Zaawansowane Technologie Bazodanowe

Wykład p.t.

DDL

Data Definition Language
Definiowanie tabel i obiektów pomocniczych

Antoni Ligęza

ligeza@agh.edu.pl
http://galaxy.uci.agh.edu.pl/~ligeza

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

DDL – Data Definition Language

Instrukcje DDL – CREATE

- `CREATE AGGREGATE` – tworzenie funkcji agregacji,
- `CREATE CONSTRAINT TRIGGER` – tworzenie procedury wyzwalanej jako ograniczenia,
- `CREATE DATABASE` – tworzenie bazy danych,
- `CREATE FUNCTION` – tworzenie nowej funkcji,
- `CREATE GROUP` – tworzenie grupy,
- `CREATE INDEX` – tworzenie indeksu,
- `CREATE LANGUAGE` – definiowanie nowego języka dla funkcji,
- `CREATE OPERATOR` – definiowanie nowego operatora użytkownika,
- `CREATE RULE` – definiowanie nowej reguły,
- `CREATE SEQUENCE` – definiowanie nowej sekwencji,
- `CREATE TABLE` – tworzenie tabel,
- `CREATE TABLE AS` – tworzenie tabel jako wyniku instrukcji SELECT,
- `CREATE TRIGGER` – tworzenie nowego wyzwalacza,
- `CREATE TYPE` – definiowanie nowego typu danych,
- `CREATE USER` – tworzenie nowego uzytkownika,
- `CREATE VIEW` – tworzenie nowej perspektywy.

DDL – Data Definition Language

Instrukcje DDL – ALTER i DROP

- `ALTER GROUP` – modyfikacja grupy (dodanie lub usunięcie użytkownika),
- `ALTER TABLE` – modyfikacja struktury tabeli,
- `ALTER USER` – modyfikacja konta uzytkownika
- `DROP AGGREGATE` – usuwanie funkcji agregacji,
- `DROP DATABASE` – usuwanie bazy danych,
- `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 RULE` – usuwanie zdefiniowanej reguły,
- `DROP SEQUENCE` – usuwanie zdefiniowanej sekwencji,
- `DROP TABLE` – usuwanie tabeli,
- `DROP TRIGGER` – usuwanie wyzwalacza,
- `DROP TYPE` – usuwanie zdefiniowanego typu danych,
- `DROP USER` – usuwanie użytkownika,
- `DROP VIEW` – usuwanie perspektywy.

DDL – Data Definition Language

DDL – inne instrukcje

- `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 uzytkownikom,
- `SET` – ustawia parametry startowe bazy danych,

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 kompatybiliności).
`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 domyslna przestrzeni tabel.

Definiowanie tablic – ograniczenia

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 | INITIAL
```

`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|niezaimplementowane|pojedyncza wartość kolumny może być NULL.
`DEFERRABLE|NOT DEFERRABLE` – możliwość opóźnienia kontroli zgodności do zakończenia transakcji.
`INITIALLY DEFERREDIIMMEDIATE` – zmieniane na potrzeby transakcji komedną `SET CONSTRAINTS`.

Ograniczenia na poziomie 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 | INITIAL

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|niezaimplementowane|pojedyncza wartość kolumny może być NULL.
`DEFERRABLE|NOT DEFERRABLE` – możliwość opóźnienia kontroli zgodności do zakończenia transakcji.
`INITIALLY DEFERREDIIMMEDIATE` – zmieniane na potrzeby transakcji komedną `SET CONSTRAINTS`.

Definiowanie tabel

Definicja ograniczeń na poziomie kolumny

```
column_constraint ::=
  [ CONSTRAINT column_constraint_name ]
  { NOT NULL | NULL | UNIQUE | PRIMARY KEY |
    DEFAULT default_value |
    CHECK (condition |
    REFERENCES foreign_table [ ( foreign_column ) ]
      [ MATCH FULL | MATCH PARTIAL ]
      [ ON DELETE action ]
      [ ON UPDATE action ]
      [ DEFERRABLE | NOT DEFERRABLE ]
      [ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]}
```

Ograniczenia na poziomie tablic

```
table constraint ::=
  [ CONSTRAINT table_constraint_name ]
  { UNIQUE ( column_name [, ... ] ) |
    PRIMARY KEY ( column_name [, ... ] ) |
    CHECK ( condition ) |
    FOREIGN KEY ( column_name [, ... ] )
      REFERENCES foreign_table
      [ ( foreign_column [, ... ] ) ]
      [ MATCH FULL | MATCH PARTIAL ]
      [ ON DELETE action ]
      [ ON UPDATE action ]
      [ DEFERRABLE | NOT DEFERRABLE ]
      [ INITIALLY DEFERRED | INITIALLY IMMEDIATE ]}
action ::= { NO ACTION | RESTRICT | CASCADE | SET NULL | SET D
```

Przykłady definicji tablic

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

booktown=# CREATE TABLE employees
booktown-#        (id integer PRIMARY KEY CHECK (id > 100),
booktown(#         last_name text NOT NULL,
booktown(#         first_name text);
NOTICE:  CREATE TABLE/PRIMARY KEY will create implicit index
'employees_pkey' for table 'employees'
CREATE

booktown=# CREATE TABLE distinguished_authors (award text)
booktown-#              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

```
booktown=# CREATE TABLE editions
booktown-#    (isbn text,
booktown(#     book_id integer,
booktown(#     edition integer,
booktown(#     publisher_id integer,
booktown(#     publication date,
booktown(#     type char,
booktown(#     CONSTRAINT pkey PRIMARY KEY (isbn),
booktown(#     CONSTRAINT integrity CHECK (book_id IS NOT NULL
booktown(#                                 AND edition IS NOT
booktown(#     CONSTRAINT book_exists FOREIGN KEY (book_id)
booktown(#                REFERENCES books (id)
booktown(#                ON DELETE CASCADE
booktown(#                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
    [ (column_name [, ...] ) ]
    [ WITH ( storage_parameter [= value] [, ... ] ) | WITH OID
    [ 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 domyslna przestrzeni tabel. `query` – instrukcja `SELECT` generująca dane.

Uwaga: definiecje ograniczeń nie są kopiowane!
Uwaga: po wykonaniu tabela jest n iezależna 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 [, ...] ] [ NOWAI
```

`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

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

`OR REPLACE` – podmiana, jeżeli `VIEW` istnieje. `TEMPORARY | TEMP` – tymczasowy widok, niszczony 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  średnia, sum(pobory) AS Suma
FROM prac
GROUP BY stanowisko;
```

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 kontrola jednego użytkowanika. Schematy moga zawierać obiekty o identycznych nazwach. Odwolanie do np. tablicy `prac` w schemacie `biuro` nastepuje poprzez uzycie notacji krpkowej: `SELECT * FROM biuro.prac;`. Użycie schematów wewnątrz bazy ma na celu `logiczną separację` obiektów, np. dla róznych uzytkowników i aplikacji. Standardowo i domyslnie 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 schematu (pustego/niepustego).

```
DROP SCHEMA myschema;

DROP SCHEMA myschema CASCADE;
```

```
SHOW search_path;

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

SET search_path TO myschema,public;
```

Schematy – przyklady

Przykłady tworzenia schematów

```
CREATE SCHEMA myschema;

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

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 N

Notice that the individual subcommands do not end with semicol

The following is an equivalent way of accomplishing the same r

CREATE SCHEMA hollywood;
CREATE TABLE hollywood.films (title text, release date, awards
CREATE VIEW hollywood.winners AS
    SELECT title, release FROM hollywood.films WHERE awards IS
```

TTT

TTT

TTT

TTT

TTT

TTT

TTT

TTT

TTT

TTT

TTT

TTT

Modyfikacja Tablic

Schemat modyfikacji – ALTER TABLE

```
 ALTER TABLE table
        ADD [ CONSTRAINT name ]
        { CHECK ( condition ) |
          FOREIGN KEY ( column [, ... ] )
                      REFERENCES table [ ( column [, ... ] ) ]
                      [ MATCH FULL | MATCH PARTIAL ]
                      [ ON DELETE action ]
                      [ ON UPDATE action ]
                      [ DEFERRABLE | NOT DEFERRABLE ]
                      [ INITIALLY DEFERRED | INITIALLY IMMEDIA
        }
```

```
booktown=# ALTER TABLE books ADD CONSTRAINT legal_subjects
booktown-#                   FOREIGN KEY (subject_id)
booktown-#                   REFERENCES subjects (id);
NOTICE:  ALTER TABLE ... ADD CONSTRAINT will create
implicit trigger(s) for FOREIGN KEY check(s)
CREATE
```

Przykład

CREATE TABLE z INSERT INTO i zmiana nazw tabel

```
booktown=# DROP INDEX books_id_pkey;
DROP
booktown=# CREATE TABLE new_books
booktown-#              (id integer CONSTRAINT books_id_pkey P
booktown(#              title text NOT NULL,
booktown(#              author_id integer,
booktown(#              subject_id integer);
NOTICE:  CREATE TABLE/PRIMARY KEY will create implicit index
’books_id_pkey’ for table ’new_books’
CREATE
booktown=# INSERT INTO new_books SELECT * FROM books;
INSERT 0 15
booktown=# ALTER TABLE books RENAME TO old_books;
ALTER
booktown=# ALTER TABLE new_books RENAME TO books;
ALTER
```

ALTER TABLE – Przykłady

Dodawanie kolumny

```
booktown=# ALTER TABLE books
booktown-#       ADD publication date;
ALTER
booktown=# \d books
          Table "books"
  Attribute  |  Type   | Modifier
-------------+---------+----------
 id          | integer | not null
 title       | text    | not null
 author_id   | integer |
 subject_id  | integer |
 publication | date    |
Index: books_id_pkey
```

Zmiana nazw kolumn

```
booktown=# \d daily_inventory
    Table "daily_inventory"
 Attribute |  Type   | Modifier
-----------+---------+----------
 isbn      | text    |
 in_stock  | boolean |
booktown=# ALTER TABLE daily_inventory
booktown-#       RENAME COLUMN in_stock TO is_in_stock;
ALTER
booktown=# ALTER TABLE daily_inventory
booktown-#       RENAME is_in_stock TO is_stocked;
ALTER
```

ALTER TABLE – Przykłady

Zmiana własności

```
booktown=# ALTER TABLE books
booktown-#       ALTER COLUMN id
booktown-#       SET DEFAULT nextval('book_ids');
ALTER
booktown=# \d books
                          Table "books"
 Attribute  |  Type   |                  Modifier
------------+---------+-------------------------------------
 id         | integer | not null default nextval('book_ids'::t
 title      | text    | not null
 author_id  | integer |
 subject_id | integer |
Index: books_id_pkey

booktown=# ALTER TABLE books
booktown-#       ALTER id
booktown-#       DROP DEFAULT;
ALTER
booktown=# \d books
          Table "books"
 Attribute  |  Type   | Modifier
------------+---------+----------
 id         | integer | not null
 title      | text    | not null
 author_id  | integer |
 subject_id | integer |
Index: books_id_pkey
```

ALTER TABLE – Przykłady

Zmiana nazwy tabeli

```
booktown=# ALTER TABLE books RENAME TO literature;
ALTER
booktown=# ALTER TABLE literature RENAME TO books;
ALTER
```

Dodawanie ograniczeń

```
booktown=# ALTER TABLE editions
booktown-#       ADD CONSTRAINT foreign_book
booktown-#       FOREIGN KEY (book_id) REFERENCES books (id);
NOTICE: ALTER TABLE ... ADD CONSTRAINT will create
implicit trigger(s) for FOREIGN KEY check(s)
CREATE
booktown=# ALTER TABLE editions
booktown-#       ADD CONSTRAINT hard_or_paper_back
booktown-#       CHECK (type = 'p' OR type = 'h');
ALTER
```

ALTER TABLE – Restrukturyzacja tablicy z CREATE TABLE AS

```
          Table "books"
  Attribute  |  Type   | Modifier
-------------+---------+----------
 id          | integer | not null
 title       | text    | not null
 author_id   | integer |
 subject_id  | integer |
 publication | date    |
Index: books_id_pkey
booktown=# CREATE TABLE new_books
booktown-#        (id, title, author_id, subject_id)
booktown-#        AS SELECT id, title, author_id, subject_id
booktown-#                  FROM books;
SELECT
booktown=# ALTER TABLE books RENAME TO old_books;
ALTER
booktown=# ALTER TABLE new_books RENAME TO books;
ALTER
booktown=# \d books
        Table "books"
 Attribute  |  Type   | Modifier
------------+---------+----------
 id         | integer |
 title      | text    |
 author_id  | integer |
 subject_id | integer |
booktown=# DROP TABLE books;
DROP
```

ALTER TABLE – Restrukturyzacja tablicy z INSERT INTO

```
booktown=# CREATE TABLE new_books (
booktown(#   id integer UNIQUE,
booktown(#   title text NOT NULL,
booktown(#   author_id integer,
booktown(#   subject_id integer,
booktown(#   CONSTRAINT books_pkey PRIMARY KEY (id)
booktown(# );
NOTICE:  CREATE TABLE/PRIMARY KEY will create implicit index ’
for table ’new_books’
CREATE
booktown=# INSERT INTO new_books
booktown-#             SELECT id, title, author_id, subject_id
booktown-#                    FROM books;
INSERT 0 12
booktown=# ALTER TABLE books RENAME TO old_books;
ALTER
booktown=# ALTER TABLE new_books RENAME TO books;
ALTER
booktown=# \d books
          Table "books"
 Attribute  |  Type   | Modifier
------------+---------+----------
 id         | integer | not null
 title      | text    | not null
 author_id  | integer |
 subject_id | integer |
Index: books_id_pkey
```

Definiowanie i użycie sekwencji

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.
Wartości sekwencji bazuja na typie `bigint` (8 bajtów) i są z zakresu:
(-9223372036854775808 do 9223372036854775807).

```
CREATE TABLE tablename (
    colname SERIAL
);

is equivalent to specifying:

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

Pełna postać definicji sekwencji:

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

`TEMPORARY | TEMP` – na potrzeby bieżącej sesji, potem niszczona. Tworzona w specjalnym schemacie. `INCREMENT [ BY ] increment` – krok zmiany sekwencji; może byc 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 rosnacych). `START [ WITH` – wartość poczatkowa. `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 domyslnie wlasności `UNIQUE`!
Uwaga: pola sekwencji dają się ustawiać ręcznie – można ich wartośc wymusić poprzez `INSERT INTO`! Powoduje to *zmylenie sekwencji*.

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

```
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;

booktown=# CREATE SEQUENCE shipments_ship_id_seq
booktown-#                 START 200 INCREMENT 1;
CREATE

booktown=# SELECT nextval ('shipments_ship_id_seq');
 nextval
---------
     200
(1 row)
```

```
booktown=# INSERT INTO shipments VALUES
booktown-#        (nextval('shipments_ship_id_seq'), 107,
   '0394800753', 'now');

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

Definiowanie i użycie dziedzin

Dziedziny to typy danych ze zdefiniowanmiy przez użytkowanika ograniczeniami.

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

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

Usuwanie tablic – DROP TABLE

Schemat usuwania

```
DROP TABLE tablename

DROP TABLE name [, ...]
```

Przykład usuwania tablic

```
booktown=# DROP TABLE employees;
DROP
```

Uwaga: Usuwanie tablicy nie usuwa automatycznie związanych z nią obiektów, np. sekwencji. Dlatego dla "wyczyszczenia" bazy przed ponownym zakładaniem tablicy trzeba usunąć "ręcznie" takie obiekty. Przykład: usuwanie sekwencji związanych z tabelą.

```
booktown=# DROP SEQUENCE shipments_ship_id_seq;
DROP
```

Wprowadzanie danych do tabel – INSERT INTO

Schemat instrukcji

```
INSERT INTO table_name
        [ ( column_name [, ...] ) ]
        VALUES ( value [, ...] )

 INSERT INTO table_name
        [ ( column_name [, ...] ) ]
        query
```

Przykłady

```
booktown=# INSERT INTO books (id, title, author_id, subject_id
booktown-#        VALUES (41472, 'Practical PostgreSQL', 1212,
INSERT 3574037 1

booktown=# INSERT INTO books (subject_id, author_id, id, title
booktown-#        VALUES (4, 7805, 41473, 'Programming Python'
INSERT 3574041 1

booktown=# INSERT INTO books (id, title, author_id, subject_id
booktown-#        SELECT nextval('book_ids'), title, author_id
booktown-#               FROM book_queue WHERE approved;
INSERT 0 2
```

Ładowanie tablic – COPY

Schemat instrukcji

```
COPY [ BINARY ] table_name [ WITH OIDS ]
     FROM { 'filename' | stdin }
     [ [USING] DELIMITERS 'delimiter' ]
     [ WITH NULL AS 'null_string' ]
```

Przykłady

```
booktown=# COPY subjects FROM '/tmp/subjects.sql'
booktown-#               USING DELIMITERS ',' WITH NULL AS '\n
COPY

booktown=# COPY books TO 'filename';
COPY

1,Business,Productivity Ave
2,Children's Books,Kids Ct
3,Classics,Academic Rd
9,Horror,Black Raven Dr
10,Mystery,Black Raven Dr
11,Poetry,Sunset Dr
13,Romance,Main St
14,Science,Productivity Ave
15,Science Fiction,Main St
0,Arts,Creativity St
6,Drama,Main St
7,Entertainment,Main St
```

Modyfikacja zawartości tabel – UPDATE

Schemat instrukcji

```
  UPDATE [ ONLY ] table SET
         column = expression [, ...]
         [ FROM source ]
         [ WHERE condition ]
```

Przykład postepowania

```
booktown=# SELECT retail FROM stock
booktown-#        WHERE isbn = '0590445065';
 retail
--------
  23.95
(1 row)

booktown=# UPDATE stock
booktown-#        SET retail = 25.95
booktown-#        WHERE isbn = '0590445065';
UPDATE 1
booktown=# SELECT retail FROM stock
booktown-#        WHERE isbn = '0590445065';
 retail
--------
  25.95
(1 row)
```

Modyfikacja zawartości tabel – UPDATE

Schemat postępowania

```
booktown=# SELECT isbn, retail, cost
booktown-#        FROM stock
booktown-#        ORDER BY isbn ASC
booktown-#        LIMIT 3;
    isbn    | retail | cost
------------+--------+-------
 0385121679 |  36.95 | 29.00
 039480001X |  32.95 | 30.00
 0394800753 |  16.95 | 16.00
(3 rows)

booktown=# UPDATE stock
booktown-#        SET retail =
booktown-#            (cost * ((retail / cost) + 0.1::numeric)
UPDATE 16

booktown=# SELECT isbn, retail, cost
booktown-#        FROM stock
booktown-#        ORDER BY isbn ASC
booktown-#        LIMIT 3;
    isbn    | retail | cost
------------+--------+-------
 0385121679 |  39.85 | 29.00
 039480001X |  35.95 | 30.00
 0394800753 |  18.55 | 16.00
(3 rows)
```

Modyfikacja zawartości tabel – UPDATE

Jednoczesna modyfikacja wielu kolumn

```
booktown=# UPDATE publishers
booktown-#        SET name = 'O\'Reilly & Associates',
booktown-#            address = 'O\'Reilly & Associates, Inc.
booktown-#                    || '101 Morris St, Sebastopol, CA
booktown-#        WHERE id = 113;
UPDATE 1
booktown=# SELECT name, substr(address, 1, 40) || '...' AS sho
booktown-#        FROM publishers
booktown-#        WHERE id = 113;
         name          |                short_address
-----------------------+-----------------------------------
 O'Reilly & Associates | O'Reilly & Associates, Inc. 101 Morri
(1 row)
```

Kasowanie rekordów – DELETE FROM

Schemat instrukcji

```
 DELETE FROM [ ONLY ] table
         [ WHERE condition ]
```

Przykłady

```
booktown=# SELECT * FROM stock
booktown-#          WHERE stock = 0;
    isbn    | cost  | retail | stock
------------+-------+--------+-------
 0394800753 | 16.00 |  16.95 |     0
 0394900014 | 23.00 |  23.95 |     0
 0451198492 | 36.00 |  46.95 |     0
 0451457994 | 17.00 |  22.95 |     0
(4 rows)

booktown=# DELETE FROM stock
booktown-#        WHERE stock = 0;
DELETE 4
```

Uwaga:

```
booktown=# DELETE FROM stock_backup;
DELETE 16
```

Tworzenie baz danych – CREATE DATABASE

Schemat instrukcji

```
CREATE DATABASE name
  [ WITH [ LOCATION = { 'dbpath' | DEFAULT } ]
         [ TEMPLATE = template | DEFAULT ]
         [ ENCODING = encoding_name | encoding_number | DEFAUL
```

Przykłady

```
template1=# CREATE DATABASE booktown;
CREATE DATABASE

template1=# CREATE DATABASE booktown WITH LOCATION = '/usr/loc
CREATE DATABASE


-- polish

DROP DATABASE nf;
CREATE DATABASE nf WITH ENCODING = 'LATIN2';
```