
Bazy Danych

Wykład p.t.

DDL

Data Definition Language

**Definiowanie obiektów pomocniczych:
domeny, sekwencje, ograniczenia, widoki**

Antoni Ligęza

ligeza@agh.edu.pl

<http://galaxy.uci.agh.edu.pl/~ligeza>

Wykorzystano materiały:

[http:](http://www.postgresql.org/docs/8.3/interactive/index.html)

[//www.postgresql.org/docs/8.3/interactive/index.html](http://www.postgresql.org/docs/8.3/interactive/index.html)

Instrukcje DDL – CREATE (28)

- CREATE AGGREGATE – tworzenie nowej funkcji agregacji,
- CREATE CAST – definiowanie nowej funkcji konwersji typów,
- CREATE CONSTRAINT TRIGGER – tworzenie procedury wyzwalanej jako ograniczenia,
- CREATE CONVERSION – definiowanie nowej funkcji konwersji kodowania,
- CREATE DATABASE – tworzenie bazy danych,
- CREATE DOMAIN – tworzenie dziedziny,
- CREATE FUNCTION – tworzenie nowej funkcji,
- CREATE GROUP – tworzenie nowej grupy,
- CREATE INDEX – tworzenie indeksu,
- CREATE LANGUAGE – definiowanie nowego języka dla funkcji,
- CREATE OPERATOR – definiowanie nowego operatora użytkownika,
- CREATE OPERATOR CLASS – definiowanie nowej klasy operatora,
- CREATE OPERATOR FAMILY – definiowanie nowej rodziny operatorów,
- CREATE ROLE – definiowanie nowej roli,
- CREATE RULE – definiowanie nowej reguły,
- CREATE SCHEMA – definiowanie nowego schematu,
- CREATE SEQUENCE – definiowanie nowej sekwencji,
- CREATE TABLE – tworzenie tabel,

- CREATE TABLESPACE – definiowanie nowej przestrzeni tablic,
- CREATE TABLE AS – tworzenie tabel jako wyniku instrukcji SELECT,
- CREATE TEXT SEARCH CONFIGURATION – definiowanie nowej konfiguracji dla przeszukiwania tekstu,
- CREATE TEXT SEARCH DICTIONARY – definiowanie nowego słownika dla przeszukiwania tekstów,
- CREATE TEXT SEARCH PARSER – definiowanie nowego parsera dla przeszukiwania tekstów,
- CREATE TEXT SEARCH TEMPLATE – definiowanie nowego wzorca dla przeszukiwania tekstów,
- CREATE TRIGGER – tworzenie nowego wyzwalacza,
- CREATE TYPE – definiowanie nowego typu danych,
- CREATE USER – tworzenie nowego użytkownika,
- CREATE VIEW – tworzenie nowej perspektywy (widoku).

Instrukcje DDL – ALTER (24)

- ALTER AGGREGATE – modyfikacji funkcji agregacji,
- ALTER CONVERSION – modyfikacja funkcji konwersji,
- ALTER DATABASE – modyfikacja bazy danych,
- ALTER DOMAIN – modyfikacja dziedziny,
- ALTER FUNCTION – modyfikacja funkcji,
- ALTER GROUP – modyfikacja grupy (dodanie lub usunięcie użytkownika),
- ALTER INDEX – modyfikacja indeksu,
- ALTER LANGUAGE – modyfikacja języka,
- ALTER OPERATOR – modyfikacja operatora,
- ALTER OPERATOR CLASS – modyfikacja klasy operatora,
- ALTER OPERATOR FAMILY – modyfikacja rodziny operatorów,
- ALTER ROLE – modyfikacja roli,
- ALTER SCHEMA – modyfikacja schematu,
- ALTER SEQUENCE – modyfikacja sekwencji,
- ALTER TABLE – modyfikacja struktury tabeli,
- ALTER TABLESPACE – modyfikacja przestrzeni tablic,
- ALTER TEXT SEARCH CONFIGURATION – modyfikacja konfiguracji przeszukiwania tekstów,
- ALTER TEXT SEARCH DICTIONARY – modyfikacja słownika przeszukiwania tekstów,

- ALTER TEXT SEARCH PARSER – modyfikacja parsera przeszukiwania tekstów,
- ALTER TEXT SEARCH TEMPLATE – modyfikacja wzorca przeszukiwania tekstów,
- ALTER TRIGGER – modyfikacja triggera,
- ALTER TYPE – modyfikacja typu,
- ALTER USER – modyfikacja konta użytkownika,
- ALTER VIEW – modyfikacja widoku (perspektywy),

Instrukcje DDL – DROP (27)

- DROP AGGREGATE – usuwanie funkcji agregacji,
- DROP CAST – usuwanie funkcji konwersji typów,
- DROP CONVERSION – usuwanie funkcji konwersji znakowej,
- DROP DATABASE – usuwanie bazy danych,
- DROP DOMAIN – usuwanie dziedziny,
- DROP FUNCTION – usuwanie funkcji,
- DROP GROUP – usuwanie grupy,
- DROP INDEX – usuwanie indeksu,
- DROP LANGUAGE – usuwanie zdefiniowanego języka dla funkcji,
- DROP OPERATOR – usuwanie zdefiniowanego operatora użytkownika,
- DROP OPERATOR CLASS – usuwanie klasy operatora,
- DROP OPERATOR FAMILY – usuwanie rodziny operatorów,
- DROP OWNED – usuwanie obiektów przypisanych do roli,
- DROP ROLE – usuwanie roli,
- DROP RULE – usuwanie zdefiniowanej reguły,
- DROP SCHEMA – usuwanie schematu,
- DROP SEQUENCE – usuwanie zdefiniowanej sekwencji,
- DROP TABLE – usuwanie tabeli,
- DROP TABLESPACE – usuwanie przestrzeni tabel,
- DROP TEXT SEARCH CONFIGURATION – usuwanie konfiguracji dla przeszukiwania tekstów,

- DROP TEXT SEARCH DICTIONARY – usuwanie słownika przeszukiwania tekstów,
- DROP TEXT SEARCH PARSER – usuwanie parsera przeszukiwania tekstów,
- DROP TEXT SEARCH TEMPLATE – usuwanie wzorca przeszukiwania tekstów,
- DROP TRIGGER – usuwanie wyzwalacza,
- DROP TYPE – usuwanie zdefiniowanego typu danych,
- DROP USER – usuwanie użytkownika,
- DROP VIEW – usuwanie perspektywy.

DDL – inne instrukcje oraz DML

- COMMENT ON – dodaje komentarz do obiektu bazy danych,
- COPY – kopiuje dane do lub z tabeli,
- DELETE FROM – usuwanie danych z tabeli,
- GRANT – definiowanie uprawnień,
- INSERT INTO – wpisywanie danych do tablicy,
- RESET – przywraca domyślną wartość parametru,
- REVOKE – odbiera uprawnienia użytkownikom,
- SET – ustawia parametry startowe bazy danych,
- UPDATE – modyfikacja danych w tabeli.

Idea sekwencji: typ serial, zastosowanie

Sekwencja = licznik; każde odwołanie generuje nową wartość.

Sekwencje bazują na typie `bigint`

(8-byte integer (-9223372036854775808 to 9223372036854775807)).

Sekwencje to obiekty służące do generowania nowych wartości licznika typu integer. Podstawowe zastosowanie to automatyczne tworzenie identyfikatorów rekordów w bazie. Typ `SERIAL` niejawnie definiuje sekwencje.

Sekwencja jest jednowierszową tabelą (specjalną) o nazwie sekwencji.

Rodzaje liczników sekwencji:

- inkrementalne vs. losowe (random),
- liniowe (skończone) vs. cyrkularne,
- skok o 1 vs skok o kwant,
- rosnące vs. malejące,
- liczby dodatnie vs. liczby ujemne.

Realizacja: typ `serial`. Przykład: `serial-create-insert.sql`

```
CREATE TEMP TABLE sprac
(
  ID_prac serial,
  Nazwisko varchar(32) NOT NULL,
  Imie varchar(16) NOT NULL,
  Data_ur date NOT NULL,
  Dzial char(5) NOT NULL,
  Stanowisko varchar(24),
  Pobory numeric(8,2),
  CONSTRAINT prac_pk PRIMARY KEY(ID_prac)
);
```

Ładowanie rekordów:

```
INSERT INTO sprac (Nazwisko, Imie, Data_ur, Dzial, Stanowisko, Pobory)
VALUES ('Kowal', 'Adam', '15/12/1989', 'PD303', 'robotnik', 1500);
INSERT INTO sprac (Nazwisko, Imie, Data_ur, Dzial, Stanowisko, Pobory)
VALUES ('Kowalik', 'Artur', '13/12/1998', 'PR202', 'majster', 1500);
INSERT INTO sprac (Nazwisko, Imie, Data_ur, Dzial, Stanowisko, Pobory)
VALUES ('Kowalewski', 'Adam', '15/11/1989', 'PR202', 'kierownik', 3500);
INSERT INTO sprac (Nazwisko, Imie, Data_ur, Dzial, Stanowisko, Pobory)
VALUES ('Kowalczyk', 'Amadeusz', '17/12/1998', 'PD303', 'robotnik', 1000);
INSERT INTO sprac (Nazwisko, Imie, Data_ur, Dzial, Stanowisko, Pobory)
VALUES ('Kowalski', 'Antoni', '15/12/1999', 'PD303', 'kierownik', 4500);
INSERT INTO sprac (Nazwisko, Imie, Data_ur, Dzial, Stanowisko, Pobory)
VALUES ('Kowalowski', 'Alojzy', '15/11/1998', 'PK101', 'kierownik', 2500);
INSERT INTO sprac (Nazwisko, Imie, Data_ur, Dzial, Stanowisko, Pobory)
VALUES ('Kowalczyk', 'Adam', '12/11/1998', 'PR202', 'majster', 2500);
```

Nie należy wstawiać 'ręcznie' wartości pól typu serial!

Jawne definiowanie sekwencji

Sekwencje można definiować jawnie instrukcją 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.column | NONE } ];
```

TEMPORARY | TEMP – na potrzeby bieżącej sesji, potem niszczone. Tworzona w specjalnym schemacie. INCREMENT [BY] increment – krok zmiany sekwencji; może być ujemny. NO MINVALUE – lub nic: defaultowe wartości to: 1 oraz $-2^{63} - 1$ (dla malejących). NO MAXVALUE – lub nic: defaultowe wartości to: $2^{63} - 1$ oraz -1 (dla rosnących). START [WITH – wartość początkowa. CACHE – liczba wartości prealokowanych w pamięci; default = 1. [NO] CYCLE – możliwość zapętlenia; default = NO. OWNED BY table.column | NONE] – przypisanie do kolumny; usuwanie kaskadowe wraz z kolumną.

Uwaga: sekwencje nie gwarantują domyślnie własności UNIQUE!

Uwaga: pola sekwencji dają się ustawiać ręcznie – można ich wartość wymusić poprzez INSERT INTO! Może to spowodować *zmylenie sekwencji*.

Przykłady: sequences-examples.sql

```
CREATE SEQUENCE prac_seq
      INCREMENT 10
      MINVALUE 0
      MAXVALUE 1000000
      START 10;
```

```
CREATE TEMP SEQUENCE seq_1
    INCREMENT 17
    MINVALUE 3
    MAXVALUE 1000000
    START 13;
```

```
CREATE TEMP SEQUENCE seq_2
    INCREMENT -7
    MINVALUE -177
    MAXVALUE 178
    START 17;
```

```
CREATE TEMP SEQUENCE seq_3
    INCREMENT 3
    MINVALUE 3
    MAXVALUE 10
    START 3
    CYCLE;
```

Definiowanie i użycie sekwencji

```
CREATE TABLE tablename (  
    colname SERIAL  
);
```

-- is equivalent to specifying:

```
CREATE SEQUENCE tablename_colname_seq;  
CREATE TABLE tablename (  
    colname integer NOT NULL  
    DEFAULT nextval('tablename_colname_seq')  
);  
ALTER SEQUENCE tablename_colname_seq  
    OWNED BY tablename.colname;
```

Odczytywanie i ustawianie sekwencji:

```
SELECT * FROM name;  
SELECT nextval('name_seq');  
SELECT currval('name_seq');  
SELECT lastval(); %dotyczy bieżącej sekwencji  
SELECT setval('name_seq',bigint\_value);
```

Definiowanie i użycie sekwencji – przykłady

```
CREATE SEQUENCE serial START 101;
```

```
SELECT nextval('serial');
```

```
nextval
-----
      101
```

```
SELECT nextval('serial');
```

```
nextval
-----
      102
```

```
INSERT INTO distributors VALUES
          (nextval('serial'), 'nothing');
```

Update the sequence value after a COPY FROM:

```
BEGIN;
COPY distributors FROM 'input_file';
SELECT setval('serial', max(id)) FROM distributors;
END;
```

```
CREATE SEQUENCE shipments_ship_id_seq
          START 200 INCREMENT 1;
```

```
CREATE
```

```
SELECT nextval ('shipments_ship_id_seq');
```

```
nextval
-----
      200
(1 row)
```

```
INSERT INTO shipments VALUES  
    (nextval('shipments_ship_id_seq'), 107,  
     '0394800753', 'now');
```

```
CREATE TABLE shipments (  
    id integer NOT NULL DEFAULT  
    nextval('shipments_ship_id_seq'),  
    customer_id integer,  
    isbn text,  
    ship_date timestamp);
```

Funkcje obsługi sekwencji

Dostępne są cztery (pięć) funkcje obsługi sekwencji: `currval()`, `nextval()`, `setval()`, `lastval()`.

```
currval(regclass)
```

```
lastval()
```

```
nextval(regclass)
```

```
setval(regclass, bigint)
```

```
setval(regclass, bigint, boolean)
```

Użycie funkcji: `SELECT <funkcja>;`

```
SELECT currval('seq_1');
```

```
SELECT nextval('seq_1');
```

```
SELECT lastval();
```

Użycie setval:

```
SELECT setval('foo', 42);           -- Next nextval will return 43
```

```
SELECT setval('foo', 42, true);    -- Same as above
```

```
SELECT setval('foo', 42, false);   -- Next nextval will return 42
```


Domeny użytkownika

Użytkownik może definiować własne domeny. Są to dziedziny atrybutów.

Domena:= typ + ograniczenie

Zastosowanie: wyabstrahowanie definicji (w jednym miejscu) dostępnej dla wielu tablic.

Syntax:

```
CREATE DOMAIN name [ AS ] data_type
    [ DEFAULT expression ]
    [ constraint [ ... ] ]
```

Definicja ograniczeń:

```
[ CONSTRAINT constraint_name ]
{ NOT NULL | NULL | CHECK (expression) }
```

Przykład:

```
CREATE DOMAIN us_postal_code AS TEXT
CHECK (
    VALUE ~ '^\\d{5}$'
OR VALUE ~ '^\\d{5}-\\d{4}$'
);
```

```
CREATE TABLE us_snail_addy (
    address_id SERIAL PRIMARY KEY,
    street1 TEXT NOT NULL,
    street2 TEXT,
    street3 TEXT,
    city TEXT NOT NULL,
    postal us_postal_code NOT NULL
);
```

Warunek CHECK może zawierać spójniki logiczne (AND, OR, NOT).

Definiowanie domen: przykład

Przykład: domain-create-insert.sql

```
CREATE DOMAIN postal_code AS char(6)
CHECK (VALUE ~ '[0-9]{2}-[0-9]{3}' OR NULL);
CREATE TEMP TABLE dprac
(
  ID_prac serial,
  Nazwisko varchar(32) NOT NULL,
  Imie varchar(16) NOT NULL,
  Data_ur date NOT NULL,
  Dzial char(5) NOT NULL,
  Stanowisko varchar(24),
  Pobory numeric(8,2),
  Kod          postal_code,
  CONSTRAINT prac_pk PRIMARY KEY(ID_prac)
);
```

Przykład: domain-create-insert-dzial.sql

```
CREATE DOMAIN dzial_dom AS varchar(10)
CHECK (VALUE IN ('PD303', 'PR202', 'PK101'));
CREATE TEMP TABLE dprac
( ID_prac serial,
  Nazwisko varchar(32) NOT NULL,
  Imie varchar(16) NOT NULL,
  Data_ur date NOT NULL,
  Dzial char(5) NOT NULL,
  Stanowisko varchar(24),
  Pobory numeric(8,2),
  CONSTRAINT prac_pk PRIMARY KEY(ID_prac));
```

Typ enum

Użytkownik może zdefiniować także własne typy wyliczalne.

```
CREATE TYPE mood AS ENUM ('sad', 'ok', 'happy');
```

Przykład zastosowania:

```
CREATE TYPE mood AS ENUM ('sad', 'ok', 'happy');
CREATE TABLE person (
    name text,
    current_mood mood
);
INSERT INTO person VALUES ('Moe', 'happy');
SELECT * FROM person WHERE current_mood = 'happy';
 name | current_mood
-----+-----
 Moe  | happy
(1 row)
```

Typ enum ma zdefiniowany porządek!

```
INSERT INTO person VALUES ('Larry', 'sad');
INSERT INTO person VALUES ('Curly', 'ok');
SELECT * FROM person WHERE current_mood > 'sad';
 name | current_mood
-----+-----
 Moe  | happy
 Curly | ok
(2 rows)

SELECT * FROM person
    WHERE current_mood > 'sad' ORDER BY current_mood;

 name | current_mood
```

-----+-----

```
Curly | ok
Moe    | happy
(2 rows)
```

```
SELECT name FROM person
  WHERE current_mood = (SELECT MIN(current_mood) FROM person);
name
```

```
Larry
(1 row)
```

Tablice słownikowe a tym enum

Tablica słownikowa: tablica zawierająca listę *legalnych* danych.

Tablica słownikowa może być:

- jednokolumnowa (tylko dane),
- dwukolumnowa (dane i ich kody),
- wielokolumnowa (dane, kody, opisy, markery).

Zastosowania:

- generowanie list wyboru (na poziomie klienta/interfejsu),
- kontrola poprawności wprowadzanych danych,
- dokumentowanie systemu.

Zastosowanie: poprzez zdefiniowanie łączenia klucz-klucz obcy.

W przeciwieństwie do typu `enum` tabele słownikowe mogą być modyfikowane w trakcie pracy z bazą! (niezależność logiczna).

Typy danych definiowane przez użytkownika

Użytkownik może definiować własne typy danych.

Syntax:

```
CREATE TYPE name AS
    ( attribute_name data_type [, ... ] )
```

```
CREATE TYPE name AS ENUM
    ( 'label' [, ... ] )
```

Pierwsza forma definiuje typ rekordu (row). Druga typ wyliczalny. Jest także możliwość definiowania typów podstawowych (vide: manual PostgreSQL).

Przykład:

```
CREATE TYPE complex AS (
    r      double precision,
    i      double precision
);
```

```
CREATE TYPE inventory_item AS (
    name          text,
    supplier_id   integer,
    price         numeric
);
```

Stosowanie:

```
CREATE TABLE on_hand (
    item          inventory_item,
    count        integer
);
```

```
INSERT INTO on_hand VALUES (ROW('fuzzy dice', 42, 1.99), 1000);
```

Zawsze gdy tworzona jest tabela, jako efekt uboczny tworzony jest także typ jej wierszy; może być stosowany jako deklaracja typu.

```
CREATE TABLE inventory_item (  
    name                text,  
    supplier_id        integer REFERENCES suppliers,  
    price               numeric CHECK (price > 0)  
);
```

Dostęp do danych: wyrażenia kropkowe.

```
SELECT (item).name  
    FROM on_hand  
    WHERE (item).price > 9.99;
```

```
SELECT (on_hand.item).name  
    FROM on_hand  
    WHERE (on_hand.item).price > 9.99;
```

Nawiasy () pozwalają odróżnić typ od nazwy tabeli.

Typy tablicowe: zagnieżdżanie tablic

Użytkownik może definiować tablice zagnieżdżone.

Rozmiar tablic może być ograniczony *explicite*.

```
CREATE TABLE sal_emp (
    name          text,
    pay_by_quarter integer[],
    schedule      text[][]
);
```

```
CREATE TABLE tictactoe (
    squares integer[3][3]
);
```

Ładowanie danych:

```
INSERT INTO sal_emp
VALUES ('Bill',
       '{10000, 10000, 10000, 10000}',
       '{{"meeting", "lunch"}, {"training", "presentation"}}');
```

```
INSERT INTO sal_emp
VALUES ('Carol',
       '{20000, 25000, 25000, 25000}',
       '{{"breakfast", "consulting"}, {"meeting", "lunch"}}');
```

```
SELECT * FROM sal_emp;
 name |          pay_by_quarter          |          schedule
-----+-----+-----
 Bill | {10000,10000,10000,10000} | {{meeting,lunch},{training,presentation}}
 Carol | {20000,25000,25000,25000} | {{breakfast,consulting},{meeting,lunch}}
(2 rows)
```

Deklaracja tablic *explicite* (ARRAY):


```
INSERT INTO sal_emp
VALUES ('Bill',
ARRAY[10000, 10000, 10000, 10000],
ARRAY[['meeting', 'lunch'], ['training', 'presentation']]);
```

```
INSERT INTO sal_emp
VALUES ('Carol',
ARRAY[20000, 25000, 25000, 25000],
ARRAY[['breakfast', 'consulting'], ['meeting', 'lunch']]);
```

Definiowanie tabel

Definicja tabeli – zasada tworzenia

CREATE TABLE – podstawowa komenda DDL: tworzenie tablicy. Po CREATE TABLE podajemy nazwę tablicy a w nawiasach argumenty rozdzielone przecinkami – tworzą one definicje kolumn. Argument to para: <nazwa kolumny> <typ danych> lub trójka <nazwa kolumny> <typ danych> <ograniczenie>; definicja ograniczeń dotyczy danej kolumny. W definicji tabeli mogą wystąpić więzy integralności dla całej tabeli, po definicji pól.

```
CREATE TABLE <nazwa_tablicy>
(
    <A_1> <typ> <ograniczenie>,
    <A_2> <typ> <ograniczenie>,
    ...
    <A_n> <typ> <ograniczenie>,
    CONSTRAINT <nazwa_1> <typ>(<definicja>),
    ...
    CONSTRAINT <nazwa_k> <typ>(<definicja>)
);
```

```
CREATE TABLE books_tab
(
    book_id    serial,
    author     varchar(32),
    title      text          NOT NULL,
    editor     varchar(64),
    place      varchar(32),
    year       integer,
    CONSTRAINT books_pk    PRIMARY KEY(book_id)
);
```

Przykłady

```
CREATE TABLE films (  
    code          char(5) CONSTRAINT firstkey PRIMARY KEY,  
    title         varchar(40) NOT NULL,  
    did           integer NOT NULL,  
    date_prod     date,  
    kind          varchar(10),  
    len           interval hour to minute  
);  
CREATE TABLE distributors (  
    did          integer PRIMARY KEY DEFAULT nextval('serial'),  
    name         varchar(40) NOT NULL CHECK (name <> '')  
);  
CREATE TABLE distributors (  
    did          integer,  
    name         varchar(40)  
    CONSTRAINT con1 CHECK (did > 100 AND name <> '')  
);  
CREATE TABLE films (  
    code          char(5),  
    title         varchar(40),  
    did           integer,  
    date_prod     date,  
    kind          varchar(10),  
    len           interval hour to minute,  
    CONSTRAINT code_title PRIMARY KEY(code,title)  
);  
CREATE TABLE distributors (  
    name         varchar(40) DEFAULT 'Luso Films',  
    did          integer DEFAULT nextval('distributors_serial'),  
    modtime      timestamp DEFAULT current_timestamp  
);
```

Definiowanie tablic – pełne możliwości

Źródło: <http://www.postgresql.org/docs/8.3/interactive/sql-createtable.html>

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } ]
    TABLE table_name ( [
        { column_name data_type [ DEFAULT default_expr ]
        [ column_constraint [ ... ] ]
        | table_constraint
        | LIKE parent_table [ { INCLUDING | EXCLUDING }
            { DEFAULTS | CONSTRAINTS | INDEXES } ] ... }
        [, ... ]
    ] )
[ INHERITS ( parent_table [, ... ] ) ]
[ WITH ( storage_parameter [= value] [, ... ] )
| WITH OIDS | WITHOUT OIDS ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ TABLESPACE tablespace ]
```

TEMPORARY | TEMP – tablica tymczasowa, dla bieżącej sesji.

GLOBAL | LOCAL – bez efektu (dla zachowania kompatybilności).

DEFAULT – wartość domyślna, dowolne wyrażenie obliczalne.

LIKE – kopiowanie wszystkich kolumn z parent_table (nazwy, typy, NOT NULL).

INHERITS – jak wyżej, ale nowa tabela jest pozostaje zależna od pierwowzoru (propagacja modyfikacji schematu).

WITH – FILLFACTOR, 10-100 (WITH | WITHOUT OIDS – deklaracja identyfikatorów).

ON COMMIT – specyfikacji akcji na końcu transakcji (dla TEMP).

TABLESPACE – specyfikacja innej niż domyślna przestrzeni tabel.

Ograniczenia w bazach danych: więzy integralności

Rodzaje integralności:

- wewnętrzna,
- zewnętrzna.

W bazach danych integralność (wewnętrzna) danych zapewniana jest na kilka sposobów:

- schemat i strukturę tabel (1NF),
- więzy lokalne: ograniczenia na poziomie kolumn (pól),
- więzy lokalne: ograniczenia na poziomie rekordu,
- więzy globalne: ograniczenia na poziomie tablicy,
- więzy globalne: ograniczenia na poziomie między-tablicowym.

Integralność zewnętrzna: problemy:

- dane nieaktualne,
- dane temporalne,
- dane nieprecyzyjne,
- brak danych,
- dane niespójne z rzeczywistością,
- dane niespójne z modelem logicznym,
- dane fałszywe,
- ...

Definiowanie tablic – ograniczenia na poziomie kolumn

```
[ CONSTRAINT constraint_name ]
{ NOT NULL | NULL |
  UNIQUE index_parameters |
  PRIMARY KEY index_parameters |
  CHECK ( expression ) |
  REFERENCES reftable [ ( refcolumn ) ]
    [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
    [ ON DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE ]
[ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

CONSTRAINT – nazwane ograniczenie; lepsza diagnostyka.

NOT NULL | NULL – nie dopuszcza wartości NULL. Default=NULL.

UNIQUE – unikalne wartości w kolumnie.

PRIMARY KEY – definicja klucza podstawowego (UNIQUE + NOT NULL).

CHECK – sprawdzanie warunku (TRUE lub UNKNOWN; FALSE nie).

REFERENCES – tablica odniesienia z kluczem.

MATCH FULL | PARTIAL | SIMPLE – wszystkie kolumny NULL lub żadna nie zaimplementowana | pojedyncza wartość kolumny może być NULL.

DEFERRABLE | NOT DEFERRABLE – możliwość opóźnienia kontroli zgodności do zakończenia transakcji.

INITIALLY DEFERRED | INITIALLY IMMEDIATE – zmieniane na potrzeby transakcji komendą SET CONSTRAINTS.

Ograniczenia na poziomie wierszy tabeli

```
[ CONSTRAINT constraint_name ]
{ UNIQUE ( column_name [, ... ] ) index_parameters |
  PRIMARY KEY ( column_name [, ... ] ) index_parameters |
  CHECK ( expression ) |
  FOREIGN KEY ( column_name [, ... ] )
    REFERENCES reftable [ ( refcolumn [, ... ] ) ]
    [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
    [ ON DELETE action ] [ ON UPDATE action ] }
[ DEFERRABLE | NOT DEFERRABLE ]
[ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]
```

index_parameters in UNIQUE and PRIMARY KEY constraints are:

```
[ WITH ( storage_parameter [= value] [, ... ] ) ]
[ USING INDEX TABLESPACE tablespace ]
```

CONSTRAINT – nazwane ograniczenie dla wielu kolumn.

UNIQUE – unikalne wartości w wielu kolumnach.

PRIMARY KEY – definicja klucza podstawowego złożonego.

CHECK – sprawdzanie warunku dla wielu kolumn (TRUE lub UNKNOWN; FALSE nie).

FOREIGN KEY – definicja klucza obcego złożonego.

REFERENCES – tablica odniesienia z kluczem.

MATCH FULL | PARTIAL | SIMPLE – wszystkie kolumny NULL lub żadna nie zaimplementowana | pojedyncza wartość kolumny może być NULL.

DEFERRABLE | NOT DEFERRABLE – możliwość opóźnienia kontroli zgodności do zakończenia transakcji.

INITIALLY DEFERRED | INITIALLY IMMEDIATE – zmieniane na potrzeby transakcji komendą SET CONSTRAINTS.

Ograniczenia na poziomie tablic (dotyczące wielu wierszy)

Ograniczenia między tablicowe

Ograniczenia dotyczące wielu wierszy

W RBD nie definiuje się (bezpośrednio) ograniczeń dotyczących wielu wierszu lub całej tabeli (dlaczego?).

Jedynym wyjątkiem jest ograniczenie typu UNIQUE lub utworzenie indeksu.

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

```
CREATE UNIQUE INDEX title_idx ON films (title);
```

```
CREATE INDEX lower_title_idx ON films ((lower(title)));
```

```
CREATE INDEX title_idx_nulls_low ON films (title NULLS FIRST);
```

```
CREATE UNIQUE INDEX title_idx ON films (title) WITH (fillfactor
```

```
CREATE INDEX code_idx ON films(code) TABLESPACE indexspace;
```

```
CREATE INDEX CONCURRENTLY sales_quantity_index ON sales_table (o
```


Ograniczenia dotyczące wielu tablic

W RDB nie definiuje się (bezpośrednio) ograniczeń dotyczących wielu tablic (dlaczego?).

Jedynym wyjątkiem jest ograniczenie typu FOREIGN KEY.

Ograniczenia można definiować pośrednio (struktura bazy) oraz operacyjnie (wyzwalacze, reguły).

Przykłady definicji tablic

```
CREATE TABLE shipments (
    id integer NOT NULL
        DEFAULT nextval('shipments_ship_id_seq'),
    customer_id integer,
    isbn text,
    ship_date timestamp);
```

CREATE

```
CREATE TABLE employees
    (id integer PRIMARY KEY CHECK (id > 100),
    last_name text NOT NULL,
    first_name text);
```

NOTICE: CREATE TABLE/PRIMARY KEY will create implicit index 'employees_pkey' for table 'employees'

CREATE

```
CREATE TABLE distinguished_authors (award text)
    INHERITS (authors);
```

CREATE

```
booktown=# \d distinguished_authors
```

```
Table "distinguished_authors"
Attribute | Type      | Modifier
-----+-----+-----
id        | integer   | not null
last_name | text      |
first_name | text      |
award     | text      |
```

Przykłady definicji tablic

Tablica z ograniczeniami i kluczem obcym

```
CREATE TABLE editions
  (isbn text,
   book_id integer,
   edition integer,
   publisher_id integer,
   publication date,
   type char,
   CONSTRAINT pkey PRIMARY KEY (isbn),
   CONSTRAINT integrity CHECK (book_id IS NOT NULL
                               AND edition IS NOT NULL),
   CONSTRAINT book_exists FOREIGN KEY (book_id)
                           REFERENCES books (id)
                           ON DELETE CASCADE
                           ON UPDATE CASCADE);
```

NOTICE: CREATE TABLE/PRIMARY KEY will create
implicit index 'pkey' for table 'editions'

NOTICE: CREATE TABLE will create implicit trigger(s) for
FOREIGN KEY check(s)

CREATE

Tworzenie tablic z danymi: CREATE TABLE AS

CREATE TABLE AS

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } ] TABLE table_name
    [ (column_name [, ...] ) ]
    [ WITH ( storage_parameter [= value] [, ...] ) | WITH OIDS
    [ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
    [ TABLESPACE tablespace ]
    AS query
```

TEMPORARY | TEMP – tablica tymczasowa, dla bieżącej sesji.

GLOBAL | LOCAL – bez efektu (dla zachowania kompatybilności).

WITH – FILLFACTOR, 10-100 (WITH | WITHOUT OIDS – deklaracja identyfikatorów.

ON COMMIT – specyfikacji akcji na końcu transakcji (dla TEMP).

TABLESPACE – specyfikacja innej niż domyślna przestrzeni tabel. query – instrukcja SELECT generująca dane.

Uwaga: definicje ograniczeń nie są kopiowane!

Uwaga: po wykonaniu tabela jest niezależna od oryginału.

Tworzenie tablic z danymi: przykłady

```
CREATE TEMP TABLE dzial_temp_tab
  AS SELECT * FROM dzial;
```

```
CREATE TEMP TABLE prac_temp_tab
  (id,nazwisko,dzial,stan,pob)
```

```
  AS SELECT ID_prac, Nazwisko, Dzial, Stanowisko, Pobory
     FROM prac
     WHERE Stanowisko <> 'kierownik'
     ORDER BY Nazwisko;
```

```
CREATE TABLE films_recent AS
  SELECT * FROM films WHERE date_prod >= '2002-01-01';
```

```
PREPARE recentfilms(date) AS
  SELECT * FROM films WHERE date_prod > $1;
CREATE TEMP TABLE films_recent WITH (OIDS) ON COMMIT DROP AS
  EXECUTE recentfilms('2002-01-01');
```

Tworzenie tabel z danymi: SELECT INTO

```
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
      * | expression [ AS output_name ] [, ...]
      INTO [ TEMPORARY | TEMP ] [ TABLE ] new_table
      [ FROM from_item [, ...] ]
      [ WHERE condition ]
      [ GROUP BY expression [, ...] ]
      [ HAVING condition [, ...] ]
      [ { UNION | INTERSECT | EXCEPT } [ ALL ] select ]
      [ ORDER BY expression [ ASC | DESC | USING operator ]
        [ NULLS { FIRST | LAST } ] [, ...] ]
      [ LIMIT { count | ALL } ]
      [ OFFSET start ]
      [ FOR { UPDATE | SHARE } [ OF table_name [, ...] ] [ NOWAIT
```

DISTINCT ON (expression) – różne z uwagi na podane wyrażenie.

FOR UPDATE – blokowanie wierszy do zakończenia wykonania.

```
SELECT * INTO films_recent
      FROM films
      WHERE date_prod >= '2002-01-01';
```

Widoki czyli perspektywy

Perspektywy (widoki): realizacja postulatu braku redundancji oraz wielu użytkowników.

Schemat wielowarstwowy baz danych.

```
CREATE [ OR REPLACE ] [ TEMP | TEMPORARY ] VIEW name
[ ( column_name [, ...] ) ]
    AS query
```

OR REPLACE – podmiana, jeżeli VIEW istnieje. TEMPORARY | TEMP – tymczasowy widok, niszczone przy końcu sesji.

```
CREATE VIEW comedies AS
    SELECT *
    FROM films
    WHERE kind = 'Comedy';
```

```
CREATE VIEW myview AS
    SELECT city, temp_lo, temp_hi, prcp, date, location
    FROM weather, cities
    WHERE city = name;
```

```
SELECT * FROM myview;
```

Widoki – przykłady

```
CREATE VIEW dzialprac
AS
SELECT *
FROM dzial d LEFT OUTER JOIN prac p
      ON d.id_dzial=p.dzial
WHERE Stanowisko <> 'kierownik';
```

```
CREATE VIEW pracdzial
AS
SELECT *
FROM prac p RIGHT OUTER JOIN dzial d
      ON p.dzial=ID_dzial
WHERE true;
```

```
CREATE VIEW dzialkierdane
AS
SELECT *
FROM dzial d LEFT OUTER JOIN prac p
      ON d.kierownik=p.ID_prac
WHERE true;
```

```
CREATE VIEW agr
AS
SELECT stanowisko, count(*) AS liczba_prac,
      avg(pobory) AS srednia, sum(pobory) AS Suma
FROM prac
GROUP BY stanowisko;
```