Kostenloses eBook

LERNEN postgresql

Free unaffiliated eBook created from
Stack Overflow contributors.

#postgresql

Inhaltsverzeichnis

Über

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

Kapitel 1: Erste Schritte mit postgresql

Bemerkungen

Dieser Abschnitt bietet einen Überblick über die Bedeutung von postgresql und warum ein Entwickler sie verwenden möchte.

Es sollte auch alle großen Themen innerhalb von postgresql erwähnen und auf die verwandten Themen verweisen. Da die Dokumentation für postgresql neu ist, müssen Sie möglicherweise erste Versionen dieser verwandten Themen erstellen.

Versionen

Ausführung	Veröffentlichungsdatum	EOL-Datum
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

Installation unter GNU + Linux

Bei den meisten GNU + Linux-Betriebssystemen kann PostgreSQL problemlos mit dem Paketmanager des Betriebssystems installiert werden.

Red Hat Familie

Ressourcen finden Sie hier: https://yum.postgresql.org/repopackages.php

Laden Sie das Repository mit dem Befehl auf den lokalen Computer herunter

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

```
redhatXX-X.X-X.noarch.rpm
```

Verfügbare Pakete anzeigen:

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

Notwendige Pakete sind: postgresqlXX postgresqlXX-server postgresqlXX-libs postgresqlXX-contrib

Diese werden mit dem folgenden Befehl installiert: yum -y install postgresqlXX postgresqlXX-server postgresqlXX-libs postgresqlXX-contrib

Nach der Installation müssen Sie den Datenbankdienst als Dienstbesitzer starten (Standard ist postgres). Dies geschieht mit dem Befehl pg_ctl.

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

Um auf die DB in CLI zuzugreifen, geben Sie `psql`

Debian-Familie

Geben Sie unter Debian und abgeleiteten Betriebssystemen Folgendes ein:

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

Dadurch wird das PostgreSQL-Serverpaket in der Standardversion installiert, die von den Paketrepositorys des Betriebssystems angeboten wird.

Wenn die standardmäßig installierte Version nicht die gewünschte ist, können Sie mit dem Paket-Manager nach bestimmten Versionen suchen, die gleichzeitig angeboten werden können.

Sie können auch das Yum-Repository verwenden, das vom PostgreSQL-Projekt (bekannt als PGDG) bereitgestellt wird , um eine andere Version zu erhalten. Dies ermöglicht möglicherweise Versionen, die noch nicht von Paketpaketen des Betriebssystems angeboten werden.

So installieren Sie PostgreSQL über MacPorts unter OSX

Um PostgreSQL unter OSX installieren zu können, müssen Sie wissen, welche Versionen derzeit unterstützt werden.

Verwenden Sie diesen Befehl, um zu sehen, welche Versionen Sie zur Verfügung haben.

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

Sie sollten eine Liste erhalten, die ungefähr wie folgt aussieht:

```
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
```

In diesem Beispiel die neueste Version von PostgreSQL, die in Version 9.6 unterstützt wird. Wir werden diese also installieren.

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

Sie sehen ein Installationsprotokoll wie folgt:

```
--->  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.
```

Das Protokoll enthält Anweisungen zu den restlichen Installationsschritten. Wir machen dies als Nächstes.

```
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'
```

Nun starten wir den Server:

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

Stellen Sie sicher, dass wir eine Verbindung zum Server herstellen können:

```
su postgres -c psql
```

Sie werden eine Aufforderung von postgres sehen:

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

postgres=#
```

Hier können Sie eine Abfrage eingeben, um festzustellen, ob der Server ausgeführt wird.

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

Und sehen Sie die Antwort:

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

Geben Sie zum Beenden \ q ein:

```
postgres=#\q
```

Und Sie werden wieder an der Shell-Eingabeaufforderung sein.

Herzliche Glückwünsche! Sie haben jetzt eine PostgreSQL-Instanz unter OS / X.

Postgres.app für Mac OSX

Ein äußerst einfaches Tool zum Installieren von PostgreSQL auf einem Mac ist durch Herunterladen von Postgres.app verfügbar.
Sie können die Einstellungen so ändern, dass PostgreSQL im Hintergrund ausgeführt wird oder nur, wenn die Anwendung ausgeführt wird.

PostgreSQL unter Windows installieren

Zwar ist die Verwendung eines Unix-basierten Betriebssystems (z. B. Linux oder BSD) als Produktionsserver empfehlenswert. Sie können PostgreSQL jedoch problemlos unter Windows installieren (hoffentlich nur als Entwicklungsserver).

Laden Sie die Windows-Installations-Binärdateien von EnterpriseDB herunter: http://www.enterprisedb.com/products-services-training/pgdownload Dies ist ein Drittanbieter-Unternehmen, das von Kernteilnehmern des PostgreSQL-Projekts gegründet wurde, die die Binärdateien für Windows optimiert haben.

Wählen Sie die neueste stabile (Nicht-Beta) Version (9.5.3 zum Zeitpunkt des Schreibens). Wahrscheinlich möchten Sie das Win x86-64-Paket. Wenn Sie jedoch eine 32-Bit-Version von Windows ausführen, die auf älteren Computern üblich ist, wählen Sie stattdessen Win x86-32 aus.

Hinweis: Das Wechseln zwischen Beta- und stabilen Versionen erfordert komplexe Aufgaben wie Dump und Restore. Ein Upgrade innerhalb der Beta- oder stabilen Version erfordert nur einen Neustart des Dienstes.

Sie können überprüfen, ob Ihre Windows-Version 32 oder 64-Bit ist, indem Sie Systemsteuerung -> System und Sicherheit -> System -> Systemtyp aufrufen. Daraufhin wird "## - bit Operating System" angezeigt. Dies ist der Pfad für Windows 7, bei anderen Windows-Versionen kann es etwas anders sein.

Wählen Sie im Installationsprogramm die Pakete aus, die Sie verwenden möchten. Zum Beispiel:

- pgAdmin (https://www.pgadmin.org) ist eine kostenlose grafische Benutzeroberfläche für die Verwaltung Ihrer Datenbank. Ich kann sie nur wärmstens empfehlen. In 9.6 wird dies standardmäßig installiert.
- PostGIS (http://postgis.net) bietet Geodaten-Analysefunktionen für GPS-Koordinaten, Entfernungen usw., die bei GIS-Entwicklern sehr beliebt sind.
- Das Sprachpaket enthält erforderliche Bibliotheken für die offiziell unterstützte Verfahrenssprache PL / Python, PL / Perl und PL / Tcl.
- Andere Pakete wie pgAgent, pgBouncer und Slony sind für größere Produktionsserver nützlich und werden nur bei Bedarf geprüft.

Alle diese optionalen Pakete können später über "Application Stack Builder" installiert werden.

Hinweis: Es gibt auch andere nicht offiziell unterstützte Sprachen wie PL / V8 , PL / Lua PL / Java.

Öffnen Sie pgAdmin und stellen Sie eine Verbindung zu Ihrem Server her, indem Sie auf den Namen doppelklicken, z. "PostgreSQL 9.5 (localhost: 5432).

Von diesem Punkt aus können Sie Anleitungen wie das ausgezeichnete Buch PostgreSQL: Up and Running, 2. Ausgabe (http://shop.oreilly.com/product/0636920032144.do) folgen.

Optional: Manueller Dienststarttyp

PostgreSQL wird als Dienst im Hintergrund ausgeführt, der sich geringfügig von den meisten Programmen unterscheidet. Dies ist bei Datenbanken und Webservern üblich. Der Standard-Starttyp ist Automatisch, was bedeutet, dass er immer ohne Eingabe von Ihnen ausgeführt wird.

Warum möchten Sie den PostgreSQL-Dienst manuell steuern? Wenn Sie Ihren PC zeitweise als Entwicklungsserver verwenden und ihn beispielsweise auch zum Spielen von Videospielen verwenden, kann PostegreSQL Ihr System während des Betriebs etwas verlangsamen.

Warum sollten Sie keine manuelle Steuerung wünschen? Das Starten und Stoppen des Dienstes kann problematisch sein, wenn Sie es häufig tun.

Wenn Sie keinen Geschwindigkeitsunterschied bemerken und den Aufwand lieber vermeiden möchten, belassen Sie den Starttyp als Automatisch und ignorieren den Rest dieser Anleitung. Andernfalls...

Gehen Sie zu Systemsteuerung -> System und Sicherheit -> Verwaltung.

Wählen Sie "Dienste" aus der Liste aus, klicken Sie mit der rechten Maustaste auf das Symbol und wählen Sie Senden an -> Desktop aus, um ein Desktop-Symbol für einen bequemeren Zugriff zu erstellen.

Schließen Sie das Fenster Verwaltung, und starten Sie Dienste über das soeben erstellte Desktopsymbol.

Scrollen Sie nach unten, bis Sie einen Dienst mit einem Namen wie postgresql-x ## - 9 sehen. # (Z. B. "postgresql-x64-9.5").

Klicken Sie mit der rechten Maustaste auf den Postgres-Dienst, und wählen Sie Eigenschaften -> Starttyp -> Manuell -> Übernehmen -> OK. Sie können es genauso leicht wieder auf automatisch umstellen.

Wenn Sie andere mit PostgreSQL in Verbindung stehende Dienste wie "pgbouncer" oder "PostgreSQL Scheduling Agent - pgAgent" in der Liste sehen, können Sie auch deren Starttyp in "Manuell" ändern, da sie nicht viel nutzen, wenn PostgreSQL nicht läuft. Dies bedeutet zwar mehr Ärger bei jedem Start und Stopp, also liegt es an Ihnen. Sie verwenden nicht so viele Ressourcen wie PostgreSQL selbst und haben möglicherweise keine spürbaren Auswirkungen auf die Systemleistung.

Wenn der Dienst ausgeführt wird, wird der Status "Gestartet" angezeigt, andernfalls wird er nicht ausgeführt.

Klicken Sie zum Starten mit der rechten Maustaste und wählen Sie Start. Eine Ladeaufforderung wird angezeigt und sollte bald darauf von selbst verschwinden. Wenn Sie einen Fehler erhalten, versuchen Sie es ein zweites Mal. Wenn dies nicht funktioniert, gab es ein Problem mit der Installation, möglicherweise weil Sie einige Einstellungen in Windows geändert haben. Die meisten Benutzer ändern sich nicht. Daher ist es möglicherweise ein wenig kompliziert, das Problem zu finden.

Klicken Sie mit der rechten Maustaste auf den Dienst, und wählen Sie Stopp.

Wenn beim Versuch, eine Verbindung zu Ihrer Datenbank herzustellen, eine Fehlermeldung angezeigt wird, überprüfen Sie die Dienste, um sicherzustellen, dass sie ausgeführt wird.

Weitere sehr spezifische Details zur EDB-Installation von PostgreSQL, z. B. die Python-Laufzeitversion im offiziellen Sprachpaket einer bestimmten PostgreSQL-Version, finden Sie immer im offiziellen EBD-Installationshandbuch . Ändern Sie die Version im Link zur Hauptversion Ihres Installationsprogramms.

Installieren Sie Postgresql with Brew auf dem Mac

Homebrew nennt sich " *den fehlenden Paketmanager für macOS* ". Es kann zum Erstellen und Installieren von Anwendungen und Bibliotheken verwendet werden. Nach der Installation können Sie den Befehl `brew` , um PostgreSQL und seine Abhängigkeiten wie folgt zu installieren:

```
brew update
brew install postgresql
```

Homebrew installiert im Allgemeinen die neueste stabile Version. Wenn Sie ein anderes `brew search postgresql` , listet `brew search postgresql` die verfügbaren Versionen auf. Wenn PostgreSQL mit bestimmten Optionen `brew info postgresql` werden soll, listet `brew info postgresql` auf, welche Optionen unterstützt werden. Wenn Sie eine nicht unterstützte Build-Option benötigen, müssen Sie den Build möglicherweise selbst durchführen, können jedoch weiterhin Homebrew verwenden, um die allgemeinen Abhängigkeiten zu installieren.

Starten Sie den Server:

```
brew services start postgresql
```

Öffnen Sie die PostgreSQL-Eingabeaufforderung

```
psql
```

Wenn psql sich darüber beschwert, dass für Ihren Benutzer keine entsprechende Datenbank vorhanden ist, führen Sie `createdb` .

Installieren Sie PostgreSQL von Source unter Linux

Abhängigkeiten:

- GNU Make Version> 3.80
- ein ISO / ANSI C-Compiler (zB gcc)
- ein Extraktor wie Teer oder Gzip
- zlib-devel
- readline-devel oder libedit-devel

Quellen: Link zur neuesten Quelle (9.6.3)

Jetzt können Sie die Quelldateien extrahieren:

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

Für die Konfiguration von PostgreSQL gibt es eine Vielzahl unterschiedlicher Optionen:

Vollständiger Link zur vollständigen Installationsprozedur

Kleine Liste der verfügbaren Optionen:

- `--prefix=PATH` für alle Dateien
- `--exec-prefix=PATH` Pfad für die von der Architektur abhängige Datei
- `--bindir=PATH` Pfad für ausführbare Programme
- `--sysconfdir=PATH` Pfad für Konfigurationsdateien
- `--with-pgport=NUMBER` einen Port für Ihren Server an
- `--with-perl` addiert Perl-Unterstützung
- `--with-python` fügt Python-Unterstützung hinzu
- `--with-openssl` fügt die Unterstützung von `--with-openssl` hinzu
- `--with-ldap` fügt LDAP-Unterstützung hinzu
- `--with-blocksize=BLOCKSIZE` Seitengröße in KB setzen
    - `BLOCKSIZE` muss eine Potenz von 2 und zwischen 1 und 32 haben
- `--with-wal-segsize=SEGSIZE` Größe des WAL-Segments in MB `--with-wal-segsize=SEGSIZE`
    - `SEGSIZE` muss eine Potenz von 2 zwischen 1 und 64 sein

Gehen Sie in den neu erstellten Ordner und führen Sie das Skript cofigure mit den gewünschten Optionen aus:

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

Führen Sie `make` , um die Objektdateien zu erstellen

Führen Sie `make install` , um PostgreSQL aus den erstellten Dateien zu installieren

Führen Sie `make clean` , `make clean`

Für die Erweiterung wechseln Sie das Verzeichnis `cd contrib` , führen Sie `make` und `make install`

Erste Schritte mit postgresql online lesen: https://riptutorial.com/de/postgresql/topic/885/erste-schritte-mit-postgresql

Kapitel 2: Aggregatfunktionen

Examples

Einfache Statistik: min (), max (), avg ()

Um einige einfache Statistiken eines Werts in einer Tabellenspalte zu ermitteln, können Sie eine Aggregatfunktion verwenden.

Wenn Ihre `individuals` ist:

Name	Alter
Allie	17
Amanda	14
Alana	20

Sie können diese Anweisung schreiben, um den Minimal-, Maximal- und Durchschnittswert zu erhalten:

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

Ergebnis:

Mindest	max	Durchschn
14	20	17

string_agg (Ausdruck, Trennzeichen)

Sie können Zeichenfolgen, die durch Trennzeichen getrennt sind, mithilfe der Funktion `string_agg()` verketten.

Wenn Ihre `individuals` ist:

Name	Alter	Land
Allie	fünfzehn	Vereinigte Staaten von Amerika
Amanda	14	Vereinigte Staaten von Amerika
Alana	20	Russland

Sie könnten eine `SELECT ... GROUP BY` Anweisung schreiben, um Namen aus jedem Land zu erhalten:

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

Beachten Sie, dass Sie eine `GROUP BY` Klausel verwenden müssen, da `string_agg()` eine Aggregatfunktion ist.

Ergebnis:

Namen	Land
Allie, Amanda	Vereinigte Staaten von Amerika
Alana	Russland

Weitere hier beschriebene PostgreSQL-Aggregatfunktion

regr_slope (Y, X): Steigung der linearen Gleichung für die Anpassung der kleinsten Quadrate, bestimmt durch die (X, Y) -Paare

Um die Verwendung von regr_slope (Y, X) zu veranschaulichen, habe ich es auf ein Problem der realen Welt angewendet. Wenn Sie in Java den Speicher nicht ordnungsgemäß bereinigen, kann der Müll hängen bleiben und den Speicher füllen. Sie legen jede Stunde Statistiken über die Speicherauslastung verschiedener Klassen ab und laden sie zur Analyse in eine Postgres-Datenbank.

Alle Speicherverlustkandidaten tendieren dazu, mehr Speicher zu verbrauchen, wenn mehr Zeit vergeht. Wenn Sie diesen Trend darstellen, können Sie sich eine Linie nach links und links vorstellen:

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

Angenommen, Sie haben eine Tabelle mit Heap-Dump-Histogrammdaten (eine Zuordnung von Klassen zu ihrem Speicherbedarf):

```
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
);
```

Um die Steigung für jede Klasse zu berechnen, gruppieren wir die Klasse über. Die HAVING-Klausel> 0 stellt sicher, dass nur Kandidaten mit einem positiven Slop (eine Linie nach oben und nach links) angezeigt werden. Wir sortieren nach der Steigung absteigend, sodass die Klassen mit der höchsten Speicherzuwachsrate an der Spitze angezeigt werden.

```
-- 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 ;
```

Ausgabe:

```
         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
```

Die Ausgabe zeigt, dass der Speicherverbrauch von java.util.ArrayList mit 71.799 Bytes pro Sekunde am schnellsten ansteigt und möglicherweise Teil des Speicherverlusts ist.

Aggregatfunktionen online lesen:
https://riptutorial.com/de/postgresql/topic/4803/aggregatfunktionen

Kapitel 3: AKTUALISIEREN

Examples

Aktualisieren Sie alle Zeilen in einer Tabelle

Sie aktualisieren alle Zeilen in der Tabelle, indem Sie einfach einen `column_name = value` :

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

Aktualisieren Sie alle Zeilen, die eine Bedingung erfüllen

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

Mehrere Spalten in der Tabelle aktualisieren

Sie können mehrere Spalten in einer Tabelle in derselben Anweisung aktualisieren, indem Sie `col=val` Paare durch Kommas trennen:

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

Aktualisieren einer Tabelle basierend auf dem Beitritt zu einer anderen Tabelle

Sie können Daten in einer Tabelle auch basierend auf Daten aus einer anderen Tabelle aktualisieren:

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

Hier sind wir den Beitritt zur `person city` Spalte in die `cities city` Säule , um die Stadt Zustand Code zu erhalten. Diese wird dann verwendet, um die `state_code` Spalte in der `person` zu aktualisieren.

AKTUALISIEREN online lesen: https://riptutorial.com/de/postgresql/topic/3136/aktualisieren

Kapitel 4: Allgemeine Tabellenausdrücke (WITH)

Examples

Häufige Tabellenausdrücke in SELECT-Abfragen

Allgemeine Tabellenausdrücke unterstützen das Extrahieren von Teilen größerer Abfragen. Zum Beispiel:

```
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)
```

Baum mit WITH RECURSIVE durchlaufen

```
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;
```

Allgemeine Tabellenausdrücke (WITH) online lesen:
https://riptutorial.com/de/postgresql/topic/1973/allgemeine-tabellenausdrucke--with-

Kapitel 5: Datentypen

Einführung

PostgreSQL verfügt über eine Vielzahl an systemeigenen Datentypen, die den Benutzern zur Verfügung stehen. Benutzer können mit dem Befehl CREATE TYPE neue Typen zu PostgreSQL hinzufügen.

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

Examples

Numerische Typen

Name	Speichergröße	Beschreibung	Angebot
`smallint`	2 Bytes	Ganzzahl mit kleinem Bereich	-32768 bis +32767
`integer`	4 Bytes	ypical Wahl für Ganzzahl	-2147483648 bis +2147483647
`bigint`	8 Bytes	Large Range Integer	-9223372036854775808 bis +9223372036854775807
`decimal`	Variable	benutzerspezifizierte Genauigkeit, genau	bis zu 131072 Stellen vor dem Dezimalpunkt; bis zu 16383 Nachkommastellen
`numeric`	Variable	benutzerspezifizierte Genauigkeit, genau	bis zu 131072 Stellen vor dem Dezimalpunkt; bis zu 16383 Nachkommastellen
`real`	4 Bytes	variable Genauigkeit, ungenau	6 Dezimalstellen Genauigkeit
`double precision`	8 Bytes	variable Genauigkeit, ungenau	15 Dezimalstellen Genauigkeit
`smallserial`	2 Bytes	kleine Ganzzahl mit automatischer Inkrementierung	1 bis 32767
`serial`	4 Bytes	Autoincrementing-Ganzzahl	1 bis 2147483647
`bigserial`	8 Bytes	große autoincrementing-	1 bis 9223372036854775807

Name	Speichergröße	Beschreibung	Angebot
		Ganzzahl	
`int4range`		Bereich der ganzen Zahl	
`int8range`		Bereich von bigint	
`numrange`		Bereich der numerischen	

Datums- / Zeittypen

Name	Speichergröße	Beschreibung	Niedriger Wert	Hochwertig	Auflösung
`timestamp` (ohne Zeitzone)	8 Bytes	Datum und Uhrzeit (keine Zeitzone)	4713 v	294276 n. Chr	1 Mikrosekunde / 14 Ziffern
`timestamp` (mit Zeitzone)	8 Bytes	sowohl Datum als auch Uhrzeit mit Zeitzone	4713 v	294276 n. Chr	1 Mikrosekunde / 14 Ziffern
`date`	4 Bytes	Datum (keine Tageszeit)	4713 v	5874897 n. Chr	1 Tag
`time` (ohne Zeitzone)	8 Bytes	Tageszeit (kein Datum)	00:00:00	24:00:00	1 Mikrosekunde / 14 Ziffern
`time` (mit Zeitzone)	12 Bytes	Nur Tageszeiten mit Zeitzone	00: 00: 00 + 1459	24: 00: 00-1459	1 Mikrosekunde / 14 Ziffern
`interval`	16 Bytes	Zeitintervall	-178000000 Jahre	178000000 Jahre	1 Mikrosekunde / 14 Ziffern
`tsrange`		Zeitstempelbereich ohne Zeitzone			
`tstzrange`		Zeitstempelbereich mit Zeitzone			
`daterange`		Datumsbereich			

Geometrische Typen

Name	Speichergröße	Beschreibung	Darstellung
`point`	16 Bytes	Zeigen Sie auf eine Ebene	(x, y)
`line`	32 Bytes	Unendliche Linie	{ABC}
`lseg`	32 Bytes	Endliches Liniensegment	((x1, y1), (x2, y2))
`box`	32 Bytes	Rechteckige Box	((x1, y1), (x2, y2))
`path`	16 + 16n Bytes	Geschlossener Pfad (ähnlich wie Polygon)	((x1, y1), ...)
`path`	16 + 16n Bytes	Pfad öffnen	[(x1, y1), ...]
`polygon`	40 + 16n Bytes	Polygon (ähnlich wie geschlossener Pfad)	((x1, y1), ...)
`circle`	24 Bytes	Kreis	<(x, y), r> (Mittelpunkt und Radius)

Typen der Netzwerkadressen

Name	Speichergröße	Beschreibung
`cidr`	7 oder 19 Bytes	IPv4- und IPv6-Netzwerke
`inet`	7 oder 19 Bytes	IPv4- und IPv6-Hosts und -Netzwerke
`macaddr`	6 Bytes	MAC-Adressen

Zeichentypen

Name	Beschreibung
`character varying(n)` , `varchar(n)`	variable Länge mit Limit
`character(n)` , `character(n) char(n)`	Feste Länge, leer aufgefüllt
`text`	variable unbegrenzte Länge

Arrays

In PostgreSQL können Sie Arrays aller integrierten, benutzerdefinierten oder Aufzählungstypen erstellen. Standardmäßig gibt es keine Begrenzung für ein Array, Sie *können* es *jedoch* angeben.

Ein Array deklarieren

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

Array erstellen

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

Zugriff auf ein Array

Standardmäßig verwendet PostgreSQL eine Ein-Nummerierungs-Konvention für Arrays, dh ein Array von n Elementen beginnt mit `array[1]` und endet mit `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)
```

Informationen zu einem Array abrufen

```
--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)
```

Array-Funktionen

wird hinzugefügt werden

Datentypen online lesen: https://riptutorial.com/de/postgresql/topic/8976/datentypen

Kapitel 6: Datumsangaben, Zeitstempel und Intervalle

Examples

Wandeln Sie einen Zeitstempel oder ein Intervall in eine Zeichenfolge

Mit der `to_char()` Funktion können Sie einen `timestamp` oder einen `interval` in eine Zeichenfolge `to_char()` :

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

Diese Anweisung wird die Zeichenfolge "12 Aug 2016 16:40:32 PM" erzeugen. Die Formatierungszeichenfolge kann auf viele verschiedene Arten geändert werden. Die vollständige Liste der Vorlagen finden Sie hier .

Beachten Sie, dass Sie auch einfachen Text in die Formatierungszeichenfolge einfügen können und die Vorlagenmuster in beliebiger Reihenfolge verwenden können:

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

Dadurch wird die Zeichenfolge "Heute ist Samstag, der 12. Tag des Monats August 2016" erzeugt. Beachten Sie jedoch, dass alle Schablonenmuster - auch die aus einem Buchstaben bestehenden Buchstaben wie "I", "D", "W" - konvertiert werden, es sei denn, der Klartext ist in Anführungszeichen gesetzt. Als Sicherheitsmaßnahme sollten Sie den gesamten Klartext wie oben angegeben in Anführungszeichen setzen.

Sie können die Zeichenfolge in der Sprache Ihrer Wahl (Tag- und Monatsnamen) lokalisieren, indem Sie den Modifikator TM (Übersetzungsmodus) verwenden. Diese Option verwendet die Lokalisierungseinstellung des Servers, auf dem PostgreSQL ausgeführt wird, oder des Clients, der eine Verbindung dazu herstellt.

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

Bei einer spanischen Einstellung ergibt sich "Sábado, 12 de Agosto del año 2016".

WÄHLEN Sie den letzten Tag des Monats aus

Sie können den letzten Tag des Monats auswählen.

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

`201608` ist durch eine Variable ersetzbar.

Zählen Sie die Anzahl der Datensätze pro Woche

SELECT date_trunc ('week', <>) AS "week", Anzahl (*) von <> GROUP BY 1 ORDER BY 1

Datumsangaben, Zeitstempel und Intervalle online lesen:
https://riptutorial.com/de/postgresql/topic/4227/datumsangaben--zeitstempel-und-intervalle

Kapitel 7: EINFÜGEN

Examples

Basic INSERT

Nehmen wir an, wir haben eine einfache Tabelle namens person:

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

Die einfachste Einfügung besteht darin, alle Werte in die Tabelle einzufügen:

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

Wenn Sie nur bestimmte Spalten einfügen möchten, müssen Sie explizit angeben, welche Spalten:

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

Wenn in der Tabelle Einschränkungen vorhanden sind, z. B. NOT NULL, müssen Sie diese Spalten in beiden Fällen angeben.

Mehrere Zeilen einfügen

Sie können mehrere Zeilen gleichzeitig in die Datenbank einfügen:

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

Einfügen aus auswählen

Sie können Daten als Ergebnis einer select-Anweisung in eine Tabelle einfügen:

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

Beachten Sie, dass die Projektion der Auswahl mit den für das Einfügen erforderlichen Spalten übereinstimmen muss. In diesem Fall hat die Tabelle `tmp_person` die gleichen Spalten wie `person` .

Daten mit COPY einfügen

COPY ist der Bulk-Insert-Mechanismus von PostgreSQL. Dies ist eine bequeme Methode zum Übertragen von Daten zwischen Dateien und Tabellen, aber auch weitaus schneller als `INSERT` wenn mehr als ein paar tausend Zeilen gleichzeitig hinzugefügt werden.

Beginnen wir mit dem Erstellen einer Beispieldatendatei.

```
cat > samplet_data.csv

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

Und wir brauchen eine zweispaltige Tabelle, in die diese Daten importiert werden können.

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

Beim eigentlichen Kopiervorgang werden nun sechs Datensätze in der Tabelle erstellt.

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

Anstatt eine Datei auf der Festplatte zu verwenden, können Daten aus `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
```

Sie können Daten auch wie folgt aus einer Tabelle in eine Datei kopieren:

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

Weitere Informationen zu COPY finden Sie hier

INSERT-Daten und RETURING-Werte

Wenn Sie Daten in eine Tabelle mit einer Auto-Increment-Spalte einfügen und den Wert der Auto-Increment-Spalte abrufen möchten.

`my_table` Sie haben eine Tabelle namens `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)
);
```

Wenn Sie Daten in `my_table` einfügen `my_table` und die ID dieser Zeile erhalten:

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

Die obige Abfrage gibt die ID der Zeile zurück, in die der neue Datensatz eingefügt wurde.

Daten in Datei auswählen.

Sie können die Tabelle KOPIEREN und in eine Datei einfügen.

```
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 AKTUALISIEREN ...

Seit Version 9.5 bietet `UPSERT` die `UPSERT` Funktionalität mit `INSERT` Anweisung.

Angenommen, Sie haben eine Tabelle namens my_table, die in mehreren vorherigen Beispielen erstellt wurde. Wir fügen eine Zeile ein und geben den PK-Wert der eingefügten Zeile zurück:

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

```
----
  2
(1 row)

INSERT 0 1
```

Wenn wir jetzt versuchen, eine Zeile mit einem vorhandenen eindeutigen Schlüssel einzufügen, wird eine Ausnahme ausgelöst:

```
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.
```

Die Upsert-Funktion bietet die Möglichkeit, sie trotzdem einzufügen und den Konflikt zu lösen:

```
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
```

EINFÜGEN online lesen: https://riptutorial.com/de/postgresql/topic/2561/einfugen

Kapitel 8: Erbe

Bemerkungen

Eine Erklärung, warum Sie die Vererbung in PostgreSQL verwenden möchten, finden Sie hier: http://stackoverflow.com/a/3075248/653378

Examples

Kindertabellen erstellen

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

Unsere drei Tische sehen so aus:

Benutzer

Säule	Art
Nutzername	Text
Email	Text

simple_users

Säule	Art
Nutzername	Text
Email	Text

Benutzer_mit_Kennwort

Säule	Art
Nutzername	Text
Email	Text
Passwort	Text

Tabellen ändern

Lassen Sie uns zwei einfache Tabellen erstellen:

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

Spalten hinzufügen

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

simple_users

Säule	Art
Nutzername	Text
Email	Text
Passwort	Text

Durch Hinzufügen derselben Spalte zur übergeordneten Tabelle wird die Definition beider Spalten zusammengeführt:

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

> HINWEIS: Zusammenführen der Definition der Spalte "Kennwort" für untergeordnete "simple_users"

Spalten löschen

Verwenden Sie unsere geänderten Tabellen:

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

Benutzer

Säule	Art
Nutzername	Text
Email	Text

simple_users

Säule	Art
Nutzername	Text
Email	Text
Passwort	Text

Da wir die Spalte zunächst zu `simple_users` hinzugefügt `simple_users` , stellt PostgreSQL sicher, dass diese Spalte nicht gelöscht wird.

Wenn wir jetzt eine andere untergeordnete Tabelle hätten, wäre die `password` natürlich gelöscht worden.

Erbe online lesen: https://riptutorial.com/de/postgresql/topic/5429/erbe

Kapitel 9: Ereignisauslöser

Einführung

Ereignisauslöser werden immer dann ausgelöst, wenn das ihnen zugeordnete Ereignis in der Datenbank auftritt.

Bemerkungen

Bitte verwenden Sie den untenstehenden Link für eine vollständige Übersicht der Ereignisauslöser in PostgreSQL

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

Examples

DDL-Befehlsstartereignisse protokollieren

Ereignistyp-

- `DDL_COMMAND_START`
- `DDL_COMMAND_END`
- SQL_DROP

In diesem Beispiel wird ein Ereignisauslöser erstellt und `DDL_COMMAND_START` Ereignisse protokolliert.

```
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();
```

Ereignisauslöser online lesen: https://riptutorial.com/de/postgresql/topic/9255/ereignisausloser

Kapitel 10: Exportieren Sie den Header und die Daten der PostgreSQL-Datenbanktabelle in die CSV-Datei

Einführung

Vom Adminer-Verwaltungstool aus gibt es den Export in die csv-Dateioption für die mysql-Datenbank, jedoch nicht für die postgresql-Datenbank. Hier werde ich den Befehl zum Exportieren von CSV für die Postgresql-Datenbank anzeigen.

Examples

Export der PostgreSQL-Tabelle in den csv-Header mit einigen Spalten

```
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;
```

Vollständige Tabellensicherung in csv mit Header

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

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

aus Abfrage kopieren

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

Exportieren Sie den Header und die Daten der PostgreSQL-Datenbanktabelle in die CSV-Datei online lesen: https://riptutorial.com/de/postgresql/topic/8643/exportieren-sie-den-header-und-die-daten-der-postgresql-datenbanktabelle-in-die-csv-datei

Kapitel 11: EXTENSION dblink und postgres_fdw

Syntax

- dblink ('dbname = name_db_distance port = PortOfDB-Host = HostOfDB-Benutzer = BenutzernameDB-Kennwort = KennwortDB', 'MEINE FRAGE')
- Datenbankname = Name der Datenbank
- port = Port der Datenbank
- host = Host der Datenbank
- user = Benutzername der Datenbank
- Passwort = Passwort der Datenbank ',
- MY QUESRY = Dies kann eine beliebige Operation sein, die ich SELECT, INSERT, ... ausführen möchte.

Examples

Erweiterung dblink

dblink EXTENSION ist eine Technik, um eine andere Datenbank zu verbinden und den Betrieb dieser Datenbank durchzuführen, damit Sie Folgendes benötigen:

1-Dblink-Erweiterung erstellen:

```
CREATE EXTENSION dblink;
```

2-Machen Sie Ihre Operation:

Wählen Sie beispielsweise ein Attribut aus einer anderen Tabelle in einer anderen Datenbank aus:

```
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);
```

Erweiterung FDW

FDW ist eine Implementierung von dblink. Es ist hilfreicher, also verwenden Sie es:

1 -Erstellen Sie eine Erweiterung:

```
CREATE EXTENSION postgres_fdw;
```

2-Create SERVER:

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

3 Erstellen Sie eine Benutzerzuordnung für den Postgres-Server

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

4 - Erstellen Sie eine fremde Tabelle:

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

Verwenden Sie diese Fremdtabelle wie in Ihrer Datenbank:

```
SELECT * FROM table_foreign;
```

Fremddaten-Wrapper

Zugriff auf das vollständige Schema der Server-Datenbank anstelle einer einzelnen Tabelle. Befolgen Sie die folgenden Schritte:

1. ERWEITERUNG erstellen:

```
  CREATE EXTENSION postgres_fdw;
```

2. SERVER erstellen:

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

3. Erstellen Sie USER MAPPING:

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

4. Erstellen Sie ein neues Schema, um auf das Schema der Server-DB zuzugreifen:

```
 CREATE SCHEMA schema_name;
```

5. Serverschema importieren:

```
   IMPORT FOREIGN SCHEMA schema_name_to_import_from_remote_db
```

```
    FROM SERVER server_name
    INTO schema_name;
```

6. Greifen Sie auf eine Tabelle mit Serverschemas zu:

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

Dies kann für den Zugriff auf mehrere Remote-DB-Schemas verwendet werden.

EXTENSION dblink und postgres_fdw online lesen:
https://riptutorial.com/de/postgresql/topic/6970/extension-dblink-und-postgres-fdw

Kapitel 12: Fensterfunktionen

Examples

allgemeines Beispiel

Daten vorbereiten:

```
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;
```

Laufen:

```
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
;
```

Ergebnis:

```
 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)
```

Erläuterung:

dist_by_i : `dense_rank() over (order by i)` ist eine Reihennummer für verschiedene Werte. Kann für die Anzahl der unterschiedlichen Werte von i verwendet werden (`count(DISTINCT i)` funktioniert nicht). Verwenden Sie einfach den Maximalwert.

prev_t : `lag(t) over ()` ist ein vorheriger Wert von t über das gesamte Fenster. Beachten Sie, dass

es für die erste Zeile null ist.

nth: `nth_value(i, 6) over ()` ist der Wert der Spalte sechster Zeilen i über die ganzen Fenster

num_by_i : `count(true) over (partition by i)` ist eine Anzahl von Zeilen für jeden Wert von i

num_all : `count(true) over ()` ist eine Anzahl von Zeilen über ein ganzes Fenster

ntile : `ntile(3) over()` teilt das gesamte Fenster in 3 (so viel wie möglich) gleich viele Teile auf

Spaltenwerte vs dense_rank vs rank vs row_number

Hier finden Sie die Funktionen.

Führen Sie mit der im vorherigen Beispiel erstellten Tabelle wf_example Folgendes aus:

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

Das Ergebnis ist:

```
 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* befiehlt **WERTE** von **i** durch Erscheinung im Fenster. `i=1` erscheint, also hat die erste Zeile dense_rank, der nächste und der dritte i-Wert ändert sich nicht, so dass es `dense_rank` zeigt, dass *1* - der `dense_rank` Wert nicht geändert wird. vierte Reihe `i=2` , es ist der zweite Wert von *i* erfüllt, so dass `dense_rank` 2 zeigt, und so für die nächste Reihe. Dann trifft es in der 6. Zeile auf den Wert `i=3` , also zeigt es 3. Gleiche Werte für die restlichen zwei Werte von *i* . Der letzte Wert von `dense_rank` ist also die Anzahl der unterschiedlichen Werte von *i* .

- *row_number* ordnet **ROWS so an,** wie sie aufgelistet sind.

- *rank* Nicht zu verwechseln mit `dense_rank` Diese Funktion ordnet **ROW NUMBER** von **i-**Werten an. Es beginnt also gleich mit drei Einsen, hat aber den nächsten Wert 4, was bedeutet, dass `i=2` (neuer Wert) in Zeile 4 erfüllt wurde. Das gleiche `i=3` wurde in Zeile 6 getroffen.

Fensterfunktionen online lesen: https://riptutorial.com/de/postgresql/topic/7421/fensterfunktionen

Kapitel 13: JSON-Unterstützung

Einführung

JSON - Java Script Object Notation, Postgresql unterstützt den JSON-Datentyp seit Version 9.2. Es gibt einige vordefinierte Funktionen und Operatoren, um auf die JSON-Daten zuzugreifen. Der Operator `->` gibt den Schlüssel der JSON-Spalte zurück. Der Operator `->>` gibt den Wert der JSON-Spalte zurück.

Examples

Eine reine JSON-Tabelle erstellen

Um eine reine JSON-Tabelle zu erstellen, müssen Sie ein einzelnes Feld mit dem Typ `JSONB` :

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

Sie sollten auch einen Basisindex erstellen:

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

An dieser Stelle können Sie Daten in die Tabelle einfügen und diese effizient abfragen.

Abfragen komplexer JSON-Dokumente

Ein komplexes JSON-Dokument in eine Tabelle aufnehmen:

```
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"
        }
    }
}
$$);
```

Abfrage für ein Element der obersten Ebene:

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

Abfrage nach einem einfachen Element in einem Array:

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

Abfrage für ein Objekt in einem Array:

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

Abfrage für ein verschachteltes Objekt:

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

Leistung von `@>` Vergleich zu `->` und `->>`

Es ist wichtig, die Leistungsunterschiede zwischen `@>` , `->` und `->>` im `WHERE` Teil der Abfrage zu verstehen. Obwohl diese beiden Abfragen weitgehend gleichwertig erscheinen:

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

Die erste Anweisung verwendet den oben erstellten Index, während die letzten beiden nicht erforderlich sind und einen vollständigen Tabellenscan erfordern.

Es ist weiterhin zulässig, den Operator `->` , wenn die resultierenden Daten abgerufen werden. Daher verwenden die folgenden Abfragen auch den Index:

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

JSONb-Operatoren verwenden

Anlegen einer Datenbank und einer Tabelle

```
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
);
```

DB füllen

```
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" } }'
);
```

Lasst uns alles in den Tabellenbüchern sehen:

```
SELECT * FROM books;
```

Ausgabe:

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

-> Operator gibt Werte aus JSON-Spalten zurück

1 Spalte auswählen:

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

Ausgabe:

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

Auswahl von 2 Spalten:

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

Ausgabe:

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 ->>

Der Operator `->` gibt den ursprünglichen JSON-Typ (möglicherweise ein Objekt) zurück, während `->>` Text zurückgibt.

NESTED-Objekte zurückgeben

Sie können mit `->` ein verschachteltes Objekt zurückgeben und somit die Operatoren verketten:

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

Ausgabe:

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

Filterung

Wählen Sie Zeilen basierend auf einem Wert in Ihrem JSON aus:

```
 SELECT
 client,
```

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

Beachten Sie, WO verwendet `->` also müssen wir mit `'"Dharma Bums"'` von JSON vergleichen

Oder wir könnten `->>` und mit `'Dharma Bums'`

Ausgabe:

client character varying	title jsonb
Jenny	"Dharma Bums"

Verschachtelte Filterung

Suchen Sie Zeilen basierend auf dem Wert eines geschachtelten JSON-Objekts:

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

Ausgabe:

client character varying	title jsonb
Jenny	"Dharma Bums"

Ein reales Beispiel

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

In dieser Tabelle werden Ereignisse wie Seitenaufrufe gespeichert. Jedes Ereignis hat Eigenschaften, die alles sein können (zB aktuelle Seite) und auch Informationen über den Browser (wie Betriebssystem, Bildschirmauflösung usw.) senden. Beide sind vollkommen frei und können sich im Laufe der Zeit ändern (wenn wir an zusätzliche Dinge denken, um sie zu verfolgen).

```
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 } }'
);
```

Jetzt können wir alles auswählen:

```
SELECT * FROM events;
```

Ausgabe:

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

JSON-Operatoren + Aggregatefunktionen von PostgreSQL

Mit den JSON-Operatoren können wir in Kombination mit den herkömmlichen PostgreSQL-Aggregatfunktionen das ausführen, was wir wollen. Sie haben alle Möglichkeiten eines RDBMS zu Ihrer Verfügung.

- Sehen Sie sich die Browserverwendung an:

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

Ausgabe:

browser text	count bigint
Firefox	4
Chrome	2

- Gesamtumsatz pro Besucher:

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

Ausgabe:

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

- Durchschnittliche Bildschirmauflösung

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

Ausgabe:

width numeric	height numeric
1397.3333333333333333	894.666666666666667

Weitere Beispiele und Dokumentation hier und hier .

JSON-Unterstützung online lesen: https://riptutorial.com/de/postgresql/topic/1034/json-unterstutzung

Kapitel 14: Kommentare in postgresql

Einführung

Der Hauptzweck von ***COMMMENT*** besteht darin, einen Kommentar zu einem Datenbankobjekt zu definieren oder zu ändern.

Für jedes Datenbankobjekt kann nur ein einzelner Kommentar (Zeichenfolge) angegeben werden. COMMENT hilft uns dabei zu wissen, was für ein bestimmtes Datenbankobjekt definiert wurde, was dessen eigentlicher Zweck ist.

Die Regel für ***COMMENT ON ROLE*** lautet, dass Sie Superuser sein müssen, um eine Superuser-Rolle zu kommentieren, oder über das ***CREATEROLE-*** Privileg verfügen, um Nicht-Superuser-Rollen zu kommentieren. Ein *Superuser kann* natürlich *alles kommentieren*

Syntax

- COMMENT ON Datenbankobjekt Objektname IS 'Text';

Bemerkungen

Vollständige Syntax siehe: http://www.postgresql.org/docs/current/static/sql-comment.html

Examples

KOMMENTAR auf dem Tisch

COMMENT ON TABLE Tabellenname IS 'Dies ist eine Studentendetails-Tabelle';

Kommentar entfernen

Kommentar zu TABLE Student ist NULL;

Der Kommentar wird mit der Ausführung der obigen Anweisung entfernt.

Kommentare in postgresql online lesen:
https://riptutorial.com/de/postgresql/topic/8191/kommentare-in-postgresql

Kapitel 15: Postgres Tipp und Tricks

Examples

DATEADD-Alternative in Postgres

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

Kommagetrennte Werte einer Spalte

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

Löschen Sie doppelte Datensätze aus der Postgres-Tabelle

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

Aktualisierungsabfrage mit Verknüpfung zwischen zwei Tabellen alternativ, da Postresql die Aktualisierungsabfrage nicht unterstützt

```
    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>
```

Unterschied zwischen zwei Datumszeitstempeln auf Monats- und Jahresebene

Monatliche Differenz zwischen zwei Datumsangaben (Zeitstempel)

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

Jahresabweichung zwischen zwei Datumsangaben (Zeitstempel)

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

Abfrage zum Kopieren / Verschieben / Übertragen von Tabellendaten aus einer Datenbank in eine andere Datenbanktabelle mit demselben Schema

Erste Ausführung

```
CREATE EXTENSION DBLINK;
```

Dann

```
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>
    );
```

Postgres Tipp und Tricks online lesen: https://riptutorial.com/de/postgresql/topic/7433/postgres-tipp-und-tricks

Kapitel 16: PostgreSQL Hochverfügbarkeit

Examples

Replikation in PostgreSQL

- **Konfigurieren des Primärservers**
  - **Bedarf:**
    - Replikationsbenutzer für Replikationsaktivitäten
    - Verzeichnis zum Speichern der WAL-Archive
  - **Erstellen Sie einen Replikationsbenutzer**

    ```
    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
    ```

  - **Erstellen Sie ein Archivverzeichnis im Datenverzeichnis**

    ```
    mkdir $PGDATA/archive
    ```

  - **Bearbeiten Sie die Datei pg_hba.conf**

    Dies ist die Host-Basisauthentifizierungsdatei, die die Einstellung für die Clientauthentifizierung enthält. Fügen Sie den folgenden Eintrag hinzu:

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

  - **Bearbeiten Sie die postgresql.conf-Datei**

    Dies ist die Konfigurationsdatei von PostgreSQL.

    ```
    wal_level = hot_standby
    ```

    Dieser Parameter bestimmt das Verhalten des Slave-Servers.

    ```
    `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
    ```

    Dieser Parameter ermöglicht das Senden von WAL-Segmenten an den

Archivspeicherort mit `archive_command` Parameter " `archive_command` .

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

Im Wesentlichen `archive_command` Sie die WAL-Segmente in das Archivverzeichnis.

`wal_senders = 5` Dies ist die maximale Anzahl von WAL- `wal_senders = 5` .

Starten Sie nun den Primärserver neu.

- **Sichern des primay-Servers auf dem Slave-Server**

  Bevor Sie Änderungen am Server vornehmen, stoppen Sie den primären Server.

  Wichtig: Starten Sie den Dienst nicht erneut, bis alle Konfigurations- und Sicherungsschritte abgeschlossen sind. Sie müssen den Standby-Server in einen Zustand versetzen, in dem er als Sicherungsserver bereit ist. Das bedeutet, dass alle Konfigurationseinstellungen vorhanden sein müssen und die Datenbanken bereits synchronisiert sein müssen. Andernfalls kann die Streaming-Replikation nicht gestartet werden

- **Führen Sie nun das Dienstprogramm pg_basebackup aus**

  `pg_basebackup` Dienstprogramm `pg_basebackup` kopiert die Daten aus dem Datenverzeichnis des primären Servers in das Slave-Datenverzeichnis.

  ```
  $ 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.
```

- **Standby-Server konfigurieren**

  Um den Standby-Server zu konfigurieren, bearbeiten Sie postgresql.conf und erstellen eine neue Konfigurationsdatei mit dem Namen recovery.conf.

  ```
  hot_standby = on
  ```

  Dies gibt an, ob Sie während der Wiederherstellung Abfragen ausführen dürfen

  - **Wiederherstellungsdatei.conf erstellen**

    ```
    standby_mode = on
    ```

    Legen Sie die Verbindungszeichenfolge auf dem primären Server fest. Ersetzen Sie durch die externe IP-Adresse des Primärservers. Ersetzen Sie das Kennwort für den Benutzer mit dem Namen "Replikation"

`primary_conninfo = 'host = port = 5432 user = Kennwort für die Replikation ='

(Optional) Legen Sie den Speicherort der Triggerdatei fest:

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

Der von Ihnen angegebene `trigger_file` ist der Ort, an dem Sie eine Datei hinzufügen können, wenn das System auf den Standby-Server `trigger_file` . Das Vorhandensein der Datei "löst" das Failover aus. Alternativ können Sie den Befehl pg_ctl promotet verwenden, um ein Failover auszulösen.

- **Starten Sie den Standby-Server**

  Sie haben jetzt alles eingerichtet und können den Standby-Server hochfahren

Zuschreibung

Dieser Artikel ist im Wesentlichen abgeleitet von PostgreSQL für Hochverfügbarkeit und Replikation mit Hot Standby , mit geringfügigen Änderungen in der Formatierung und Beispielen sowie einigen Texten. Die Quelle wurde unter der Creative Commons Public License 3.0 veröffentlicht , die hier gepflegt wird.

PostgreSQL Hochverfügbarkeit online lesen:
https://riptutorial.com/de/postgresql/topic/5478/postgresql-hochverfugbarkeit

Kapitel 17: Postgres-Verschlüsselungsfunktionen

Einführung

In Postgres können kryptographische Funktionen mithilfe des pgcrypto-Moduls entsperrt werden.
CREATE EXTENSION pgcrypto;

Examples

verdauen

`DIGEST()` Funktionen erzeugen einen binären Hash der angegebenen Daten. Dies **kann** verwendet werden, um einen zufälligen Hash zu erstellen.

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

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

Beispiele:

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

Postgres-Verschlüsselungsfunktionen online lesen:
https://riptutorial.com/de/postgresql/topic/9230/postgres-verschlusselungsfunktionen

Kapitel 18: Programmgesteuert auf Daten zugreifen

Examples

Zugriff auf Postgresql von .NET aus über den Npgsql-Anbieter

Einer der bekannteren .NET-Anbieter für Postgresql ist Npgsql , das ADO.NET-kompatibel ist und fast identisch mit anderen .NET-Datenbankanbietern verwendet wird.

Eine typische Abfrage wird ausgeführt, indem ein Befehl erstellt, Parameter gebunden und dann der Befehl ausgeführt wird. In 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
```

Zugriff auf PostgreSQL mit der C-API

Die C-API ist der leistungsfähigste Weg, um auf PostgreSQL zuzugreifen, und ist überraschend komfortabel.

Zusammenstellung und Verlinkung

Während des Kompilierens müssen Sie das Include-Verzeichnis von PostgreSQL, das mit `pg_config --includedir` gefunden `pg_config --includedir` , zum Include-Pfad `pg_config --includedir` .
Sie müssen eine Verknüpfung mit der gemeinsam genutzten Bibliothek des PostgreSQL-Clients herstellen (`libpq.so` unter UNIX, `libpq.dll` unter Windows). Diese Bibliothek befindet sich im PostgreSQL-Bibliotheksverzeichnis, das mit `pg_config --libdir` gefunden `pg_config --libdir` .

Hinweis: Aus historischen Gründen heißt die Bibliothek `libpq.so` und *nicht* `libpg.so` ist eine beliebte Falle für Anfänger.

Da sich das folgende Codebeispiel in der Datei `coltype.c` , würde das Kompilieren und Verknüpfen mit erfolgen

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

mit dem GNU C-Compiler (in Betracht `-Wl,-rpath,"$(pg_config --libdir)"` , um den Bibliotheksuchpfad hinzuzufügen) oder mit

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

unter Windows mit Microsoft Visual C.

Beispielprogramm

```
/* 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;
}
```

Zugriff auf PostgreSQL von Python mit psycopg2

Eine Beschreibung des Treibers finden Sie hier .

Das schnelle Beispiel ist:

```
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())
```

Wird resultieren:

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

Zugriff auf PostgreSQL von PHP aus mit Pomm2

Auf den Schultern der Low-Level-Treiber befindet sich Pomm . Es bietet einen modularen Ansatz, Datenkonverter, Listen / Notification-Support, Datenbankinspektor und vieles mehr.

Angenommen, Pomm wurde mit Composer installiert. Hier ein vollständiges Beispiel:

```
<?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]' : '');
    }
}
```

Das Abfragemanager-Modul von Pomm umgeht Abfrageargumente, um die SQL-Injektion zu verhindern. Wenn die Argumente umgewandelt werden, werden sie auch von einer PHP-Darstellung in gültige Postgres-Werte konvertiert. Das Ergebnis ist ein Iterator, der intern einen Cursor verwendet. Jede Zeile wird im laufenden Betrieb konvertiert, Booleans in Booleans, Zeitstempel in \ DateTime usw.

Programmgesteuert auf Daten zugreifen online lesen:
https://riptutorial.com/de/postgresql/topic/2014/programmgesteuert-auf-daten-zugreifen

Kapitel 19: Programmierung mit PL / pgSQL

Bemerkungen

PL / pgSQL ist eine in PostgreSQL integrierte Programmiersprache zum Schreiben von Funktionen, die in der Datenbank selbst ausgeführt werden und in anderen Datenbanken als gespeicherte Prozeduren bezeichnet werden. Es erweitert SQL um Schleifen, Bedingungen und Rückgabetypen. Obwohl die Syntax für viele Entwickler ungewöhnlich ist, ist sie viel schneller als alles, was auf dem Anwendungsserver ausgeführt wird, da der Verbindungsaufwand für die Verbindung zur Datenbank entfällt. Dies ist besonders nützlich, wenn Sie andernfalls eine Abfrage ausführen müssen. Warten Sie auf das Ergebnis. und senden Sie eine weitere Abfrage.

Obwohl es viele andere prozedurale Sprachen für PostgreSQL gibt, wie PL / Python, PL / Perl und PLV8, ist PL / pgSQL ein allgemeiner Ausgangspunkt für Entwickler, die ihre erste PostgreSQL-Funktion schreiben möchten, da ihre Syntax auf SQL basiert. Es ist auch ähnlich zu PL / SQL, der muttersprachlichen Verfahrenssprache von Oracle, so dass jeder, der sich mit PL / SQL auskennt, die Sprache als bekannt kennt und jeder Entwickler, der beabsichtigt, Oracle-Anwendungen in der Zukunft zu entwickeln, aber mit einer freien Datenbank beginnen möchte, den Übergang vornehmen kann von PL / pgSQL zu PL / SQL relativ einfach.

Es sollte betont werden, dass andere prozedurale Sprachen existieren und PL / pgSQL ihnen in keiner Weise überlegen ist, einschließlich der Geschwindigkeit. Beispiele in PL / pgSQL können jedoch als allgemeiner Bezugspunkt für andere Sprachen dienen, die zum Schreiben von PostgreSQL-Funktionen verwendet werden. PL / pgSQL bietet die meisten Tutorials und Bücher aller PLs und kann ein Sprungbrett für das Erlernen der Sprachen mit weniger Dokumentation sein.

Hier sind Links zu einigen kostenlosen Handbüchern und Büchern zu PL / pgSQL:

- Die offizielle Dokumentation: https://www.postgresql.org/docs/current/static/plpgsql.html
- w3resource.com-Tutorial: http://www.w3resource.com/PostgreSQL/pl-pgsql-tutorial.php
- Tutorial postgres.cz: http://postgres.cz/wiki/PL/pgSQL_(en)
- PostgreSQL Server Programming, 2. Ausgabe: https://www.packtpub.com/big-data-and-business-intelligence/postgresql-server-programming-secondedition
- PostgreSQL-Entwicklerhandbuch: https://www.packtpub.com/big-data-and-business-intelligence/postgresql-developers-guide

Examples

Grundlegende PL / pgSQL-Funktion

Eine einfache PL / pgSQL-Funktion:

```
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;
```

Dies hätte nur mit der SQL-Anweisung erreicht werden können, zeigt aber die grundlegende Struktur einer Funktion.

Um die Funktion auszuführen, machen Sie:

```
select active_subscribers();
```

PL / pgSQL-Syntax

```
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;
```

RÜCKGABE Block

Optionen für die Rückgabe in einer PL / pgSQL-Funktion:

- `Datatype` Liste aller Datentypen
- `Table(column_name column_type, ...)`
- `Setof 'Datatype' or 'table_column'`

benutzerdefinierte Ausnahmen

Benutzerdefinierte Ausnahme 'P2222' erstellen:

```
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
;
```

Benutzerdefinierte Ausnahme erstellen, die kein Errm zuweist:

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

Berufung:

```
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
```

hier wurde das benutzerdefinierte P0001 verarbeitet, und P2222 nicht, die Ausführung wurde abgebrochen.

Es ist auch sehr sinnvoll, eine Tabelle mit Ausnahmen zu führen, wie hier:
http://stackoverflow.com/a/2700312/5315974

Programmierung mit PL / pgSQL online lesen:
https://riptutorial.com/de/postgresql/topic/5299/programmierung-mit-pl---pgsql

Kapitel 20: Rekursive Abfragen

Einführung

Es gibt keine rekursiven Abfragen!

Examples

Summe der ganzen Zahlen

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

Link zur Dokumentation

Rekursive Abfragen online lesen: https://riptutorial.com/de/postgresql/topic/9025/rekursive-abfragen

Kapitel 21: Rollenverwaltung

Syntax

- `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

Erstellen Sie einen Benutzer mit einem Kennwort

Im Allgemeinen sollten Sie die Verwendung der Standarddatenbankrolle (häufig `postgres`) in Ihrer Anwendung vermeiden. Sie sollten stattdessen einen Benutzer mit niedrigeren Berechtigungen erstellen. Hier machen wir einen Namen namens `niceusername` und geben ihm ein `very-strong-password`

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

Das Problem dabei ist, dass Abfragen, die in die `psql` Konsole eingegeben werden, in der History-Datei `.psql_history` im Home-Verzeichnis des Benutzers gespeichert werden und `.psql_history` im PostgreSQL-Datenbankserverprotokoll protokolliert werden können, wodurch das Kennwort angezeigt wird.

Um dies zu vermeiden, verwenden Sie den Befehl `\password` , um das Benutzerkennwort festzulegen. Wenn der Benutzer, der den Befehl ausgibt, ein Superuser ist, wird das aktuelle Kennwort nicht abgefragt. (Muss ein Superuser sein, um die Passwörter der Superuser zu ändern.)

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

Erstellen Sie eine Rollen- und passende Datenbank

Um eine bestimmte Anwendung zu unterstützen, erstellen Sie häufig eine neue Rolle und Datenbank, die übereinstimmen soll.

Die auszuführenden Shell-Befehle lauten wie folgt:

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

```
$ createdb -O blogger blogger
```

Dies setzt voraus, dass `pg_hba.conf` richtig konfiguriert wurde, was wahrscheinlich so aussieht:

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

Gewähren und Widerruf von Berechtigungen.

Angenommen, wir haben drei Benutzer:

1. Der Administrator der Datenbank> admin
2. Die Anwendung mit vollem Zugriff auf ihre Daten> read_write
3. Der Nur-Lese-Zugriff> read_only

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

Mit den obigen Abfragen können nicht vertrauenswürdige Benutzer keine Verbindung zur Datenbank mehr herstellen.

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

Die nächste Gruppe von Abfragen sperrt alle Berechtigungen von nicht authentifizierten Benutzern und bietet eingeschränkte Berechtigungen für den Benutzer `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;
```

Ändern Sie den Standard-Suchpfad des Benutzers

Mit den folgenden Befehlen kann der Standard-Suchpfad des Benutzers festgelegt werden.

1. Überprüfen Sie den Suchpfad, bevor Sie das Standardschema festlegen.

```
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. `search_path` Sie `search_path` mit dem Befehl `alter user` , um ein neues Schema `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. Ergebnis nach Ausführung prüfen.

```
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)
```

Alternative:

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

Gewähren Sie Zugriffsberechtigungen für Objekte, die in der Zukunft erstellt werden.

Angenommen, wir haben `three users` :

1. Der Administrator der Datenbank> `admin`
2. Die Anwendung mit vollem Zugriff auf ihre Daten> `read_write`
3. Der Nur-Lese-Zugriff> `read_only`

Mit den folgenden Abfragen können Sie Zugriffsberechtigungen für Objekte festlegen, die in der Zukunft in einem bestimmten Schema erstellt werden.

```
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;
```

Sie können auch Zugriffsberechtigungen für Objekte festlegen, die in der Zukunft von einem bestimmten Benutzer erstellt werden.

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

Schreibgeschützten Benutzer erstellen

```
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;
```

Rollenverwaltung online lesen: https://riptutorial.com/de/postgresql/topic/1572/rollenverwaltung

Kapitel 22: Sichern und Wiederherstellen

Bemerkungen

`pg_dumpall` des Dateisystems anstatt `pg_dumpall` und `pg_dump`

Es ist sehr wichtig, dass Sie in diesem `pg_start_backup()` Funktionen `pg_start_backup()` vor und `pg_stop_backup()` danach `pg_stop_backup()` . Das Sichern von Dateisystemen ist ansonsten nicht sicher. Selbst ein ZFS- oder FreeBSD-Snapshot des Dateisystems, das ohne diese Funktionsaufrufe gesichert wurde, versetzt die Datenbank in den Wiederherstellungsmodus und kann Transaktionen verlieren.

Ich würde vermeiden, Dateisystem-Backups anstelle von regulären Postgres-Backups durchzuführen, sowohl aus diesem Grund als auch, weil Postgres-Backup-Dateien (insbesondere im benutzerdefinierten Format) äußerst vielseitig sind, um alternative Wiederherstellungen zu unterstützen. Da es sich um Einzeldateien handelt, sind sie auch weniger umständlich zu verwalten.

Examples

Eine Datenbank sichern

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

Das `-Fc` wählt das "benutzerdefinierte Sicherungsformat" aus, das Ihnen mehr Leistung als `-Fc` SQL bietet. Weitere Informationen finden Sie unter `pg_restore` . Wenn Sie eine Vanilla-SQL-Datei benötigen, können Sie stattdessen Folgendes tun:

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

oder auch

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

Backups wiederherstellen

```
psql < backup.sql
```

Eine sicherere Alternative verwendet `-1` , um die Wiederherstellung in eine Transaktion einzuwickeln. `-f` gibt den Dateinamen an, anstatt die Shell-Umleitung zu verwenden.

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

Benutzerdefinierte `pg_restore` müssen mit `pg_restore` und der Option `-d` wiederhergestellt werden, um die Datenbank anzugeben:

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

Das benutzerdefinierte Format kann auch wieder in SQL konvertiert werden:

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

Die Verwendung des benutzerdefinierten Formats wird empfohlen, da Sie auswählen können, welche Elemente wiederhergestellt werden sollen, und optional die parallele Verarbeitung aktivieren.

Möglicherweise müssen Sie einen pg_dump gefolgt von einem pg_restore ausführen, wenn Sie von einer postgresql-Version auf eine neuere aktualisieren.

Das gesamte Cluster sichern

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

Dies funktioniert im Hintergrund, indem mehrere Verbindungen zum Server einmal für jede Datenbank hergestellt und `pg_dump` auf dieser Datenbank ausgeführt werden.

Manchmal können Sie versucht sein, dies als Cron-Job einzurichten. Sie möchten also das Datum sehen, an dem die Sicherung als Teil des Dateinamens erstellt wurde:

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

Beachten Sie jedoch, dass dies täglich große Dateien erzeugen kann. Postgresql hat einen viel besseren Mechanismus für regelmäßige Backups - WAL-Archive

Die Ausgabe von pg_dumpall reicht aus, um eine identisch konfigurierte Postgres-Instanz wiederherzustellen. Die Konfigurationsdateien in `$PGDATA` (`pg_hba.conf` und `postgresql.conf`) sind jedoch nicht Teil der Sicherung. Sie müssen sie daher separat sichern.

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

Um eine Sicherung des Dateisystems durchzuführen, müssen Sie diese Funktionen verwenden, um sicherzustellen, dass sich Postgres in einem konsistenten Zustand befindet, während die Sicherung vorbereitet wird.

Kopieren zum Importieren verwenden

So kopieren Sie Daten aus einer CSV-Datei in

eine Tabelle

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

Einfügen eines `user` in eine Tabelle mit dem Namen `user_data.csv` in `/home/user/` :

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

So kopieren Sie Daten aus einer durch Pipes getrennten Datei in eine Tabelle

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

Hinweis: Ohne die Option `with delimiter` ist das Standardbegrenzungszeichen Komma ,

Kopfzeile beim Importieren ignorieren

Verwenden Sie die Header-Option:

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

Hinweis: Wenn Daten in Anführungszeichen stehen, sind die Anführungszeichen der Daten standardmäßig in doppelte Anführungszeichen gesetzt. Wenn die Daten mit einem anderen Zeichen zitiert werden, verwenden Sie die Option `QUOTE` . Diese Option ist jedoch nur bei Verwendung des CSV-Formats zulässig.

Kopieren zum Exportieren verwenden

So kopieren Sie die Tabelle in den Standardbetrieb

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

So exportieren Sie den Tabellenbenutzer in die Standardausgabe:

Benutzer KOPIEREN ZU STDOUT (DELIMITER '|');

Tabelle in Datei kopieren

Benutzer KOPIEREN FROM '/ home / user / user_data' WITH DELIMITER '|';

So kopieren Sie die Ausgabe der SQL-Anweisung in eine Datei

COPY (SQL-Anweisung) TO '<Dateiname mit Pfad>';

COPY (SELECT * FROM Benutzer WO Benutzer_Name LIKE 'A%') TO '/ Home / Benutzer / Benutzerdaten';

So kopieren Sie in eine komprimierte Datei

Benutzer KOPIEREN ZUM PROGRAMM 'gzip> /home/user/user_data.gz';

Hier wird das Programm gzip ausgeführt, um die Daten der Benutzertabellen zu komprimieren.

Verwenden von psql zum Exportieren von Daten

Daten können mit dem Kopierbefehl oder mit den Befehlszeilenoptionen des Befehls psql exportiert werden.

So exportieren Sie CSV-Daten aus dem Tabellenbenutzer in die CSV-Datei:

```
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
```

Hier macht die Kombination von -A und -F den Trick.

`-F` ist als Trennzeichen anzugeben

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

Wechselt in den nicht ausgerichteten Ausgabemodus. (Der Standardausgabemodus ist ansonsten ausgerichtet.)

Sichern und Wiederherstellen online lesen: https://riptutorial.com/de/postgresql/topic/2291/sichern-und-wiederherstellen

Kapitel 23: Sicherungsskript für eine Produktionsdatenbank

Syntax

- Mit dem Skript können Sie für jede Ausführung ein Sicherungsverzeichnis mit der folgenden Syntax erstellen: Name des Datenbanksicherungsverzeichnisses + Datum und Uhrzeit der Ausführung
- Beispiel: prodDir22-11-2016-19h55
- Nach der Erstellung werden zwei Sicherungsdateien mit der folgenden Syntax erstellt: **Name der Datenbank + Datum und Uhrzeit der Ausführung**
- Beispiel:
- dbprod22-11-2016-19h55.backup **(** Sicherungsdatei **)**
- dbprod22-11-2016-19h55.sql **(SQL-Datei)**
- Am Ende einer Hinrichtung am **22-11-2016** @ **19h55 erhalten** wir:
- /save_bd/prodDir22-11-2016-19h55/dbprod22-11-2016-19h55.backup
- /save_bd/prodDir22-11-2016-19h55/dbprod22-11-2016-19h55.sql

Parameter

Parameter	Einzelheiten
save_db	Das Hauptsicherungsverzeichnis
dbProd	Das sekundäre Sicherungsverzeichnis
DATUM	Das Datum der Sicherung im angegebenen Format
dbprod	Der Name der Datenbank, die gespeichert werden soll
/opt/postgres/9.0/bin/pg_dump	Der Pfad zur pg_dump-Binärdatei
-h	Gibt den Hostnamen des Computers an, auf dem der Server ausgeführt wird. Beispiel: localhost
-p	Gibt den TCP-Port oder die lokale Unix-Domänensocket-Dateierweiterung an, an dem der Server Verbindungen überwacht, Beispiel 5432
-U	Benutzername als Verbindung.

Bemerkungen

1. Wenn es ein Sicherungsprogramm wie HDPS oder Symantec Backup gibt , ... Das

Sicherungsverzeichnis muss **vor jedem Start** geleert werden .

Um das Backup-Tool nicht durcheinander zu bringen, da die Sicherung alter Dateien durchgeführt werden soll.

Um diese Funktion zu aktivieren, markieren Sie bitte die Kommentarzeile 3.

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

2. Für den Fall, dass das Budget keine Sicherungswerkzeuge zulässt, kann immer der Aufgabenplaner (Befehl cron) verwendet werden.

   Mit dem folgenden Befehl können Sie die Cron-Tabelle für den aktuellen Benutzer bearbeiten.

```
crontab -e
```

   Planen Sie den Start des Skripts mit dem Kalender um 23:00 Uhr.

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

Examples

saveProdDb.sh

Im Allgemeinen neigen wir dazu, die Datenbank mit dem pgAdmin-Client zu sichern. Das folgende sh-Skript dient zum Speichern der Datenbank (unter Linux) in zwei Formaten:

- **SQL-Datei** : Für eine mögliche Zusammenfassung der Daten in einer beliebigen Version von PostgreSQL.
- **Dump-Datei** : für eine höhere Version als die aktuelle Version.

```
#!/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
```

Sicherungsskript für eine Produktionsdatenbank online lesen:

https://riptutorial.com/de/postgresql/topic/7974/sicherungsskript-fur-eine-produktionsdatenbank

Kapitel 24: Stellen Sie von Java aus eine Verbindung zu PostgreSQL her

Einführung

Die API zur Verwendung einer relationalen Datenbank aus Java ist JDBC.

Diese API wird von einem JDBC-Treiber implementiert.

Um es zu verwenden, setzen Sie die JAR-Datei mit dem Treiber in den JAVA-Klassenpfad.

Diese Dokumentation zeigt Beispiele, wie Sie mit dem JDBC-Treiber eine Verbindung zu einer Datenbank herstellen.

Bemerkungen

JDBC-URL

Die JDBC-URL kann eine der folgenden Formen annehmen:

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

  `host` standardmäßig auf `localhost` , *`port`* auf 5432.
  Wenn der *`host`* eine IPv6-Adresse ist, muss diese in eckigen Klammern stehen.
  Der Standard-Datenbankname entspricht dem Namen des verbindenden Benutzers.

  Um ein Failover zu implementieren, können mehrere *`host`* `[:` *`port`* `]` -Einträge durch ein Komma getrennt werden.
  Sie werden der Reihe nach versucht, bis eine Verbindung erfolgreich ist.

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

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

  Diese Formulare dienen zur Verbindung mit `localhost` .

`parameters` ist eine Liste von *`key`* `[=` *`value`* `]` , die von `?` und durch `&` getrennt. Wenn der *`value`* fehlt, wird davon ausgegangen, dass er `true` .

Ein Beispiel:

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

Verweise

- JDBC-Spezifikation: http://download.oracle.com/otndocs/jcp/jdbc-4_2-mrel2-eval-spec/

- PostgreSQL-JDBC-Treiber: https://jdbc.postgresql.org/
- PostgreSQL-JDBC-Treiberdokumentation: https://jdbc.postgresql.org/documentation/head/index.html

Examples

Verbindung mit java.sql.DriverManager

Dies ist der einfachste Weg, um eine Verbindung herzustellen.

Zunächst muss der Treiber bei `java.sql.DriverManager` *registriert* werden, damit er weiß, welche Klasse verwendet werden soll.
Dazu wird die Treiberklasse in der Regel mit `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);
}
```

Benutzer und Kennwort können nicht in der JDBC-URL enthalten sein. In diesem Fall müssen Sie sie nicht im Aufruf der `getConnection` Methode angeben.

Verbindung mit java.sql.DriverManager und Eigenschaften

Anstatt Verbindungsparameter wie Benutzer und Kennwort (siehe eine vollständige Liste hier) in der URL oder in separaten Parametern anzugeben, können Sie sie in ein `java.util.Properties` Objekt packen:

```
/**
 * 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);
}
```

Verbindung mit javax.sql.DataSource über einen Verbindungspool

Es ist üblich, `javax.sql.DataSource` mit JNDI in Anwendungsservercontainern zu verwenden, wo Sie eine Datenquelle unter einem Namen registrieren und bei Bedarf nachschlagen.

Dieser Code zeigt, wie Datenquellen funktionieren:

```
/**
 * 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;
}
```

Nachdem Sie eine Datenquelle durch Aufrufen dieser Funktion erstellt haben, würden Sie diese wie folgt verwenden:

```
/* 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();
```

Stellen Sie von Java aus eine Verbindung zu PostgreSQL her online lesen: https://riptutorial.com/de/postgresql/topic/9633/stellen-sie-von-java-aus-eine-verbindung-zu-postgresql-her

Kapitel 25: Tabellenerstellung

Examples

Tabellenerstellung mit Primärschlüssel

```
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)
);
```

Alternativ können Sie die `PRIMARY KEY` Einschränkung direkt in die `PRIMARY KEY` :

```
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)
);
```

Es wird empfohlen, dass Sie für die Tabelle und alle Spalten Kleinbuchstaben verwenden. Wenn Sie Großbuchstaben wie `Person` Sie diesen Namen in doppelte Anführungszeichen (`"Person"`) in jede Abfrage einschließen, da PostgreSQL die Fallverfolgung erzwingt.

Tabellendefinition anzeigen

Öffnen Sie das Befehlszeilentool `psql` , das mit der Datenbank verbunden ist, in der sich Ihre Tabelle befindet. Geben Sie dann den folgenden Befehl ein:

```
\d tablename
```

Um einen erweiterten Informationstyp zu erhalten

```
\d+ tablename
```

Wenn Sie den Namen der Tabelle vergessen haben, geben Sie einfach \ d in psql ein, um eine Liste der Tabellen und Ansichten in der aktuellen Datenbank zu erhalten.

Tabelle erstellen aus auswählen

Nehmen wir an, Sie haben eine Tabelle namens person:

```
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)
);
```

Sie können eine neue Tabelle mit Personen über 30 wie folgt erstellen:

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

Erstellen Sie eine nicht protokollierte Tabelle

Sie können nicht protokollierte Tabellen erstellen, um die Tabellen erheblich schneller zu machen. Nicht protokollierte Tabelle überspringt das `write-ahead` Protokoll, sodass es nicht absturzsicher ist und nicht repliziert werden kann.

```
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)
);
```

Erstellen Sie eine Tabelle, die auf andere Tabelle verweist.

In diesem Beispiel enthält die Benutzertabelle eine Spalte, die auf die Agency-Tabelle verweist.

```
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.
)
```

Tabellenerstellung online lesen: https://riptutorial.com/de/postgresql/topic/2430/tabellenerstellung

Kapitel 26: Trigger- und Triggerfunktionen

Einführung

Der Trigger wird der angegebenen Tabelle oder Sicht zugeordnet und führt die angegebene Funktion Funktionsname aus, wenn bestimmte Ereignisse auftreten.

Bemerkungen

Bitte benutzen Sie den untenstehenden Link für eine vollständige Übersicht über:

- **Auslöser** : https://www.postgresql.org/docs/current/static/sql-createtrigger.html
- **Triggerfunktionen** : https://www.postgresql.org/docs/current/static/plpgsql-trigger.html

Examples

Grundlegende PL / pgSQL-Triggerfunktion

Dies ist eine einfache Triggerfunktion.

```
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;
```

Hinzufügen dieser Auslöserfunktion zur `users`

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

Art der Auslöser

Auslöser kann zum Auslösen angegeben werden:

- `BEFORE` der Vorgang in einer Zeile versucht wird - Einfügen, Aktualisieren oder Löschen.
- `AFTER` der Vorgang abgeschlossen ist - Einfügen, Aktualisieren oder Löschen.
- `INSTEAD OF` the operation bei Einfügungen, Aktualisierungen oder Löschungen in einer Ansicht.

Auslöser, der markiert ist:

- `FOR EACH ROW` heißt einmal für jede Zeile , dass die Operation modifiziert;
- `FOR EACH STATEMENT` wird für jede gegebene Operation aufgerufen.

Beispiele für die Ausführung vorbereiten

```
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()
)
```

Single-Insert-Abzug

Schritt 1: Erstellen Sie Ihre Funktion

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

Schritt 2: Erstellen Sie Ihren Abzug

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

Schritt 3: testen Sie es

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

Auslöser für mehrere Zwecke

Schritt 1: Erstellen Sie Ihre Funktion

```
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;
```

Schritt 2: Erstellen Sie Ihren Abzug

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

Schritt 3: testen Sie es

```
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;
```

Trigger- und Triggerfunktionen online lesen: https://riptutorial.com/de/postgresql/topic/6957/trigger--und-triggerfunktionen

Kapitel 27: VERSCHMELZEN

Einführung

Coalesce gibt das erste Argument ohne Nullen aus einer Reihe von Argumenten zurück. Nur das erste Nicht-Null-Argument ist return, alle nachfolgenden Argumente werden ignoriert. Die Funktion wird zu Null ausgewertet, wenn alle Argumente Null sind.

Examples

Einzelnes Nicht-Null-Argument

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

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

Mehrere Nicht-Null-Argumente

PGSQL> SELECT COALESCE (NULL, NULL, 'erster nicht null', null, null, 'zweiter nicht null')

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

Alle Nullargumente

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

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

VERSCHMELZEN online lesen: https://riptutorial.com/de/postgresql/topic/10576/verschmelzen

Kapitel 28: WÄHLEN

Examples

WÄHLEN Sie mit WHERE

In diesem Thema stützen wir uns auf diese Benutzertabelle:

```
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 |
+----+------------+-----------+----------+------+
```

Syntax

Wähle alles aus:

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

Wählen Sie einige Felder aus:

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

Beispiele

```
-- 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;
```

WÄHLEN online lesen: https://riptutorial.com/de/postgresql/topic/9528/wahlen

Kapitel 29: Zeichenkettenlänge / Zeichenlänge suchen

Einführung

Um die Länge der Felder "Zeichen ändern", "Text" zu erhalten, verwenden Sie char_length () oder character_length ().

Examples

Beispiel, um die Länge eines Zeichenänderungsfelds zu erhalten

Beispiel 1, Abfrage: `SELECT char_length('ABCDE')`

Ergebnis:

    5

Beispiel 2, Abfrage: `SELECT character_length('ABCDE')`

Ergebnis:

    5

Zeichenkettenlänge / Zeichenlänge suchen online lesen:
https://riptutorial.com/de/postgresql/topic/9695/zeichenkettenlange---zeichenlange-suchen

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit 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	Aggregatfunktionen	Alison S, joseph, Kirill Sokolov, Patrick
3	AKTUALISIEREN	frlan, leeor
4	Allgemeine Tabellenausdrücke (WITH)	Daniel Lyons, Jakub Fedyczak, Kevin Sylvestre
5	Datentypen	Ben H, user_0
6	Datumsangaben, Zeitstempel und Intervalle	KIM, Nuri Tasdemir, Patrick, Tom Gerken
7	EINFÜGEN	chalitha geekiyanage, e4c5, gpdude_, KIM, lamorach, leeor, Nathaniel Waisbrot, Patrick, Vao Tsun
8	Erbe	evuez
9	Ereignisauslöser	Ben H, Tajinder, Udlei Nati
10	Exportieren Sie den Header und die Daten der PostgreSQL-Datenbanktabelle in die CSV-Datei	Vao Tsun, wOwhOw
11	EXTENSION dblink und postgres_fdw	Riya Bansal, YCF_L
12	Fensterfunktionen	mnoronha, Vao Tsun
13	JSON-Unterstützung	Clodoaldo Neto, commonSenseCode, jgm, KIRAN KUMAR MATAM, mnoronha, Peter Krauss
14	Kommentare in postgresql	Ben, KIRAN KUMAR MATAM
15	Postgres Tipp und Tricks	Ben H, skj123, user_0, YCF_L
16	PostgreSQL	gpdude_, Patrick

	Hochverfügbarkeit	
17	Postgres-Verschlüsselungsfunktionen	Ben H, skj123
18	Programmgesteuert auf Daten zugreifen	AstraSerg, brichins, greg, Laurenz Albe
19	Programmierung mit PL / pgSQL	AndrewCichocki, Ben H, Goerman, Laurenz Albe, Vao Tsun
20	Rekursive Abfragen	Ben H
21	Rollenverwaltung	Ben, Ben H, bilelovitch, Blackus, Daniel Lyons, e4c5, greg, KIM, Laurenz Albe, mnoronha, Reboot
22	Sichern und Wiederherstellen	ankidaemon, Ben H, Daniel Lyons, e4c5, Laurel, mnoronha
23	Sicherungsskript für eine Produktionsdatenbank	bilelovitch
24	Stellen Sie von Java aus eine Verbindung zu PostgreSQL her	Laurenz Albe
25	Tabellenerstellung	e4c5, Jefferson, KIM, leeor, Patrick
26	Trigger- und Triggerfunktionen	chalitha geekiyanage, mnoronha, Udlei Nati
27	VERSCHMELZEN	Mokadillion
28	WÄHLEN	YCF_L
29	Zeichenkettenlänge / Zeichenlänge suchen	Mohamed Navas