Bazy Danych

Podstawy administracji PostgreSQL. DCL — Data Control Language. Użytkownicy, role, uprawnienia, kopie zapasowe.

Antoni Ligęza, Marcin Szpyrka

Wykorzystano materiały:
`http://www.postgresql.org/docs/8.3/interactive/extend.html`

Administrowanie bazą danych

Administrator bazy danych (ang. database administrator, DBA) — osoba odpowiedzialna za całokształt zadań związanych z pracą SZBD i konkretnej bazy (baz) w określonym środowisku.

Administrator pracuje na styku następujących komponentów:

- system operacyjny,
- SZBD,
- baza (bazy) danych,
- oprogramowanie klienckie,
- sieć.

Administrowanie bazą danych:

- utrzymanie działania (zapewnienie ciągłości działania),
- organizacja dostępu,
- zapewnienie bezpieczeństwa (kopie, odtwarzanie, dostęp),
- zapewnienie efektywności.

Zadania administratora bazy danych

- Sterowanie serwerem:
  - uruchamianie, zatrzymywanie i monitorowanie serwera,
  - organizacja i kontrola dostępu (lokalnie, przez sieć),
  - administrowanie logami (dziennikami),
- administracja danymi:
  - tworzenie i usuwanie baz danych,
  - tworzenie kopii zapasowych, odtwarzanie po awarii,
  - przydzielanie zasobów, odzyskiwanie pamięci,
  - uaktualnianie wersji serwera,
  - archiwizacja danych,
- organizacja i kontrola dostępu użytkowników:
  - tworzenie, usuwanie i modyfikacja kont użytkowników,
  - zarządzanie prawami dostępu (grupy, role, uprawnienia),
  - monitorowanie działania użytkowników,
- zarządzanie bezpieczeństwem,
- zarządanie konfiguracją,
- zarządzanie wydajnością, strojenie, optymalizacja:
  - tworzenie i usuwanie indeksów,
  - monitorowanie i dostrajanie bazy,
  - zarządzanie pamięcią.

Architektura systemu PostgreSQL

PostgreSQL jest oparty na architekturze klient-serwer. Sesja zawiera trzy współpracujące procesy:

- proces administratora (`postmaster`),
- `frontend` – aplikacja użytkownika końcowego (np. program psql),
- `backend` – jeden lub więcej serwerów bazy danych (`proces postgres`).

Proces postmaster zarządza kolekcją baz danych (klastrem) na danym komputerze.

Aplikacja kliencka, chcąca otrzymać dostęp do bazy w klastrze tworzy zapytania do biblioteki `libpq`. Biblioteka przesyła żądania poprzez sieć do postmastera, który rozpoczyna nowy proces serwera i łączy proces klienta z nowym serwerem. Od tej chwili komunikacja odbywa się bez udziału postmastera.

Proces postmaster jest uruchomiony zawsze, czekając na żądania, podczas gdy procesy klienckie i procesy serwera mają określony *czas życia*.

Biblioteka `libpq` zezwala pojedynczemu klientowi na nawiązanie wielu połączeń do procesu serwera. Postmaster i serwer zawsze są uruchomione na tej samej maszynie (serwerze bazy danych), podczas gdy klient może być wszędzie.

Katalogi bazy PostgreSQL

Wg literatury, System PostgreSQL jest (bywał) instalowany w dedykowanym katalogu (np. `/usr/local/pgsql` lub `/var/local/pgsql`). Nie jest to aktualne.

Obecnie często lokacją danych `PGDATA` jest `/var/lib/pgsql/data`.

W Ubuntu: `/var/lib/postgresql/8.2/`

System PostgreSQL 8.3 posiada katalogi opisane w sekcji *53.1. Database File Layout*.

Pliki konfiguracyjne PostgreSQL znajdują się w katalogu `/etc/postgresql/8.2/main`; są to:

pg_hba.conf — plik tekstowy zawierający konfigurację uwierzytelniania hostów PostgreSQL,

pg_ident.conf — plik tekstowy zawierający konfigurację uwierzytelniania użytkowników PostgreSQL,

postgresql.conf — plik konfiguracyjny PostgreSQL,

start.conf — plik konfiguracji trybu uruchamiania (`auto/manual/disabled`).

Oprócz plików konfiguracyjnych w katalogu `/etc/postgresql/8.2/main` znajduje się dowiązanie do katalogu `pgdata` oraz pliku logów `log`.

Konstrukcja fizyczna bazy danych PostgreSQL bazuje na technice *stronicowania pamięci*. Typowy rozmiar strony to 8kB (rozmiar bloku).

Wiersze tablicy nie mogą być (bezpośrednio) dzielone na kilka stron. PostgreSQL stosuje specjalną technologię kompresji dużych wartości atrybutów i rozdziału ich pomiędzy strony (technologia TOAST).

Sterowanie serwerem

Serwer bazy danych PostgreSQL uruchamia się jako proces nasłuchu (UNIX/Linux). Proces `postmaster` musi działać, aby możliwy był dostęp do baz danych.

Sprawdzenie: `ps -el | grep post`.

Zatrzymywanie i uruchamianie serwera:

```
root@alir:/home/ali# su - postgres
postgres@alir:~$ /etc/init.d/postgresql-8.2 start
 * Starting PostgreSQL 8.2 database server
postgres@alir:~$ ps -el | grep post
0 S   109 15356     1  1  80   0 -  9518 -
      ?        00:00:00 postgres
1 S   109 15358 15356  0  80   0 -  9518 -
      ?        00:00:00 postgres
1 S   109 15359 15356  0  80   0 -  2447 -
     ?        00:00:00 postgres
postgres@alir:~$ /etc/init.d/postgresql-8.2 stop
 * Stopping PostgreSQL 8.2 database server
postgres@alir:~$ ps -el | grep post
postgres@alir:~$ /etc/init.d/postgresql-8.2 start
 * Starting PostgreSQL 8.2 database server
postgres@alir:~$ ps -el | grep post
0 S   109 15453     1 12  80   0 -  9518 -
     ?        00:00:00 postgres
1 S   109 15455 15453  0  80   0 -  9518 -
     ?        00:00:00 postgres
1 S   109 15456 15453  0  80   0 -  2447 -
      ?        00:00:00 postgres
postgres@alir:~$
```

Przykład

- Instalacja potrzebnych pakietów (Ubuntu):

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

  Przy instalacji tego pakietu automatycznie instalowane są inne pakiety, od których ten pakiet jest zależny.

  Przy instalacji tworzony jest użytkownik `postgres`, serwer jest domyślnie konfigurowany i na koniec uruchamiany.

- Uruchomienie serwera bazy danych (z konta roota lub postgresa):

  ```
  /etc/init.d/postgresql-... start
  ```

  W oparciu o skrypt `/etc/init.d/postgresql` można sterować serwerem baz danych, dostępne opcje to m.in.: `start`, `stop`, `status`, `restart`.

- Utworzenie *zwykłego* użytkownika (najwygodniej jest utworzyć użytkownika o nazwie pokrywającej się z nazwą konta w systemie UNIX/Linux):

  ```
  su
  su postgres
  createuser bzik
  ```

- Utworzenie pierwszej bazy danych (z konta użytkownika *bzik*):

  ```
  createdb -E 'latin2' pracownicy
  ```

- Połączenie się z utworzoną bazą danych:

  ```
  psql -d pracownicy
  ```

Opcje sterowania serwerem

Polecenie

`/etc/init.d/postgresql-8.2 <option>` może realizować następujące operacje:

start — uruchamianie serwera bazy danych,

stop — zatrzymywanie serwera bazy danych,

restart — restartowanie serwera bazy danych,

force-reload — wymuszenie ponownego załadowania serwera,

status — wyświetlenie informacji o statusie serwera,

autovac-start — uruchamianie serwera bazy danych z autovacuum,

autovac-stop — zatrzymywanie serwera bazy danych z autovacuum,

autovac-restart — restartowanie serwera bazy danychz autovacuum,

Polecenia systemowe PostgreSQL

Obok komend języka SQL, PostgreSQL oferuje wiele poleceń uruchamianych z poziomu systemu operacyjnego.

Polecenia te mogą być plikami binarnymi lub skryptami opakowującymi (wrappers) np. w PERL-u, sh.

Polecenia uruchamiane z poziomu systemu operacyjnego umieszczone są w katalogu `/usr/bin`.

Lista ważniejszych poleceń:

psql — narzędzie wiersza poleceń PostgreSQL;

createuser — narzędzie do tworzenia użytkownika bazy danych;

dropuser — narzędzie do usuwania użytkownika bazy danych;

createdb — narzędzie do tworzenia bazy danych;

dropdb — narzędzie do usuwania bazy danych;

pg_dump — narzędzie do tworzenia kopii zapasowej bazy danych;

pg_dumpall — narzędzie do tworzenia kopii zapasowych wszystkich baz danych w instalacji;

pg_restore — narzędzie do odtwarzania bazy danych z kopii zapasowej;

pg_config — dostęp do informacji o wersji i instalacji PostgreSQL;

reindexdb — reindeksacja bazy danych;

vacuumdb — garbage collection (odzyskiwanie pamięci) i analiza bazy danych.

Pełna lista poleceń jest dostępna w dokumentacji PostgreSQL `II. PostgreSQL Client Applications.`

Użytkownicy bazy danych

Dostęp do bazy danych możliwy jest tylko dla zarejestrowanych użytkowników.
Użytkownikiem uprzywilejowanym jest `postgres`.
Lista użytkowników dostępna jest w tablicy katalogowej `pg_user`.

```
pracownicy=> select * from pg_user ;
 usename  | usesysid | usecreatedb | usesuper | usecatupd |  pas
----------+----------+-------------+----------+-----------+-----
 postgres |       10 | t           | t        | t         | ****
 ali      |    16384 | t           | f        | f         | ****
 bzik     |    16519 | t           | f        | f         | ****
(3 rows)

pracownicy=> \du
                             List of roles
 Role name | Superuser | Create role | Create DB | Connections |
-----------+-----------+-------------+-----------+-------------+
 ali       | no        | yes         | yes       | no limit    |
 bzik      | no        | yes         | yes       | no limit    |
 postgres  | yes       | yes         | yes       | no limit    |
(3 rows)

pracownicy=>
```

Każdy użytkownik ma unikalną nazwę `usename` oraz unikalny identyfikator `usesysid`. ma też uprawnienia (np. do tworzenia baz danych).

Uwaga: użytkownik PostgreSQL to niekoniecznie użytkownik systemu operacyjnego (chociaż można stosować taka konwencję). Login użytkownika jest domyślnie przedstawiany serwerowi bazy danych, ale można wymusić logowanie z nazwą przekazaną jako parametr.

Tworzenie i usuwanie użytkowników baz danych

Tworzenie użytkownika baz danych może odbywać się z poziomu systemu operacyjnego lub z poziomu SQL (`CREATE USER`).

Polecenie systemu operacyjnego:

```
createuser [option...] [username]
```

Polecenie to tworzy nowego użytkownika (dokładniej: rolę).

Tylko superuser i użytkownicy posiadający odpowiednie uprawnienia mogą tworzyć nowych użytkowników.

Polecenie `createuser` posiada wiele parametrów (vide: dokumentacja).

Przykład:

```
ali@alir:~$ dropuser bzik
DROP ROLE
ali@alir:~$ createuser bzik
Shall the new role be a superuser? (y/n) n
Shall the new role be allowed to create databases? (y/n) y
Shall the new role be allowed to create more new roles? (y/n) y
CREATE ROLE
ali@alir:~$
```

Usuwanie użytkownika:

```
ali@alir:~$ dropuser bzik
DROP ROLE
ali@alir:~$
```

SQL - CREATE USER

Z poziomu SQL użytkownika tworzymy poleceniem:

```
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 rolename [, ...]
    | IN GROUP rolename [, ...]
    | ROLE rolename [, ...]
    | ADMIN rolename [, ...]
    | USER rolename [, ...]
    | SYSID uid
```

Aktualnie `CREATE USER` jest aliasem dla `CREATE ROLE`.
Usuwanie użytkownika:

```
DROP USER [ IF EXISTS ] name [, ...]
```

Modyfikacja użytkownika

```
ALTER 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'

ALTER USER name RENAME TO newname

ALTER USER name SET configuration_parameter { TO | = } { value |
ALTER USER name SET configuration_parameter FROM CURRENT
ALTER USER name RESET configuration_parameter
ALTER USER name RESET ALL
```

Aktualnie `ALTER USER` jest aliasem dla `ALTER ROLE`.

Uprawnienia użytkowników

- W PostgreSQL zarządzanie uprawnieniami użytkowników odbywa się poprzez tzw. *role*. W zależności od ustawień rola może być postrzegana jako pojedynczy użytkownik, lub grupa użytkowników. Pojęcie roli zastępuje występujące we wcześniejszych wersjach PostgreSQL pojęcia użytkownika i grupy użytkowników.
- Rola może być właścicielem obiektów bazy danych (np. tabel) i może przypisywać uprawnienia do tych obiektów innym rolom. Ponadto można ustanowić jedną rolę członkiem innej, przez co przejmuje ona uprawnienia jakie posiada ta druga.
- Polecenia `createuser` i `dropuser` odpowiadają w rzeczywistości poleceniom SQL:

```
CREATE ROLE <name>;
DROP ROLE NAME;
```

- Informacje o zdefiniowanych rolach przechowywane są w tablicy `pg_roles`. Można je również wyświetlić w psql za pomocą polecenia `\du`.
- Każdy dostęp do serwera jest wykonywany *w imieniu* pewnej roli.
- Tworzenie roli zawierającej inne role:

```
create role sprzedawcy;
grant sprzedawcy to Ewa;
grant sprzedawcy to Adam;
revoke sprzedawcy from Jan;
```

Uprawnienia użytkowników

PostgreSQL steruje dostępem do bazy danych poprzez wykorzystanie systemu uprawnień, które mogą być udzielane lub odbierane za pomocą poleceń `GRANT` i `REVOKE`.
Dozwolone uprawnienia to m.in.:

SELECT – umożliwia odczytywanie wierszy,

INSERT – umożliwia tworzenie nowych wierszy,

DELETE – umożliwia usuwanie wierszy,

UPDATE – umożliwia aktualizację istniejących wierszy,

RULE – umożliwia tworzenie reguł dla tabeli lub perspektywy,

ALL – daje wszystkie uprawnienia,

```
grant uprawnienie [, ...] on obiekt [, ...]
     to {public | rola}
revoke uprawnienie [, ...] on obiekt [, ...]
     from {public | rola}
```

Słowo kluczowe `PUBLIC` jest skrótem oznaczającym wszystkie role, w tym również nie istniejących jeszcze w momencie nadawania przywilejów.

Połączenie uprawnień użytkowników z perspektywami umożliwia ograniczenie dla pewnych użytkowników dostępu do niektórych informacji zawartych w tabelach. Dostęp do takich informacji można zwykłym użytkownikom uniemożliwić poprzez utworzenie perspektywy wybierającej tylko dozwolone kolumny i nadanie uprawnień wszystkim użytkownikom do tej właśnie perspektywy.

Definiowanie ról

```
CREATE ROLE 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 rolename [, ...]
    | IN GROUP rolename [, ...]
    | ROLE rolename [, ...]
    | ADMIN rolename [, ...]
    | USER rolename [, ...]
    | SYSID uid
```

Pełny opis parametrów w dokumentacji.
Usuwanie roli:

```
DROP ROLE [ IF EXISTS ] name [, ...]
```

Modyfikacja roli

```
ALTER ROLE 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'

ALTER ROLE name RENAME TO newname

ALTER ROLE name SET configuration_parameter { TO | = }
    { value | DEFAULT }
ALTER ROLE name SET configuration_parameter FROM CURRENT
ALTER ROLE name RESET configuration_parameter
ALTER ROLE name RESET ALL
```

Pełny opis parametrów w dokumentacji.

Grupy użytkowników

Grupy użytkowników pozwalają na jednolite traktowanie użytkowników w bazie danych (łatwe zarządzanie uprawnieniami).

Tworzenie grup:

```
CREATE GROUP name [ [ WITH ] option [ ... ] ]

where option can be:

      SUPERUSER | NOSUPERUSER
    | CREATEDB | NOCREATEDB
    | CREATEROLE | NOCREATEROLE
    | CREATEUSER | NOCREATEUSER
    | INHERIT | NOINHERIT
    | LOGIN | NOLOGIN
    | [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'
    | VALID UNTIL 'timestamp'
    | IN ROLE rolename [, ...]
    | IN GROUP rolename [, ...]
    | ROLE rolename [, ...]
    | ADMIN rolename [, ...]
    | USER rolename [, ...]
    | SYSID uid
```

Przykład

```
pracownicy=> \du
                                   List of roles
 Role name | Superuser | Create role | Create DB | Connections |
-----------+-----------+-------------+-----------+-------------+
 ali       | no        | yes         | yes       | no limit    |
 bzik      | no        | yes         | yes       | no limit    |
 postgres  | yes       | yes         | yes       | no limit    |
(3 rows)

pracownicy=> create group nasi with user ali, bzik;
CREATE ROLE
pracownicy=> select * from pg_group;
 groname | grosysid |    grolist
---------+----------+---------------
 nasi    |    16760 | {16384,16759}
(1 row)

pracownicy=>
```

Usuwanie grup:

```
DROP GROUP [ IF EXISTS ] name [, ...]
```

Modyfikacja grup:

```
ALTER GROUP groupname ADD USER username [, ... ]
ALTER GROUP groupname DROP USER username [, ... ]

ALTER GROUP groupname RENAME TO newname
```

Instrukcje GRANT

Polecenie `GRANT` pozwala przypisywać uprawnienia dla użytkowników do obiektów bazy danych (table, view, sequence, database, function, procedural language, schema, or tablespace) oraz przypisywać role do użytkowników.

```
GRANT { { SELECT | INSERT | UPDATE
        | DELETE | REFERENCES | TRIGGER }
    [,...] | ALL [ PRIVILEGES ] }
    ON [ TABLE ] tablename [, ...]
    TO { [ GROUP ] rolename | PUBLIC }
    [, ...] [ WITH GRANT OPTION ]

GRANT { { USAGE | SELECT | UPDATE }
    [,...] | ALL [ PRIVILEGES ] }
    ON SEQUENCE sequencename [, ...]
    TO { [ GROUP ] rolename | PUBLIC }
    [, ...] [ WITH GRANT OPTION ]

GRANT { { CREATE | CONNECT | TEMPORARY | TEMP }
    [,...] | ALL [ PRIVILEGES ] }
    ON DATABASE dbname [, ...]
    TO { [ GROUP ] rolename | PUBLIC } [, ...]
    [ WITH GRANT OPTION ]

GRANT { EXECUTE | ALL [ PRIVILEGES ] }
    ON FUNCTION funcname ( [ [ argmode ]
    [ argname ] argtype [, ...] ] ) [, ...]
    TO { [ GROUP ] rolename | PUBLIC }
    [, ...] [ WITH GRANT OPTION ]
```

Instrukcje GRANT c.d.

```
GRANT { USAGE | ALL [ PRIVILEGES ] }
    ON LANGUAGE langname [, ...]
    TO { [ GROUP ] rolename | PUBLIC }
    [, ...] [ WITH GRANT OPTION ]

GRANT { { CREATE | USAGE } [,...]
    | ALL [ PRIVILEGES ] }
    ON SCHEMA schemaname [, ...]
    TO { [ GROUP ] rolename | PUBLIC }
    [, ...] [ WITH GRANT OPTION ]

GRANT { CREATE | ALL [ PRIVILEGES ] }
    ON TABLESPACE tablespacename [, ...]
    TO { [ GROUP ] rolename | PUBLIC }
    [, ...] [ WITH GRANT OPTION ]
```

Specyficzną postać ma delegowanie uprawnień poprzez przypisanie ról:

```
GRANT role [, ...] TO rolename [, ...]
    [ WITH ADMIN OPTION ]
```

Polecenia REVOKE

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | INSERT | UPDATE
    | DELETE | REFERENCES | TRIGGER }
    [,...] | ALL [ PRIVILEGES ] }
    ON [ TABLE ] tablename [, ...]
    FROM { [ GROUP ] rolename | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { { USAGE | SELECT | UPDATE }
    [,...] | ALL [ PRIVILEGES ] }
    ON SEQUENCE sequencename [, ...]
    FROM { [ GROUP ] rolename | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { { CREATE | CONNECT | TEMPORARY
    | TEMP } [,...] | ALL [ PRIVILEGES ] }
    ON DATABASE dbname [, ...]
    FROM { [ GROUP ] rolename | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { EXECUTE | ALL [ PRIVILEGES ] }
    ON FUNCTION funcname ( [ [ argmode ]
    [ argname ] argtype [, ...] ] ) [, ...]
    FROM { [ GROUP ] rolename | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]
```

Polecenia REVOKE c.d.

```
REVOKE [ GRANT OPTION FOR ]
    { USAGE | ALL [ PRIVILEGES ] }
    ON LANGUAGE langname [, ...]
    FROM { [ GROUP ] rolename | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { { CREATE | USAGE } [,...]
    | ALL [ PRIVILEGES ] }
    ON SCHEMA schemaname [, ...]
    FROM { [ GROUP ] rolename | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { CREATE | ALL [ PRIVILEGES ] }
    ON TABLESPACE tablespacename [, ...]
    FROM { [ GROUP ] rolename | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ ADMIN OPTION FOR ]
    role [, ...] FROM rolename [, ...]
    [ CASCADE | RESTRICT ]
```

Mechanizmy zarządzania dostępem do danych

W RBD istnieje kilka mechanizmów i związanych z nimi narzędzi dla organizacji dostępu do obiektów baz danych; są to:

role — zdefiniowani użytkownicy baz danych (`CREATE ROLE`),

grupy — użytkownicy mogą być łączeni w grupy (`CREATE GROUP`),

uprawnienia — przypisanie uprawnień dla ról (`GRANT, REVOKE`),

perspektywy — wybór danych i przypisanie uprawnień (`CREATE VIEW`),

schematy — obiekty bazodanowe mogą być łączone w schematy (`CREATE SCHEMA`)

W PostgreSQL tabela `pg_user` jest w istocie widokiem na `pg_shadow`; jednak zwykli użytkownicy nie mają praw dostępu do tej ostatniej.

W praktyce kombinacje wskazanych powyżej 5 mechanizmów pozwalaja na elastyczne zarządzanie uprawnieniami.

Schematy

`SCHEMA` – schemat to przestrzeń nazw w bazie danych. Baza danych może zawierać jeden lub więcej schematów (ang. `schema`). Zwyczajowo schemat to zestaw obiektów (tablic) pod kontrolą jednego użytkowanika (roli).

Schematy mogą zawierać obiekty o identycznych nazwach. Odwołanie do np. tablicy `prac` w schemacie `biuro` następuje poprzez użycie notacji kropkowej: `SELECT * FROM biuro.prac;`.

Użycie schematów wewnątrz bazy ma na celu *logiczną separację* obiektów, np. dla różnych użytkowników i aplikacji. Standardowo i domyślnie wszystkie obiekty tworzone są w schemacie `public`.

Tworzenie schematu:

```
CREATE SCHEMA schemaname
     [ AUTHORIZATION username ]
     [ schema_element [ ... ] ]

CREATE SCHEMA AUTHORIZATION username
     [ schema_element [ ... ] ]
```

`username` – nazwa właściciela schematu; default – bieżący użytkownik.
`schema_element` – tworzony element schematu; aktualnie tylko `CREATE TABLE, CREATE VIEW, CREATE INDEX, CREATE SEQUENCE, CREATE TRIGGER, GRANT`.

Tworzenie tabeli (obiektu) w schemacie:

```
CREATE SCHEMA myschema;

CREATE TABLE myschema.mytable (
 ...
);
```

Usuwanie schematów

Usuwanie schematu (pustego/niepustego):

```
DROP SCHEMA myschema;

DROP SCHEMA myschema CASCADE;

SHOW search_path;

 search_path
--------------
 "$user",public

SET search_path TO myschema,public;
```

Przykłady tworzenia schematów

```
CREATE SCHEMA myschema;

Create a schema for user joe;
   the schema will also be named joe:

CREATE SCHEMA AUTHORIZATION joe;

Create a schema and create a table
   and view within it:

CREATE SCHEMA hollywood
    CREATE TABLE films
    (title text, release date, awards text[])
    CREATE VIEW winners AS
        SELECT title, release FROM films
        WHERE awards IS NOT NULL;

Notice that the individual subcommands
   do not end with semicolons.

The following is an equivalent way of
   accomplishing the same result:

CREATE SCHEMA hollywood;
CREATE TABLE hollywood.films
   (title text, release date, awards text[]);
CREATE VIEW hollywood.winners AS
    SELECT title, release FROM hollywood.films
    WHERE awards IS NOT NULL;
```

Tworzenie bazy danych

W PostgreSQL bazę danych można utworzyć z poziomu SQL (`CREATE DATABASE`) lub z poziomu systemu operacyjnego (`createdb`).

Z poziomu SQL:

```
CREATE DATABASE name
    [ [ WITH ] [ OWNER [=] dbowner ]
      [ TEMPLATE [=] template ]
      [ ENCODING [=] encoding ]
      [ TABLESPACE [=] tablespace ]
      [ CONNECTION LIMIT [=] connlimit ] ]
```

Dla utworzenia bazy danych potrzebne są uprawnienia do zakładania baz.

Opcje kodowania (`ENCODING`) dostępne są w dokumentacji; np. `LATIN2`, `UTF8` `SQL_ASCII`.

Limit połączeń równoległych (`CONNECTION LIMIT = -1`) dopuszcza nielimitowaną liczbę połączeń (default).

Aby utworzyć pierwszą bazę należy połączyć się z bazą `template1`.

Z poziomu systemu operacyjnego:

```
createdb [option...] [dbname] [description]
```

Tworzenie i usuwanie baz danych

Opcje tworzenia bazy:

-D tablespace — specyfikacja przestrzeni tablic,

-e — echo,

-E encoding — ustalenie kodowania,

-O owner — okreslenie właściciela bazy,

-T template — okreslenie bazy wzorcowej,

-h host — definicja hosta,

-p port — definicja portu (default: 5432),

-U username — użytkownika,

-W — wymaganie hasła przed wykonaniem.

Kodowania: np. `LATIN2`, `UTF8` `SQL_ASCII`.
Usuwanie bazy (przez właściciela):

```
DROP DATABASE [ IF EXISTS ] name
```

```
dropdb [option...] dbname
```

Kopie zapasowe – przykłady i bezpieczeństwo

PostgreSQL posiada własne mechanizmy tworzenia kopii zapasowych i ich odtwarzania: pg_dump, pg_dumpall oraz pg_restore.

```
pg_dump -Fp kwiaciarnia > kopia.sql
pg_dump -Ft kwiaciarnia > kopia.tar
pg_restore -d kwiaciarnia2 kopia.tar
```

Działanie systemu kopii zapasowej polega na utworzeniu dużego skryptu złożonego z poleceń SQL (oraz wewnętrznych poleceń psql), które po wykonaniu odtworzą całość bazy danych. Skrypt zawiera polecenia tworzenia i wypełniania tabel, ale nie obejmuje utworzenia samej bazy danych.

PostgreSQL zabezpiecza bazy danych na kilka sposobów.

- Prawa dostępu do wszystkich plików, które PostgreSQL wykorzystuje do przechowywania danych są ustawione tak, że pliki te są dostępne tylko dla użytkownika PostgreSQL postgres (oraz root).
- Po uruchomieniu serwera baz danych połączenia zdalne nie są dozwolone. Dopuszczalne są tylko żądania klientów na komputerze lokalnym.
- Aby zezwolić na dostęp z sieci, należy jawnie w tym celu skonfigurować PostgreSQL. Konfigurację systemu bezpieczeństwa w PostgreSQL przeprowadza się za pomocą pliku pg_hba.conf. Można między innymi ograniczyć zakres adresów sieciowych, z których połączenia są dopuszczalne.

Tworzenie kopii zapasowych

Możliwe jest tworzenie kopii *gorących* (w trakcie pracy) i *zimnych* (przy wyłączonym serwerze; pod systemem operacyjnym).

Do tworzenia kopii zapasowych `gorących` należy użyć specjalnych poleceń systemu PostgreSQL.

```
pg_dump [option...] [dbname]
```

Przykład:

```
pg_dump pracownicy > pracownicy.backup

pg_dump -Ft pracownicy > pracownicy.backup.tar

pg_dump -Ft pracownicy | gzip > pracownicy.backup.tar.gz
```

Wybrane opcje:

-a — tylko dane (bez schematu),

-d — składuje dane w formie komend `INSERT` (zamiast `copy`),

-D — składuje dane w formie komend `INSERT` (zamiast `copy`) z jawny okresleniem kolumn,

-E encoding — jawna specyfikacja zmiany kodowania,

-f file — przesłanie do pliku (inaczej przez przekierowanie `pg_dump baza > baza.backup`),

F <format — specyfikacja formatu (p – plain, c – custom, t – tar (wymagapg_restore)),

-s — tylko schemat bazy (bez danych),

-t table — tylko wybraną tablicę bazy.

Kopia zapasowa wszystkich baz

Awaryjny zrzut wszystkich baz; pozwala zachować też informacje wspólne (o użytkownikach).

Należy wykonać jako postgres:

```
pg_dumpall [option...]
```

```
su - postgres
pg_dumpall > all.bacup
-- lub
pg_dumpall | gzip > all.bacup.gz
```

Odtwarzanie bazy:

```
pg_restore [option...] [filename]
```

Wybrane opcje:

-a — tylko dane (bez schematu),

-C — tworzy bazę danych przed odtworzeniem,

-d dbname — łączy się z bazą o podanej nazwie i tam odtwarza bazę,

F <format> — specyfikacja formatu (c – custom, t – tar),

-s — tylko schemat bazy (bez danych),

-t table — tylko wybraną tablicę bazy.

Przykłady

```
$ pg_dump mydb > db.sql

$ psql -d newdb -f db.sql

$ pg_dump -Fc mydb > db.dump

$ pg_restore -d newdb db.dump

$ pg_dump -t mytab mydb > db.sql

$ pg_dump -t 'detroit.emp*' -T detroit.employee_log
     mydb > db.sql

$ pg_dump -n 'east*gsm' -n 'west*gsm' -N '*test*'
     mydb > db.sql

$ pg_dump -n '(east|west)*gsm' -N '*test*' mydb > db.sql

$ pg_dump -T 'ts_*' mydb > db.sql


$ pg_dump -Fc mydb > db.dump

$ dropdb mydb
$ pg_restore -C -d postgres db.dump
```

Klonowanie bazy template0:

```
$ createdb -T template0 newdb
$ pg_restore -d newdb db.dump
```

Konfiguracja serwera — podstawy: konta użytkowników i pg_hba.conf

Aplikacja klienta łącząca się z serwerem bazy danych musi podać nazwę użytkownika PostgreSQL, która będzie używana w trakcie połączenia. Na podstawie tej nazwy określane są przywileje użytkownika do korzystania z obiektów bazy danych (łącznie z tym czy może się on łączyć z daną bazą). Nazwy użytkowników PostgreSQL są logicznie inne niż nazwy użytkowników systemu operacyjnego, choć jeśli użytkownik ma konto w systemie, to zazwyczaj stosowane są takie same nazwy.

Autoryzacja klientów jest kontrolowana przez plik konfiguracyjny `pg_hba.conf` (host-based authentication).

```
/etc/postgresql/8.?/main/pg_hba.conf
```

Plik pg_hba.conf ma postać zbioru rekordów, z których każdy zajmuje dokładnie jedną linię. Ignorowane są puste linie i tekst po symbolu #. Poszczególne pola w rekordzie oddzielone są spacjami lub znakami tabulacji.

Każdy rekord określa: typ połączenia, zakres adresów IP (jeśli są istotne w danym przypadku), nazwę bazy danych, nazwę użytkownika i metodę autoryzacji. Do autoryzacji klienta wykorzystywane jest pierwszy znaleziony rekord, który pasuje do określonego typu połączenia.

Plik pg_hba.conf jest czytany przy starcie serwera. Po wprowadzeniu zmian należy serwer zrestartować.

Rekordy w pliku pg_hba.conf

```
local     database user authentication-method
host      database user IP-address IP-mask authentication-method
hostssl   database user IP-address IP-mask authentication-method
hostnossl database user IP-address IP-mask authentication-method
```

- **local** – połączenia lokalne (łączymy się nie korzystając z portów TCP/IP);
- **host** – połączenia zdalne poprzez TCP/IP (zarówno kodowane jak i niekodowane);
- **hostssl** – połączenia zdalne kodowane za pośrednictwem protokołu SSL;
- **hostnossl** – połączenia zdalne niekodowane;
- **database** – nazwa lub nazwy (oddzielone przecinkami) baz danych (opcja all);
- **user** – nazwa lub nazwy (oddzielone przecinkami) użytkowników lub grup (nazwy grup poprzedzane są symbolem +);
- **IP-address IP-mask** – adres IP i maska podsieci (dla host, hostssl, hostnossl);
- **authentication-method** – metoda autoryzacji, np.: trust – bez podawania hasła, reject – bezwarunkowa odmowa, md5 – wymaga hasła kodowanego algorytmem md5, password – wymaga niekodowanego hasła.

```
local all trust
host all 127.0.0.1 255.255.255.255 trust
host all 192.168.1.1 255.255.255.0 password
host kwiaciarnia 192.168.1.1 255.255.255.0 password
```

Odzyskiwanie pamięci

Polecenie `VACUUM` ma dwa zastosowania:

- odzyskiwanie pamięci w bazie danych,
- aktualizacja statystyk optymalizatora.

W bazie gromadzą się nieużywane dane (np. po wycofanych transakcjach, po `DROP TABLE`, etc.).
Wykonanie tego polecenia `VACUUM` pozwala uporządkować bazę.

```
VACUUM [ FULL ] [ FREEZE ] [ VERBOSE ] [ table ]

VACUUM [ FULL ] [ FREEZE ] [ VERBOSE ] ANALYZE
     [ table [ (column [, ...] ) ] ]
```

Polecenie `VACUUM ANALYZE` pozwala uaktualnić statystyki optymalizatora.
Z poziomu systemu operacyjnego:

```
vacuumdb [connection-option...] [--full | -f]
     [--verbose | -v] [--analyze | -z]
     [--table | -t table [( column [,...] )] ] [dbname]

vacuumdb [connection-options...] [--all | -a]
     [--full | -f] [--verbose | -v] [--analyze | -z]
```

Konfiguracja do pracy lokalnej – PostgreSQL, Apache, PHP

- W Ubuntu należy zainstalować pakiety: `postgresql-8.?`, `apache2`, `php5`, `libapache-mod-auth-pgsql`, `php5-pqsql`.

```
/etc/init.d/postgresql-8.? stop
/etc/init.d/apache2 start
/etc/init.d/apache2 stop
```

- Konfiguracja PostgreSQL `/etc/postgresql/8.?/main/postgresql.con`:

```
listen_addresses = 'localhost'
port = 5432
max_connections = 100
```

- Konfiguracja PostgreSQL `/etc/postgresql/8.?/main/pg_hba.conf`:

```
local all all trust
host all all 127.0.0.1/32 trust
```

- Konfiguracja PHP `/etc/php5/apache2/php.ini`

```
extension_dir = "/usr/lib/php5/20060613+lfs/"
extension=pgsql.so

/etc/init.d/apache start
/etc/init.d/postgresql-8.? start
```