на основе присоединения к другой таблице

Вы также можете обновлять данные в таблице на основе данных из другой таблицы:

```
UPDATE person
SET state_code = cities.state_code
FROM cities
WHERE cities.city = city;
```

Здесь мы присоединяемся к колонке `city person` столбцу `cities city` , чтобы получить код состояния города. Затем он используется для обновления `state_code` столбца в `person` таблицу.

Прочитайте ОБНОВИТЬ онлайн: https://riptutorial.com/ru/postgresql/topic/3136/обновить

глава 14: Общие выражения таблицы (WITH)

Examples

Общие выражения таблицы в SELECT-запросах

Общие выражения таблиц поддерживают извлечение частей более крупных запросов. Например:

```
WITH sales AS (
  SELECT
    orders.ordered_at,
    orders.user_id,
    SUM(orders.amount) AS total
  FROM orders
  GROUP BY orders.ordered_at, orders.user_id
)
SELECT
  sales.ordered_at,
  sales.total,
  users.name
FROM sales
JOIN users USING (user_id)
```

Перемещение дерева с помощью RECURSIVE

```
create table empl (
    name text primary key,
    boss text null
        references name
            on update cascade
            on delete cascade
        default null
);

insert into empl values ('Paul',null);
insert into empl values ('Luke','Paul');
insert into empl values ('Kate','Paul');
insert into empl values ('Marge','Kate');
insert into empl values ('Edith','Kate');
insert into empl values ('Pam','Kate');
insert into empl values ('Carol','Luke');
insert into empl values ('John','Luke');
insert into empl values ('Jack','Carol');
insert into empl values ('Alex','Carol');

with recursive t(level,path,boss,name) as (
        select 0,name,boss,name from empl where boss is null
    union
        select
            level + 1,
```

```
            path || ' > ' || empl.name,
            empl.boss,
            empl.name
        from
            empl join t
                on empl.boss = t.name
) select * from t order by path;
```

Прочитайте Общие выражения таблицы (WITH) онлайн:
https://riptutorial.com/ru/postgresql/topic/1973/общие-выражения-таблицы--with-

глава 15: Поддержка JSON

Вступление

JSON - Java Script Object Notation, Postgresql поддерживает JSON Тип данных с версии 9.2. Существуют некоторые предопределенные функции и операторы для доступа к данным JSON. Оператор `->` возвращает ключ столбца JSON. Оператор `->>` возвращает значение столбца JSON.

Examples

Создание чистой таблицы JSON

Чтобы создать чистую таблицу JSON, вам нужно предоставить одно поле с типом `JSONB` :

```
CREATE TABLE mytable (data JSONB NOT NULL);
```

Вы также должны создать базовый индекс:

```
CREATE INDEX mytable_idx ON mytable USING gin (data jsonb_path_ops);
```

На этом этапе вы можете вставлять данные в таблицу и запрашивать ее эффективно.

Запрос сложных документов JSON

Принимая сложный документ JSON в таблице:

```
CREATE TABLE mytable (data JSONB NOT NULL);
CREATE INDEX mytable_idx ON mytable USING gin (data jsonb_path_ops);
INSERT INTO mytable VALUES($$
{
    "name": "Alice",
    "emails": [
        "alice1@test.com",
        "alice2@test.com"
    ],
    "events": [
        {
            "type": "birthday",
            "date": "1970-01-01"
        },
        {
            "type": "anniversary",
            "date": "2001-05-05"
        }
    ],
    "locations": {
        "home": {
```

```
            "city": "London",
            "country": "United Kingdom"
        },
        "work": {
            "city": "Edinburgh",
            "country": "United Kingdom"
        }
    }
}
$$);
```

Запрос элемента верхнего уровня:

```
SELECT data->>'name' FROM mytable WHERE data @> '{"name":"Alice"}';
```

Запрос простого элемента в массиве:

```
SELECT data->>'name' FROM mytable WHERE data @> '{"emails":["alice1@test.com"]}';
```

Запрос объекта в массиве:

```
SELECT data->>'name' FROM mytable WHERE data @> '{"events":[{"type":"anniversary"}]}';
```

Запрос для вложенного объекта:

```
SELECT data->>'name' FROM mytable WHERE data @> '{"locations":{"home":{"city":"London"}}}';
```

Производительность `@>` по сравнению с `->` и `->>`

Важно понимать разницу в производительности между использованием `@>` , `->` и `->>` в части `WHERE` запроса. Хотя эти два запроса в целом эквивалентны:

```
SELECT data FROM mytable WHERE data @> '{"name":"Alice"}';
SELECT data FROM mytable WHERE data->'name' = '"Alice"';
SELECT data FROM mytable WHERE data->>'name' = 'Alice';
```

первый оператор будет использовать созданный выше индекс, тогда как последние два не будут, требующие полного сканирования таблицы.

По-прежнему можно использовать оператор- `->` при получении результирующих данных, поэтому следующие запросы также будут использовать индекс:

```
SELECT data->'locations'->'work' FROM mytable WHERE data @> '{"name":"Alice"}';
SELECT data->'locations'->'work'->>'city' FROM mytable WHERE data @> '{"name":"Alice"}';
```

Использование операторов JSONb

Создание базы данных и таблицы

```
DROP DATABASE IF EXISTS books_db;
CREATE DATABASE books_db WITH ENCODING='UTF8' TEMPLATE template0;

DROP TABLE IF EXISTS books;

CREATE TABLE books (
  id SERIAL PRIMARY KEY,
  client TEXT NOT NULL,
  data JSONb NOT NULL
);
```

Заполнение БД

```
INSERT INTO books(client, data) values (
    'Joe',
    '{ "title": "Siddhartha", "author": { "first_name": "Herman", "last_name": "Hesse" } }'
),(
    'Jenny',
    '{ "title": "Dharma Bums", "author": { "first_name": "Jack", "last_name": "Kerouac" } }'
),(
    'Jenny',
    '{ "title": "100 años de soledad", "author": { "first_name": "Gabo", "last_name":
"Marquéz" } }'
);
```

Давайте посмотрим все в книгах:

```
SELECT * FROM books;
```

Выход:

id integer	client character varying	data jsonb
1	Joe	{"title": "Siddhartha", "author": {"last name": "Hesse", "first name": "Herman"}}
2	Jenny	{"title": "Dharma Bums", "author": {"last name": "Kerouac", "first name": "Jack"}}
3	Jenny	{"title": "100 años de soledad", "author": {"last name": "Marquéz", "first name": "G

-> возвращает значения из столбцов JSON

Выбор 1 столбца:

```
SELECT client,
    data->'title' AS title
    FROM books;
```

Выход:

	client character varying	title jsonb
1	Joe	"Siddhartha"
2	Jenny	"Dharma Bums"
3	Jenny	"100 años de soledad"

Выбор 2 столбцов:

```
SELECT client,
   data->'title' AS title, data->'author' AS author
   FROM books;
```

Выход:

client character varying	title jsonb	author jsonb
Joe	"Siddhartha"	{"last_name": "Hesse", "first_name": "Herman"}
Jenny	"Dharma Bums"	{"last name": "Kerouac", "first name": "Jack"}
Jenny	"100 años de soledad"	{"last name": "Marquéz", "first name": "Gabo"}

-> vs ->>

Оператор `->` возвращает исходный тип JSON (который может быть объектом), тогда как `->>` возвращает текст.

Вернуть объекты NESTED

Вы можете использовать `->` для возврата вложенного объекта и, таким образом, привязать операторов:

```
SELECT client,
   data->'author'->'last_name' AS author
   FROM books;
```

Выход:

client character varying	author jsonb
Joe	"Hesse"
Jenny	"Kerouac"
Jenny	"Marquéz"

фильтрация

Выберите строки, основанные на значении внутри вашего JSON:

```
SELECT
client,
data->'title' AS title
FROM books
 WHERE data->'title' = '"Dharma Bums"';
```

Обратите внимание, что WHERE использует `->` так что мы должны сравнить с JSON `'"Dharma Bums"'`

Или мы могли бы использовать `->>` и сравнить с `'Dharma Bums'`

Выход:

client character varying	title jsonb
Jenny	"Dharma Bums"

Вложенная фильтрация

Найти строки на основе значения вложенного объекта JSON:

```
SELECT
 client,
 data->'title' AS title
 FROM books
  WHERE data->'author'->>'last_name' = 'Kerouac';
```

Выход:

client character varying	title jsonb
Jenny	"Dharma Bums"

Пример реального мира

```
CREATE TABLE events (
  name varchar(200),
  visitor_id varchar(200),
  properties json,
  browser json
);
```

Мы собираемся хранить события в этой таблице, такие как просмотры страниц. Каждое событие имеет свойства, которые могут быть любыми (например, текущая страница), а также отправляет информацию о браузере (например, ОС, разрешение экрана и т. Д.). Оба они полностью бесплатны и могут меняться со временем (поскольку мы думаем о дополнительных материалах для отслеживания).

```
INSERT INTO events (name, visitor_id, properties, browser) VALUES
(
  'pageview', '1',
  '{ "page": "/" }',
  '{ "name": "Chrome", "os": "Mac", "resolution": { "x": 1440, "y": 900 } }'
),(
  'pageview', '2',
  '{ "page": "/" }',
  '{ "name": "Firefox", "os": "Windows", "resolution": { "x": 1920, "y": 1200 } }'
),(
  'pageview', '1',
  '{ "page": "/account" }',
  '{ "name": "Chrome", "os": "Mac", "resolution": { "x": 1440, "y": 900 } }'
),(
  'purchase', '5',
  '{ "amount": 10 }',
  '{ "name": "Firefox", "os": "Windows", "resolution": { "x": 1024, "y": 768 } }'
),(
  'purchase', '15',
  '{ "amount": 200 }',
  '{ "name": "Firefox", "os": "Windows", "resolution": { "x": 1280, "y": 800 } }'
),(
  'purchase', '15',
  '{ "amount": 500 }',
  '{ "name": "Firefox", "os": "Windows", "resolution": { "x": 1280, "y": 800 } }'
);
```

Теперь давайте выберем все:

```
SELECT * FROM events;
```

Выход:

name character varying(200)	visitor_id character varying(200)	properties json	browser json
pageview	1	{ "page": "/" }	{ "name": "Chrome", "os": "Mac", "resolution
pageview	2	{ "page": "/" }	{ "name": "Firefox", "os": "Windows", "resol
pageview	1	{ "page": "/account" }	{ "name": "Chrome", "os": "Mac", "resolution
purchase	5	{ "amount": 10 }	{ "name": "Firefox", "os": "Windows", "resol
purchase	15	{ "amount": 200 }	{ "name": "Firefox", "os": "Windows", "resol
purchase	15	{ "amount": 500 }	{ "name": "Firefox", "os": "Windows", "resol

Операторы JSON + агрегированные функции PostgreSQL

Используя JSON-операторы в сочетании с традиционными функциями PostgreSQL, мы можем вытащить все, что захотим. У вас есть полная мощь РСУБД в вашем распоряжении.

- Давайте посмотрим на использование браузера:

```
  SELECT browser->>'name' AS browser,
```

```
  count(browser)
  FROM events
  GROUP BY browser->>'name';
```

Выход:

browser text	count bigint
Firefox	4
Chrome	2

- Общий доход на одного посетителя:

```
SELECT visitor_id, SUM(CAST(properties->>'amount' AS integer)) AS total
FROM events
WHERE CAST(properties->>'amount' AS integer) > 0
GROUP BY visitor_id;
```

Выход:

visitor_id character varying(200)	total bigint
5	10
15	700

- Среднее разрешение экрана

```
SELECT AVG(CAST(browser->'resolution'->>'x' AS integer)) AS width,
  AVG(CAST(browser->'resolution'->>'y' AS integer)) AS height
FROM events;
```

Выход:

width numeric	height numeric
1397.3333333333333333	894.666666666666667

Дополнительные примеры и документация здесь и здесь .

Прочитайте Поддержка JSON онлайн: https://riptutorial.com/ru/postgresql/topic/1034/поддержка-json

глава 16: Подключение к PostgreSQL с Java

Вступление

API для использования реляционной базы данных с Java - это JDBC.

Этот API реализован драйвером JDBC.

Чтобы использовать его, поместите JAR-файл с драйвером в путь класса JAVA.

В этой документации приведены примеры использования драйвера JDBC для подключения к базе данных.

замечания

URL JDBC

URL JDBC может принимать одну из следующих форм:

- `jdbc:postgresql://` *`host`* `[:` *`port`* `]/[` *`database`* `][` *`parameters`* `]`

  `host` умолчанию - `localhost` , *`port`* - 5432.
  Если *`host`* является адресом IPv6, он должен быть заключен в квадратные скобки.
  Имя базы данных по умолчанию совпадает с именем подключаемого пользователя.

  Чтобы реализовать переход на другой ресурс, возможно наличие нескольких записей *`host`* `[:` *`port`* `]` разделенных запятой.
  Они стараются в свою очередь, пока соединение не удастся.

- `jdbc:postgresql:` *`database`* `[` *`parameters`* `]`

- `jdbc:postgresql:/[` *`parameters`* `]`

  Эти формы предназначены для соединений с `localhost` .

`parameters` - это список *`key`* `[=` *`value`* `]` пар *`key`* `[=` *`value`* `]` , возглавляемых `?` и разделяются символом `&` . Если *`value`* отсутствует, оно считается `true` .

Пример:

```
jdbc:postgresql://localhost/test?user=fred&password=secret&ssl&sslfactory=org.postgresql.ssl.NonValidat
```

Рекомендации

- Спецификация JDBC: http://download.oracle.com/otndocs/jcp/jdbc-4_2-mrel2-eval-spec/
- Драйвер PostgreSQL JDBC: https://jdbc.postgresql.org/
- Документация драйвера PostgreSQL JDBC: https://jdbc.postgresql.org/documentation/head/index.html

Examples

Подключение с помощью java.sql.DriverManager

Это самый простой способ подключения.

Во-первых, драйвер должен быть *зарегистрирован* с помощью `java.sql.DriverManager` чтобы он знал, какой класс использовать.
Это делается путем загрузки класса драйвера, как правило, с помощью `java.lang.Class.forname( <driver class name> ) .`

```
/**
 * Connect to a PostgreSQL database.
 * @param url the JDBC URL to connect to; must start with "jdbc:postgresql:"
 * @param user the username for the connection
 * @param password the password for the connection
 * @return a connection object for the established connection
 * @throws ClassNotFoundException if the driver class cannot be found on the Java class path
 * @throws java.sql.SQLException if the connection to the database fails
 */
private static java.sql.Connection connect(String url, String user, String password)
    throws ClassNotFoundException, java.sql.SQLException
{
    /*
     * Register the PostgreSQL JDBC driver.
     * This may throw a ClassNotFoundException.
     */
    Class.forName("org.postgresql.Driver");
    /*
     * Tell the driver manager to connect to the database specified with the URL.
     * This may throw an SQLException.
     */
    return java.sql.DriverManager.getConnection(url, user, password);
}
```

Не тот пользователь и пароль также могут быть включены в URL JDBC, и в этом случае вам не нужно указывать их в вызове метода `getConnection` .

Подключение с помощью java.sql.DriverManager и свойств

Вместо указания параметров соединения, таких как пользователь и пароль (см. Полный список здесь) в URL-адресе или отдельных параметрах, вы можете упаковать их в объект `java.util.Properties` :

```
/**
 * Connect to a PostgreSQL database.
```

```
 * @param url the JDBC URL to connect to. Must start with "jdbc:postgresql:"
 * @param user the username for the connection
 * @param password the password for the connection
 * @return a connection object for the established connection
 * @throws ClassNotFoundException if the driver class cannot be found on the Java class path
 * @throws java.sql.SQLException if the connection to the database fails
 */
private static java.sql.Connection connect(String url, String user, String password)
    throws ClassNotFoundException, java.sql.SQLException
{
    /*
     * Register the PostgreSQL JDBC driver.
     * This may throw a ClassNotFoundException.
     */
    Class.forName("org.postgresql.Driver");
    java.util.Properties props = new java.util.Properties();
    props.setProperty("user", user);
    props.setProperty("password", password);
    /* don't use server prepared statements */
    props.setProperty("prepareThreshold", "0");
    /*
     * Tell the driver manager to connect to the database specified with the URL.
     * This may throw an SQLException.
     */
    return java.sql.DriverManager.getConnection(url, props);
}
```

Подключение с помощью javax.sql.DataSource с использованием пула соединений

Обычно используется `javax.sql.DataSource` с JNDI в контейнерах сервера приложений, где вы регистрируете источник данных под именем и просматриваете его, когда вам нужно соединение.

Это код, который показывает, как работают источники данных:

```
/**
 * Create a data source with connection pool for PostgreSQL connections
 * @param url the JDBC URL to connect to. Must start with "jdbc:postgresql:"
 * @param user the username for the connection
 * @param password the password for the connection
 * @return a data source with the correct properties set
 */
private static javax.sql.DataSource createDataSource(String url, String user, String password)
{
    /* use a data source with connection pooling */
    org.postgresql.ds.PGPoolingDataSource ds = new org.postgresql.ds.PGPoolingDataSource();
    ds.setUrl(url);
    ds.setUser(user);
    ds.setPassword(password);
    /* the connection pool will have 10 to 20 connections */
    ds.setInitialConnections(10);
    ds.setMaxConnections(20);
    /* use SSL connections without checking server certificate */
    ds.setSslMode("require");
    ds.setSslfactory("org.postgresql.ssl.NonValidatingFactory");
```

```
    return ds;
}
```

Как только вы создали источник данных, вызвав эту функцию, вы будете использовать ее следующим образом:

```
/* get a connection from the connection pool */
java.sql.Connection conn = ds.getConnection();

/* do some work */

/* hand the connection back to the pool - it will not be closed */
conn.close();
```

Прочитайте Подключение к PostgreSQL с Java онлайн:
https://riptutorial.com/ru/postgresql/topic/9633/подключение-к-postgresql-с-java

глава 17: Программирование с помощью PL / pgSQL

замечания

PL / pgSQL - это встроенный язык программирования PostgreSQL для написания функций, которые выполняются в самой базе данных, известной как хранимые процедуры в других базах данных. Он расширяет SQL с помощью циклов, условных выражений и возвращаемых типов. Хотя его синтаксис может быть странным для многих разработчиков, он намного быстрее, чем все, что работает на сервере приложений, потому что устранены накладные расходы на подключение к базе данных, что особенно полезно, когда вам в противном случае нужно было бы выполнить запрос, дождаться результата, и отправить другой запрос.

Хотя для PostgreSQL существует много других процедурных языков, таких как PL / Python, PL / Perl и PLV8, PL / pgSQL является общей отправной точкой для разработчиков, которые хотят написать свою первую функцию PostgreSQL, потому что ее синтаксис построен на SQL. Он также похож на PL / SQL, собственный процедурный язык Oracle, поэтому любой разработчик, знакомый с PL / SQL, найдет язык знакомым, и любой разработчик, который намеревается разрабатывать приложения Oracle в будущем, но хочет начать с бесплатной базы данных, может перейти от PL / pgSQL до PL / SQL с относительной легкостью.

Следует подчеркнуть, что существуют другие процедурные языки, и PL / pgSQL не обязательно превосходит их в любом случае, включая скорость, но примеры в PL / pgSQL могут служить общей точкой отсчета для других языков, используемых для написания функций PostgreSQL. PL / pgSQL имеет большинство учебников и книг всех PL и может стать плацдармом для изучения языков с меньшей документацией.

Вот ссылки на некоторые бесплатные руководства и книги по PL / pgSQL:

- Официальная документация: https://www.postgresql.org/docs/current/static/plpgsql.html
- w3resource.com учебник: http://www.w3resource.com/PostgreSQL/pl-pgsql-tutorial.php
- postgres.cz учебник: http://postgres.cz/wiki/PL/pgSQL_(en)
- PostgreSQL Server Programming, 2nd Edition: https://www.packtpub.com/big-data-and-business-intelligence/postgresql-server-programming-second-edition
- Руководство для разработчиков PostgreSQL: https://www.packtpub.com/big-data-and-business-intelligence/postgresql-developers-guide

Examples

Основная функция PL / pgSQL

Простая функция PL / pgSQL:

```
CREATE FUNCTION active_subscribers() RETURNS bigint AS $$
DECLARE
    -- variable for the following BEGIN ... END block
    subscribers integer;
BEGIN
    -- SELECT must always be used with INTO
    SELECT COUNT(user_id) INTO subscribers FROM users WHERE subscribed;
    -- function result
    RETURN subscribers;
EXCEPTION
    -- return NULL if table "users" does not exist
    WHEN undefined_table
    THEN RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

Это могло быть достигнуто только с помощью инструкции SQL, но демонстрирует основную структуру функции.

Для выполнения функции выполните:

```
select active_subscribers();
```

Синтаксис PL / pgSQL

```
CREATE [OR REPLACE] FUNCTION functionName (someParameter 'parameterType')
RETURNS 'DATATYPE'
AS $_block_name_$
DECLARE
    --declare something
BEGIN
    --do something
    --return something
END;
$_block_name_$
LANGUAGE plpgsql;
```

RETURNS Block

Параметры для возврата в PL / pgSQL:

- `Datatype` Список всех типов данных
- `Table(column_name column_type, ...)`
- `Setof 'Datatype' or 'table_column'`

пользовательские исключения

создание настраиваемого исключения «P2222»:

```
create or replace function s164() returns void as
```

```
$$
begin
raise exception using message = 'S 164', detail = 'D 164', hint = 'H 164', errcode = 'P2222';
end;
$$ language plpgsql
;
```

создание настраиваемого исключения, не присваивающего errm:

```
create or replace function s165() returns void as
$$
begin
raise exception '%','nothing specified';
end;
$$ language plpgsql
;
```

призвание:

```
t=# do
$$
declare
 _t text;
begin
  perform s165();
  exception when SQLSTATE 'P0001' then raise info '%','state P0001 caught: '||SQLERRM;
  perform s164();

end;
$$
;
INFO:  state P0001 caught: nothing specified
ERROR:  S 164
DETAIL:  D 164
HINT:  H 164
CONTEXT:  SQL statement "SELECT s164()"
PL/pgSQL function inline_code_block line 7 at PERFORM
```

здесь пользовательский P0001 обрабатывается, а P2222, а не, прерывает выполнение.

Также имеет смысл хранить таблицу исключений, например здесь:
http://stackoverflow.com/a/2700312/5315974

Прочитайте Программирование с помощью PL / pgSQL онлайн:
https://riptutorial.com/ru/postgresql/topic/5299/программирование-с-помощью-pl---pgsql

глава 18: Резервное копирование и восстановление

замечания

Резервное копирование файловой системы вместо использования `pg_dumpall` и `pg_dump`

Очень важно, что если вы используете это, вы вызываете `pg_start_backup()` перед и `pg_stop_backup()` после. Выполнение резервных копий файловой системы в противном случае небезопасно; даже моментальный снимок ZFS или FreeBSD из резервной копии файловой системы без этих вызовов функций поместит базу данных в режим восстановления и может потерять транзакции.

По этой причине я бы не стал делать резервные копии файловой системы вместо обычных резервных копий Postgres и потому, что файлы резервного копирования Postgres (особенно в пользовательском формате) чрезвычайно универсальны в поддержке альтернативных восстановлений. Поскольку они представляют собой одиночные файлы, они также менее трудны в управлении.

Examples

Резервное копирование одной базы данных

```
pg_dump -Fc -f DATABASE.pgsql DATABASE
```

`-Fc` выбирает «настраиваемый формат резервного копирования», который дает вам больше мощности, чем исходный SQL; см. `pg_restore` для более подробной информации. Если вам нужен файл vanilla SQL, вы можете сделать это вместо этого:

```
pg_dump -f DATABASE.sql DATABASE
```

или даже

```
pg_dump DATABASE > DATABASE.sql
```

Восстановление резервных копий

```
psql < backup.sql
```

Более безопасная альтернатива использует `-1` для обертывания восстановления в транзакции. Параметр `-f` указывает имя файла, а не перенаправление оболочки.

```
psql -1f backup.sql
```

Файлы настраиваемого формата должны быть восстановлены с помощью `pg_restore` с опцией `-d` чтобы указать базу данных:

```
pg_restore -d DATABASE DATABASE.pgsql
```

Пользовательский формат также можно преобразовать обратно в SQL:

```
pg_restore backup.pgsql > backup.sql
```

Рекомендуется использовать пользовательский формат, поскольку вы можете выбрать, какие вещи нужно восстановить и, возможно, включить параллельную обработку.

Возможно, вам понадобится pg_dump, за которым следует pg_restore, если вы перейдете из одной версии postgresql в более новую.

Резервное копирование всего кластера

```
$ pg_dumpall -f backup.sql
```

Это работает за кулисами, делая несколько подключений к серверу один раз для каждой базы данных и выполняя `pg_dump` на нем.

Иногда может возникнуть соблазн установить это как задание cron, поэтому вы хотите увидеть дату, когда резервная копия была взята как часть имени файла:

```
$ postgres-backup-$(date +%Y-%m-%d).sql
```

Однако обратите внимание, что это может ежедневно создавать большие файлы. Postgresql имеет гораздо лучший механизм для регулярных резервных копий - архивы WAL

Вывод из pg_dumpall достаточен для восстановления экземпляра Postgres с идентичной конфигурацией, но файлы конфигурации в `$PGDATA` (`pg_hba.conf` и `postgresql.conf`) не являются частью резервной копии, поэтому вам придется резервировать их отдельно.

```
postgres=# SELECT pg_start_backup('my-backup');
postgres=# SELECT pg_stop_backup();
```

Чтобы выполнить резервное копирование файловой системы, вы должны использовать эти функции, чтобы гарантировать, что Postgres находится в согласованном состоянии, пока резервная копия подготовлена.

Использование копирования для импорта

Копировать данные из файла CSV в таблицу

```
COPY <tablename> FROM '<filename with path>';
```

Чтобы вставить в `user` таблицы из файла с именем `user_data.csv` помещенным внутри `/home/user/` :

```
COPY user FROM '/home/user/user_data.csv';
```

Копировать данные из файла, разделенного на трубы, в таблицу

```
COPY user FROM '/home/user/user_data' WITH DELIMITER '|';
```

Примечание. В отсутствие опции `with delimiter` по умолчанию является запятая ,

Чтобы игнорировать строку заголовка при импорте файла

Используйте параметр заголовка:

```
COPY user FROM '/home/user/user_data' WITH DELIMITER '|' HEADER;
```

Примечание. Если данные цитируются, по умолчанию данные котировки символов являются двойными. Если данные цитируются с использованием любого другого символа, используйте опцию `QUOTE` ; однако эта опция разрешена только при использовании формата CSV.

Использование копирования для экспорта

Скопировать таблицу в стандартный файл o / p

COPY <имя_таблицы> TO STDOUT (DELIMITER '|');

Чтобы экспортировать пользователя таблицы в стандартный вывод:

Пользователь COPY TO STDOUT (DELIMITER '|');

Скопировать таблицу в файл

COPY user FROM '/ home / user / user_data' WITH DELIMITER '|';

Копировать вывод инструкции SQL в файл

COPY (оператор sql) TO '<имя_файла с контуром>';

COPY (SELECT * FROM user WHERE user_name LIKE 'A%') TO '/ home / user / user_data';

Скопировать в сжатый файл

COPY user TO PROGRAM 'gzip> /home/user/user_data.gz';

Здесь программа gzip выполняется для сжатия данных пользовательской таблицы.

Использование psql для экспорта данных

Данные могут быть экспортированы с помощью команды копирования или с использованием параметров командной строки команды psql.

Чтобы экспортировать данные csv из пользователя таблицы в файл csv:

```
psql -p \<port> -U \<username> -d \<database> -A -F<delimiter> -c\<sql to execute> \> \<output filename with path>

psql -p 5432 -U postgres -d test_database -A -F, -c "select * from user" > /home/user/user_data.csv
```

Здесь комбинация -A и -F делает трюк.

`-F` - указать разделитель

```
-A or --no-align
```

Переключает в режим выровненного вывода. (Режим вывода по умолчанию в противном случае выровнен.)

Прочитайте Резервное копирование и восстановление онлайн:
https://riptutorial.com/ru/postgresql/topic/2291/резервное-копирование-и-восстановление

глава 19: Резервный сценарий для производственной БД

Синтаксис

- Сценарий позволяет создать резервный каталог для каждого исполнения со следующим синтаксисом: Имя каталога резервного копирования базы данных + дата и время выполнения
- Пример: prodDir22-11-2016-19h55
- После его создания он создает два файла резервной копии со следующим синтаксисом: **Имя базы данных + дата и время выполнения**
- Пример :
- dbprod22-11-2016-19h55.backup **(файл дампа)**
- dbprod22-11-2016-19h55.sql **(файл sql)**
- В конце одного исполнения в **22-11-2016 гг. В 19 ч. 55** г. мы получаем:
- /save_bd/prodDir22-11-2016-19h55/dbprod22-11-2016-19h55.backup
- /save_bd/prodDir22-11-2016-19h55/dbprod22-11-2016-19h55.sql

параметры

параметр	подробности
save_db	Основной каталог резервного копирования
dbProd	Каталог вторичной резервной копии
ДАТА	Дата резервного копирования в указанном формате
dbprod	Имя сохраняемой базы данных
/opt/postgres/9.0/bin/pg_dump	Путь к бинарнику pg_dump
-час	Указывает имя хоста на компьютере, на котором работает сервер. Пример: localhost
-п	Задает порт TCP или локальное расширение файла сокета домена Unix, на котором сервер прослушивает подключения, пример 5432
-U	Имя пользователя для подключения.

замечания

1. Если есть инструмент резервного копирования, такой как HDPS или Symantec Backup , ... **Перед каждым запуском** необходимо освободить каталог резервного копирования.

Во избежание загромождения инструмента резервного копирования, поскольку предполагается, что резервная копия старых файлов будет выполнена.

   Чтобы включить эту функцию, пожалуйста, удалите строку № 3.

```
rm -R / save_db / *
```

2. В случае, когда бюджет не позволяет использовать инструмент резервного копирования, всегда можно использовать планировщик задач (команда cron).

   Следующая команда используется для редактирования таблицы cron для текущего пользователя.

```
crontab -e
```

   Запланируйте запуск сценария с календарем в 11 часов.

```
0 23 * * * /saveProdDb.sh
```

Examples

saveProdDb.sh

В общем, мы пытаемся создать резервную копию базы данных с клиентом pgAdmin. Ниже приведен сценарий sh, используемый для сохранения базы данных (под Linux) в двух форматах:

- **Файл SQL** : для возможного возобновления данных в любой версии PostgreSQL.
- **Файл дампа** : для более высокой версии, чем текущая версия.

```
#!/bin/sh
cd /save_db
#rm -R /save_db/*
DATE=$(date +%d-%m-%Y-%Hh%M)
echo -e "Sauvegarde de la base du ${DATE}"
mkdir prodDir${DATE}
cd prodDir${DATE}

#dump file
/opt/postgres/9.0/bin/pg_dump -i -h localhost -p 5432 -U postgres -F c -b -w -v -f
```

```
"dbprod${DATE}.backup" dbprod

#SQL file
/opt/postgres/9.0/bin/pg_dump -i -h localhost -p 5432 -U postgres --format plain --verbose  -f
"dbprod${DATE}.sql" dbprod
```

Прочитайте Резервный сценарий для производственной БД онлайн:
https://riptutorial.com/ru/postgresql/topic/7974/резервный-сценарий-для-производственной-бд

глава 20: Рекурсивные запросы

Вступление

Рекурсивных запросов нет!

Examples

Сумма целых чисел

```
WITH RECURSIVE t(n) AS (
    VALUES (1)
  UNION ALL
    SELECT n+1 FROM t WHERE n < 100
)
SELECT sum(n) FROM t;
```

Ссылка на документацию

Прочитайте Рекурсивные запросы онлайн: https://riptutorial.com/ru/postgresql/topic/9025/рекурсивные-запросы

глава 21: Советы и подсказки Postgres

Examples

Альтернатива DATEADD в Postgres

- `SELECT CURRENT_DATE + '1 day'::INTERVAL`
- `SELECT '1999-12-11'::TIMESTAMP + '19 days'::INTERVAL`
- `SELECT '1 month'::INTERVAL + '1 month 3 days'::INTERVAL`

Comma разделяет значения столбца

```
SELECT
    string_agg(<TABLE_NAME>.<COLUMN_NAME>, ',')
FROM
    <SCHEMA_NAME>.<TABLE_NAME> T
```

Удаление дубликатов записей из таблицы postgres

```
DELETE
    FROM <SCHEMA_NAME>.<Table_NAME>
WHERE
    ctid NOT IN
        (
        SELECT
            MAX(ctid)
        FROM
            <SCHEMA_NAME>.<TABLE_NAME>
        GROUP BY
            <SCHEMA_NAME>.<TABLE_NAME>.*
        )
;
```

Обновить запрос с объединением двух альтернативных таблиц, поскольку Postresql не поддерживает соединение в запросе обновления.

```
    update <SCHEMA_NAME>.<TABLE_NAME_1> AS A
    SET <COLUMN_1> = True
    FROM <SCHEMA_NAME>.<TABLE_NAME_2> AS B
    WHERE
        A.<COLUMN_2> = B.<COLUMN_2> AND
        A.<COLUMN_3> = B.<COLUMN_3>
```

Разница между двумя датами даты по месяцам и годам

Месячная разница между двумя датами (временная метка)

```
select
```

```
    (
        (DATE_PART('year', AgeonDate) - DATE_PART('year', tmpdate)) * 12
        +
        (DATE_PART('month', AgeonDate) - DATE_PART('month', tmpdate))
    )
from dbo."Table1"
```

Разница в два раза между двумя датами (временная метка)

```
select (DATE_PART('year', AgeonDate) - DATE_PART('year', tmpdate)) from dbo."Table1"
```

Запрос на копирование / перемещение / перенос данных таблицы из одной базы данных в другую таблицу базы данных с той же схемой

Первое исполнение

```
CREATE EXTENSION DBLINK;
```

затем

```
INSERT INTO
    <SCHEMA_NAME>.<TABLE_NAME_1>
SELECT *
FROM
    DBLINK(
    'HOST=<IP-ADDRESS> USER=<USERNAME> PASSWORD=<PASSWORD> DBNAME=<DATABASE>',
    'SELECT * FROM <SCHEMA_NAME>.<TABLE_NAME_2>')
    AS <TABLE_NAME>
    (
    <COLUMN_1> <DATATYPE_1>,
    <COLUMN_1> <DATATYPE_2>,
    <COLUMN_1> <DATATYPE_3>
    );
```

Прочитайте Советы и подсказки Postgres онлайн:
https://riptutorial.com/ru/postgresql/topic/7433/советы-и-подсказки-postgres

глава 22: Совокупные функции

Examples

Простая статистика: min (), max (), avg ()

Чтобы определить некоторую простую статистику значения в столбце таблицы, вы можете использовать агрегатную функцию.

Если ваш `individuals` стол:

название	Возраст
Алли	17
Аманда	14
Alana	20

Вы можете написать этот оператор, чтобы получить минимальное, максимальное и среднее значение:

```
SELECT min(age), max(age), avg(age)
FROM individuals;
```

Результат:

мин	Максимум	средний
14	20	17

string_agg (выражение, разделитель)

Вы можете конкатенировать строки, разделенные разделителем, с помощью функции `string_agg()` .

Если ваш `individuals` стол:

название	Возраст	Страна
Алли	15	Соединенные Штаты Америки
Аманда	14	Соединенные Штаты Америки

название	Возраст	Страна
Alana	20	Россия

Вы можете написать `SELECT ... GROUP BY` для получения имен из каждой страны:

```
SELECT string_agg(name, ', ') AS names, country
FROM individuals
GROUP BY country;
```

Обратите внимание, что вам нужно использовать предложение `GROUP BY` , потому что `string_agg()` является агрегированной функцией.

Результат:

имена	страна
Элли, Аманда	Соединенные Штаты Америки
Alana	Россия

Подробнее описанная здесь функция агрегации PostgreSQL

regr_slope (Y, X): наклон линейного уравнения с наименьшим квадратом, определяемого парами (X, Y)

Чтобы проиллюстрировать, как использовать regr_slope (Y, X), я применил его к реальной проблеме. В Java, если вы не правильно очищаете память, мусор может застрять и заполнить память. Каждый час вы отправляете статистику о загрузке памяти различными классами и загружаете ее в базу данных postgres для анализа.

Все кандидаты на утечку памяти будут иметь тенденцию потреблять больше памяти с большим количеством времени. Если вы планируете эту тенденцию, вы можете представить линию, идущую вверх и влево:

```
    ^
    |
s   |  Legend:
i   |  *  - data point
z   |  -- - trend
e   |
(   |
b   |                  *
y   |                      --
t   |                   --
e   |              * --     *
s   |           --
)   |       *--        *
    |     --     *
```

```
    |  -- *
   ------------------------------------->
                      time
```

Предположим, у вас есть таблица, содержащая данные гистограммы дампа кучи (сопоставление классов с тем, сколько памяти они потребляют):

```
CREATE TABLE heap_histogram (
    -- when the heap histogram was taken
    histwhen timestamp without time zone NOT NULL,
    -- the object type bytes are referring to
    -- ex: java.util.String
    class character varying NOT NULL,
    -- the size in bytes used by the above class
    bytes integer NOT NULL
);
```

Чтобы вычислить наклон для каждого класса, мы группируем по классу. Предложение HAVING> 0 гарантирует, что мы получим только кандидатов с положительным отскоком (линия вверх и влево). Мы сортируем по нисходящему склону, чтобы мы получили классы с наибольшим увеличением объема памяти вверху.

```
-- epoch returns seconds
SELECT class, REGR_SLOPE(bytes,extract(epoch from histwhen)) as slope
    FROM public.heap_histogram
    GROUP BY class
    HAVING REGR_SLOPE(bytes,extract(epoch from histwhen)) > 0
    ORDER BY slope DESC ;
```

Выход:

```
         class              |        slope
----------------------------+---------------------
 java.util.ArrayList        |     71.7993806279174
 java.util.HashMap          |     49.0324576155785
 java.lang.String           |     31.7770770326123
 joe.schmoe.BusinessObject  |     23.2036817108056
 java.lang.ThreadLocal      |     20.9013528767851
```

Из вывода видно, что потребление памяти java.util.ArrayList увеличивается быстрее всего на 71.799 байт в секунду и потенциально является частью утечки памяти.

Прочитайте Совокупные функции онлайн: https://riptutorial.com/ru/postgresql/topic/4803/совокупные-функции

глава 23: Создание таблицы

Examples

Создание таблицы с помощью основного ключа

```
CREATE TABLE person (
    person_id BIGINT NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    first_name VARCHAR(255),
    address VARCHAR(255),
    city VARCHAR(255),
    PRIMARY KEY (person_id)
);
```

Кроме того, вы можете поместить ограничение `PRIMARY KEY` непосредственно в определение столбца:

```
CREATE TABLE person (
    person_id BIGINT NOT NULL PRIMARY KEY,
    last_name VARCHAR(255) NOT NULL,
    first_name VARCHAR(255),
    address VARCHAR(255),
    city VARCHAR(255)
);
```

Рекомендуется использовать имена нижнего регистра для таблицы, а также все столбцы. Если вы используете имена верхнего регистра, такие как `Person` вы должны были бы обернуть это имя в двойные кавычки (`"Person"`) в каждом запросе, потому что PostgreSQL обеспечивает свертывание фреймов.

Показать определение таблицы

Откройте инструмент командной строки `psql` , подключенный к базе данных, где находится ваша таблица. Затем введите следующую команду:

```
\d tablename
```

Чтобы получить расширенный тип информации

```
\d+ tablename
```

Если вы забыли имя таблицы, просто введите \ d в psql, чтобы получить список таблиц и представлений в текущей базе данных.

Создать таблицу из выбранного

Допустим, у вас есть таблица, называемая человеком:

```
CREATE TABLE person (
    person_id BIGINT NOT NULL,
    last_name VARCHAR(255) NOT NULL,
    first_name VARCHAR(255),
    age INT NOT NULL,
    PRIMARY KEY (person_id)
);
```

Вы можете создать новую таблицу людей старше 30 лет следующим образом:

```
CREATE TABLE people_over_30 AS SELECT * FROM person WHERE age > 30;
```

Создать таблицу с разметкой

Вы можете создавать таблицы с разметкой, чтобы значительно ускорить выполнение таблиц. Незарегистрированный стол пропускает `write-ahead` журнала записи, что означает, что он не является аварийным и неспособен реплицироваться.

```
CREATE UNLOGGED TABLE person (
    person_id BIGINT NOT NULL PRIMARY KEY,
    last_name VARCHAR(255) NOT NULL,
    first_name VARCHAR(255),
    address VARCHAR(255),
    city VARCHAR(255)
);
```

Создайте таблицу, которая ссылается на другую таблицу.

В этом примере таблица пользователя будет иметь столбец, который ссылается на таблицу Агентства.

```
CREATE TABLE agencies ( -- first create the agency table
  id SERIAL PRIMARY KEY,
  name TEXT NOT NULL
)

CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  agency_id NOT NULL INTEGER REFERENCES agencies(id) DEFERRABLE INITIALLY DEFERRED -- this is
going to references your agency table.
)
```

Прочитайте Создание таблицы онлайн: https://riptutorial.com/ru/postgresql/topic/2430/создание-таблицы

глава 24: Типы данных

Вступление

PostgreSQL имеет богатый набор собственных типов данных, доступных пользователям. Пользователи могут добавлять новые типы в PostgreSQL с помощью команды CREATE TYPE.

https://www.postgresql.org/docs/9.6/static/datatype.html

Examples

Числовые типы

название	Размер хранилища	Описание	Спектр
`smallint`	2 байта	малое целое число	-32768 - +32767
`integer`	4 байта	ypical выбор для целого	-2147483648 - +2147483647
`bigint`	8 байт	целочисленный целочисленный	-9223372036854775808 - +9223372036854775807
`decimal`	переменная	заданная пользователем точность, точная	до 131072 цифр до десятичной точки; до 16383 цифр после десятичной точки
`numeric`	переменная	заданная пользователем точность, точная	до 131072 цифр до десятичной точки; до 16383 цифр после десятичной точки
`real`	4 байта	переменная точность, неточная	6 десятичных цифр
`double precision`	8 байт	переменная точность, неточная	15 десятичных цифр
`smallserial`	2 байта	небольшое автоинкрементное целое число	1 до 32767
`serial`	4 байта	автоинкрементное	1 до 2147483647

название	Размер хранилища	Описание	Спектр
		целое число	
bigserial	8 байт	большое число автоинкрементных чисел	1 до 9223372036854775807
int4range		Диапазон целых чисел	
int8range		Диапазон bigint	
numrange		Диапазон числовых	

Типы даты и времени

название	Размер хранилища	Описание	Низкая стоимость	Высокое значение	разрешение
timestamp (без часового пояса)	8 байт	и дата и время (без часового пояса)	4713 г. до н.	294276 н.э.	1 микросекунда / 14 цифр
timestamp (с часовым поясом)	8 байт	дата и время, с часовым поясом	4713 г. до н.	294276 н.э.	1 микросекунда / 14 цифр
date	4 байта	дата (нет времени суток)	4713 г. до н.	5874897 гг. Н.э.	1 день
time (без часового пояса)	8 байт	время суток (нет даты)	00:00:00	24:00:00	1 микросекунда / 14 цифр
time (с часовым поясом)	12 байт	время суток, с часовым поясом	00: 00: 00 + 1459	24: 00: 00-1459	1 микросекунда / 14 цифр
interval	16 байт	интервал времени	-178000000 лет	178000000 лет	1 микросекунда / 14 цифр
tsrange		диапазон			

название	Размер хранилища	Описание	Низкая стоимость	Высокое значение	разрешение
		временной метки без часового пояса			
`tstzrange`		диапазон временной метки с часовым поясом			
`daterange`		диапазон дат			

Геометрические типы

название	Размер хранилища	Описание	Представление
`point`	16 байт	Точка на плоскости	(X, y)
`line`	32 байт	Бесконечная линия	{A, B, C},
`lseg`	32 байт	Конечный сегмент	((X1, y1), (x2, y2))
`box`	32 байт	Прямоугольная коробка	((X1, y1), (x2, y2))
`path`	16 + 16n байт	Закрытый путь (аналогично полигону)	((X1, y1), ...)
`path`	16 + 16n байт	Открытый путь	[(X1, y1), ...]
`polygon`	40 + 16n байт	Многоугольник (аналогично замкнутому пути)	((X1, y1), ...)
`circle`	24 байта	Круг	<(x, y), r> (центральная точка и радиус)

Типы сетевых адресов

название	Размер хранилища	Описание
`cidr`	7 или 19 байт	Сети IPv4 и IPv6

название	Размер хранилища	Описание
`inet`	7 или 19 байт	Хосты и сети IPv4 и IPv6
`macaddr`	6 байт	MAC-адреса

Типы символов

название	Описание
`character varying(n)` , `varchar(n)`	переменная длина с ограничением
`character(n)` , `char(n)`	фиксированная длина, пустая прокладка
`text`	переменная неограниченная длина

Массивы

В PostgreSQL вы можете создавать массивы любого встроенного, определенного пользователем или перечисляемого типа. По умолчанию для массива нет ограничений, но вы *можете* указать его.

Объявление массива

```
SELECT integer[];
SELECT integer[3];
SELECT integer[][];
SELECT integer[3][3];
SELECT integer ARRAY;
SELECT integer ARRAY[3];
```

Создание массива

```
SELECT '{0,1,2}';
SELECT '{{0,1},{1,2}}';
SELECT ARRAY[0,1,2];
SELECT ARRAY[ARRAY[0,1],ARRAY[1,2]];
```

Доступ к массиву

По умолчанию PostgreSQL использует одноуровневое соглашение о нумерации для массивов, то есть массив из n элементов начинается с `array[1]` и заканчивается `array[n]` .

```
--accesing a spefific element
WITH arr AS (SELECT ARRAY[0,1,2] int_arr) SELECT int_arr[1] FROM arr;
```

```
int_arr
---------
        0
(1 row)

--sclicing an array
WITH arr AS (SELECT ARRAY[0,1,2] int_arr) SELECT int_arr[1:2] FROM arr;

int_arr
---------
    {0,1}
(1 row)
```

Получение информации о массиве

```
--array dimensions (as text)
with arr as (select ARRAY[0,1,2] int_arr) select array_dims(int_arr) from arr;

array_dims
------------
       [1:3]
(1 row)

--length of an array dimension
 WITH arr AS (SELECT ARRAY[0,1,2] int_arr) SELECT array_length(int_arr,1) FROM arr;

 array_length
 --------------
              3
 (1 row)

--total number of elements across all dimensions
 WITH arr AS (SELECT ARRAY[0,1,2] int_arr) SELECT cardinality(int_arr) FROM arr;

 cardinality
 -------------
             3
 (1 row)
```

Функции массива

будет добавлено

Прочитайте Типы данных онлайн: https://riptutorial.com/ru/postgresql/topic/8976/типы-данных

глава 25: Триггеры и триггерные функции

Вступление

Триггер будет связан с указанной таблицей или представлением и будет выполнять указанную функцию function_name при возникновении определенных событий.

замечания

Пожалуйста, используйте ниже ссылку для полного обзора:

- **Триггеры** : https://www.postgresql.org/docs/current/static/sql-createtrigger.html
- **Триггерные функции** : https://www.postgresql.org/docs/current/static/plpgsql-trigger.html

Examples

Базовая функция триггера PL / pgSQL

Это простая триггерная функция.

```
CREATE OR REPLACE FUNCTION my_simple_trigger_function()
RETURNS trigger AS
$BODY$

BEGIN
    -- TG_TABLE_NAME :name of the table that caused the trigger invocation
IF (TG_TABLE_NAME = 'users') THEN

    --TG_OP : operation the trigger was fired
  IF (TG_OP = 'INSERT') THEN
    --NEW.id is holding the new database row value (in here id is the id column in users
table)
    --NEW will return null for DELETE operations
    INSERT INTO log_table (date_and_time, description) VALUES (now(), 'New user inserted. User
ID: '|| NEW.id);
    RETURN NEW;

  ELSIF (TG_OP = 'DELETE') THEN
    --OLD.id is holding the old database row value (in here id is the id column in users
table)
    --OLD will return null for INSERT operations
    INSERT INTO log_table (date_and_time, description) VALUES (now(), 'User deleted.. User ID:
' || OLD.id);
    RETURN OLD;

  END IF;

RETURN null;
END IF;
```

```
END;
$BODY$
LANGUAGE plpgsql VOLATILE
COST 100;
```

Добавление этой триггерной функции в таблицу `users`

```
CREATE TRIGGER my_trigger
AFTER INSERT OR DELETE
ON users
FOR EACH ROW
EXECUTE PROCEDURE my_simple_trigger_function();
```

Тип триггеров

Триггер можно указать для запуска:

- `BEFORE` выполнением операции в строке - вставка, обновление или удаление;
- `AFTER` завершения операции - вставка, обновление или удаление;
- `INSTEAD OF` операции в случае вставки, обновления или удаления в представлении.

Триггер, который отмечен:

- `FOR EACH ROW` вызывается один раз для каждой строки, которую операция модифицирует;
- `FOR EACH STATEMENT` называется anyde для любой операции.

Подготовка к выполнению примеров

```
CREATE TABLE company (
    id          SERIAL PRIMARY KEY NOT NULL,
    name        TEXT NOT NULL,
    created_at  TIMESTAMP,
    modified_at TIMESTAMP DEFAULT NOW()
)

CREATE TABLE log (
    id          SERIAL PRIMARY KEY NOT NULL,
    table_name  TEXT NOT NULL,
    table_id    TEXT NOT NULL,
    description TEXT NOT NULL,
    created_at  TIMESTAMP DEFAULT NOW()
)
```

Одиночный триггер

Шаг 1: создайте свою функцию

```
CREATE OR REPLACE FUNCTION add_created_at_function()
  RETURNS trigger AS $BODY$
BEGIN
  NEW.created_at := NOW();
  RETURN NEW;
END $BODY$
LANGUAGE plpgsql;
```

Шаг 2: создайте триггер

```
CREATE TRIGGER add_created_at_trigger
BEFORE INSERT
ON company
FOR EACH ROW
EXECUTE PROCEDURE add_created_at_function();
```

Шаг 3: проверьте его

```
INSERT INTO company (name) VALUES ('My company');
SELECT * FROM company;
```

Триггер для нескольких целей

Шаг 1: создайте свою функцию

```
CREATE OR REPLACE FUNCTION add_log_function()
  RETURNS trigger AS $BODY$
DECLARE
  vDescription TEXT;
  vId INT;
  vReturn RECORD;
BEGIN
    vDescription := TG_TABLE_NAME || ' ';
     IF (TG_OP = 'INSERT') THEN
        vId := NEW.id;
        vDescription := vDescription || 'added. Id: ' || vId;
        vReturn := NEW;
    ELSIF (TG_OP = 'UPDATE') THEN
        vId := NEW.id;
        vDescription := vDescription || 'updated. Id: ' || vId;
        vReturn := NEW;
    ELSIF (TG_OP = 'DELETE') THEN
        vId := OLD.id;
        vDescription := vDescription || 'deleted. Id: ' || vId;
        vReturn := OLD;
    END IF;
```

```
    RAISE NOTICE 'TRIGER called on % - Log: %', TG_TABLE_NAME, vDescription;

    INSERT INTO log
        (table_name, table_id, description, created_at)
        VALUES
        (TG_TABLE_NAME, vId, vDescription, NOW());

    RETURN vReturn;
END $BODY$
  LANGUAGE plpgsql;
```

Шаг 2: создайте триггер

```
CREATE TRIGGER add_log_trigger
AFTER INSERT OR UPDATE OR DELETE
ON company
FOR EACH ROW
EXECUTE PROCEDURE add_log_function();
```

Шаг 3: проверьте его

```
INSERT INTO company (name) VALUES ('Company 1');
INSERT INTO company (name) VALUES ('Company 2');
INSERT INTO company (name) VALUES ('Company 3');
UPDATE company SET name='Company new 2' WHERE name='Company 2';
DELETE FROM company WHERE name='Company 1';
SELECT * FROM log;
```

Прочитайте Триггеры и триггерные функции онлайн:
https://riptutorial.com/ru/postgresql/topic/6957/триггеры-и-триггерные-функции

глава 26: Триггеры событий

Вступление

Событие Триггеры будут запущены, когда связанное с ними событие происходит в базе данных.

замечания

Используйте приведенную ниже ссылку для полного просмотра триггеров событий в PostgreSQL.

https://www.postgresql.org/docs/9.3/static/event-trigger-definition.html

Examples

Запуск событий запуска DDL

Тип события-

- `DDL_COMMAND_START`
- `DDL_COMMAND_END`
- SQL_DROP

Это пример создания Event Trigger и регистрации событий `DDL_COMMAND_START` .

```
CREATE TABLE TAB_EVENT_LOGS(
  DATE_TIME TIMESTAMP,
  EVENT_NAME TEXT,
  REMARKS TEXT
);

CREATE OR REPLACE FUNCTION FN_LOG_EVENT()
  RETURNS EVENT_TRIGGER
  LANGUAGE SQL
  AS
  $main$
    INSERT INTO TAB_EVENT_LOGS(DATE_TIME,EVENT_NAME,REMARKS)
      VALUES(NOW(),TG_TAG,'Event Logging');
  $main$;

CREATE EVENT TRIGGER TRG_LOG_EVENT ON DDL_COMMAND_START
  EXECUTE PROCEDURE FN_LOG_EVENT();
```

Прочитайте Триггеры событий онлайн: https://riptutorial.com/ru/postgresql/topic/9255/триггеры-событий

глава 27: Управление ролями

Синтаксис

- `CREATE ROLE name [ [ WITH ] option [ ... ] ]`
- `CREATE USER name [ [ WITH ] option [ ... ] ]`
- `where option can be: SUPERUSER | NOSUPERUSER | CREATEDB | NOCREATEDB | CREATEROLE | NOCREATEROLE | CREATEUSER | NOCREATEUSER | INHERIT | NOINHERIT | LOGIN | NOLOGIN | CONNECTION LIMIT connlimit | [ENCRYPTED | UNENCRYPTED] PASSWORD 'password' | VALID UNTIL 'timestamp' | IN ROLE role_name [, ...] | IN GROUP role_name [, ...] | ROLE role_name [, ...] | ADMIN role_name [, ...] | USER role_name [, ...] | SYSID uid`

Examples

Создайте пользователя с паролем

Как правило, вам следует избегать использования роли базы данных по умолчанию (часто `postgres`) в вашем приложении. Вместо этого вы должны создать пользователя с более низким уровнем привилегий. Здесь мы делаем одно имя `niceusername` и даем ему пароль `very-strong-password`

```
CREATE ROLE niceusername with PASSWORD 'very-strong-password' LOGIN;
```

Проблема заключается в том, что запросы, введенные в консоль `psql` сохраняются в файле истории `.psql_history` в домашнем каталоге пользователя и могут также регистрироваться в журнале сервера базы данных PostgreSQL, тем самым раскрывая пароль.

Чтобы этого избежать, используйте команду `\password` для установки пароля пользователя. Если пользователь, выдающий команду, является суперпользователем, текущий пароль не будет задан. (Должен быть суперпользователем, чтобы изменять пароли суперпользователей)

```
CREATE ROLE niceusername with LOGIN;
\password niceusername
```

Создание роли и соответствующей базы данных

Чтобы поддерживать данное приложение, вы часто создаете новую роль и базу данных для соответствия.

Командами оболочки для запуска будут следующие:

```
$ createuser -P blogger
Enter password for the new role: ********
```

```
Enter it again: ********

$ createdb -O blogger blogger
```

Это предполагает, что `pg_hba.conf` настроен правильно, что, вероятно, выглядит так:

```
# TYPE  DATABASE        USER            ADDRESS                 METHOD
host    sameuser        all             localhost               md5
local   sameuser        all                                     md5
```

Предоставить и отменить привилегии.

Предположим, что у нас три пользователя:

1. Администратор базы данных> admin
2. Приложение с полным доступом к ее данным> read_write
3. Доступ только для чтения> read_only

```
--ACCESS DB
REVOKE CONNECT ON DATABASE nova FROM PUBLIC;
GRANT  CONNECT ON DATABASE nova  TO user;
```

С помощью вышеуказанных запросов недоверенные пользователи больше не могут подключаться к базе данных.

```
--ACCESS SCHEMA
REVOKE ALL     ON SCHEMA public FROM PUBLIC;
GRANT  USAGE   ON SCHEMA public  TO user;
```

Следующий набор запросов отменяет все привилегии от пользователей, не прошедших проверку подлинности, и предоставляет ограниченный набор привилегий для пользователя `read_write` .

```
--ACCESS TABLES
REVOKE ALL ON ALL TABLES IN SCHEMA public FROM PUBLIC ;
GRANT SELECT                         ON ALL TABLES IN SCHEMA public TO read_only ;
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO read_write ;
GRANT ALL                            ON ALL TABLES IN SCHEMA public TO admin ;

--ACCESS SEQUENCES
REVOKE ALL   ON ALL SEQUENCES IN SCHEMA public FROM PUBLIC;
GRANT SELECT ON ALL SEQUENCES IN SCHEMA public TO read_only; -- allows the use of CURRVAL
GRANT UPDATE ON ALL SEQUENCES IN SCHEMA public TO read_write; -- allows the use of NEXTVAL and
SETVAL
GRANT USAGE  ON ALL SEQUENCES IN SCHEMA public TO read_write; -- allows the use of CURRVAL and
NEXTVAL
GRANT ALL    ON ALL SEQUENCES IN SCHEMA public TO admin;
```

Изменить поиск по умолчанию для пользователя

С помощью приведенных ниже команд пользовательский путь поиска по умолчанию может быть установлен.

1. Проверьте путь поиска до установки схемы по умолчанию.

```
postgres=# \c postgres user1
You are now connected to database "postgres" as user "user1".
postgres=> show search_path;
  search_path
----------------
 "$user",public
(1 row)
```

2. Установите `search_path` с `alter user` команды `alter user` чтобы добавить новую схему `my_schema`

```
postgres=> \c postgres postgres
You are now connected to database "postgres" as user "postgres".
postgres=# alter user user1 set search_path='my_schema, "$user", public';
ALTER ROLE
```

3. Проверьте результат после выполнения.

```
postgres=# \c postgres user1
Password for user user1:
You are now connected to database "postgres" as user "user1".
postgres=> show search_path;
 search_path
-------------
 my_schema, "$user", public
(1 row)
```

Альтернатива:

```
postgres=# set role user1;
postgres=# show search_path;
 search_path
-------------
 my_schema, "$user", public
(1 row)
```

Предоставьте права доступа на объекты, созданные в будущем.

Предположим, что у нас `three users` :

1. Администратор базы данных> `admin`
2. Приложение с полным доступом к ее данным> `read_write`
3. Доступ только для чтения> `read_only`

С помощью нижеуказанных запросов вы можете установить права доступа для объектов, созданных в будущем в указанной схеме.

```
ALTER DEFAULT PRIVILEGES IN SCHEMA myschema GRANT SELECT                    ON TABLES TO
read_only;
ALTER DEFAULT PRIVILEGES IN SCHEMA myschema GRANT SELECT,INSERT,DELETE,UPDATE ON TABLES TO
read_write;
ALTER DEFAULT PRIVILEGES IN SCHEMA myschema GRANT ALL                       ON TABLES TO
admin;
```

Или вы можете установить права доступа для объектов, созданных в будущем указанным пользователем.

```
ALTER DEFAULT PRIVILEGES FOR ROLE admin GRANT SELECT  ON TABLES TO read_only;
```

Создать пользователя только для чтения

```
CREATE USER readonly WITH ENCRYPTED PASSWORD 'yourpassword';
GRANT CONNECT ON DATABASE <database_name> to readonly;

GRANT USAGE ON SCHEMA public to readonly;
GRANT SELECT ON ALL SEQUENCES IN SCHEMA public TO readonly;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO readonly;
```

Прочитайте Управление ролями онлайн: https://riptutorial.com/ru/postgresql/topic/1572/управление-ролями

глава 28: Функции окна

Examples

общий пример

Подготовка данных:

```
create table wf_example(i int, t text,ts timestamptz,b boolean);
insert into wf_example select 1,'a','1970.01.01',true;
insert into wf_example select 1,'a','1970.01.01',false;
insert into wf_example select 1,'b','1970.01.01',false;
insert into wf_example select 2,'b','1970.01.01',false;
insert into wf_example select 3,'b','1970.01.01',false;
insert into wf_example select 4,'b','1970.02.01',false;
insert into wf_example select 5,'b','1970.03.01',false;
insert into wf_example select 2,'c','1970.03.01',true;
```

Бег:

```
select *
  , dense_rank() over (order by i) dist_by_i
  , lag(t) over () prev_t
  , nth_value(i, 6) over () nth
  , count(true) over (partition by i) num_by_i
  , count(true) over () num_all
  , ntile(3) over() ntile
from wf_example
;
```

Результат:

```
 i | t |           ts           | b | dist_by_i | prev_t | nth | num_by_i | num_all | ntile
---+---+------------------------+---+-----------+--------+-----+----------+---------+-------
 1 | a | 1970-01-01 00:00:00+01 | f |         1 |        |   3 |        3 |       8 |     1
 1 | a | 1970-01-01 00:00:00+01 | t |         1 | a      |   3 |        3 |       8 |     1
 1 | b | 1970-01-01 00:00:00+01 | f |         1 | a      |   3 |        3 |       8 |     1
 2 | c | 1970-03-01 00:00:00+01 | t |         2 | b      |   3 |        2 |       8 |     2
 2 | b | 1970-01-01 00:00:00+01 | f |         2 | c      |   3 |        2 |       8 |     2
 3 | b | 1970-01-01 00:00:00+01 | f |         3 | b      |   3 |        1 |       8 |     2
 4 | b | 1970-02-01 00:00:00+01 | f |         4 | b      |   3 |        1 |       8 |     3
 5 | b | 1970-03-01 00:00:00+01 | f |         5 | b      |   3 |        1 |       8 |     3
(8 rows)
```

Объяснение:

dist_by_i : `dense_rank() over (order by i)` - это как row_number для разных значений. Может использоваться для количества различных значений i (`count(DISTINCT i)` wold не работает). Просто используйте максимальное значение.

prev_t : `lag(t) over ()` - это предыдущее значение t во всем окне. помните, что для первой строки она равна нулю.

nth : `nth_value(i, 6) over ()` - значение столбца i шестой строки во всем окне

num_by_i : `count(true) over (partition by i)` - количество строк для каждого значения i

num_all : `count(true) over ()` - количество строк по всему окну

ntile : `ntile(3) over()` разбивает все окно на 3 (насколько это возможно), равное по количеству частей

значения столбцов vs dense_rank vs rank vs row_number

здесь вы можете найти функции.

С помощью таблицы wf_example, созданной в предыдущем примере, запустите:

```
select i
  , dense_rank() over (order by i)
  , row_number() over ()
  , rank() over (order by i)
from wf_example
```

Результат:

```
 i | dense_rank | row_number | rank
---+------------+------------+------
 1 |          1 |          1 |    1
 1 |          1 |          2 |    1
 1 |          1 |          3 |    1
 2 |          2 |          4 |    4
 2 |          2 |          5 |    4
 3 |          3 |          6 |    6
 4 |          4 |          7 |    7
 5 |          5 |          8 |    8
```

- *dense_rank* заказывает **VALUES i** по внешнему виду в окне. `i=1` , поэтому первая строка имеет плотность, то следующее и третье значение i не изменяется, поэтому отображается значение `dense_rank` *1* - значение FIRST не изменяется. четвертая строка `i=2` , это второе значение *i* встречается, поэтому `dense_rank` показывает 2, а также для следующей строки. Затем он соответствует значению `i=3` в 6-й строке, поэтому он показывает 3. То же самое для остальных двух значений *i* . Таким образом, последнее значение `dense_rank` - это количество различных значений *i* .
- *row_number* заказывает **ROWS,** как они перечислены.
- *rank* Чтобы не путать с `dense_rank` эта функция заказывает значение **ROW NUMBER** из **i** . Таким образом, он начинается с трех, но имеет следующее значение 4, что

означает, что `i=2` (новое значение) встречается в строке 4. Тот же `i=3` был встречен в строке 6. Etc ..

Прочитайте Функции окна онлайн: https://riptutorial.com/ru/postgresql/topic/7421/функции-окна

глава 29: Экспортировать заголовок таблицы базы данных PostgreSQL и данные в файл CSV

Вступление

Из инструмента управления Adminer у него есть экспорт в файл csv-файла для базы данных mysql, но недоступен для базы данных postgresql. Здесь я покажу команду для экспорта CSV для базы данных postgresql.

Examples

Экспортировать таблицу PostgreSQL в csv с заголовком для некоторого столбца (ов)

```
COPY products(is_public, title, discount) TO 'D:\csv_backup\products_db.csv' DELIMITER ',' CSV
HEADER;

COPY categories(name) TO 'D:\csv_backup\categories_db.csv' DELIMITER ',' CSV HEADER;
```

Полное резервное копирование таблицы в csv с заголовком

```
COPY products TO 'D:\csv_backup\products_db.csv' DELIMITER ',' CSV HEADER;

COPY categories TO 'D:\csv_backup\categories_db.csv' DELIMITER ',' CSV HEADER;
```

копировать из запроса

```
copy (select oid,relname from pg_class limit 5) to stdout;
```

Прочитайте Экспортировать заголовок таблицы базы данных PostgreSQL и данные в файл CSV онлайн: https://riptutorial.com/ru/postgresql/topic/8643/экспортировать-заголовок-таблицы-базы-данных-postgresql-и-данные-в-файл-csv

кредиты

S. No	Главы	Contributors
1	Начало работы с postgresql	a_horse_with_no_name, Alison S, AndrewCichocki, Ben, Ben H , bignose, Community, Dakota Wagner, DeadEye, Demircan Celebi, Dmitri Goldring, e4c5, , jasonszhao, Kirk Roybal, Marek Skiba, Mokadillion, Patrick, user_0
2	COALESCE	Mokadillion
3	EXTENSION dblink и postgres_fdw	Riya Bansal, YCF_L
4	ВСТАВИТЬ	chalitha geekiyanage, e4c5, gpdude_, KIM, lamorach, leeor, Nathaniel Waisbrot, Patrick, Vao Tsun
5	ВЫБРАТЬ	YCF_L
6	Высокая доступность PostgreSQL	gpdude_, Patrick
7	Даты, отметки времени и интервалы	KIM, Nuri Tasdemir, Patrick, Tom Gerken
8	Доступ к данным программно	AstraSerg, brichins, greg, Laurenz Albe
9	Комментарии в postgresql	Ben, KIRAN KUMAR MATAM
10	Криптографические функции Postgres	Ben H, skj123
11	Найти длину строки / длину символа	Mohamed Navas
12	наследование	evuez
13	ОБНОВИТЬ	frlan, leeor
14	Общие выражения таблицы (WITH)	Daniel Lyons, Jakub Fedyczak, Kevin Sylvestre

15	Поддержка JSON	Clodoaldo Neto, commonSenseCode, jgm, KIRAN KUMAR MATAM, mnoronha, Peter Krauss
16	Подключение к PostgreSQL с Java	Laurenz Albe
17	Программирование с помощью PL / pgSQL	AndrewCichocki, Ben H, Goerman, Laurenz Albe, Vao Tsun
18	Резервное копирование и восстановление	ankidaemon, Ben H, Daniel Lyons, e4c5, Laurel, mnoronha
19	Резервный сценарий для производственной БД	bilelovitch
20	Рекурсивные запросы	Ben H
21	Советы и подсказки Postgres	Ben H, skj123, user_0, YCF_L
22	Совокупные функции	Alison S, joseph, Kirill Sokolov, Patrick
23	Создание таблицы	e4c5, Jefferson, KIM, leeor, Patrick
24	Типы данных	Ben H, user_0
25	Триггеры и триггерные функции	chalitha geekiyanage, mnoronha, Udlei Nati
26	Триггеры событий	Ben H, Tajinder, Udlei Nati
27	Управление ролями	Ben, Ben H, bilelovitch, Blackus, Daniel Lyons, e4c5, greg, KIM, Laurenz Albe, mnoronha, Reboot
28	Функции окна	mnoronha, Vao Tsun
29	Экспортировать заголовок таблицы базы данных PostgreSQL и	Vao Tsun, wOwhOw

	данные в файл CSV	