Elementy bazy danych

- Tabele
- Dziedziny i typy danych
- Bazy, schematy, przestrzenie nazw i tabel
- Użytkownicy, role, prawa
- Indeksy i sekwencje
- Perspektywy
- Asercje i wyzwalacze
- Wyzwalacze, funkcje i reguły

Dziedziny - definiowanie i stosowanie

Definicja dziedziny

```
CREATE DOMAIN nazwa [ AS ] dziedzina_bazowa
            [ DEFAULT wyrażenie] [ warunek [ ... ] ]
```

Warunek:

```
[ CONSTRAINT nazwa\_warunku ]
{ NOT NULL | NULL | CHECK (wyrażenie\_logiczne) }
```

Dziedziny - definiowanie i stosowanie

Definicja dziedziny

```
CREATE DOMAIN nazwa [ AS ] dziedzina_bazowa
            [ DEFAULT wyrażenie] [ warunek [ ... ] ]
```

Warunek:

```
[ CONSTRAINT nazwa\_warunku ]
{ NOT NULL | NULL | CHECK (wyrażenie\_logiczne) }
```

Przykład:

```
CREATE DOMAIN zajecia_dom AS char(1)
DEFAULT 'w' NOT NULL
CHECK (VALUE IN ('w','c','p','r','e','s','C','P','R'));

ALTER TABLE grupa
ALTER COLUMN rodzaj_zajec
SET DATA TYPE zajecia_dom;
```

Dziedziny - uwagi

- Wartości z różnych dziedzin są zgodne, jesli ich dziedziny bazowe są zgodne. Np. można zrobić:

```
SELECT * FROM grupa WHERE rodzaj_zajec='c';
```

Dziedziny - uwagi

- Wartości z różnych dziedzin są zgodne, jesli ich dziedziny bazowe są zgodne. Np. można zrobić:
  ```
  SELECT * FROM grupa WHERE rodzaj_zajec='c';
  ```
- Usunięcie dziedziny z opcją CASCADE usuwa obiekty zależne, czyli np. kolumny tego typu:
  ```
  DROP DOMAIN zajecia_dom CASCADE;
  ```

Dziedziny - uwagi

- Wartości z różnych dziedzin są zgodne, jesli ich dziedziny bazowe są zgodne. Np. można zrobić:
  ```
  SELECT * FROM grupa WHERE rodzaj_zajec='c';
  ```
- Usunięcie dziedziny z opcją CASCADE usuwa obiekty zależne, czyli np. kolumny tego typu:
  ```
  DROP DOMAIN zajecia_dom CASCADE;
  ```
- Warunek dziedziny musi być NOT FALSE! Ta dziedzina akceptuje wszystkie znaki:
  ```
  CREATE DOMAIN abc AS char(1) CHECK (VALUE IN ('a','b','c',NULL));
  ```

Dziedziny - uwagi

- Wartości z różnych dziedzin są zgodne, jesli ich dziedziny bazowe są zgodne. Np. można zrobić:
  ```
  SELECT * FROM grupa WHERE rodzaj_zajec='c';
  ```
- Usunięcie dziedziny z opcją CASCADE usuwa obiekty zależne, czyli np. kolumny tego typu:
  ```
  DROP DOMAIN zajecia_dom CASCADE;
  ```
- Warunek dziedziny musi być NOT FALSE! Ta dziedzina akceptuje wszystkie znaki:
  ```
  CREATE DOMAIN abc AS char(1) CHECK (VALUE IN ('a','b','c',NULL));
  ```
- Warunek dziedziny jest sprawdzany w czasie: INSERT/UPDATE:

```
CREATE DOMAIN dobra_data AS timestamp
             CHECK (VALUE >= CURRENT_TIMESTAMP);
CREATE TABLE test (a dobra_data);
INSERT INTO test VALUES(CURRENT_TIMESTAMP);
SELECT * FROM test WHERE a < CURRENT_TIMESTAMP;
-- jest OK
ALTER DOMAIN dobra_data ADD COSNTRAINT c2 CHECK
             (VALUE>CURRENT_TIMESTAMP);
ERROR:  column "a" of table "test" contains values
   that violate the new constraint.
```

Definicja i przykład

Definicja typu

- stwórzmy typ wyliczeniowy i użyjmy go w definicji tablicy:

```
CREATE TYPE typ_zajec AS ENUM('w','c','p');

CREATE TABLE grupa ( ... rodzaj_zajec typ_zajec ... );
```

Definicja i przykład

Definicja typu

- stwórzmy typ wyliczeniowy i użyjmy go w definicji tablicy:

```
CREATE TYPE typ_zajec AS ENUM('w','c','p');

CREATE TABLE grupa ( ... rodzaj_zajec typ_zajec ... );
```

- próba użycia w porównananiu wartości nowego typu z typem char(1) skończy się niepowodzeniem:

```
SELECT * FROM grupa WHERE rodzaj_zajec='w';
-- wynik pusty i ewentualne ostrzeżenie
-- o niezgodności typów kolumny i stałej;
```

Definicja i przykład

Definicja typu

- stwórzmy typ wyliczeniowy i użyjmy go w definicji tablicy:

```
CREATE TYPE typ_zajec AS ENUM('w','c','p');

CREATE TABLE grupa ( ... rodzaj_zajec typ_zajec ... );
```

- próba użycia w porównananiu wartości nowego typu z typem char(1) skończy się niepowodzeniem:

```
SELECT * FROM grupa WHERE rodzaj_zajec='w';
-- wynik pusty i ewentualne ostrzeżenie
-- o niezgodności typów kolumny i stałej;
```

- do skutecznego wykonania porównania potrzebna jest jawna zmiana typu (CAST, ::); może być tez konieczne podanie metody konwersji typu (tutaj nie jest to potrzebne):

```
SELECT * FROM grupa WHERE rodzaj_zajec::char(1)='w';
-- w wyniku dostajemy wszystkie grupy wykładowe;
```

Struktura serwera bazy danych

Struktura danych na serwerze bazodanowym (w jednej instalacji SZBD):

- bazy danych (`DATABASE`)
  - elementy bazy danych tj. użytkownicy, uprawnienia
  - schematy vel. katalogi (`SCHEMA`)
    - obiekty bazy danych w katalogu, tj. tabela, perspektywa, funkcja,...
- katalogi systemowe, czyli katalogi z metadanymi (`pg_calatog`)
  - `pg_tables`
  - `pg_database`
  - `pg_views`
  - `pg_user`
  - `pg_settings`
  - ...
- przestrzenie tabel (`TABLESPACE`)

CREATE DATABASE

Przykłady:

```
CREATE DATABASE nazwa;
CREATE DATABASE nazwabd OWNER nazwa_wlasciciela;
```

CREATE DATABASE

Przykłady:

```
CREATE DATABASE nazwa;
CREATE DATABASE nazwabd OWNER nazwa_wlasciciela;
```

Składnia polecenia:

```
CREATE DATABASE name -- domyślnie jak current_user
   [ [ WITH ] [ OWNER [=] user_name ] -- domyślnie current_user
          [ TEMPLATE [=] template ]  -- domyślnie template1
           [ ENCODING [=] encoding ]
          [ LC_COLLATE [=] lc_collate ]
          [ LC_CTYPE [=] lc_ctype ]
          [ TABLESPACE [=] tablespace_name ]
          [ CONNECTION LIMIT [=] connlimit ] ] -- domyślnie -1
```

CREATE DATABASE — uwagi

- Do stworzenia bazy danych potrzebne jest uprawnienie `createdb`.
- Do istniejącej bazy łączymy się:
  - `psql`
  - `EXEC SQL CONNECT TO nazwa_bazy;`
  - itd.
- W ramach jednego połączenia mamy swobodny dostęp do jednej bazy.
- Bazę usuwamy poleceniem `DROP DATABASE nazwa_bazy;`

CREATE SCHEMA

`CREATE SCHEMA nazwa_schematu;` — utworzenie nowego schematu, który jest praktycznie tylko przestrzenią nazw;

`SET search_path TO lista_nazwschematów;` — ustawienie ścieżki domyślnie przeszukiwanych schematów;

`SELECT current_schema;` — sprawdzenie aktualnego schematu;

`SELECT * FROM public.grupa;` — korzystanie z nazw obiektów z jawnie podanym schematem.

Użytkownik = rola

Użytkownik ma identyfikator, hasło, zakres uprawnień i prawa:

tworzenia obiektów — jest ich właścicielem,

użytkowania obiektów — może to wynikać z tego, że jest ich właścicielem lub otrzymał te prawa od innych.

Użytkownik = rola

Użytkownik ma identyfikator, hasło, zakres uprawnień i prawa:

- tworzenia obiektów — jest ich właścicielem,
- użytkowania obiektów — może to wynikać z tego, że jest ich właścicielem lub otrzymał te prawa od innych.

Rola to zestaw uprawnień w bazie; tworzenie ról ułatwia zarządzanie uprawnieniami.

CREATE ROLE

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

where option can be:

      SUPERUSER | NOSUPERUSER -- może prawie wszystko
    | CREATEDB | NOCREATEDB -- może tworzyć nowe bazy danych
    | CREATEROLE | NOCREATEROLE -- ma prawo tworzenia innych ról
    | CREATEUSER | NOCREATEUSER -- może tworzyć innych użytkowników
    | INHERIT | NOINHERIT
    | LOGIN | NOLOGIN
    | REPLICATION | NOREPLICATION -- ma prawo wykonywania repliki bd
    | CONNECTION LIMIT connlimit
    | [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'
    | VALID UNTIL 'timestamp'
    | IN ROLE role_name [, ...] -- należy do wskazanych ról
    | IN GROUP role_name [, ...]
    | ROLE role_name [, ...] -- te role zostaną włączone do danej
    | ADMIN role_name [, ...] -- ma prawo przyznawania tej roli
    | USER role_name [, ...]
    | SYSID uid
```

Rodzaje praw do obiektów bazy

TABLE, VIEW — SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER

column TABLE, VIEW — SELECT | INSERT | UPDATE | REFERENCES

SEQUENCE — USAGE | SELECT | UPDATE

DATABASE — CREATE | CONNECT | TEMPORARY | TEMP

FUNCTION — EXECUTE | ALL

LANGUAGE — USAGE | ALL

SCHEMA — CREATE | USAGE

DOMAIN, TYPE — USAGE

Nadawanie praw: GRANT

Polecenie GRANT:

```
GRANT lista_praw ON obiekt TO uprawniony [WITH GRANT OPTION];
```

- `lista_praw` — może być `ALL`;
- `uprawniony` — może być `PUBLIC`;
- `WITH GRANT OPTION` — pozwala obdarowanemu prawem przekazywać je innym (także `WITH GRANT OPTION`;

Nadawanie praw: GRANT

Polecenie GRANT:

```
GRANT lista_praw ON obiekt TO uprawniony [WITH GRANT OPTION];
```

- `lista_praw` — może być `ALL`;
- `uprawniony` — może być `PUBLIC`;
- `WITH GRANT OPTION` — pozwala obdarowanemu prawem przekazywać je innym (także `WITH GRANT OPTION`;

Prawa są sygnowane — obdarowujący ”podpisuje się” pod udzielonym prawem (licencją) i system to zapamiętuje;

Wiele licencji — jeden użytkownik/rola może otrzymać to samo prawo od wielu innych użytkowników;

Nadanie prawa dalej — prawo posiadane `WITH GRANT OPTION` przekazane innym jest sygnowane przez oryginalnego ”dawcę”, ale pamiętana jest także ścieżka otrzymania prawa.

Odbieranie praw: REVOKE

Polecenie REVOKE:

```
REVOKE [GRANT OPTION FOR] lista_praw ON obiekt
FROM  uprawniony [RESTRICT|CASCADE];
```

- `uprawniony` — może być `PUBLIC`;
- `REVOKE GRANT OPTION FOR` — pozwala wycofać samo prawo przekazywania praw innym;
- `odebranie "tylko" prawa` — zabiera prawo i prawo przekazywania go dalej.

Odbieranie praw: REVOKE

Polecenie REVOKE:

```
REVOKE [GRANT OPTION FOR] lista_praw ON obiekt
FROM  uprawniony [RESTRICT|CASCADE];
```

- `uprawniony` — może być `PUBLIC`;
- `REVOKE GRANT OPTION FOR` — pozwala wycofać samo prawo przekazywania praw innym;
- `odebranie "tylko" prawa` — zabiera prawo i prawo przekazywania go dalej.

PUBLIC — odebranie prawa "roli" PUBLIC zamyka określony dostęp do obiektu bez posiadania sygnowanego prawa;

Wycofanie kaskadowe — odbierając prawo trzeba uporządkować konsekwencje tego, że było przyznane; **trzeba** wycofać wszystkie prawa przekazane dalej w oparciu o to prawo;

Wycofanie GRANT OPTION — wycofanie tylko GRANT OPTION powoduje wycofanie praw nadanych dalej, ale zachowuje prawo obdarowanego bezpośrednio licencją.

Ścieżka nadania — prawa trzeba zabierać tym użytkownikom/rolom, którym się je nadało (nie można iść na skróty).

Prawa domyślne

Tabele i schematy — domyślnie PUBLIC nie ma żadnych praw,

Funkcje i języki — domyślnie PUBLIC ma prawo EXECUTE i USAGE,

Prawa właściciela — nie ma prawa DROP; to jest (niezbywalne) prawo właściciela obiektu i prawo superusera.

Zarządzanie rolami

Przypisanie roli:

```
GRANT rola TO użytkownik;
GRANT WITH ADMIN OPTION rola TO użytkownik;
```

Odebranie roli:

```
REVOKE rola FROM użytkownik;
REVOKE ADMIN OPTION FOR rola FROM użytkownik;
```

CREATE INDEX

Indeks:

przyśpiesza: wyszukiwanie, złączenia, kontrolę unikalności i innych warunków;

spowalania: wstawianie, usuwanie i modyfikację indeksowanych krotek.

CREATE INDEX

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ name ]
   ON table_name [ USING method ]
   ( { column_name | ( expression ) }
       [ COLLATE collation ] [ opclass ] [ ASC | DESC ]
       [ NULLS { FIRST | LAST } ] [, ...] )
   [ WITH ( storage_parameter = value [, ... ] ) ]
   [ TABLESPACE tablespace_name ]
   [ WHERE predicate ]
```

Metody indeksowania

B-tree — indeksowanie wg klucza, dostosowane do dużych danych przechowowanych na dysku; zbalansowane drzewo poszukiwań o bardzo "grubych"wierzchołkach i bardzo dużej armości (w związku z tym płytkie);

hash — rozrzucanie indeksowanych danych wg funkcji haszującej do "dużych"kubełków; specyficzne metody obsługi przepełnienia kubełków (podwajanie, haszowanie linioowe, haszowanie rozszerzalne);

R-tree — indeksowanie obiektów 2-wymiarowych ”obrysowanych” prostokątami;

inverter file — indeksowanie tesktów tak, by łatwo było wyszukiwać całe słowa;

suffix tree — indeksowanie tekstów tak, by łatwo było wyszukiwać wszystkie podsłowa.

CREATE SEQUENCE

Sekwencja służy do automatycznego generowania unikalnych wartości.

CREATE SEQUENCE

```
CREATE [ TEMPORARY | TEMP ] SEQUENCE name
    [ INCREMENT [ BY ] increment ]
    [ MINVALUE minvalue | NO MINVALUE ]
    [ MAXVALUE maxvalue | NO MAXVALUE ]
    [ START [ WITH ] start ] [ CACHE cache ] [ [ NO ] CYCLE ]
    [ OWNED BY { table_name.column_name | NONE } ]
```

`nexttval('nazwa_sekwencji');` — zwraca ostatnią wartość sekwencji i zwiększa ją;

`settval('nazwa_sekwencji', wartość);` — pozwala ustawić ostatnią wartość sekwencji,

`currval('nazwa_sekwencji');` — zwraca ostatnią wartość sekwencji bez zwiększania;

Pole SERIAL

- Dla kolumny typu SERIAL system automatycznie tworzy sekwencję, której właścicielem jest ta kolumna.
- Samodzielne stworzenie sekwencjii przypisanie jej do kolumny też jest możliwe:

```
CREATE SEQUENCE s1;
SELECT setval('s1',max(kod_uz)) FROM uzytkownik;
INSERT INTO uzytkownik(kod_uz,imie,nazwisko)
  VALUES (nextval('s1'),'Anna','Abacka');

CREATE SEQUENCE s1 OWNED BY uzytkownik.kod_uz;
SELECT setval('s1',max(kod_uz)) FROM uzytkownik;
INSERT INTO uzytkownik(imie,nazwisko)
  VALUES (nextval('Anna','Abacka');
```

CREATE VIEW

Tworzenie perspektywy:

```
CREATE VIEW name [ ( column_name [, ...] ) ]
    AS query
    [ WITH [ CASCADED | LOCAL ] CHECK OPTION ]
```

tabela wirtualna — w zapytaniach SELECT można ją traktować jak dowolną tabelę bazy danych;

nazwane zapytanie — można ją uważać za zapytanie zapisane pod nazwą, które jest obliczane zawsze, gdy użytkownik się do niego odwoła.

Perspektywy modyfikowalne

Przykład:

```
CREATE VIEW niezapisani AS
  SELECT * FROM uzytkownik
  WHERE semestr IS NOT NULL AND
              NOT EXISTS (SELECT * FROM wybor
                          WHERE wybor.kod_uz=uzytkownik.kod_uz);
```

1. Operacja: `DELETE FROM niezapisani WHERE kod_uz=123;` powinna usunąć użytkownika o kodzie 123 z tabeli uzytkownicy, co w efekcie spowoduje, że zniknie on z tabeli niezapisani.
2. `WITH CHECK OPTION` w definicji perspektywy niezapisani powoduje sprawdzenie, czy 123 spełnia warunek selekcji do tej perspektywy — jeśli nie spełnia, to usunięcie jest blokowane.

Obliczanie perspektyw:

1. *ad hoc* — za każdym razem, na żądanie,
2. *materializowanie perspektyw* — przez system
3. *materializowanie perspektyw* — samodzielnie; np. srwynik za zadanie w tabeli Zad:

Obliczanie perspektyw:

1. *ad hoc* — za każdym razem, na żądanie,
2. *materializowanie perspektyw* — przez system
3. *materializowanie perspektyw* — samodzielnie; np. srwynik za zadanie w tabeli Zad:
   1. dodaj do tabeli Zad kolumny suma_wynikow i liczba_zgloszen;
   2. odebierz"zwykły" użytkownikom prawa do tych kolumn;
   3. wstępnie wypełnij kolumny dla każdego zadania sumą wyników i liczbą zgłoszeń;
   4. napisz wyzwalacze, które przy dodaniu/usunięciu/modyfikacji wyniku spowodują aktualizacje wartości w kolumnach suma_wynikow i liczba_zgloszen tylko dla odpowiednich zadań dotkniętych modyfikacją w tabeli wynik;
   5. dostęp do średniego wyniku możemy zagwarantować sobie na jeden z dwóch sposobów:
      1. po zmianach w suma_wynikow i liczba_zgloszen wyzwalacz automatycznie oblicza srwynik dla danego zadania
      2. usuwamy redundantną kolumnę srwynik; definiujemy perspektywę zawierającą wszystkie kolumny tabeli zad i dodatkowo kolumnę srwynik wyliczaną jako iloraz suma_wynikow/liczba_zgloszen (o ile liczba_zgloszen<>0)

Zastosowania perspektyw:

1. upraszczanie zapytań,
2. unikanie wielokrotnego pisania tych samych zapytań
3. utworzenie ”obrazu” bazy danych dla konkretnej grupy użytkowników
4. zasłonięcie zmienionego schematu bazy danych przez perspektywy naśladujące stary schemat.

CREATE ASSERTION

Przykład asercji:

```
CREATE TABLE powiadom (
nick varchar(8) REFERENCES Osoba,
id int REFERENCES Zad);

CREATE ASSERTION powiadom_tylko_10 AS CHECK
(NOT EXISTS (SELECT * FROM powiadom
                    GROUP BY nick
                   HAVING count(*)>10));
```

1. Asercja odpowiada za zapewnienie, że warunek jest zawsze spełniony w bazie danych.
2. Asercje nie są implementowane.
3. Zamiast asercji trzeba stosować wyzwalacze, by zapewnić działanie bazy danych równoważne asercji.

CREATE TRIGGER (składnia standardowa)

Tworzenie wyzwalacza:

```
CREATE TRIGGER nazwa { BEFORE | AFTER | INSTEAD OF }
{ zdarzenie [ OR ... ] }
   ON tabela[.kolumna]
REFERENCING OLD [ROW] AS stary
REFERENCING NEW [ROW] AS nowy
WHEN warunek
BEGIN ... END
[ FOR [ EACH ] { ROW | STATEMENT } ]
```

W wyzwalaczu określamy:

- zdarzenie — INSERT, UPDATE lub DELETE dla tabeli (kolumny);
- warunek — gdy występuje wyzwalacz zostaje uruchomiony;
- moment odpalenia wyzwalacza — przed/po/zamiast zdarzenia ;
- identyfikatory dla starej i/lub nowej wersji zmienianej w wyniku zdarzenia krotki/tabeli;
- granulację wyzwalacza, czyli czy należy go uruchomić po zmianie każdej krotki w ramach jednej instrukcji FOR EACH ROW: (old row/new row) czy raz dla całej instrukcji i całej tabeli zmienionej instrukcją FOR EACH STATEMENT (old/new)

Wyzwalacze — uwagi:

1. Wyzwalacz może wywołać zdarzenia, z którymi są związane wyzwalacze — one także zostaną uruchomieone.
2. Operacje wykonywane w ramach wyzwalacza należą do tej samej transakcji, co instrukcja, która wyzwalacz wywołała.
3. Jeśli nie powiedzie się którakolwiek z operacji wywołanych przez wyzwalacz, wszystkie działania wyzwalacza wraz z instrukcją początkową, która go zainicjowała, są wycofywane.
4. Poprawne zdefiniowanie zbioru wyzwalaczy "pilnujących"poprawności danych w bazie wymaga uważnego zlokalizowania wszystkich zdarzeń naruszających poprawność danych. Np. chcąc zdefiniować wyzwalacze analogiczne do asercji `powiadom_tylko_10`, trzeba zdefiniować wyzwalacze:
   1. dla operacji dodawania do tabeli powiadom
   2. dla operacji modyfikacji w tabeli powiadom (obu kolumn)

   Gdybyśmy dodatkowo chcieli różnicować limit możliwych do śledzenia zadań w zależności od roli użytkownika (admin,naucz,uczen,...), to trzeba by dodać wyzwalacz do modyfikacji roli, bo np. “degradacja” z nauczyciela na ucznia mogłaby obniżyć dopuszczalny limit i wymagać zablokowania lub usunięcia kilku śledzonych zadań.

CREATE TRIGGER (składnia PostgreSQL)

Składnia wyzwalaczy w dialektach SQL istotnie różnią się od składni standardowej.

Tworzenie wyzwalacza PostgreSQL:

```
CREATE TRIGGER nazwa { BEFORE | AFTER }
   { zdarzenie [ OR ... ] }
   ON tabela [ FOR [ EACH ] { ROW | STATEMENT } ]
   EXECUTE PROCEDURE nazwa_funkcji(argumenty)
```