無料電子ブック

学習

postgresql

Free unaffiliated eBook created from
Stack Overflow contributors.

#postgresql

1
1: postgresql 2
2
2
Examples 2
GNU + Linux 2
Red Hat 2
Debian 3
PostgreSQLMacPortsOSX 3
Postgres.app for Mac OSX 5
WindowsPostgreSQL 5
Macbrewpostgresql 7
LinuxPostgreSQL 8
2: JavaPostgreSQL 10
10
10
Examples 10
java.sql.DriverManager 10
java.sql.DriverManager 11
javax.sql.DataSource 12
3: JSON 13
13
Examples 13
JSON 13
JSON 13
@>->->> -> 14
JSONb 14
DB 14
DB 15
->JSON 15
->->> 16

NESTED 16
16
16
17
JSON+ PostgreSQL 18
4: PL / pgSQL 19
19
Examples 19
PL / pgSQL 19
PL / pgSQL 20
20
20
5: postgresql 22
22
22
22
Examples 22
22
22
6: PostgreSQLCSV 23
23
Examples 23
PostgreSQLcsv 23
csv 23
23
7: PostgreSQL 24
Examples 24
PostgreSQL 24
8: Postgres 27
Examples 27
PostgresDATEADD 27

27

postgres 27

Postresql2 27

2 27

1// 28

9: Postgres 29

..... 29

Examples 29

..... 29

10: 30

..... 30

..... 30

Examples 30

DDL 30

11: 31

Examples 31

INSERT 31

..... 31

..... 31

COPY 31

INSERTRETURING 32

..... 33

UPSERT - INSERT ... ON CONFLICT DO UPDATE 33

12: 35

Examples 35

..... 35

vs dense_rank vs rank vs row_number 36

13: 37

..... 37

Examples 37

..... 37

..... 37

37
14: 38
Examples 38
WHERESELECT 38
15: 39
39
Examples 39
39
/ 40
40
41
41
41
41
41
41
42
42
16: 43
Examples 43
43
43
43
44
44
17: 45
45
45
Examples 45
PL / pgSQL 45
46
46

46
46
46
1 46
2 47
3 47
47
1 47
2 48
3 48
18: 49
49
pg_dumpallpg_dump 49
Examples 49
1 49
49
50
50
CSV 50
50
51
51
o / p 51
51
SQL 51
51
psql 52
19: 53
Examples 53
Npgsql.NETPostgreSQL 53
C-APIPostgreSQL 54

54
54
psycopg2PythonPostgreSQL 57
Pomm2PHPPostgreSQL 57
20: DB 59
59
59
59
Examples 60
saveProdDb.sh 60
21: 61
61
Examples 61
61
61
62
search_path 62
63
63
22: WITH 65
Examples 65
SELECT 65
WITH RECURSIVE 65
23: 67
67
Examples 67
67
24: dblinkpostgres_fdw 68
68
Examples 68
dblink 68

FDW ... 68
... 69
25: / ... 71
... 71
Examples ... 71
... 71
26: ... 72
Examples ... 72
... 72
... 72
1 ... 72
27: ... 73
Examples ... 73
... 73
... 73
... 73
... 73
28: ... 74
... 74
Examples ... 74
... 74
... 74
... 74
users_with_password ... 74
... 75
... 75
... 75
... 75
... 75
... 76
29: ... 77

Examples....77
minmaxavg....77
string_agg....77
regr_slopeYXXY....78
....80

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

1: postgresqlをいめる

このでは、postgresqlのと、なぜがそれをいたいのかをします。

また、postgresqlのきなについてもし、するトピックにリンクするがあります。 postgresqlのドキュメントはしくなっているので、それらのトピックのバ―ジョンをするがあります。

バ―ジョン

バ―ジョン		EOLの
9.6	2016-09-29	2021-09-01
9.5	2016-01-07	2021-01-01
9.4	20141218	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

GNU + Linuxへのインストール

ほとんどのGNU + Linuxオペレ―ティングシステムでは、PostgreSQLはオペレ―ティングシステムのパッケ―ジマネ―ジャをしてにインストールできます。

Red Hatファミリ―

リポジトリはhttps://yum.postgresql.org/repopackages.phpにあります。

コマンドをしてリポジトリをロ―カルマシンにダウンロ―ドする

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

なパッケ―ジをする

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

Neccesaryパッケージはのとおりです。postgresqlXX postgresqlXX-server postgresqlXX-libs postgresqlXX-contrib

これらは、のコマンドでインストールされます。yum -y install postgresqlXX postgresqlXX-server postgresqlXX-libs postgresqlXX-contrib

インストールがしたら、サービスとしてデータベースサービスをするがありますデフォルトは postgresです。これはpg_ctlコマンドでいます。

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

CLIでDBにアクセスするには、 `psql`

Debianファミリー

Debianとしたオペレーティングシステムでは、のようにします。

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

これにより、PostgreSQLサーバパッケージが、オペレーティングシステムのパッケージリポジトリによってされるデフォルトバージョンでインストールされます。

でインストールされているバージョンがのバージョンでないは、パッケージマネージャをして、にされるのバージョンをすることができます。

PostgreSQLプロジェクト PGDG がするYumリポジトリをして、のバージョンをすることもできます。これにより、まだオペレーティングシステムパッケージリポジトリによってされていないバージョンがされるがあります。

PostgreSQLをMacPortsでOSXにインストールする

PostgreSQLをOSXにインストールするには、サポートされているバージョンをるがあります。

なバージョンをするには、このコマンドをします。

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

のようなリストがされます。

```
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
```

このでは9.6でサポートされているPostgreSQLのバージョンですので、それをインストールします。

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

のようなインストールログがされます。

```
--->  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.
```

このログには、インストールのりのについてのがされていますので、にそれをいます。

```
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'
```

ここでサーバーをします。

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

サーバーにできることをします。

```
su postgres -c psql
```

postgresからのプロンプトがされます。

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

postgres=#
```

ここでは、サーバーがであることをするためにをできます。

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

そして、そのをてください

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

\ qとしてします。

```
postgres=#\q
```

そしてあなたはあなたのシェルプロンプトにります。

おめでとう OS / XでPostgreSQLインスタンスがしています。

Postgres.app for Mac OSX

PostgreSQLをMacにインストーールするためのになツーールは、 Postgres.appをダウンロードすることでできます。
PostgreSQLをバックグラウンドでさせるようにプリファレンスをすることも、アプリケーションがされているにのみプリファレンスをすることもできます。

WindowsへのPostgreSQLのインストール

サーバーとしてUnixベースのオペレーティングシステムLinuxやBSDなどをすることをおめしますが、PostgreSQLをWindowsににインストールできますうまくいけばサーバーとしてのみ。

Windowsインストーラバイナリを EnterpriseDB からダウンロードしてください http://www.enterprisedb.com/products-services-training/pgdownload これは、Windowsのバイナリをしたプロジェクトのなによってめられたサードパーティのです。

のベータでは9.5.3をしてください。ほとんどの、Win x86-64パッケージがですが、いコンピュータではなWindowsの32ビットバージョンをしているは、わりにWin x86-32をします。

BetaとStableのりえには、ダンプやなどのながになります。ベータまたはでアップグレードするには、サービスをするがあります。

コントロールパネル -> システムとセキュリティ -> システム -> システムタイプの "## - bit Operating System"にくと、Windowsのバージョンが32または64ビットかどうかをできます。これはWindows 7のパスですが、のバージョンのWindowsではなるがあります。

インストーラで、するパッケージをします。えば

- pgAdmin https://www.pgadmin.org はあなたのデータベースをするためののGUIです。はそれをくします。 9.6では、これはデフォルトでインストールされます。
- PostGIS http://postgis.net は、GISのでにしているGPS、などにするをします。
- パッケージは、にサポートされているきPL / Python、PL / Perl、PL / Tclになライブラリをします。
- pgAgent、pgBouncer、Slonyのようなのパッケージは、よりなサーバーにはです。にしてチェックするだけです。

これらのオプションパッケージは、で "Application Stack Builder"をしてインストールすることができます。

PL / V8 、 PL / Lua PL / Javaなどのにサポートされていないもあります。

pgAdminをき、そのをダブルクリックしてサーバーにします。 "PostgreSQL 9.5localhost5432。

このから、なのPostgreSQLUp and Running、2 http://shop.oreilly.com/product/0636920032144.do などのガイドにうことができます。

オプションサービスタイプ

PostgreSQLは、ほとんどのプログラムとはしなるバックグラウンドでサービスとしてされます。これは、データベースとWebサーバーでです。そのデフォルトのスタートアップのはです。つまり、あなたからのなしでにされます。

なぜPostgreSQLサービスをでしたいのですか PCをサーバーとしてしているや、ビデオゲームなどにする、PostegreSQLをすると、のシステムがしくなるがあります。

なぜでコントロールしたくないのですかサービスのとは、にうとです。

スピードのいにかず、をけたいは、「スタートアップの」を「」のままにして、このガイドのりのはしてください。さもないと...

コントロールパネル - >システムとセキュリティ - >ツールにします。

リストから「Services」をし、そのアイコンをクリックし、Send To - > Desktopをして、よりなアクセスのためのデスクトップアイコンをします。

[ツール]ウィンドウをじ、にしたデスクトップアイコンから[サービス]をします。

postgresql-x ## - 9.postgresql-x64-9.5のようなのサービスがされるまでにスクロールします。

postgresサービスをクリックし、[プロパティ] - > [スタートアップの] - > [] - > [] - > [OK]をします。あなたはそれをににしすことができます。

"pgbouncer"や "PostgreSQL Scheduling Agent - pgAgent"などのPostgreSQLサービスがされているは、PostgreSQLがされていないでもStartup TypeをManualにすることができます。これはあなたがし、するたびにくのをするが、それはあなたである。 PostgreSQLのリソースはそれほどくわれておらず、システムのパフォーマンスにはほとんどしません。

サービスがされている、StatusはStartedとされ、そうでないはされていません。

するには、クリックして[]をします。みみプロンプトがされ、すぐにえるはずです。それはあなたにエラーをえるはもうしてください。それがうまくいかないは、ほとんどのがしないWindowsのをしたことがで、インストールにがしていたがあります。

postgresをするには、サービスをクリックし、[]をします。

データベースにしようとしているときにエラーがしたは、そのサービスがしていることをしてください。

のPostgreSQLバージョンのパックにあるPythonランタイムバージョンなどのEDB PostgreSQLインストールにするそののについては、のEBDインストールガイドをしてください。インストーラのメジャーバージョンにリンクしてバージョンをしてください。

Macにbrewをインストールしてpostgresqlをインストールする

Homebrewが'*macOS*のパッケージマネージャーがない'とんでいます。これは、アプリケーションとライブラリをビルドしてインストールするためにできます。 インストールがすると、 `brew`コマンドをしてPostgreSQLをインストールすることができ、そのはのようになります。

```
brew update
brew install postgresql
```

Homebrewはにのバージョンをインストールします。のものがなは、 `brew search postgresql`はなバージョンをします。のオプションでビルドされたPostgreSQLがなは、 `brew info postgresql`はどのオプションがサポートされているかをします。サポートされていないビルドオプションがなは、ビルドをでうがありますが、きHomebrewをしてのをインストールすることができます。

サーバーをします。

```
brew services start postgresql
```

PostgreSQLのプロンプトをきます

```
psql
```

psqlがあなたのユーザにするデータベースがないとうと、 `createdb`します。

Linuxのソースから**PostgreSQL**をインストールする

- GNU Makeバージョン> 3.80
- ISO / ANSI Cコンパイラgccなど
- tarやgzipのようなプログラム
- zlib-devel
- readline-devel oder libedit-devel

ソース ソースへのリンク9.6.3

これで、ソースファイルをすることができます

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

PostgreSQLのには、さまざまなオプションがあります。

なインストールへのなリンク

なオプションのさなリスト

- `--prefix=PATH`すべてのファイルのパス
- `--exec-prefix=PATH` architectur-dependetファイルのパス
- `--bindir=PATH`プログラムのパス
- `--sysconfdir=PATH`ファイルの`--sysconfdir=PATH`パス
- `--with-pgport=NUMBER`サーバーのポートをする
- `--with-perl` add perlサポート
- `--with-python` pythonのサポートをする
- `--with-openssl`はopensslサポートをします
- `--with-ldap` add ldapサポート
- `--with-blocksize=BLOCKSIZE`はKB `--with-blocksize=BLOCKSIZE`し`--with-blocksize=BLOCKSIZE`
  - `BLOCKSIZE`は2ので、 `BLOCKSIZE`なければなりません
- `--with-wal-segsize=SEGSIZE`サイズをWALに - セグメントサイズMB
  - `SEGSIZE`は1から64のの2の`SEGSIZE`なければなりません

しくしたフォルダにし、なオプションをしてcofigureスクリプトをします。

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

`make`をしてオブジェクトファイルをする

`make install`をしてビルドされたファイルからPostgreSQLをインストールする

`make clean`に`make clean`にきれいにする

スイッチの、 `cd contrib`ディレクトリに`make`を`make` 、 `make install`

オンラインでpostgresqlをいめるをむ https://riptutorial.com/ja/postgresql/topic/885/postgresqlをいめる

2: JavaからPostgreSQLにする

き

JavaからのリレーショナルデータベースをするapiはJDBCです。

このapiはJDBCドライバによってされています。

それをするには、ドライバをむJARファイルをJAVAクラスパスにします。

このドキュメントでは、JDBCドライバをしてデータベースにするのサンプルをします。

JDBC URL

JDBC URLは、のいずれかのをることができます。

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

  `host`デフォルトは`localhost`、 `port`は5432です。
  `host`がIPv6アドレスのは、でむがあります。
  デフォルトのデータベースは、するユーザーのとじです。

  フェールオーバーをするには、の`host [: port ]`エントリをカンマでって`host [: port ]`することができます。
  がするまでにされます。

- `jdbc:postgresql: database [ parameters ]`

- `jdbc:postgresql:/[ parameters ]`

  これらのは、 `localhost`へのです。

`parameters`は`key [= value ]`ペアのリストで、は`? &`でられています。 `value`がないは`true`とみなされ`true`。

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

- JDBC http : //download.oracle.com/otndocs/jcp/jdbc-4_2-mrel2-eval-spec/
- PostgreSQL JDBCドライバ https : //jdbc.postgresql.org/
- PostgreSQL JDBCドライバのドキュメント https : //jdbc.postgresql.org/documentation/head/index.html

Examples

java.sql.DriverManagerをした

これはするもなです。

まず、するクラスをるためにドライバを`java.sql.DriverManager`にするがあります。
これは、ドライバクラスをは`java.lang.Class.forname( <driver class name> )`ロードすることによってわれ`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);
}
```

そのユーザーとパスワードをJDBC URLにめることはできません。その、`getConnection`メソッドびしでユーザーとパスワードをするはありません。

`java.sql.DriverManager`とのとプロパティ

URLやのパラメータにuserやpasswordなどのパラメータなリストを をするわりに、それらを`java.util.Properties`オブジェクトにパックすることができます。

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

プールをしたjavax.sql.DataSourceとの

アプリケーションサーバーコンテナでJNDIとともに`javax.sql.DataSource`をするのがです。データソースをでし、がなときはいつでもすることができます。

これは、データソースのみをすコードです。

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

このをびしてデータソースをしたら、のようにします。

```
/* 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();
```

オンラインでJavaからPostgreSQLにするをむ https://riptutorial.com/ja/postgresql/topic/9633/javaからpostgresqlにする

3: JSONのサポート

き

JSON - Java Script Object Notation、PostgreSQLはJSON 9.2のデータをサポートしています。JSONデータにアクセスするためのいくつかのされたとがあります。 `->`は、JSONのキーをします。 `->>`は、JSON Columnのをします。

Examples

なJSONテーブルをする

なJSONテーブルをするには、 `JSONB`ののフィールドをするがあります。

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

インデックスもするがあります。

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

こので、テーブルにデータをしてにクエリをできます。

なJSONドキュメントのクエリ

テーブルでなJSONドキュメントをする

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

トップレベルのクエリ

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

のなをする

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

のオブジェクトをする

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

ネストされたオブジェクトのクエリ

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

@>パフォーマンスは->と->> ->として

クエリの`WHERE`で`@>` 、 `->`および`->>`をするのパフォーマンスのいをすることがです。これらの2つのクエリはほぼじですが、

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

のステートメントはでしたインデックスをしますが、のステートメントはなテーブルスキャンをとしません。

のデータをするときに`->`をすることはですので、のクエリもインデックスをします

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

JSONbの

DBとテーブルの

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

DROP TABLE IF EXISTS books;

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

DBへのデータ

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

テーブルブックののすべてをてみましょう

```
SELECT * FROM books;
```

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

->はJSONからをします

1をする

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

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

2の

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

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"}

->->>

`->`はのJSONオブジェクトでもよいをしますが、`->>`はテキストをします。

NESTEDオブジェクトをす

`->`をすると、ネストしたオブジェクトをすことができ、をできます。

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

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

フィルタリング

JSONのについてをします。

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

JSON `'"Dharma Bums"'`とするがあるので、どこで`->`しているかをしてください

あるいは、`->>`をって`'Dharma Bums'`とすることもできます

client character varying	title jsonb
Jenny	"Dharma Bums"

ネストされたフィルタリング

れになったJSONオブジェクトのについてをする

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

client character varying	title jsonb
Jenny	"Dharma Bums"

の

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

このには、ページビューなどのイベントをします。イベントにはプロパティのページなどがあり、ブラウザにするOS、などもされます。これらはともにフリーフォームであり、のとにするがありますするなものをえると。

```
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 } }'
);
```

すぐすべてをしてください

```
SELECT * FROM events;
```

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

JSON+ PostgreSQL

のPostgreSQLとみわされたJSONをして、なものをきすことができます。あなたはにRDBMSをにれることができます。

- ブラウザのをてみましょう

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

browser text	count bigint
Firefox	4
Chrome	2

- ビジターあたりの

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

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

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

width numeric	height numeric
1397.3333333333333333	894.666666666666667

よりくのとドキュメントはこことここにあります 。

オンラインでJSONのサポートをむ https://riptutorial.com/ja/postgresql/topic/1034/jsonのサポート

4: PL / pgSQLによるプログラミング

PL / pgSQLは、のデータベースのストアドプロシージャとばれるデータベースでされるをくためのPostgreSQLのみみプログラミングです。これは、ループ、、およびりのでSQLをします。くのにとってがなことはありますが、データベースへののオーバーヘッドがなくなるため、アプリケーションサーバーでされているものよりもはるかにです。クエリをしてをつがある、のクエリをします。

PL / Python、PL / Perl、PLV8など、PostgreSQLにはのくのきがしますが、そのがSQLにされているため、のPostgreSQLをするにとってのです。また、PL / SQLにしているであれば、そのにしていることがわかります。、Oracleアプリケーションをしようとしているが、フリー・データベースからしたいは、 PL / pgSQLからPL / SQLへのがです。

のきがし、PL / pgSQLがスピードをめていかなるでもれているわけではないことをするがありますが、PL / pgSQLのはPostgreSQLをくためにされるののとしてちます。 PL / pgSQLには、すべてのPLのチュートリアルとがあります。

PL / pgSQLにするいくつかのガイドとへのリンクがあります

- ドキュメント https : //www.postgresql.org/docs/current/static/plpgsql.html
- w3resource.comチュートリアル http : //www.w3resource.com/PostgreSQL/pl-pgsql-tutorial.php
- postgres.czチュートリアル http : //postgres.cz/wiki/PL/pgSQL_(en
- PostgreSQLサーバプログラミング、2 https : //www.packtpub.com/big-data-and-business-intelligence/postgresql-server-programming-secondededition
- PostgreSQLガイド https : //www.packtpub.com/big-data-and-business-intelligence/postgresql-developers-guide

Examples

なPL / pgSQL

な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;
```

これは、SQLだけでできたものの、のをしています。

このをするには

```
select active_subscribers();
```

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;
```

ブロックをす

PL / pgSQLですためのオプション

- `Datatype` すべてのデータ `Datatype` リスト
- `Table(column_name column_type, ...)`
- `Setof 'Datatype' or 'table_column'`

カスタム

カスタム「P2222」をします。

```
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
;
```

errmをりてないカスタムをする

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

```
;
```

びし

```
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
```

ここでカスタムP0001がされ、P2222がされず、がされます。

また、ここでのようなのテーブルをすることはにがあります http : //stackoverflow.com/a/2700312/5315974

オンラインでPL / pgSQLによるプログラミングをむ https://riptutorial.com/ja/postgresql/topic/5299/pl---pgsqlによるプログラミング

5: postgresqlのコメント

き

COMMMENTは、デ―タベ―ス･オブジェクトにするコメントをまたはすることです。

のデ―タベ―スオブジェクトにしてコメントを1つだけえることができます.COMMENTは、のデ―タベ―スオブジェクトがのとはかをするのにちます。

COMMENT ON ROLEのルールは、ス―パ―ユ―ザ―にコメントするにはス―パ―ユ―ザ―でなければならず、ス―パ―ユ―ザ―のについてコメントする***CREATEROLE***をつがあります。もちろん、 ス―パ―ユ―ザ―はでもコメントできます

- COMMENT ON database_object object_name IS 'テキスト';

なはhttp://www.postgresql.org/docs/current/static/sql-comment.htmlをしてください。

Examples

のコメント

COMMENT ON TABLE table_name IS 'これはのです';

コメントを

TABLEのコメントはNULLです。

コメントはののとともにされます。

オンラインでpostgresqlのコメントをむ https://riptutorial.com/ja/postgresql/topic/8191/postgresqlのコメント

6: PostgreSQLのデータベーステーブルヘッダとデータをCSVファイルにエクスポートする

き

ツールからは、mysqlデータベースのcsvファイルオプションにエクスポートできますが、postgresqlデータベースではできません。ここでは、postgresqlデータベースのCSVをエクスポートするコマンドをします。

Examples

PostgreSQLのテーブルをいくつかのカラムのヘッダとともにcsvにエクスポートする

```
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;
```

ヘッダーきのcsvへのフルテーブルバックアップ

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

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

クエリからコピーする

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

オンラインでPostgreSQLのデータベーステーブルヘッダとデータをCSVファイルにエクスポートするをむ https://riptutorial.com/ja/postgresql/topic/8643/postgresqlのデータベーステーブルヘッダとデータをcsvファイルにエクスポートする

7: PostgreSQLの

Examples

PostgreSQLのレプリケーション

- プライマリサーバの
  - レプリケーション・アクティビティーのレプリケーション・ユーザー
  - WALアーカイブをするディレクトリ
- レプリケーションユーザーの

```
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
```

- データディレクトリにアーカイブディレクトリをする

```
mkdir $PGDATA/archive
```

- **pg_hba.conf**ファイルをする

これはホストベースのファイルで、クライアントのautherntication のがまれています。のエントリを

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

- **postgresql.conf**ファイルをする

これはPostgreSQLのファイルです。

```
wal_level = hot_standby
```

このパラメータは、スレーブサーバのをします。

```
`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
```

このパラメータにより、 `archive_command`パラメータをしてWALセグメントをアーカイブにできます。

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

に`archive_command`は、WALセグメントをア―カイブディレクトリにコピ―することです。

`wal_senders = 5`これは、WALプロセスのです。

プライマリサ―バをします。

- プライマリ―・サ―バ―をスレ―ブ・サ―バ―にバックアップする

  サ―バ―をするに、プライマリサ―バ―をしてください。

  すべてのとバックアップのがするまで、サ―ビスをしないでください。スタンバイ・サ―バ―は、バックアップ・サ―バ―のがったでちげるがあります。つまり、すべてのをし、デ―タベ―スをすでにしておくがあります。そうでなければ、ストリ―ミングのはできません

- すぐ**pg_basebackup**ユ―ティリティをしてください

  `pg_basebackup`ユ―ティリティは、プライマリサ―バのデ―タディレクトリからスレ―ブのデ―タディレクトリにデ―タをコピ―します。

  ```
  $ 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.
```

- スタンバイサ―バの

  スタンバイサ―バをするには、postgresql.confをして、recovery.confというのしいファイルをします。

  `hot_standby = on`

  これは、にをできるかどうかをします

  - **recovery.conf**ファイルの

    `standby_mode = on`

    プライマリサ―バ―にをします。プライマリサ―バのIPアドレスにきえてください。レプリケ―ションというのユ―ザ―のパスワ―ドできえます

    `primary_conninfo = 'host = port = 5432 user =パスワ―ド='

オプショントリガーファイルのをします。

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

する`trigger_file`パスは、システムをスタンバイ・サーバーにフェールオーバーするときにファイルをできるです。ファイルがすると、フェールオーバーがトリガーされます。または、pg_ctl promoteコマンドをしてフェールオーバーをトリガすることもできます。

- スタンバイサーバーをする

これで、すべてのがい、スタンバイサーバーをするがいました

アトリビューション

このはにPostgreSQLをとホットスタンバイでするためのと 、フォーマットとサンプルのの、およびのテキストのによるものです。ソースは、ここでされているCreative Commons Public License 3.0のでされました。

オンラインでPostgreSQLのをむ https://riptutorial.com/ja/postgresql/topic/5478/postgresqlの

8: Postgresのヒントとテクニック

Examples

PostgresのDATEADDの

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

のカンマり

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

postgresテーブルからレコードをする

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

Postresqlはクエリのをサポートしていないため、2つのテーブルのでクエリをします。

```
    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>
```

2つのタイムスタンプのの

2つののののタイムスタンプ

```
select
```

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

2つのタイムスタンプ

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

1つのデータベースからじスキーマをつのデータベーステーブルにテーブルデータをコピー//するクエリ

の

```
CREATE EXTENSION DBLINK;
```

その、

```
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>
    );
```

オンラインでPostgresのヒントとテクニックをむ
https://riptutorial.com/ja/postgresql/topic/7433/postgresのヒントとテクニック

9: Postgresの

き

Postgresでは、pgcryptoモジュールをってをロックすることができます。 CREATE EXTENSION pgcrypto;

Examples

ダイジェスト

`DIGEST()`は、えられたデータのバイナリハッシュをします。これは 、ランダムなハッシュをするためにできます。

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

または `digest(data bytea, type text) returns bytea`

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

オンラインでPostgresのをむ https://riptutorial.com/ja/postgresql/topic/9230/postgresの

10: イベントトリガー

き

ィベントトリガーは、けられたィベントがデータベースするたびにします。

PostgreSQLのイベントトリガのなについては、のリンクをしてください

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

Examples

DDLコマンドのイベントのロギング

ィベントタイプ-

- DDL_COMMAND_START
- DDL_COMMAND_END
- SQL_DROP

これは、ィベント・トリガーをし、 DDL_COMMAND_STARTイベントをロギングするためのです。

```
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();
```

オンラインでイベントトリガーをむ https://riptutorial.com/ja/postgresql/topic/9255/イベントトリガー

11: インサート

Examples

INSERT

personというなテーブルがあるとしましょう

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

もなは、テーブルにすべてのをすることです。

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

ののみをするは、にをするがあります。

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

NOT NULLのようながテーブルにするは、どちらのでもこれらのをめるがあることにしてください。

のをする

にのをデータベースにすることができます。

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

から

selectのとしてにデータをできます。

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

selectのは、になとするがあることにしてください。この、`tmp_person`は`person`とじをちます。

COPYをしてデータをする

COPYはPostgreSQLのメカニズムです。これは、ファイルとテーブルでデータをするなですが、

にをえるをするは、 `INSERT`よりもはるかにです。

サンプルデ―タファイルをしましょう。

```
cat > samplet_data.csv

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

そして、このデ―タをインポ―トできる2つのカラムテ―ブルがです。

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

のコピ―では、テ―ブルに6つのレコ―ドがされます。

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

ディスクのファイルをするわりに、 `stdin`からデ―タをすることができます

```
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
```

また、のようにテ―ブルからファイルにデ―タをコピ―することもできます

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

COPYのについてはこちらをごください

INSERTデ―タとRETURING

をつにデ―タをする、およびのをするは、のようにします。

`my_table`というテーブルがあるとし`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)
);
```

`my_table`データをし、そののIDをするは、のようにします。

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

のクエリは、しいレコードがされたのIDをします。

データをファイルにします。

テーブルをCOPYしてファイルにりけることができます。

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

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

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

バージョン9.5の postgresは`INSERT`で`UPSERT`をしているからです。

のいくつかのでされたmy_tableというテーブルがあるとします。をし、されたのPKをします。

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

```
INSERT 0 1
```

のキーをつをしようとすると、がします。

```
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.
```

Upsertは、とにかくそれをするをし、をします

```
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
```

オンラインでインサートをむ https://riptutorial.com/ja/postgresql/topic/2561/インサート

12: ウィンドウ

Examples

な

データの

```
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;
```

ランニング

```
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
;
```

```
 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)
```

dist_by_i `dense_rank() over (order by i)`は、なるごとのrow_numberにています。 i `count(DISTINCT i)` wold not workのなるのにできます。 をしてください。

prev_t `lag(t) over ()`は、ウィンドウののtのです。 のではnullであることにしてください。

n `nth_value(i, 6) over ()`ウィンドウにわたってiのであります

num_by_i `count(true) over (partition by i)`は、iのにするのです

num_all `count(true) over ()`は、ウィンドウにわたるのです

ntile `ntile(3) over()`は、ウィンドウを3なりのにします

のvs dense_rank vs rank vs row_number

ここでをつけることができます。

のでされたテーブルwf_exampleをして、をします。

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

はのとおりです。

```
 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*は、ウィンドウのによってiの をします。 `i=1`がされるので、のはdense_rankをち、と3のiはしないので、 `dense_rank`は*1* - FIRSTがされていないことをします。4`i=2` 、それは*i*の2のなので、 `dense_rank`は2をし、のについてもします。それはたし`i=3` 6では、それはのりのつの3.じします。したがって、 `dense_rank`ののは、 *i*ののののです。
- *row_number*は、リストされているROWSをべえます。
- *rank* `dense_rank`としないで`dense_rank`このは、 iのROW NUMBERをします。それで3つのものとじようにまりますが、の4があります。これは、4で`i=2` しいがたされたことをします。じ`i=3`が6でたされました。

オンラインでウィンドウをむ https://riptutorial.com/ja/postgresql/topic/7421/ウィンドウ

13: ゴールデン

き

Coalesceは、のセットからのnone nullをします。のヌルだけがreturnであり、そののすべてのはされます。すべてのがnullの、はnullにされます。

Examples

のヌル

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

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

のヌル

PGSQL> SELECT COALESCENULL、NULL、 'のNULLでない'、null、NULL、 '2のNULL';

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

すべてのヌル

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

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

オンラインでゴールデンをむ https://riptutorial.com/ja/postgresql/topic/10576/ゴールデン

14: セレクト

Examples

WHEREをしたSELECT

このトピックでは、のについてします。

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

すべてのものをしてください

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

いくつかのフィールドをしてください

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

```
-- 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;
```

オンラインでセレクトをむ https://riptutorial.com/ja/postgresql/topic/9528/セレクト

15: データ

き

PostgreSQLには、ユーザができるなネイティブデータがあります。ユーザーは、CREATE TYPE コマンドをしてPostgreSQLにしいタイプをできます。

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

Examples

タイプ

	ストレージサイズ		
smallint	2バイト		-32768+32767
integer	4バイト	のためのypical の	-2147483648+2147483647
bigint	8バイト		-9223372036854775808 +9223372036854775807
decimal		ユーザーの、	のに131072。16383まで
numeric		ユーザーの、	のに131072。16383まで
real	4バイト	、	10の6の
double precision	8バイト	、	15の
smallserial	2バイト	さなインクリメント	132767
serial	4バイト	インクリメント	12147483647
bigserial	8バイト	きなインクリメント	19223372036854775807
int4range		の	
int8range		のbigint	
numrange		の	

/タイプ

	ストレージサイズ		い	い	
`timestamp` タイムゾーンなし	8バイト	とのタイムゾーンなし	4713	294276 AD	1マイクロ/ 14
`timestamp` タイムゾーンき	8バイト	との、タイムゾーンき	4713	294276 AD	1マイクロ/ 14
`date`	4バイト	なし	4713	5874897 AD	1
`time` タイムゾーンなし	8バイト	なし	00:00:00	24:00:00	1マイクロ/ 14
`time` あり	12バイト	のみ	000000 + 1459	240000-1459	1マイクロ/ 14
`interval`	16バイト		-17800	17800	1マイクロ/ 14
`tsrange`		タイムゾーンなしのタイムスタンプの			
`tstzrange`		タイムゾーンきのタイムスタンプの			
`daterange`		の			

タイプ

	ストレージサイズ		
`point`	16バイト	の	x、y
`line`	32バイト	ライン	{A、B、C}
`lseg`	32バイト	セグメント	x1、y1、x2、y2
`box`	32バイト	のボックス	x1、y1、x2、y2
`path`	16 + 16nバイト	ポリゴンにている	x1、y1、...
`path`	16 + 16nバイト	オープンパス	[x1、y1、...]

	ストレージサイズ		
`polygon`	40 + 16nバイト	ポリゴンじたパスに	x1、y1、...
`circle`	24バイト	サークル	<x、y、r>と

ネットワークアドレスタイプ

	ストレージサイズ	
`cidr`	7または19バイト	IPv4およびIPv6ネットワーク
`inet`	7または19バイト	IPv4とIPv6のホストとネットワーク
`macaddr`	6バイト	MACアドレス

の

`character varying(n)` 、 `varchar(n)`	
`character(n)` 、 `char(n)`	、ブランク
`text`	

PostgreSQLでは、みみ、ユーザ、またはのをできます。デフォルトでは、にははありませんが、することはできます。

の

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

の

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

へのアクセス

デフォルトでは、PostgreSQLはに`array[1]`からまるけをいます。つまり、nのは`array[1]`でまり、`array[n]`わります。

```
--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)
```

にするの

```
--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)
```

されます

オンラインでデータをむ https://riptutorial.com/ja/postgresql/topic/8976/データ

16: テーブル

Examples

キーによるテーブルの

```
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)
);
```

あるいは、 `PRIMARY KEY`をにすることもできます。

```
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)
);
```

およびすべてのにのをすることをおめします。 `Person`などののをするは、PostgreSQLがとをするため、せでそのを `"Person"` でむがあります。

テーブルをする

あなたのテーブルがあるデータベースにされた`psql`コマンドラインツールをきます。に、のコマンドをします。

```
\d tablename
```

タイプをするには

```
\d+ tablename
```

テーブルのをれたは、\ dをpsqlにして、のデータベースのテーブルとビューのリストをします。

からテーブルをする

personというがあるとします。

```
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)
);
```

あなたはこのように30の々のしいテーブルをすることができます

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

ログインのテーブルをする

ログインのテーブルをして、テーブルをかなりすることができます。のテーブルは`write-ahead`ログをスキップ`write-ahead`ます。 つまり、クラッシュセーフではなく、できません。

```
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)
);
```

のテーブルをするテーブルをします。

このでは、ユーザーテーブルにはテーブルをするがあります。

```
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.
)
```

オンラインでテーブルをむ https://riptutorial.com/ja/postgresql/topic/2430/テーブル

17: トリガとトリガ

き

トリガ—は、されたまたはビュ—にけられ、のイベントがしたときに、されたfunction_nameをします。

のリンクをにしてください

- トリガ— https : //www.postgresql.org/docs/current/static/sql-createtrigger.html
- トリガ https : //www.postgresql.org/docs/current/static/plpgsql-trigger.html

Examples

なPL / pgSQLトリガ

これはなトリガです。

```
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;
```

このトリガーを`users`テーブルにする

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

トリガーのタイプ

トリガをしてすることができます

- 、、のいずれかのにしてがされる`BEFORE`。
- `AFTER`がしました-、、または。
- ビューの、、またはのは、の`INSTEAD OF`にをします。

マークされたトリガー

- `FOR EACH ROW`は、がするごとに1びされます。
- `FOR EACH STATEMENT`は、のにしてondeとばれます。

サンプルの

```
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()
)
```

シングルインサートトリガー

ステップ1あなたのをする

```
CREATE OR REPLACE FUNCTION add_created_at_function()
```

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

ステップ2トリガーをする

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

ステップ3テストする

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

のためのトリガー

ステップ1あなたのをする

```
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;
```

ステップ2トリガーをする

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

ステップ3テストする

```
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;
```

オンラインでトリガとトリガをむ https://riptutorial.com/ja/postgresql/topic/6957/トリガとトリガ

18: バックアップと

pg_dumpallとpg_dumpをうわりにファイルシステムをバックアップする

それはあなたがこれをするは、あなたがびすことがにですpg_start_backup()のとpg_stop_backup()のにを。ファイルシステムのバックアップをうことはではありません。それらのびしなしでバックアップされたファイルシステムのZFSまたはFreeBSDスナップショットでも、データベースはリカバリモードになり、トランザクションがわれるがあります。

は、Postgresのバックアップファイルにカスタムフォーマットのものがリストアをサポートすることでにがあるため、のPostgresバックアップのわりにファイルシステムバックアップをうことはけています。 1つのファイルなので、もではありません。

Examples

1つのデータベースをバックアップする

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

-Fcは、「カスタムバックアップ」をします。これにより、のSQLよりもです。はpg_restoreをしてください。バニラのSQLファイルがなは、わりにこれをうことができます

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

あるいは

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

バックアップの

```
psql < backup.sql
```

よりなでは、 -1をしてリストアをトランザクションにラップします。シェルリダイレクトをするのではなく、 -fがファイルをします。

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

データベースをするには、 -dオプションをしてpg_restoreをしてカスタムファイルをリストアするがあります。

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

カスタムは、SQLにすることもできます。

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

リストアするをし、にじてをにすることができるので、カスタムをすることをおめします。

あるpostgresqlリリースからしいものにアップグレードするは、pg_dumpにいてpg_restoreをするがあります。

クラスタをバックアップする

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

これは、データベースごとにのをい、 `pg_dump` をすることで、でします。

によっては、これをcronジョブとしてしたくなるかもしれないので、バックアップがファイルのとしてられたをたいとうかもしれません

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

ただし、きなファイルがされるがあることにしてください。 Postgresqlには、なバックアップのためのよりいメカニズムがあります - WALアーカイブ

pg_dumpallからのは、じのPostgresインスタンスにするのにですが、 `$PGDATA pg_hba.conf` と `postgresql.conf` のファイルはバックアップのではないため、々にバックアップするがあります。

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

ファイルシステムのバックアップをるには、これらのをして、バックアップのにPostgresがのあるになるようにするがあります。

コピーをしてインポートする

CSVファイルからテーブルにデータをコピーするには

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

/home/user/ user_data.csvというのファイルからテーブルの**user**するには

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

パイプでられたファイルからテーブルにデータをコピーするには

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

`with delimiter`きのオプション`with delimiter`ない、デフォルトのりはカンマで,

ファイルのインポートにヘッダをする

ヘッダーオプションをします。

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

データがされると、デフォルトでデータはでまれます。のをしてデータをするは、 `QUOTE`オプションをします。ただし、このオプションは、CSVをするにのみされます。

コピーをしてエクスポートする

テーブルを**o / p**にコピーする

<tablename>をSTDOUTにコピーDELIMITER '|';

テーブルユーザーをにエクスポートするには

ユーザーをSTDOUTにコピーしますDELIMITER '|';

テーブルをファイルにコピーするには

COPYユーザーFROM '/ home / user / user_data' WITH DELIMITER '|';

SQLのをファイルにコピーするには

COPYSQLTO '<ファイル>';

COPYSELECT * FROM user WHERE user_name LIKE 'A' '/ home / user / user_data';

ファイルにコピーするには

プログラムをユーザーにコピー 'gzip> /home/user/user_data.gz';

ここでプログラムgzipがされ、ユーザーテーブルのデータがされます。

psqlをってデータをエクスポートする

コピーコマンドをするか、psqlコマンドのコマンドラインオプションをしてデータをエクスポートすることができます。

csvデータをテーブルユーザーから**csv**ファイルにエクスポートするには

```
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
```

ここで、-Aと-Fをみわせることができます。

-Fはりをすることです

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

アラインされていないモードにりえます。 デフォルトのモードは、そうでなければされます。

オンラインでバックアップとをむ https://riptutorial.com/ja/postgresql/topic/2291/バックアップと

19: プログラムによるデータへのアクセス

Examples

Npgsqlプロバイダをして.NETからPostgreSQLにアクセスする

Postgresqlのポピュラーな.NETプロバイダの1つはNpgsqlで、これはADO.NETとがあり、の.NETデータベースプロバイダとほぼじようにされます。

なクエリは、コマンドをし、パラメータをバインドしてコマンドをすることによってされます。Cでは

```
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
```

C-APIをしたPostgreSQLへのアクセス

C-APIはPostgreSQLにアクセスするもなであり、くほどです。

コンパイルとリンク

コンパイルには、 `pg_config --includedir`でつかるPostgreSQLインクルードディレクトリをインクルードパスにするがあります。
あなたは、PostgreSQLのクライアントライブラリとリンクしなければなりません`libpq.so` UNIX 、 `libpq.dll` Windowsの。このライブラリはPostgreSQLのライブラリディレクトリにあります。これは`pg_config --libdir`でつけることができます。

なから、ライブラリは`libpq.so`とばれ、のためのなトラップである`libpg.so`ではありません 。

のコードサンプルが`coltype.c`ファイルにあると`coltype.c` 、コンパイルとリンクは

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

GNU Cコンパイラでライブラリパスをするために`-Wl,-rpath,"$(pg_config --libdir)"`をすることをして`-Wl,-rpath,"$(pg_config --libdir)"` 、または

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

WindowsでMicrosoft Visual Cをする

サンプルプログラム

```
/* 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;
}
```

psycopg2をってPythonからPostgreSQLにアクセスする

ここでドライバのをつけることができます。

なはのとおりです。

```
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())
```

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

Pomm2をしたPHPからのPostgreSQLへのアクセス

レベルのドライバーのには、 pommがあります 。モジュラーアプローチ、データコンバーター、リスン/サポート、データベースインスペクターなどをします。

Pommがをってインストールされているとすると、ながここにあります

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

Pommのクエリマネ―ジャモジュールは、SQLインジェクションをぐためにクエリをエスケ―プします。がキャストされると、をPHPからなPostgresにします。はイテレ―タであり、にカ―ソルをします。すべてのがオンザフライでされ、ブ―ルはブ―ルにされ、タイムスタンプは\DateTimeにされます。

オンラインでプログラムによるデータへのアクセスをむ
https://riptutorial.com/ja/postgresql/topic/2014/プログラムによるデータへのアクセス

20: プロダクションDBのバックアップスクリプト

- このスクリプトをすると、ごとにのでバックアップディレクトリをできます。データベースバックアップディレクトリの+
- prodDir22-11-2016-19h55
- 、ので2つのバックアップファイルがされます。 データベース+
-
- dbprod22-11-2016-19h55.backup ダンプファイル
- dbprod22-11-2016-19h55.sql **sql**ファイル
- **22-11-2016 @ 19h55**での1ののに、
- /save_bd/prodD2222-11-2016-19h55/dbprod22-11-2016-19h55.backup
- /save_bd/prodDir22-11-2016-19h55/dbprod22-11-2016-19h55.sql

パラメーター

パラメータ	
save_db	メインバックアップディレクトリ
dbProd	セカンダリバックアップディレクトリ
	したのバックアップの
dbprod	するデータベースの
/opt/postgres/9.0/bin/pg_dump	pg_dumpバイナリへのパス
-h	サーバーがしているマシンのホストをします。localhost
-p	サーバがをリスンするTCPポートまたはローカルUnixドメインソケットファイルをします5432。
-U	するユーザー。

1. HDPSやSymantec Backupなどのバックアップツールがある... するたびにバックアップディレクトリをにするがあります。

いファイルのバックアップがわれているため、バックアップツールのをけるため。

このをにするには、N°3のをコメントしてください。

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

2. でバックアップツールをできないは、にタスクプランナー cronコマンド をできます。

   のコマンドをして、のユーザーのcronテーブルをします。

```
crontab -e
```

   11にカレンダーきのスクリプトのをスケジュールします。

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

Examples

saveProdDb.sh

に、たちはpgAdminクライアントでDBをバックアップするがあります。は、データベースlinuxを2つのでするためにされるshスクリプトです

- **SQL**ファイル PostgreSQLのどのバージョンでもデータのがです。
- ダンプファイル のバージョンよりものバージョン。

```
#!/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
```

オンラインでプロダクションDBのバックアップスクリプトをむ
https://riptutorial.com/ja/postgresql/topic/7974/プロダクションdbのバックアップスクリプト

21: ロール

- `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

パスワードをしてユーザーをする

に、アプリケーションではデフォルトのデータベースロールは`postgres`をしないでください。わりに、よりいレベルのをつユーザーをするがあります。ここでは、`niceusername`という`niceusername`のパスワードをし、パスワード`very-strong-password`

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

そのは、`psql`コンソールにされたクエリがユーザのホームディレクトリの`.psql_history`というファイルにされ、PostgreSQLデータベースサーバのログにされ、パスワードがされるがあることです。

これをけるには、`\password`コマンドをしてユーザーパスワードをします。コマンドをしたユーザーがスーパーユーザーである、のパスワードはねられません。 スーパーユーザのパスワードをするには、スーパーユーザでなければなりません

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

とするデータベースをする

のアプリケーションをサポートするには、しいロールとデータベースをしてさせることがよくあります。

するシェルコマンドはのとおりです。

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

$ createdb -O blogger blogger
```

これは、`pg_hba.conf`がしくされていることをとしています。これはおそらくのようになります。

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

のとりし。

たとえば、3のユーザーがいるとします。

1. データベースの>
2. のデータへのなアクセスをつアプリケーション> read_write
3. みりアクセス> read_only

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

のクエリでは、できないユーザーはもはやデータベースにできなくなります。

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

のクエリのセットは、されていないユーザーからのすべてのをりし、 `read_write`ユーザーにしてられたのセットをします。

```
--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;
```

ユーザーのデフォルトの**search_path**をする

のコマンドで、ユーザのデフォルトのsearch_pathをすることができます。

1. デフォルトスキーマをするにパスをしてください。

```
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. `alter user`コマンドで`search_path`をして、しいスキーマ`my_schema`をする

```
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. にをする。

```
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)
```

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

されるオブジェクトにするアクセスをします。

たとえば、 `three users`。

1. データベースの`admin` > `admin`
2. のデータへのなアクセスをつアプリケーション> `read_write`
3. みりアクセス> `read_only`

のクエリでは、されたスキーマでされるオブジェクトにするアクセスをできます。

```
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;
```

または、されるオブジェクトにして、されたユーザーがアクセスをすることもできます。

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

みりユーザーをする

```
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;
```

オンラインでロールをむ https://riptutorial.com/ja/postgresql/topic/1572/ロール

22: テーブルWITH

Examples

SELECTクエリのテーブル

テーブルは、よりきなクエリのをすることをサポートしています。えば

```
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)
```

WITH RECURSIVEをしたトラバースツリー

```
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;
```

オンラインでテーブルWITHをむ https://riptutorial.com/ja/postgresql/topic/1973/テーブル-with-

23: クエリ

き

のなクエリ—はありません

Examples

の

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

ドキュメントへのリンク

オンラインでクエリをむ https://riptutorial.com/ja/postgresql/topic/9025/クエリ

24: dblinkとpostgres_fdw

- dblink 'dbname = name_db_distanceポート= PortOfDBホスト= HostOfDBユーザー= usernameDBパスワード= passwordDB'、 'MY QUESRY'
- dbname =データベースの
- port =データベースのポート
- host =データベースのホスト
- user =データベースのユーザ
- パスワード=データベースのパスワード '、
- のQUESRY =これはがしたい、SELECT、INSERT、...することができます

Examples

dblink

dblink EXTENSIONは、のデータベースにし、このデータベースをなだけできるようにするためのです。

1 - dblinkエクステンションをする

```
CREATE EXTENSION dblink;
```

2 - あなたのをいます

exempleのデータベースののテーブルからいくつかのをします。

```
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);
```

エクステンションFDW

FDWはdblinkのです。それはもっとにちますので、それをしてください

1 - エクステンションをする

```
CREATE EXTENSION postgres_fdw;
```

2 - サーバーの

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

3 - postgresサーバーのユーザーマッピングをする

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

4 - の

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

5 - あなたのデータベースにあるように、このテーブルをしてください

```
SELECT * FROM table_foreign;
```

データラッパー

1つのテーブルのわりに、サーバーデータベースのなスキーマにアクセスする。のにってください

1. エクステンションを

```
  CREATE EXTENSION postgres_fdw;
```

2. サーバーの

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

3. ユーザーマッピングの

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

4. サーバーDBのスキーマにアクセスするためのしいスキーマの

```
 CREATE SCHEMA schema_name;
```

5. サーバースキーマのインポート

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

6. サーバースキーマののテーブルにアクセスする

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

これは、リモートDBののスキーマにアクセスするためにできます。

オンラインでdblinkとpostgres_fdwをむ https://riptutorial.com/ja/postgresql/topic/6970/dblinkとpostgres-fdw

25: のさ/のさをつける

き

「character varying」、「text」フィールドのさをするには、char_lengthまたはcharacter_length をします。

Examples

フィールドのさをする

1、クエリ `SELECT char_length('ABCDE')`

5

2、クエリ `SELECT character_length('ABCDE')`

5

オンラインでのさ/のさをつけるをむ https://riptutorial.com/ja/postgresql/topic/9695/のさ-のさをつける

26: 、タイムスタンプ、および

Examples

にタイムスタンプまたはをキャストする

`to_char()`をすると、 `timestamp`または`interval`をにできます。

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

このは、 "12 Aug 2016 04:40:32 PM"をします。は、さまざまなでできます。テンプレートパターンのなリストはここでつけることができます 。

にプレーンテキストをすることもできます。また、のでテンプレートパターンをできます。

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

これにより、「は、20168の12」というがされます。ただし、プレーンテキストがでまれていないり、テンプレートパターン「I」、「D」、「W」などののもがされることにしてください。として、のようにすべてのプレーンテキストをでむがあります。

TMモードをして、したおよびのにをローカライズすることができます。このオプションは、PostgreSQLをしているサーバまたはそれにしているクライアントのローカライズをします。

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

スペインのロケールでは、 "Sábado、12 de Agosto delaño2016"がされます。

のを

のをできます。

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

`201608`はできえです。

1あたりのレコードをえる

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

オンラインで、タイムスタンプ、およびをむ https://riptutorial.com/ja/postgresql/topic/4227/-タイムスタンプ-および

27:

Examples

テーブルのすべてのをする

`column_name = value`をするだけで、テーブルのすべてのをでき`column_name = value`。

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

をたすすべてのをする

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

テーブルののをする

じステートメントののののをできます。つまり、 `col=val`ペアをカンマでります。

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

のテーブルをすることについてテーブルをする

のテーブルのデータについてテーブルのデータをすることもできます。

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

ここでは、のコードをするために、 `cities city`に`person city`をします。これをして、 `person`の`state_code`をします。

オンラインでをむ https://riptutorial.com/ja/postgresql/topic/3136/

28:

PostgreSQLでをするについては、ここでしています。http : //stackoverflow.com/a/3075248/653378

Examples

テーブルの

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

3つのテーブルはのようになります。

ユーザー

カラム	タイプ
ユーザー	テキスト
Eメール	テキスト

シンプルユーザー

カラム	タイプ
ユーザー	テキスト
Eメール	テキスト

users_with_password

カラム	タイプ
ユーザー	テキスト
Eメール	テキスト
パスワード	テキスト

テーブルの

2つのなテーブルをしましょう

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

をする

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

シンプルユーザー

カラム	タイプ
ユーザー	テキスト
Eメール	テキスト
パスワード	テキスト

テーブルにじをすると、ののがマージされます。

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

「simple_users」のカラム「password」のをマージ

をする

されたテーブルをする

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

ユーザー

カラム	タイプ
ユーザー	テキスト
Eメール	テキスト

シンプルユーザー

カラム	タイプ
ユーザー	テキスト
Eメール	テキスト
パスワード	テキスト

にカラムを`simple_users`してから、PostgreSQLはこのカラムがされないようにします。

、のテーブルがあれば、その`password`はされています。

オンラインでをむ https://riptutorial.com/ja/postgresql/topic/5429/

29:

Examples

なmin、max、avg

ののののなをべるには、をします。

あなたのindividualsテーブルが

アリー	17
アマンダ	14
アラナ	20

このをくと、、、をることができます

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

14	20	17

string_agg、り

string_agg()をして、りでられたをすることができます。

あなたのindividualsテーブルが

アリー	15	
アマンダ	14	
アラナ	20	ロシア

SELECT ... GROUP BYをくと、のをできます。

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

string_agg()

はであるため、 `GROUP BY`をするがあることにしてください。

アリー、アマンダ	
アラナ	ロシア

PostgreSQLのはこちら

regr_slopeY、XX、Yのペアでまるのき

regr_slopeY、Xのいをするために、のにしました。 Javaでは、メモリをにクリーンアップしないと、ゴミがスタックしてメモリをいっぱいにすることがあります。さまざまなクラスのメモリについてをダンプし、のためにpostgresデータベースにロードします。

すべてのメモリリークは、よりくのがするにつれてよりくのメモリをするがあります。このトレンドをプロットすると、ラインがにかっているとできます。

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

ヒープ.ダンプ.ヒストグラム.データするメモリーにするクラスのマッピングをむがあるとします。

```
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
);
```

クラスのをするには、クラスごとにグループします。 HAVING>が0の、のスロープがになり、にするののみがされます。たちはをでべえて、のメモリでクラスをします。

```
-- 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 ;
```

```
         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
```

から、java.util.ArrayListのメモリは、71.799バイト/でにしており、にメモリリークのであることがわかります。

オンラインでをむ https://riptutorial.com/ja/postgresql/topic/4803/

クレジット

S. No		Contributors
1	postgresqlをいめる	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	JavaからPostgreSQLにする	Laurenz Albe
3	JSONのサポート	Clodoaldo Neto, commonSenseCode, jgm, KIRAN KUMAR MATAM, mnoronha, Peter Krauss
4	PL / pgSQLによるプログラミング	AndrewCichocki, Ben H, Goerman, Laurenz Albe, Vao Tsun
5	postgresqlのコメント	Ben, KIRAN KUMAR MATAM
6	PostgreSQLのデータベーステーブルヘッダとデータをCSVファイルにエクスポートする	Vao Tsun, wOwhOw
7	PostgreSQLの	gpdude_, Patrick
8	Postgresのヒントとテクニック	Ben H, skj123, user_0, YCF_L
9	Postgresの	Ben H, skj123
10	イベントトリガー	Ben H, Tajinder, Udlei Nati
11	インサート	chalitha geekiyanage, e4c5, gpdude_, KIM, lamorach, leeor, Nathaniel Waisbrot, Patrick, Vao Tsun
12	ウィンドウ	mnoronha, Vao Tsun
13	ゴールデン	Mokadillion
14	セレクト	YCF_L

15	データ	Ben H, user_0
16	テーブル	e4c5, Jefferson, KIM, leeor, Patrick
17	トリガとトリガ	chalitha geekiyanage, mnoronha, Udlei Nati
18	バックアップと	ankidaemon, Ben H, Daniel Lyons, e4c5, Laurel, mnoronha
19	プログラムによるデータへのアクセス	AstraSerg, brichins, greg, Laurenz Albe
20	プロダクションDBのバックアップスクリプト	bilelovitch
21	ロール	Ben, Ben H, bilelovitch, Blackus, Daniel Lyons, e4c5, greg, KIM, Laurenz Albe, mnoronha, Reboot
22	テーブルWITH	Daniel Lyons, Jakub Fedyczak, Kevin Sylvestre
23	クエリ	Ben H
24	dblinkとpostgres_fdw	Riya Bansal, YCF_L
25	のさ/のさをつける	Mohamed Navas
26	、タイムスタンプ、および	KIM, Nuri Tasdemir, Patrick, Tom Gerken
27		frlan, leeor
28		evuez
29		Alison S, joseph, Kirill Sokolov, Patrick