EBook Gratis

APRENDIZAJE postgresql

Free unaffiliated eBook created from
Stack Overflow contributors.

#postgresql

Tabla de contenido

Acerca de

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

Capítulo 1: Empezando con postgresql

Observaciones

Esta sección proporciona una descripción general de qué es postgresql y por qué un desarrollador puede querer usarlo.

También debe mencionar cualquier tema importante dentro de postgresql y vincular a los temas relacionados. Dado que la Documentación para postgresql es nueva, es posible que deba crear versiones iniciales de esos temas relacionados.

Versiones

Versión	Fecha de lanzamiento	Fecha de EOL
9.6	2016-09-29	2021-09-01
9.5	2016-01-07	2021-01-01
9.4	2014-12-18	2019-12-01
9.3	2013-09-09	2018-09-01
9.2	2012-09-10	2017-09-01
9.1	2011-09-12	2016-09-01
9.0	2010-09-20	2015-09-01
8.4	2009-07-01	2014-07-01

Examples

Instalación en GNU + Linux

En la mayoría de los sistemas operativos GNU + Linux, PostgreSQL se puede instalar fácilmente usando el administrador de paquetes del sistema operativo.

Familia red hat

Los respositorios se pueden encontrar aquí: https://yum.postgresql.org/repopackages.php

Descargue el repositorio a la máquina local con el comando

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

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

Ver los paquetes disponibles:

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

Los paquetes necesarios son: postgresqlXX postgresqlXX-server postgresqlXX-libs postgresqlXX-contrib

Estos se instalan con el siguiente comando: yum -y install postgresqlXX postgresqlXX-server postgresqlXX-libs postgresqlXX-contrib

Una vez instalado, deberá iniciar el servicio de base de datos como propietario del servicio (el valor predeterminado es postgres). Esto se hace con el comando pg_ctl.

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

Para acceder al DB en CLI ingresa `psql`

Familia debian

En Debian y sistemas operativos derivados , escriba:

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

Esto instalará el paquete del servidor PostgreSQL, en la versión predeterminada que ofrecen los repositorios de paquetes del sistema operativo.

Si la versión que está instalada de forma predeterminada no es la que usted desea, puede usar el administrador de paquetes para buscar versiones específicas que pueden ofrecerse simultáneamente.

También puede usar el repositorio de Yum proporcionado por el proyecto PostgreSQL (conocido como PGDG) para obtener una versión diferente. Esto puede permitir versiones aún no ofrecidas por los repositorios de paquetes del sistema operativo.

Cómo instalar PostgreSQL a través de MacPorts en OSX

Para instalar PostgreSQL en OSX, necesita saber qué versiones son compatibles actualmente.

Utilice este comando para ver qué versiones tiene disponibles.

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

Debería obtener una lista que se parece a la siguiente:

```
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
```

En este ejemplo, la versión más reciente de PostgreSQL que se admite en 9.6, así que la instalaremos.

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

Verás un registro de instalación como este:

```
--->  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.
```

El registro proporciona instrucciones sobre el resto de los pasos para la instalación, por lo que hacemos eso a continuación.

```
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'
```

Ahora iniciamos el servidor:

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

Verifique que podamos conectarnos al servidor:

```
su postgres -c psql
```

Verá un aviso de postgres:

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

postgres=#
```

Aquí puede escribir una consulta para ver si el servidor se está ejecutando.

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

Y ver la respuesta:

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

Escribe \ q para salir:

```
postgres=#\q
```

Y volverá a su indicador de shell.

¡Felicidades! Ahora tiene una instancia de PostgreSQL en ejecución en OS / X.

Postgres.app para Mac OSX

Una herramienta extremadamente simple para instalar PostgreSQL en una Mac está disponible descargando Postgres.app .
Puede cambiar las preferencias para que PostgreSQL se ejecute en segundo plano o solo cuando la aplicación se está ejecutando.

Instalando PostgreSQL en Windows

Si bien es una buena práctica usar un sistema operativo basado en Unix (por ejemplo, Linux o BSD) como servidor de producción, puede instalar fácilmente PostgreSQL en Windows (con suerte solo como servidor de desarrollo).

Descargue los binarios de instalación de Windows de EnterpriseDB: http://www.enterprisedb.com/products-services-training/pgdownload Esta es una compañía de terceros iniciada por colaboradores principales del proyecto PostgreSQL que han optimizado los binarios para Windows.

Seleccione la última versión estable (no Beta) (9.5.3 en el momento de la escritura). Lo más probable es que desee el paquete Win x86-64, pero si está ejecutando una versión de Windows de 32 bits, que es común en las computadoras antiguas, seleccione Win x86-32 en su lugar.

Nota: el cambio entre las versiones Beta y Estable implicará tareas complejas como volcado y restauración. La actualización dentro de la versión beta o estable solo necesita un reinicio del servicio.

Puede verificar si su versión de Windows es de 32 o 64 bits yendo a Panel de control -> Sistema y seguridad -> Sistema -> Tipo de sistema, que dirá "## - Sistema operativo de bits". Esta es la ruta para Windows 7, puede ser ligeramente diferente en otras versiones de Windows.

En el instalador, seleccione los paquetes que desea utilizar. Por ejemplo:

- pgAdmin (https://www.pgadmin.org) es una GUI gratuita para administrar su base de datos y lo recomiendo altamente. En 9.6 esto se instalará por defecto.
- PostGIS (http://postgis.net) proporciona funciones de análisis geoespaciales sobre coordenadas GPS, distancias, etc., muy populares entre los desarrolladores de SIG.
- El paquete de idiomas proporciona las bibliotecas necesarias para el lenguaje de procedimientos oficialmente admitido PL / Python, PL / Perl y PL / Tcl.
- Otros paquetes como pgAgent, pgBouncer y Slony son útiles para servidores de producción más grandes, solo se verifican según sea necesario.

Todos esos paquetes opcionales se pueden instalar posteriormente a través del "Application Stack Builder".

Nota: También hay otros idiomas compatibles no oficiales como PL / V8 , PL / Lua PL / Java disponibles.

Abra pgAdmin y conéctese a su servidor haciendo doble clic en su nombre, ej. "PostgreSQL 9.5 (localhost: 5432).

Desde este punto, puede seguir guías como el excelente libro PostgreSQL: Up and Running, 2nd Edition (http://shop.oreilly.com/product/0636920032144.do).

Opcional: Tipo de inicio de servicio manual

PostgreSQL se ejecuta como un servicio en segundo plano que es ligeramente diferente a la mayoría de los programas. Esto es común para las bases de datos y servidores web. Su tipo de inicio predeterminado es Automático, lo que significa que siempre se ejecutará sin ninguna

entrada de usted.

¿Por qué querría controlar manualmente el servicio PostgreSQL? Si está utilizando su PC como servidor de desarrollo algunas veces y también lo usa para jugar videojuegos, por ejemplo, PostegreSQL podría ralentizar un poco su sistema mientras se está ejecutando.

¿Por qué no quieres control manual? Iniciar y detener el servicio puede ser una molestia si lo haces a menudo.

Si no nota ninguna diferencia en la velocidad y prefiere evitar la molestia, deje su Tipo de inicio como Automático e ignore el resto de esta guía. De otra manera...

Vaya a Panel de control -> Sistema y seguridad -> Herramientas administrativas.

Seleccione "Servicios" de la lista, haga clic con el botón derecho en su icono y seleccione Enviar a -> Escritorio para crear un icono de escritorio para un acceso más conveniente.

Cierre la ventana Herramientas administrativas y luego inicie Servicios desde el icono del escritorio que acaba de crear.

Desplácese hacia abajo hasta que vea un servicio con un nombre como postgresql-x ## - 9. # (ej. "Postgresql-x64-9.5").

Haga clic derecho en el servicio postgres, seleccione Propiedades -> Tipo de inicio -> Manual -> Aplicar -> Aceptar. Puedes cambiarlo de nuevo a automático con la misma facilidad.

Si ve otros servicios relacionados con PostgreSQL en la lista como "pgbouncer" o "PostgreSQL Scheduling Agent - pgAgent", también puede cambiar su Tipo de inicio a Manual porque no son muy útiles si PostgreSQL no se está ejecutando. Aunque esto significará más problemas cada vez que comiences y te detengas, depende de ti. No utilizan tantos recursos como PostgreSQL en sí y es posible que no tengan un impacto notable en el rendimiento de sus sistemas.

Si el servicio se está ejecutando, su estado dirá Iniciado, de lo contrario no se estará ejecutando.

Para iniciarlo haz click derecho y selecciona Iniciar. Se mostrará un mensaje de carga y desaparecerá por sí solo poco después. Si le da un error intente una segunda vez. Si eso no funciona, entonces hubo algún problema con la instalación, posiblemente porque cambió alguna configuración en Windows que la mayoría de la gente no cambia, por lo que encontrar el problema puede requerir cierta investigación.

Para detener postgres, haga clic derecho en el servicio y seleccione Detener.

Si alguna vez recibe un error al intentar conectarse a su base de datos, compruebe los Servicios para asegurarse de que se está ejecutando.

Para otros detalles muy específicos sobre la instalación de EDB PostgreSQL, por ejemplo, la versión de Python Runtime en el paquete de idioma oficial de una versión específica de PostgreSQL, siempre consulte la guía de instalación oficial de EBD , cambie la versión en enlace a la versión principal de su instalador.

Instalar postgresql con cerveza en Mac

Homebrew se llama a sí mismo ' *el administrador de paquetes faltantes para macOS* '. Se puede utilizar para construir e instalar aplicaciones y bibliotecas. Una vez instalado , puede usar el comando `brew` para instalar PostgreSQL y sus dependencias de la siguiente manera:

```
brew update
brew install postgresql
```

Homebrew generalmente instala la última versión estable. Si necesita uno diferente, `brew search postgresql` las versiones disponibles. Si necesita PostgreSQL creado con opciones particulares, `brew info postgresql` las opciones que son compatibles. Si necesita una opción de compilación no admitida, puede que tenga que hacer la compilación usted mismo, pero aún puede usar Homebrew para instalar las dependencias comunes.

Iniciar el servidor:

```
brew services start postgresql
```

Abra el prompt de PostgreSQL

```
psql
```

Si psql se queja de que no hay una base de datos correspondiente para su usuario, ejecute `createdb` .

Instala PostgreSQL desde Source en Linux

Dependencias:

- Versión GNU Make> 3.80
- un compilador C de ISO / ANSI (por ejemplo, gcc)
- Un extractor como el alquitrán o el gzip.
- zlib-devel
- readline-devel oder libedit-devel

Fuentes: Enlace a la última fuente (9.6.3)

Ahora puedes extraer los archivos fuente:

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

Hay una gran cantidad de opciones diferentes para la configuración de PostgreSQL:

Enlace completo al procedimiento completo de instalación

Pequeña lista de opciones disponibles:

- `--prefix=PATH` ruta de acceso `--prefix=PATH` para todos los archivos
- `--exec-prefix=PATH` ruta de acceso `--exec-prefix=PATH` para el archivo architectur-dependet
- `--bindir=PATH` ruta `--bindir=PATH` para programas ejecutables
- `--sysconfdir=PATH` ruta de acceso `--sysconfdir=PATH` para archivos de configuración
- `--with-pgport=NUMBER` especifica un puerto para su servidor
- `--with-perl` agregar soporte perl
- `--with-python` agrega soporte python
- `--with-openssl` agrega soporte openssl
- `--with-ldap` añadir soporte ldap
- `--with-blocksize=BLOCKSIZE` establece tamaño de página en KB
    - `BLOCKSIZE` debe tener una potencia de 2 y entre 1 y 32
- `--with-wal-segsize=SEGSIZE` establece el tamaño del tamaño del segmento WAL en MB
    - `SEGSIZE` debe ser una potencia de 2 entre 1 y 64

Vaya a la nueva carpeta creada y ejecute el script cofigure con las opciones deseadas:

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

Ejecutar `make` para crear los archivos de objeto.

Ejecute `make install` para instalar PostgreSQL desde los archivos construidos

Ejecutar `make clean` para poner en orden

Para la extensión, cambie el directorio `cd contrib` , ejecute `make` y `make install`

Lea Empezando con postgresql en línea:
https://riptutorial.com/es/postgresql/topic/885/empezando-con-postgresql

Capítulo 2: Accediendo a los datos programáticamente

Examples

Acceso a Postgresql desde .NET utilizando el proveedor Npgsql

Uno de los proveedores de .NET más populares para Postgresql es Npgsql , que es compatible con ADO.NET y se utiliza casi idénticamente como otros proveedores de base de datos de .NET.

Una consulta típica se realiza creando un comando, vinculando parámetros y luego ejecutando el comando. Cía#:

```
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
```

Accediendo a PostgreSQL con la C-API

El C-API es la forma más poderosa de acceder a PostgreSQL y es sorprendentemente cómodo.

Recopilación y vinculación.

Durante la compilación, debe agregar el directorio de inclusión de PostgreSQL, que se puede encontrar con `pg_config --includedir` , a la ruta de inclusión.
Debe vincularse con la biblioteca compartida del cliente PostgreSQL (`libpq.so` en UNIX, `libpq.dll` en Windows). Esta biblioteca se encuentra en el directorio de bibliotecas de PostgreSQL, que se puede encontrar con `pg_config --libdir` .

Nota: por razones históricas, la biblioteca se llama `libpq.so` y *no* `libpg.so` , que es una trampa popular para los principiantes.

Dado que el ejemplo de código siguiente está en el archivo `coltype.c` , la compilación y el enlace se realizarían con

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

con el compilador GNU C (considere agregar `-Wl,-rpath,"$(pg_config --libdir)"` para agregar la ruta de búsqueda de la biblioteca) o con

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

en Windows con Microsoft Visual C.

Programa de muestra

```
/* 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;
}
```

Accediendo a PostgreSQL desde python usando psycopg2

Puedes encontrar la descripción del driver aquí .

El ejemplo rápido es:

```
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())
```

Resultará:

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

Accediendo a PostgreSQL desde PHP usando Pomm2

En los hombros de los conductores de bajo nivel, hay pomm . Propone un enfoque modular, convertidores de datos, soporte de escucha / notificación, inspector de base de datos y mucho más.

Suponiendo que Pomm se ha instalado usando el compositor, aquí hay un ejemplo completo:

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

El módulo del administrador de consultas de Pomm escapa de los argumentos de consulta para evitar la inyección de SQL. Cuando los argumentos se emiten, también los convierte de una representación de PHP a valores de Postgres válidos. El resultado es un iterador, utiliza un cursor internamente. Cada fila se convierte sobre la marcha, valores booleanos a valores booleanos, marcas de tiempo a \ DateTime, etc.

Lea Accediendo a los datos programáticamente en línea:
https://riptutorial.com/es/postgresql/topic/2014/accediendo-a-los-datos-programaticamente

Capítulo 3: Activadores de eventos

Introducción

Los disparadores de eventos se activarán cuando el evento asociado con ellos ocurra en la base de datos.

Observaciones

Utilice el siguiente enlace para obtener una descripción completa de los activadores de eventos en PostgreSQL

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

Examples

Registro de eventos de inicio de comando DDL

Tipo de evento-

- `DDL_COMMAND_START`
- `DDL_COMMAND_END`
- SQL_DROP

Este es un ejemplo para crear un activador de eventos y registrar eventos `DDL_COMMAND_START` .

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

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

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

Lea Activadores de eventos en línea: https://riptutorial.com/es/postgresql/topic/9255/activadores-de-eventos

Capítulo 4: ACTUALIZAR

Examples

Actualizar todas las filas en una tabla

Actualizas todas las filas en la tabla simplemente proporcionando un `column_name = value` :

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

Actualizar todas las filas que cumplan una condición

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

Actualizando columnas múltiples en tabla

Puede actualizar varias columnas en una tabla en la misma declaración, separando los pares `col=val` con comas:

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

Actualización de una tabla basada en unirse a otra tabla

También puede actualizar datos en una tabla basándose en datos de otra tabla:

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

Aquí nos estamos uniendo la `person city` columna a la `cities city` columna con el fin de obtener el código de estado de la ciudad. Esto se usa para actualizar la columna `state_code` en la tabla de `person` .

Lea ACTUALIZAR en línea: https://riptutorial.com/es/postgresql/topic/3136/actualizar

Capítulo 5: Alta disponibilidad de PostgreSQL

Examples

Replicación en PostgreSQL

- **Configurando el servidor primario**
  - **Requisitos:**
    - Usuario de replicación para actividades de replicación
    - Directorio para almacenar los archivos WAL
  - **Crear usuario de replicación**

    ```
    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
    ```

  - **Crear un directorio de archivos en el directorio de datos.**

    ```
    mkdir $PGDATA/archive
    ```

  - **Edita el archivo pg_hba.conf**

    Este es el archivo de autenticación de la base de host, contiene la configuración para la autenticación automática del cliente. Añadir a continuación la entrada:

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

  - **Edita el archivo postgresql.conf**

    Este es el archivo de configuración de PostgreSQL.

    ```
    wal_level = hot_standby
    ```

    Este parámetro decide el comportamiento del servidor esclavo.

    ```
    `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
    ```

Este parámetro permite enviar segmentos WAL a la ubicación de archivo utilizando el parámetro `archive_command` .

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

Básicamente, lo que hace arriba en `archive_command` es que copia los segmentos de WAL al directorio de archivo.

`wal_senders = 5` Este es el número máximo de procesos de remitente WAL.

Ahora reinicie el servidor primario.

- **Copia de seguridad del servidor primay al servidor esclavo**

  Antes de hacer cambios en el servidor, detenga el servidor primario.

  Importante: no vuelva a iniciar el servicio hasta que todos los pasos de configuración y copia de seguridad estén completos. Debe activar el servidor en espera en un estado en el que esté listo para ser un servidor de respaldo. Esto significa que todos los ajustes de configuración deben estar en su lugar y las bases de datos ya deben estar sincronizadas. De lo contrario, la replicación de secuencias no se iniciará

- **Ahora ejecuta la utilidad pg_basebackup**

  `pg_basebackup` utilidad `pg_basebackup` copia los datos del directorio de datos del servidor primario al directorio de datos esclavos.

  ```
  $ 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.
```

- **Configurando el servidor standby**

  Para configurar el servidor en espera, editará postgresql.conf y creará un nuevo archivo de configuración llamado recovery.conf.

  ```
  hot_standby = on
  ```

  Esto especifica si se le permite ejecutar consultas mientras se recupera

  - **Creando el archivo recovery.conf**

    ```
    standby_mode = on
    ```

    Establezca la cadena de conexión al servidor primario. Reemplace con la dirección IP externa del servidor primario. Reemplazar con la contraseña para el usuario llamado

replicación

`primary_conninfo = 'host = puerto = 5432 usuario = contraseña de replicación ='

(Opcional) Establezca la ubicación del archivo disparador:

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

La ruta del `trigger_file` que especifique es la ubicación donde puede agregar un archivo cuando desee que el sistema realice una conmutación por error al servidor en espera. La presencia del archivo "activa" la conmutación por error. Alternativamente, puede usar el comando pg_ctl Promote para activar la conmutación por error.

- **Iniciar el servidor en espera**

  Ahora tiene todo en su lugar y está listo para activar el servidor en espera

Atribución

Este artículo se deriva sustancialmente de y se atribuye a Cómo configurar PostgreSQL para alta disponibilidad y replicación con Hot Standby , con pequeños cambios en el formato y ejemplos y algunos textos eliminados. La fuente se publicó bajo la licencia pública de Creative Commons 3.0 , que se mantiene aquí.

Lea Alta disponibilidad de PostgreSQL en línea:
https://riptutorial.com/es/postgresql/topic/5478/alta-disponibilidad-de-postgresql

Capítulo 6: Comentarios en postgresql

Introducción

El propósito principal de ***COMMMENT*** es definir o cambiar un comentario en un objeto de base de datos.

Solo se puede dar un solo comentario (cadena) en cualquier objeto de base de datos. COMPROMISO nos ayudará a saber qué se definió para el objeto de base de datos en particular, cuál es su propósito real.

La regla para ***COMMENT ON ROLE*** es que debe ser superusuario para comentar sobre un rol de superusuario, o tener el privilegio ***CREATEROLE*** para comentar sobre roles que no sean de superusuario. Por supuesto, un *superusuario puede comentar cualquier cosa.*

Sintaxis

- COMENTARIO SOBRE database_object object_name IS 'Text';

Observaciones

Sintaxis completa, consulte: http://www.postgresql.org/docs/current/static/sql-comment.html

Examples

COMENTARIO sobre la mesa

COMENTARIO EN LA TABLA table_name IS 'esta es la tabla de detalles del estudiante';

Eliminar comentario

COMENTARIO sobre la tabla el estudiante es nulo;

El comentario será eliminado con la ejecución de la declaración anterior.

Lea Comentarios en postgresql en línea:
https://riptutorial.com/es/postgresql/topic/8191/comentarios-en-postgresql

Capítulo 7: Conéctate a PostgreSQL desde Java

Introducción

La API para usar una base de datos relacional de Java es JDBC.

Esta API es implementada por un controlador JDBC.

Para usarlo, coloque el archivo JAR con el controlador en la ruta de la clase JAVA.

Esta documentación muestra ejemplos de cómo utilizar el controlador JDBC para conectarse a una base de datos.

Observaciones

URL de JDBC

La URL de JDBC puede tomar una de estas formas:

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

  `host` defecto es `localhost` , *`port`* a 5432.
  Si el *`host`* es una dirección IPv6, debe estar entre corchetes.
  El nombre predeterminado de la base de datos es el mismo que el nombre del usuario que se conecta.

  Para implementar la conmutación por error, es posible tener varias entradas de *`host`* `[:` *`port`* `]` separadas por una coma.
  Se prueban a su vez hasta que una conexión tiene éxito.

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

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

  Estas formas son para conexiones a `localhost` .

`parameters` es una lista de pares *`key`* `[=` *`value`* `]` , encabezados por `?` y separados por `&` . Si falta el *`value`* , se asume que es `true` .

Un ejemplo:

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

Referencias

- Especificación JDBC: http://download.oracle.com/otndocs/jcp/jdbc-4_2-mrel2-eval-spec/
- Controlador JDBC de PostgreSQL: https://jdbc.postgresql.org/
- Documentación del controlador JDBC de PostgreSQL: https://jdbc.postgresql.org/documentation/head/index.html

Examples

Conectando con java.sql.DriverManager

Esta es la forma más sencilla de conectarse.

Primero, el controlador debe estar *registrado* con `java.sql.DriverManager` para que sepa qué clase usar.
Esto se hace cargando la clase de controlador, generalmente con `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);
}
```

No es posible que el usuario y la contraseña también se puedan incluir en la URL de JDBC, en cuyo caso no tiene que especificarlos en la llamada al método `getConnection` .

Conectando con java.sql.DriverManager y Propiedades

En lugar de especificar parámetros de conexión como usuario y contraseña (vea una lista completa aquí) en la URL o en parámetros separados, puede empaquetarlos en un objeto `java.util.Properties` :

```
/**
 * Connect to a PostgreSQL database.
 * @param url the JDBC URL to connect to. Must start with "jdbc:postgresql:"
```

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

Conectando con javax.sql.DataSource usando un grupo de conexiones

Es común usar `javax.sql.DataSource` con JNDI en los contenedores del servidor de aplicaciones, donde registra una fuente de datos con un nombre y la busca cada vez que necesita una conexión.

Este es un código que demuestra cómo funcionan las fuentes de datos:

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

Una vez que haya creado una fuente de datos al llamar a esta función, la usaría así:

```
/* 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();
```

Lea Conéctate a PostgreSQL desde Java en línea:
https://riptutorial.com/es/postgresql/topic/9633/conectate-a-postgresql-desde-java

Capítulo 8: Consejos y trucos de Postgres

Examples

Alternativa DATEADD en Postgres

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

Valores separados por comas de una columna

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

Eliminar registros duplicados de la tabla de postgres

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

La consulta de actualización con una combinación de dos tablas es una alternativa ya que Postresql no admite la unión en la consulta de actualización.

```
    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>
```

Diferencia entre dos marcas de fecha y hora mes y año

Diferencia de un mes entre dos fechas (marca de tiempo)

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

Diferencia anual entre dos fechas (marca de tiempo)

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

Consulta de copiar / mover / transferir datos de tablas de una base de datos a otra tabla de bases de datos con el mismo esquema

Primera ejecución

```
CREATE EXTENSION DBLINK;
```

Entonces

```
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>
    );
```

Lea Consejos y trucos de Postgres en línea:
https://riptutorial.com/es/postgresql/topic/7433/consejos-y-trucos-de-postgres

Capítulo 9: Consultas recursivas

Introducción

No hay consultas reales recursivas!

Examples

Suma de enteros

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

Enlace a la documentación

Lea Consultas recursivas en línea: https://riptutorial.com/es/postgresql/topic/9025/consultas-recursivas

Capítulo 10: Copia de seguridad y restaurar

Observaciones

Copia de seguridad del sistema de archivos en lugar de usar `pg_dumpall` y `pg_dump`

Es muy importante que si usa esto, llame a la función `pg_start_backup()` antes y a la función `pg_stop_backup()` después. Hacer copias de seguridad del sistema de archivos no es seguro de lo contrario; incluso una instantánea de ZFS o FreeBSD del sistema de archivos respaldado sin esas llamadas de función colocará la base de datos en modo de recuperación y puede perder transacciones.

Evitaría hacer copias de seguridad del sistema de archivos en lugar de copias de seguridad regulares de Postgres, tanto por este motivo como porque los archivos de copia de seguridad de Postgres (especialmente en el formato personalizado) son extremadamente versátiles para admitir restauraciones alternativas. Ya que son archivos únicos, también son menos complicados de administrar.

Examples

Copia de seguridad de una base de datos

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

`-Fc` selecciona el "formato de copia de seguridad personalizado" que le da más poder que el SQL sin formato; ver `pg_restore` para más detalles. Si desea un archivo SQL de vainilla, puede hacer esto en su lugar:

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

o incluso

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

Restaurando copias de seguridad

```
psql < backup.sql
```

Una alternativa más segura usa `-1` para envolver la restauración en una transacción. La `-f` especifica el nombre de archivo en lugar de usar la redirección de shell.

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

Los archivos de formato personalizado deben restaurarse utilizando `pg_restore` con la opción `-d` para especificar la base de datos:

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

El formato personalizado también se puede convertir de nuevo a SQL:

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

Se recomienda el uso del formato personalizado porque puede elegir qué cosas restaurar y, opcionalmente, habilitar el procesamiento paralelo.

Es posible que deba realizar un pg_dump seguido de un pg_restore si actualiza de una versión postgresql a una más nueva.

Copia de seguridad de todo el grupo

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

Esto funciona entre bastidores al hacer múltiples conexiones al servidor una vez para cada base de datos y ejecutar `pg_dump` en él.

A veces, puede tener la tentación de configurarlo como un trabajo cron, por lo que desea ver la fecha en que se tomó la copia de seguridad como parte del nombre de archivo:

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

Sin embargo, tenga en cuenta que esto podría producir archivos grandes diariamente. Postgresql tiene un mecanismo mucho mejor para copias de seguridad regulares: archivos WAL

La salida de pg_dumpall es suficiente para restaurar una instancia de Postgres configurada de forma idéntica, pero los archivos de configuración en `$PGDATA` (`pg_hba.conf` y `postgresql.conf`) no forman parte de la copia de seguridad, por lo que tendrá que hacer una copia de seguridad por separado.

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

Para realizar una copia de seguridad del sistema de archivos, debe usar estas funciones para asegurarse de que Postgres se encuentre en un estado constante mientras se prepara la copia de seguridad.

Usando Copiar para importar

Para copiar datos de un archivo CSV a una

tabla

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

Para insertar en el `user` tabla desde un archivo llamado `user_data.csv` ubicado dentro de `/home/user/` :

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

Para copiar datos de un archivo separado por tuberías a la tabla

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

Nota: en ausencia de la opción `with delimiter` , el delimitador predeterminado es una coma `,`

Para ignorar la línea del encabezado al importar el archivo

Utilice la opción de encabezado:

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

Nota: Si se citan datos, los caracteres de comillas de datos predeterminados son comillas dobles. Si los datos se citan utilizando cualquier otro carácter, use la opción `QUOTE` ; sin embargo, esta opción solo está permitida cuando se utiliza el formato CSV.

Usando Copiar para exportar

Para copiar la tabla al estándar o / p

COPY <tablename> A STDOUT (DELIMITER '|');

Para exportar el usuario de la tabla a la salida estándar:

COPY user TO STUTUT (DELIMITER '|');

Para copiar tabla a archivo

COPIAR usuario DESDE '/ home / user / user_data' WITH DELIMITER '|';

Para copiar la salida de la sentencia SQL al archivo

COPY (sentencia sql) TO '<nombre de archivo con ruta>';

COPIAR (SELECCIONAR * DEL usuario WHERE nombre_usuario COMO "A%") A '/ home / user / user_data';

Para copiar en un archivo comprimido

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

Aquí se ejecuta el programa gzip para comprimir los datos de la tabla de usuario.

Usando psql para exportar datos

Los datos se pueden exportar utilizando el comando de copia o utilizando las opciones de la línea de comandos del comando psql.

Para exportar datos csv del usuario de la tabla al archivo csv:

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

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

Aquí la combinación de -A y -F hace el truco.

`-F` es especificar delimitador

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

Cambia al modo de salida no alineado. (De lo contrario, el modo de salida predeterminado está alineado.)

Lea Copia de seguridad y restaurar en línea: https://riptutorial.com/es/postgresql/topic/2291/copia-de-seguridad-y-restaurar

Capítulo 11: Creación de tablas

Examples

Creación de tablas con clave primaria

```
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)
);
```

Alternativamente, puede colocar la restricción `PRIMARY KEY` directamente en la definición de columna:

```
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)
);
```

Se recomienda utilizar nombres en minúsculas para la tabla y también para todas las columnas. Si usa nombres en mayúsculas, como `Person` , tendría que ajustar ese nombre entre comillas dobles (`"Person"`) en todas y cada una de las consultas, porque PostgreSQL impone el plegado de casos.

Mostrar definición de tabla

Abra la herramienta de línea de comandos `psql` conectada a la base de datos donde se encuentra su tabla. Luego escribe el siguiente comando:

```
\d tablename
```

Para obtener el tipo de información extendida

```
\d+ tablename
```

Si ha olvidado el nombre de la tabla, simplemente escriba \ d en psql para obtener una lista de tablas y vistas en la base de datos actual.

Crear tabla desde seleccionar

Digamos que tienes una mesa llamada 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)
);
```

Puedes crear una nueva tabla de personas mayores de 30 como esta:

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

Crear tabla no registrada

Puede crear tablas no registradas para que pueda hacer las tablas considerablemente más rápido. La tabla no registrada salta la escritura `write-ahead` registro de `write-ahead` que significa que no es seguro y no se puede replicar.

```
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)
);
```

Cree una tabla que haga referencia a otra tabla.

En este ejemplo, la tabla de usuario tendrá una columna que hace referencia a la tabla de agencia.

```
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.
)
```

Lea Creación de tablas en línea: https://riptutorial.com/es/postgresql/topic/2430/creacion-de-tablas

Capítulo 12: Disparadores y funciones de disparador

Introducción

El activador se asociará con la tabla o vista especificada y ejecutará la función function_name especificada cuando se produzcan ciertos eventos.

Observaciones

Utilice el siguiente enlace para obtener una descripción completa de:

- **Desencadenadores** : https://www.postgresql.org/docs/current/static/sql-createtrigger.html
- **Funciones de activación** : https://www.postgresql.org/docs/current/static/plpgsql-trigger.html

Examples

Función básica de disparo PL / pgSQL

Esta es una función de disparo simple.

```
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;
```

Agregando esta función de disparo a la tabla de `users`

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

Tipo de disparadores

El disparador se puede especificar para disparar:

- `BEFORE` intentar la operación en una fila: insertar, actualizar o eliminar;
- `AFTER` la operación haya finalizado: insertar, actualizar o eliminar;
- `INSTEAD OF` la operación en el caso de inserciones, actualizaciones o eliminaciones en una vista.

Gatillo que está marcado:

- `FOR EACH ROW` se llama una vez por cada fila que modifica la operación;
- `FOR EACH STATEMENT` se llama onde para cualquier operación dada.

Preparando para ejecutar ejemplos.

```
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()
)
```

Gatillo de inserción individual

Paso 1: crea tu función

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

Paso 2: crea tu disparador

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

Paso 3: probarlo

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

Trigger para múltiples propósitos

Paso 1: crea tu función

```
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;
```

Paso 2: crea tu disparador

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

Paso 3: probarlo

```
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;
```

Lea Disparadores y funciones de disparador en línea:
https://riptutorial.com/es/postgresql/topic/6957/disparadores-y-funciones-de-disparador

Capítulo 13: Encontrar la longitud de la cadena / longitud del carácter

Introducción

Para obtener la longitud de los campos de "variación de caracteres", "texto", use char_length () o character_length ().

Examples

Ejemplo para obtener la longitud de un carácter que varía el campo

Ejemplo 1, Consulta: `SELECT char_length('ABCDE')`

Resultado:

    5

Ejemplo 2, Consulta: `SELECT character_length('ABCDE')`

Resultado:

    5

Lea Encontrar la longitud de la cadena / longitud del carácter en línea: https://riptutorial.com/es/postgresql/topic/9695/encontrar-la-longitud-de-la-cadena---longitud-del-caracter

Capítulo 14: Exportar el encabezado y los datos de la tabla de la base de datos PostgreSQL a un archivo CSV

Introducción

Desde la herramienta de administración Adminer, tiene opción de exportación a archivo csv para la base de datos mysql, pero no está disponible para la base de datos postgresql. Aquí mostraré el comando para exportar CSV para la base de datos postgresql.

Examples

Exporte la tabla PostgreSQL a csv con encabezado para algunas columnas

```
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;
```

Copia de seguridad completa de la tabla a CSV con encabezado

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

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

copia de consulta

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

Lea Exportar el encabezado y los datos de la tabla de la base de datos PostgreSQL a un archivo CSV en línea: https://riptutorial.com/es/postgresql/topic/8643/exportar-el-encabezado-y-los-datos-de-la-tabla-de-la-base-de-datos-postgresql-a-un-archivo-csv

Capítulo 15: Expresiones de tabla comunes (CON)

Examples

Expresiones de tablas comunes en consultas SELECT

Las expresiones de tabla comunes admiten la extracción de porciones de consultas más grandes. Por ejemplo:

```
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)
```

Atravesando árbol utilizando CON RECURSIVO

```
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;
```

Lea Expresiones de tabla comunes (CON) en línea:
https://riptutorial.com/es/postgresql/topic/1973/expresiones-de-tabla-comunes--con-

Capítulo 16: EXTENSION dblink y postgres_fdw

Sintaxis

- dblink ('dbname = name_db_distance port = PortOfDB host = HostOfDB usuario = usernameDB contraseña = passwordDB', 'MY QUESRY')
- dbname = nombre de la base de datos
- puerto = puerto de la base de datos
- host = host de la base de datos
- usuario = nombre de usuario de la base de datos
- contraseña = contraseña de la base de datos ',
- MI PREGUNTA = esto puede ser cualquier operación que quiera hacer SELECCIONAR, INSERTAR, ...

Examples

Extensión dblink

EXTENSIÓN de dblink es una técnica para conectar otra base de datos y hacer que esta base de datos funcione, para hacer lo que necesita:

1-Crear una extensión dblink:

```
CREATE EXTENSION dblink;
```

2-Haz tu operación:

Por ejemplo, seleccione algún atributo de otra tabla en otra base de datos:

```
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);
```

Extensión FDW

FDW es una implementación de dblink, es más útil, así que para usarlo:

1-Crear una extensión:

```
CREATE EXTENSION postgres_fdw;
```

2-Crear SERVIDOR:

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

3-Crear mapeo de usuario para servidor postgres

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

4-Crear tabla extranjera:

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

5-usa esta tabla externa como está en tu base de datos:

```
SELECT * FROM table_foreign;
```

Contenedor de datos extranjeros

Para acceder al esquema completo de la base de datos del servidor en lugar de una sola tabla. Siga los siguientes pasos:

1. Crear EXTENSIÓN:

```
CREATE EXTENSION postgres_fdw;
```

2. Crear SERVIDOR:

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

3. Crear mapeo de usuario:

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

4. Cree un nuevo esquema para acceder al esquema de la base de datos del servidor:

```
CREATE SCHEMA schema_name;
```

5. Importar esquema del servidor:

```
IMPORT FOREIGN SCHEMA schema_name_to_import_from_remote_db
```

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

6. Acceda a cualquier tabla de esquema de servidor:

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

Esto se puede usar para acceder a múltiples esquemas de bases de datos remotas.

Lea EXTENSION dblink y postgres_fdw en línea:
https://riptutorial.com/es/postgresql/topic/6970/extension-dblink-y-postgres-fdw

Capítulo 17: Fechas, sellos de tiempo e intervalos

Examples

Convertir una marca de tiempo o intervalo a una cadena

Puede convertir una `timestamp` o un valor de `interval` en una cadena con la función `to_char()` :

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

Esta declaración producirá la cadena "12 ago 2016 04:40:32 PM". La cadena de formato se puede modificar de muchas maneras diferentes; La lista completa de patrones de plantillas se puede encontrar aquí .

Tenga en cuenta que también puede insertar texto sin formato en la cadena de formato y puede usar los patrones de plantilla en cualquier orden:

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

Esto producirá la cadena "Hoy es sábado, el día 12 del mes de agosto de 2016". Sin embargo, debe tener en cuenta que cualquier patrón de plantilla, incluso los de una sola letra, como "I", "D", "W", se convierten, a menos que el texto simple esté entre comillas dobles. Como medida de seguridad, debe poner todo el texto sin formato entre comillas dobles, como se hizo anteriormente.

Puede localizar la cadena a su idioma de elección (nombres de día y mes) usando el modificador TM (modo de traducción). Esta opción utiliza la configuración de localización del servidor que ejecuta PostgreSQL o el cliente que se conecta a él.

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

Con una configuración regional española, esto produce "Sábado, 12 de agosto del año 2016".

SELECCIONA el último día del mes

Puede seleccionar el último día del mes.

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

`201608` es reemplazable con una variable.

Cuente el número de registros por semana

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

Lea Fechas, sellos de tiempo e intervalos en línea:
https://riptutorial.com/es/postgresql/topic/4227/fechas--sellos-de-tiempo-e-intervalos

Capítulo 18: Funciones agregadas

Examples

Estadísticas simples: min (), max (), avg ()

Para determinar algunas estadísticas simples de un valor en una columna de una tabla, puede usar una función agregada.

Si su tabla de `individuals` es:

Nombre	Años
Allie	17
Amanda	14
Alana	20

Podría escribir esta declaración para obtener el valor mínimo, máximo y promedio:

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

Resultado:

min	max	avg
14	20	17

string_agg (expresión, delimitador)

Puede concatenar cadenas separadas por delimitador utilizando la función `string_agg()` .

Si su tabla de `individuals` es:

Nombre	Años	País
Allie	15	Estados Unidos
Amanda	14	Estados Unidos
Alana	20	Rusia

Puede escribir la `SELECT ... GROUP BY` para obtener nombres de cada país:

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

Tenga en cuenta que necesita usar una cláusula `GROUP BY` porque `string_agg()` es una función agregada.

Resultado:

nombres	país
Allie amanda	Estados Unidos
Alana	Rusia

Más función agregada de PostgreSQL descrita aquí

regr_slope (Y, X): pendiente de la ecuación lineal de ajuste por mínimos cuadrados determinada por los pares (X, Y)

Para ilustrar cómo usar regr_slope (Y, X), lo apliqué a un problema del mundo real. En Java, si no limpia la memoria correctamente, la basura se puede atascar y llenar la memoria. Vuelca estadísticas cada hora sobre la utilización de la memoria de diferentes clases y la carga en una base de datos de Postgres para su análisis.

Todos los candidatos a pérdida de memoria tendrán una tendencia a consumir más memoria a medida que pase más tiempo. Si trazas esta tendencia, te imaginas una línea que va hacia arriba y hacia la izquierda:

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

Supongamos que tiene una tabla que contiene datos de histograma de volcado de pila (una asignación de clases a la cantidad de memoria que consumen):

```
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
);
```

Para calcular la pendiente de cada clase, agrupamos por sobre la clase. La cláusula HAVING> 0 garantiza que solo obtengamos candidatos con una pendiente positiva (una línea que va hacia arriba y hacia la izquierda). Clasificamos por la pendiente descendente para obtener las clases con la mayor tasa de aumento de memoria en la parte superior.

```
-- 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 ;
```

Salida:

```
         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
```

Desde la salida, vemos que el consumo de memoria de java.util.ArrayList está aumentando más rápidamente a 71.799 bytes por segundo y es potencialmente parte de la pérdida de memoria.

Lea Funciones agregadas en línea: https://riptutorial.com/es/postgresql/topic/4803/funciones-agregadas

Capítulo 19: Funciones criptográficas de Postgres.

Introducción

En Postgres, las funciones criptográficas se pueden desbloquear utilizando el módulo pgcrypto. CREAR EXTENSIÓN pgcrypto;

Examples

digerir

`DIGEST()` funciones `DIGEST()` generan un hash binario de los datos dados. Esto **se** puede utilizar para crear un hash aleatorio.

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

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

Ejemplos:

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

Lea Funciones criptográficas de Postgres. en línea: https://riptutorial.com/es/postgresql/topic/9230/funciones-criptograficas-de-postgres-

Capítulo 20: Funciones de ventana

Examples

ejemplo genérico

Preparando los datos:

```
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;
```

Corriendo:

```
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
;
```

Resultado:

```
 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)
```

Explicación:

dist_by_i : `dense_rank() over (order by i)` es como un número de fila por valores distintos. Puede usarse para el número de valores distintos de i (`count(DISTINCT i)` no funcionará). Solo usa el valor máximo.

prev_t : `lag(t) over ()` es un valor anterior de t en toda la ventana. importa que sea nulo para la

primera fila.

nth : `nth_value(i, 6) over ()` es el valor de la sexta fila columna i en toda la ventana

num_by_i : `count(true) over (partition by i)` es una cantidad de filas para cada valor de i

num_all : `count(true) over ()` es una cantidad de filas en una ventana completa

ntile : `ntile(3) over()` divide toda la ventana a 3 (la mayor cantidad posible) igual en cantidad de partes

valores de columna vs dense_rank vs rango vs row_number

Aquí puedes encontrar las funciones.

Con la tabla wf_example creada en el ejemplo anterior, ejecute:

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

El resultado es:

```
 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* ordena **VALORES** de **i** por aparición en ventana. `i=1` aparece, por lo que la primera fila tiene dense_rank, la siguiente y la tercera i el valor no cambia, por lo que es `dense_rank` muestra *1* : el valor FIRST no ha cambiado. la cuarta fila `i=2` , es el segundo valor de *i se* reunió, por lo que `dense_rank` muestra 2, y también para la siguiente fila. Luego cumple con el valor `i=3` en la 6ª fila, por lo que muestra 3. Lo mismo para el resto de los dos valores de *i* . Entonces, el último valor de `dense_rank` es el número de valores distintos de *i* .
- órdenes *ROW_NUMBER* **FILAS** en que se enumeran.
- *Rango Para* no confundirse con `dense_rank` esta función ordena el **NÚMERO DE FILA** de los valores **i** . Así que comienza igual con tres, pero tiene el siguiente valor 4, lo que significa que `i=2` (nuevo valor) se cumplió en la fila 4. Igual `i=3` se cumplió en la fila 6. Etc ..

Lea Funciones de ventana en línea: https://riptutorial.com/es/postgresql/topic/7421/funciones-de-ventana

Capítulo 21: Gestión de roles

Sintaxis

- `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

Crear un usuario con una contraseña.

En general, debe evitar usar la función de base de datos predeterminada (a menudo `postgres`) en su aplicación. En su lugar, debe crear un usuario con niveles más bajos de privilegios. Aquí hacemos uno llamado `niceusername` y le damos una contraseña `very-strong-password`

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

El problema con eso es que las consultas escritas en la consola `psql` se guardan en un archivo histórico `.psql_history` en el directorio de inicio del usuario y también pueden registrarse en el registro del servidor de base de datos PostgreSQL, exponiendo así la contraseña.

Para evitar esto, use el comando `\password` para establecer la contraseña del usuario. Si el usuario que emite el comando es un superusuario, no se solicitará la contraseña actual. (Debe ser superusuario para alterar contraseñas de superusuarios)

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

Crear rol y base de datos coincidentes

Para admitir una aplicación determinada, a menudo creas un nuevo rol y una base de datos para que coincida.

Los comandos de shell para ejecutar serían estos:

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

$ createdb -O blogger blogger
```

Esto supone que `pg_hba.conf` se ha configurado correctamente, lo que probablemente tenga este aspecto:

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

Otorgar y revocar privilegios.

Supongamos que tenemos tres usuarios:

1. El administrador de la base de datos> admin
2. La aplicación con acceso completo para sus datos> read_write
3. El acceso de solo lectura> read_only

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

Con las consultas anteriores, los usuarios que no son de confianza ya no pueden conectarse a la base de datos.

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

El siguiente conjunto de consultas revoca todos los privilegios de usuarios no autenticados y proporciona un conjunto limitado de privilegios para el usuario `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;
```

Modificar la ruta de búsqueda predeterminada del usuario

Con los siguientes comandos, se puede establecer la ruta de búsqueda predeterminada del usuario.

1. Compruebe la ruta de búsqueda antes de establecer el esquema predeterminado.

```
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. Configure `search_path` con `alter user` comando `alter user` para agregar un nuevo esquema `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. Verifique el resultado después de la ejecución.

```
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)
```

Alternativa:

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

Otorgar privilegios de acceso a objetos creados en el futuro.

Supongamos que tenemos `three users` :

1. El administrador de la base de datos> `admin`
2. La aplicación con acceso completo para sus datos> `read_write`
3. El acceso de solo lectura> `read_only`

Con las siguientes consultas, puede establecer privilegios de acceso en objetos creados en el futuro en un esquema específico.

```
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;
```

O bien, puede establecer privilegios de acceso en objetos creados en el futuro por un usuario específico.

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

Crear usuario de solo lectura

```
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;
```

Lea Gestión de roles en línea: https://riptutorial.com/es/postgresql/topic/1572/gestion-de-roles

Capítulo 22: Herencia

Observaciones

Una explicación de por qué querría usar la herencia en PostgreSQL está disponible aquí: http://stackoverflow.com/a/3075248/653378

Examples

Creando mesas infantiles

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

Nuestras tres mesas se ven así:

usuarios

Columna	Tipo
nombre de usuario	texto
correo electrónico	texto

simples_usuarios

Columna	Tipo
nombre de usuario	texto
correo electrónico	texto

users_with_password

Columna	Tipo
nombre de usuario	texto
correo electrónico	texto
contraseña	texto

Alterando mesas

Vamos a crear dos tablas simples:

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

Añadiendo columnas

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

simples_usuarios

Columna	Tipo
nombre de usuario	texto
correo electrónico	texto
contraseña	texto

Agregar la misma columna a la tabla principal combinará la definición de ambas columnas:

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

AVISO: fusionar la definición de la columna "contraseña" para el niño "simple_users"

Caída de columnas

Usando nuestras tablas alteradas:

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

usuarios

Columna	Tipo
nombre de usuario	texto
correo electrónico	texto

simples_usuarios

Columna	Tipo
nombre de usuario	texto
correo electrónico	texto
contraseña	texto

Desde que agregamos la columna a `simple_users` , PostgreSQL se asegura de que esta columna no se `simple_users` .

Ahora, si tuviéramos otra tabla secundaria, su columna de `password` , por supuesto, se habría eliminado.

Lea Herencia en línea: https://riptutorial.com/es/postgresql/topic/5429/herencia

Capítulo 23: INSERTAR

Examples

Básico INSERTAR

Digamos que tenemos una tabla simple llamada persona:

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

La inserción más básica consiste en insertar todos los valores en la tabla:

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

Si desea insertar solo columnas específicas, debe indicar explícitamente qué columnas:

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

Tenga en cuenta que si existe alguna restricción en la tabla, como NOT NULL, se le pedirá que incluya esas columnas en cualquier caso.

Insertando múltiples filas

Puede insertar varias filas en la base de datos al mismo tiempo:

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

Insertar desde seleccionar

Puede insertar datos en una tabla como resultado de una declaración de selección:

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

Tenga en cuenta que la proyección de la selección debe coincidir con las columnas necesarias para la inserción. En este caso, la tabla `tmp_person` tiene las mismas columnas que `person` .

Insertar datos utilizando COPY

COPY es el mecanismo de inserción masiva de PostgreSQL. Es una forma conveniente de

transferir datos entre archivos y tablas, pero también es mucho más rápido que `INSERT` cuando se agregan más de unos pocos miles de filas a la vez.

Vamos a empezar por crear un archivo de datos de muestra.

```
cat > samplet_data.csv

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

Y necesitamos una tabla de dos columnas en la que se pueden importar estos datos.

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

Ahora la operación de copia real, esto creará seis registros en la tabla.

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

En lugar de usar un archivo en el disco, puede insertar datos desde la `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
```

También puede copiar datos de una tabla a un archivo de la siguiente manera:

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

Para más detalles sobre COPY puede consultar aquí

Insertar datos y valores RETORNOS

Si está insertando datos en una tabla con una columna de incremento automático y si desea

obtener el valor de la columna de incremento automático.

Digamos que tienes una tabla llamada `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)
);
```

Si desea insertar datos en `my_table` y obtener el ID de esa fila:

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

La consulta anterior devolverá el ID de la fila donde se insertó el nuevo registro.

SELECCIONE los datos en el archivo.

Puede copiar la tabla y pegarla en un archivo.

```
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 - INSERTAR ... EN CONFLICTO ACTUALIZAR ...

desde la versión 9.5, postgres ofrece la funcionalidad `UPSERT` con la declaración `INSERT` .

Digamos que tienes una tabla llamada my_table, creada en varios ejemplos anteriores.
Insertamos una fila, devolviendo el valor PK de la fila insertada:

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

```
(1 row)

INSERT 0 1
```

Ahora, si intentamos insertar una fila con una clave única existente, se generará una excepción:

```
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.
```

La funcionalidad Upsert ofrece la capacidad de insertarlo de todos modos, resolviendo el conflicto:

```
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
```

Lea INSERTAR en línea: https://riptutorial.com/es/postgresql/topic/2561/insertar

Capítulo 24: JUNTARSE

Introducción

Coalesce devuelve el primer argumento ninguno nulo de un conjunto de argumentos. Solo se devuelve el primer argumento no nulo, se ignoran todos los argumentos subsiguientes. La función se evaluará como nula si todos los argumentos son nulos.

Examples

Solo argumento no nulo

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

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

Múltiples argumentos no nulos

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

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

Todos los argumentos nulos

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

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

Lea JUNTARSE en línea: https://riptutorial.com/es/postgresql/topic/10576/juntarse

Capítulo 25: Programación con PL / pgSQL

Observaciones

PL / pgSQL es el lenguaje de programación incorporado de PostgreSQL para escribir funciones que se ejecutan dentro de la base de datos, conocido como procedimientos almacenados en otras bases de datos. Extiende SQL con bucles, condicionales y tipos de retorno. Aunque su sintaxis puede ser extraña para muchos desarrolladores, es mucho más rápida que cualquier cosa que se ejecute en el servidor de aplicaciones porque se elimina la sobrecarga de conexión a la base de datos, lo cual es particularmente útil cuando de lo contrario necesitaría ejecutar una consulta, espere el resultado. y enviar otra consulta.

Aunque existen muchos otros lenguajes de procedimiento para PostgreSQL, como PL / Python, PL / Perl y PLV8, PL / pgSQL es un punto de partida común para los desarrolladores que desean escribir su primera función PostgreSQL porque su sintaxis se basa en SQL. También es similar a PL / SQL, el lenguaje de procedimiento nativo de Oracle, por lo que cualquier desarrollador familiarizado con PL / SQL encontrará el lenguaje familiar, y cualquier desarrollador que pretenda desarrollar aplicaciones de Oracle en el futuro pero que quiera comenzar con una base de datos gratuita puede hacer la transición. desde PL / pgSQL a PL / SQL con relativa facilidad.

Se debe enfatizar que existen otros lenguajes de procedimiento y PL / pgSQL no es necesariamente superior a ellos de ninguna manera, incluida la velocidad, pero los ejemplos en PL / pgSQL pueden servir como un punto de referencia común para otros lenguajes utilizados para escribir funciones de PostgreSQL. PL / pgSQL tiene la mayoría de los tutoriales y libros de todos los PL y puede ser un trampolín para aprender los idiomas con menos documentación.

Aquí hay enlaces a algunas guías y libros gratuitos sobre PL / pgSQL:

- La documentación oficial: https://www.postgresql.org/docs/current/static/plpgsql.html
- Tutorial de w3resource.com: http://www.w3resource.com/PostgreSQL/pl-pgsql-tutorial.php
- postgres.cz tutorial: http://postgres.cz/wiki/PL/pgSQL_(en)
- Programación del servidor PostgreSQL, 2ª edición: https://www.packtpub.com/big-data-and-business-intelligence/postgresql-server-programming-second-edition
- Guía del desarrollador de PostgreSQL: https://www.packtpub.com/big-data-and-business-intelligence/postgresql-developers-guide

Examples

Función PL / pgSQL básica

Una simple función PL / pgSQL:

```
CREATE FUNCTION active_subscribers() RETURNS bigint AS $$
DECLARE
    -- variable for the following BEGIN ... END block
    subscribers integer;
```

```
BEGIN
    -- SELECT must always be used with INTO
    SELECT COUNT(user_id) INTO subscribers FROM users WHERE subscribed;
    -- function result
    RETURN subscribers;
EXCEPTION
    -- return NULL if table "users" does not exist
    WHEN undefined_table
    THEN RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

Esto podría haberse logrado solo con la instrucción SQL, pero demuestra la estructura básica de una función.

Para ejecutar la función haz:

```
select active_subscribers();
```

Sintaxis de PL / pgSQL

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

Bloque de devoluciones

Opciones para regresar en una función PL / pgSQL:

- Lista de tipos de `Datatype` de todos los tipos de datos
- `Table(column_name column_type, ...)`
- `Setof 'Datatype' or 'table_column'`

excepciones personalizadas

creando la excepción personalizada 'P2222':

```
create or replace function s164() returns void as
$$
begin
raise exception using message = 'S 164', detail = 'D 164', hint = 'H 164', errcode = 'P2222';
end;
$$ language plpgsql
;
```

creando una excepción personalizada que no asigna errm:

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

vocación:

```
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
```

Aquí se procesa el P0001 personalizado y P2222, no, abortando la ejecución.

También tiene mucho sentido mantener una tabla de excepciones, como aquí: http://stackoverflow.com/a/2700312/5315974

Lea Programación con PL / pgSQL en línea: https://riptutorial.com/es/postgresql/topic/5299/programacion-con-pl---pgsql

Capítulo 26: Script de respaldo para un DB de producción

Sintaxis

- La secuencia de comandos le permite crear un directorio de respaldo para cada ejecución con la siguiente sintaxis: Nombre del directorio de respaldo de la base de datos + fecha y hora de ejecución
- Ejemplo: prodDir22-11-2016-19h55
- Una vez creado, crea dos archivos de copia de seguridad con la siguiente sintaxis: **Nombre de la base de datos + fecha y hora de ejecución**
- Ejemplo:
- dbprod22-11-2016-19h55.backup **(archivo de volcado)**
- dbprod22-11-2016-19h55.sql **(archivo sql)**
- Al final de una ejecución el **22-11-2016** @ **19h55** , obtenemos:
- /save_bd/prodDir22-11-2016-19h55/dbprod22-11-2016-19h55.backup
- /save_bd/prodDir22-11-2016-19h55/dbprod22-11-2016-19h55.sql

Parámetros

parámetro	detalles
save_db	El directorio principal de copia de seguridad.
dbProd	El directorio secundario de respaldo.
FECHA	La fecha de la copia de seguridad en el formato especificado.
dbprod	El nombre de la base de datos a guardar.
/opt/postgres/9.0/bin/pg_dump	El camino al binario pg_dump
-h	Especifica el nombre de host de la máquina en la que se está ejecutando el servidor, Ejemplo: localhost
-pag	Especifica el puerto TCP o la extensión de archivo de socket de dominio Unix local en el que el servidor está escuchando las conexiones, Ejemplo 5432
-U	Nombre de usuario para conectar como.

Observaciones

1. Si hay una herramienta de copia de seguridad como HDPS , o Symantec Backup , ... Es necesario vaciar el directorio de copia de seguridad **antes de cada inicio** .

Para evitar el desorden de la herramienta de copia de seguridad porque se supone que la copia de seguridad de los archivos antiguos se realiza.

Para habilitar esta característica, por favor descomente la línea N ° 3.

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

2. En el caso de que el presupuesto no permita tener una herramienta de respaldo, siempre se puede usar el planificador de tareas (comando cron).

   El siguiente comando se usa para editar la tabla cron para el usuario actual.

```
crontab -e
```

Programe el lanzamiento del script con el calendario a las 23:00.

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

Examples

saveProdDb.sh

En general, tendemos a hacer una copia de seguridad de la base de datos con el cliente pgAdmin. La siguiente es una secuencia de comandos sh utilizada para guardar la base de datos (en linux) en dos formatos:

- **Archivo SQL** : para un posible resumen de datos en cualquier versión de PostgreSQL.
- **Volcar archivo** : para una versión más alta que la versión actual.

```
#!/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
```

Lea Script de respaldo para un DB de producción en línea:

https://riptutorial.com/es/postgresql/topic/7974/script-de-respaldo-para-un-db-de-produccion

Capítulo 27: SELECCIONAR

Examples

SELECCIONAR utilizando DONDE

En este tema nos basaremos en esta tabla de usuarios:

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

Sintaxis

Selecciona cada cosa:

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

Seleccione algunos campos:

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

Ejemplos

```
-- 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;
```

Lea SELECCIONAR en línea: https://riptutorial.com/es/postgresql/topic/9528/seleccionar

Capítulo 28: Soporte JSON

Introducción

JSON: Java Script Object Notation, Postgresql admite el tipo de datos JSON desde la versión 9.2. Hay algunas funciones y operadores predefinidos para acceder a los datos JSON. El operador `->` devuelve la clave de la columna JSON. El operador `->>` devuelve el valor de la columna JSON.

Examples

Creando una tabla JSON pura

Para crear una tabla JSON pura, debe proporcionar un único campo con el tipo `JSONB` :

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

También debe crear un índice básico:

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

En este punto, puede insertar datos en la tabla y consultarlos de manera eficiente.

Consultar documentos JSON complejos.

Tomando un documento JSON complejo en una tabla:

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

Consulta por un elemento de nivel superior:

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

Consulta por un elemento simple en una matriz:

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

Consulta de un objeto en una matriz:

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

Consulta de un objeto anidado:

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

Rendimiento de `@>` comparado con `->` y `->>`

Es importante comprender la diferencia de rendimiento entre usar `@>` , `->` y `->>` en la parte `WHERE` de la consulta. Aunque estas dos consultas parecen ser ampliamente equivalentes:

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

la primera declaración utilizará el índice creado anteriormente, mientras que las dos últimas no lo harán, lo que requerirá una exploración completa de la tabla.

Todavía es posible usar el operador `->` al obtener datos resultantes, por lo que las siguientes consultas también usarán el índice:

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

Usando operadores JSONb

Creando un DB y una tabla

```
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
);
```

Poblando el DB

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

Veamos todo dentro de los libros de mesa:

```
SELECT * FROM books;
```

Salida:

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

-> operador devuelve valores fuera de columnas JSON

Seleccionando 1 columna:

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

Salida:

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

Seleccionando 2 columnas:

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

Salida:

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 ->>

El operador `->` devuelve el tipo JSON original (que podría ser un objeto), mientras que `->>` devuelve texto.

Devuelve objetos anidados

Puede usar `->` para devolver un objeto anidado y así encadenar los operadores:

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

Salida:

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

Filtración

Seleccione filas basadas en un valor dentro de su JSON:

```
 SELECT
 client,
```

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

Observe DÓNDE usa `->` así que debemos comparar con JSON `'"Dharma Bums"'`

O podríamos usar `->>` y comparar con `'Dharma Bums'`

Salida:

client character varying	title jsonb
Jenny	"Dharma Bums"

Filtrado anidado

Encuentre filas basadas en el valor de un objeto JSON anidado:

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

Salida:

client character varying	title jsonb
Jenny	"Dharma Bums"

Un ejemplo del mundo real.

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

Vamos a almacenar eventos en esta tabla, como páginas vistas. Cada evento tiene propiedades, que pueden ser cualquier cosa (por ejemplo, la página actual) y también envía información sobre el navegador (como el sistema operativo, la resolución de la pantalla, etc.). Ambos son completamente libres y podrían cambiar con el tiempo (ya que pensamos en cosas adicionales para rastrear).

```
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 } }'
);
```

Ahora vamos a seleccionar todo:

```
SELECT * FROM events;
```

Salida:

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

Operadores JSON + funciones agregadas de PostgreSQL

Usando los operadores JSON, combinados con las funciones agregadas tradicionales de PostgreSQL, podemos sacar lo que queramos. Tienes toda la potencia de un RDBMS a tu disposición.

- Permite ver el uso del navegador:

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

Salida:

browser text	count bigint
Firefox	4
Chrome	2

- Ingresos totales por visitante:

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

Salida:

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

- Resolución media de la pantalla

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

Salida:

width numeric	height numeric
1397.3333333333333333	894.666666666666667

Más ejemplos y documentación aquí y aquí .

Lea Soporte JSON en línea: https://riptutorial.com/es/postgresql/topic/1034/soporte-json

Capítulo 29: Tipos de datos

Introducción

PostgreSQL tiene un amplio conjunto de tipos de datos nativos disponibles para los usuarios. Los usuarios pueden agregar nuevos tipos a PostgreSQL usando el comando CREAR TIPO.

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

Examples

Tipos numericos

Nombre	Tamaño de almacenamiento	Descripción	Distancia
`smallint`	2 bytes	entero de rango pequeño	-32768 a +32767
`integer`	4 bytes	elección ypical para entero	-2147483648 a +2147483647
`bigint`	8 bytes	entero de gran rango	-9223372036854775808 a +9223372036854775807
`decimal`	variable	precisión especificada por el usuario, exacta	hasta 131072 dígitos antes del punto decimal; hasta 16383 dígitos después del punto decimal
`numeric`	variable	precisión especificada por el usuario, exacta	hasta 131072 dígitos antes del punto decimal; hasta 16383 dígitos después del punto decimal
`real`	4 bytes	precisión variable, inexacta	Precisión de 6 dígitos decimales
`double precision`	8 bytes	precisión variable, inexacta	Precisión de 15 dígitos decimales
`smallserial`	2 bytes	pequeño entero autoincremento	1 a 32767
`serial`	4 bytes	autoincremento entero	1 a 2147483647
`bigserial`	8 bytes	entero	1 a 9223372036854775807

Nombre	Tamaño de almacenamiento	Descripción	Distancia
		autoincremento grande	
`int4range`		Rango de enteros	
`int8range`		Gama de bigint	
`numrange`		Rango de numérico	

Tipos de fecha / hora

Nombre	Tamaño de almacenamiento	Descripción	Bajo valor	Alto valor	Resolución
`timestamp` (sin zona horaria)	8 bytes	fecha y hora (sin zona horaria)	4713 aC	294276 dC	1 microsegundo / 14 dígitos
`timestamp` (con zona horaria)	8 bytes	fecha y hora, con zona horaria	4713 aC	294276 dC	1 microsegundo / 14 dígitos
`date`	4 bytes	fecha (sin hora del día)	4713 aC	5874897 dC	1 día
`time` (sin zona horaria)	8 bytes	hora del día (sin fecha)	00:00:00	24:00:00	1 microsegundo / 14 dígitos
`time` (con zona horaria)	12 bytes	solo horas del día, con zona horaria	00: 00: 00 + 1459	24: 00: 00-1459	1 microsegundo / 14 dígitos
`interval`	16 bytes	intervalo de tiempo	-178000000 años	178000000 años	1 microsegundo / 14 dígitos
`tsrange`		rango de marca de tiempo sin zona horaria			
`tstzrange`		rango de marca de tiempo con zona horaria			

Nombre	Tamaño de almacenamiento	Descripción	Bajo valor	Alto valor	Resolución
`daterange`		rango de fecha			

Tipos geometricos

Nombre	Tamaño de almacenamiento	Descripción	Representación
`point`	16 bytes	Punto en un avion	(x, y)
`line`	32 bytes	Linea infinita	{A B C}
`lseg`	32 bytes	Segmento de línea finita	((x1, y1), (x2, y2))
`box`	32 bytes	Caja rectangular	((x1, y1), (x2, y2))
`path`	16 + 16n bytes	Ruta cerrada (similar a polígono)	((x1, y1), ...)
`path`	16 + 16n bytes	Camino abierto	[(x1, y1), ...]
`polygon`	40 + 16n bytes	Polígono (similar a la ruta cerrada)	((x1, y1), ...)
`circle`	24 bytes	Circulo	<(x, y), r> (punto central y radio)

Tipos de direcciones de red

Nombre	Tamaño de almacenamiento	Descripción
`cidr`	7 o 19 bytes	Redes IPv4 e IPv6
`inet`	7 o 19 bytes	IPv4 e IPv6 hosts y redes
`macaddr`	6 bytes	Direcciones MAC

Tipos de personajes

Nombre	Descripción
`character varying(n)` , `varchar(n)`	longitud variable con límite
`character(n)` , `char(n)`	longitud fija, acolchado en blanco

Nombre	Descripción
`text`	longitud ilimitada variable

Arrays

En PostgreSQL puede crear matrices de cualquier tipo integrado, definido por el usuario o enumeración. De forma predeterminada, no hay límite para una matriz, pero *puede* especificarla.

Declarando una matriz

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

Creando un Array

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

Accediendo a un Array

Por defecto, PostgreSQL usa una convención de numeración basada en uno para los arreglos, es decir, un arreglo de n elementos comienza con el `array[1]` y termina con el `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)
```

Obtener información sobre una matriz

```
--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)
```

Funciones de matriz

será añadido

Lea Tipos de datos en línea: https://riptutorial.com/es/postgresql/topic/8976/tipos-de-datos

Creditos

S. No	Capítulos	Contributors
1	Empezando con 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	Accediendo a los datos programáticamente	AstraSerg, brichins, greg, Laurenz Albe
3	Activadores de eventos	Ben H, Tajinder, Udlei Nati
4	ACTUALIZAR	frlan, leeor
5	Alta disponibilidad de PostgreSQL	gpdude_, Patrick
6	Comentarios en postgresql	Ben, KIRAN KUMAR MATAM
7	Conéctate a PostgreSQL desde Java	Laurenz Albe
8	Consejos y trucos de Postgres	Ben H, skj123, user_0, YCF_L
9	Consultas recursivas	Ben H
10	Copia de seguridad y restaurar	ankidaemon, Ben H, Daniel Lyons, e4c5, Laurel, mnoronha
11	Creación de tablas	e4c5, Jefferson, KIM, leeor, Patrick
12	Disparadores y funciones de disparador	chalitha geekiyanage, mnoronha, Udlei Nati
13	Encontrar la longitud de la cadena / longitud del carácter	Mohamed Navas
14	Exportar el	Vao Tsun, wOwhOw

	encabezado y los datos de la tabla de la base de datos PostgreSQL a un archivo CSV	
15	Expresiones de tabla comunes (CON)	Daniel Lyons, Jakub Fedyczak, Kevin Sylvestre
16	EXTENSION dblink y postgres_fdw	Riya Bansal, YCF_L
17	Fechas, sellos de tiempo e intervalos	KIM, Nuri Tasdemir, Patrick, Tom Gerken
18	Funciones agregadas	Alison S, joseph, Kirill Sokolov, Patrick
19	Funciones criptográficas de Postgres.	Ben H, skj123
20	Funciones de ventana	mnoronha, Vao Tsun
21	Gestión de roles	Ben, Ben H, bilelovitch, Blackus, Daniel Lyons, e4c5, greg, KIM, Laurenz Albe, mnoronha, Reboot
22	Herencia	evuez
23	INSERTAR	chalitha geekiyanage, e4c5, gpdude_, KIM, lamorach, leeor, Nathaniel Waisbrot, Patrick, Vao Tsun
24	JUNTARSE	Mokadillion
25	Programación con PL / pgSQL	AndrewCichocki, Ben H, Goerman, Laurenz Albe, Vao Tsun
26	Script de respaldo para un DB de producción	bilelovitch
27	SELECCIONAR	YCF_L
28	Soporte JSON	Clodoaldo Neto, commonSenseCode, jgm, KIRAN KUMAR MATAM, mnoronha, Peter Krauss
29	Tipos de datos	Ben H, user_0