PostgreSQL - най-напредничавата свободна база данни

представена от Георги Чорбаджийски
http://georgi.unixsol.org/
georgi@unixsol.org

Какво е PostgreSQL?

- BSD лицензирана обектно релационна база данни (ORDBMS)
- Пионер, чиито концепции по-късно са се появили в комерсиалните бази данни
- SQL92 / SQL99 съвместима
- Поддържа се голяма част от SQL:2003
- ORDBMS с поддръжка на много модерни възможности
- ORDBMS разширяема от потребителя по много начини

Кратка история 1/2

- 1977-1985
  - Базиран на проекта POSTGRES в Бъркли
- 1986-1993
  - Трансформиран е в обектно ориентирана релационна база данни (ORDBMS)
- 1995
  - Добавена е поддръжка на SQL и проектът е преименуван на Postgres95
- 1996
  - Преименува се на PostgreSQL и продължава да се разработва като свободен софтуер

Кратка история 2/2

- 2000
  - Излиза версия 7.0 с поддръжка на FK, много подобрения в оптимизатора на заявки, бързодействието и много други
- 2000-2004
  - Излизат версии 7.1, 7.2, 7.3 и 7.4, като във всяка има много подобрения спрямо предишната
- 2005
  - Излиза версия 8.0 с вградена поддръжка на Windows, Savepoints, PITR, Table spaces и много други

Текущо положение

- Много стабилен
- Високо надежден
- Има постоянен цикъл на разработка
- Нови възможности се добавят във всяка версия
- Поддържа всички типове данни, дефинирани в SQL стандартите, плюс още
- Поддържа оператори за работа с всички типове данни, плюс доста други

Използване и поддръжка

- Има работещи инсталации, поддържащи терабайти данни
- Използва се от .org регистъра
- Комерсиална поддръжка се предлага от PostgreSQL Inc, Red Hat и много други консултанти по света
- Има няколко решения за репликация, най-известното е Slony-I
- Най-голямата японска IT компания Fujitsu помага при разработката
- Има чудесна документация и пощенски списъци

Поддържани възможности

- Стандартни за RDBMS
  - изгледи (views)
  - външни ключове (foreign keys)
  - ограничения (constraints)
  - тригери (triggers)
  - вградени процедури (stored procedures)
  - комплексни заявки (complex queries)
  - транзакционна цялост (transactional integrity)
- ORDBMS възможности
  - наследяване на таблици (table inheritance)
  - данни тип record (composite types)
  - разширяема от потребителя
  - поддръжка на масиви (arrays)
- Уникални възможности
  - multiversion concurrency control

Разширяемост

- Потребителят може да добавя собствени:
  - типове данни (data types)
  - функции (functions)
  - оператори (operators)
  - агрегиращи функции (aggregate functions)
  - методи за индексиране (index methods)
  - процедурни езици (procedural languages)

Стандартни типове данни

- Numeric Types
- Character Types
- Binary Data Types
- Date/Time Types
- Boolean Type
- Geometric Types
- Network Address Types
- Bit String Types
- Arrays
- Composite Types

Езици за програмиране

- За вложени процедури
  - C / C++
  - PL/pgSQL
  - PL/Tcl
  - PL/Perl
  - PL/Python
  - PL/Java
  - PL/PHP
  - PL/sh
- За достъп до базата
  - Всички по-горе изредени
  - Драйвери за ODBC
  - Драйвери за JDBC
  - Драйвери за Delphi, Kylix, Borland C++, .NET, Qt

Неподдържани SQL възможности

- Писане на вградени процедури на Ada, COBOL, Fortran, MUMPS, Pascal, PL/I
- Привилегии при UPDATE и REFERENCES на ниво колона
- Възможност да се дава WITH GRANT OPTION при GRANT
- Позициониран UPDATE и DELETE при курсорите
- Специфични тригери за колони и твърдо подреждане на тригери
- Йерархични JOIN заявки

Основни обекти и функционалности

Table spaces

- Алтернативни места във файловата система за разполагане на данни
- Създаване на конкретни обекти на различни дискове
- Обектите могат да бъдат: бази данни, таблици, индекси и ограничения
- Примери:

```
CREATE TABLESPACE indexspace LOCATION '/scsi/disk1';

CREATE TABLESPACE slowspace LOCATION '/ide/disk2';
```

Бази данни (databases)

- Основният обект, в който се разполагат всички други
- Кодови таблици
- Table spaces
- Template databases
- Примери:

```
CREATE DATABASE test;

CREATE DATABASE music ENCODING 'WIN';

CREATE DATABASE sales OWNER salesapp TABLESPACE salesspace;

CREATE DATABASE user_xxx OWNER user_xxx TABLESPACE users TEMPLATE users_template;
```

Схеми (schemas)

- Логическо разделение в базата данни
- Полезни са при достъп на различни приложения към една БД
- Използване на различни кодови таблици в една БД (не се поддържа в момента)
- Примери:

```
CREATE SCHEMA myschema;

CREATE SCHEMA AUTHORIZATION someuser;

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;
```

Таблици (tables) 1/2

- Основният обект, в който се пазят данни
- Наследяване
- Временни таблици (temporary tables)
- Примери:

```
CREATE TABLE desc_movies (
  name      text,
  director  text
);

CREATE TABLE tech_movies (
  length    int,
  aspect    real
);
```

Таблици (tables) 2/2

- Примери:

```
CREATE TABLE movies (
   movie_id  serial
) INHERITS (desc_movies, tech_movies);

test=> \d movies
Table "public.movies"
  Column  |  Type   |                            Modifiers
----------+---------+-------------------------------------------------------------
 name     | text    |
 director | text    |
 length   | integer |
 aspect   | real    |
 movie_id | integer | not null default nextval('public.movies_movie_id_seq'::text)
Inherits: desc_movies,
          tech_movies

DROP TABLE desc_movies CASCADE;

CREATE TEMP TABLE test (id int);
```

Броячи (sequences)

- Генератори на числа
- Винаги връщат уникален отговор
- Поредността на числата НЕ Е гарантирана
- Примери:

```
CREATE SEQUENCE test_seq START 101;

test=> SELECT nextval('test_seq');
nextval
---------
102

CREATE TABLE test_table (
  id     integer,
  id2    serial,
);

INSERT INTO test_table (id) VALUES (nextval('test_seq'));
```

Изгледи (views) 1/2

- Ограничаване на видимостта на данни
- Удобно представяне на данни от много таблици
- Примери:

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

CREATE VIEW users_active AS
  SELECT
    users.login, users.name,
    groups.name AS group_name,
      CASE WHEN groups.name = ''
        THEN users.login
        ELSE users.login || '@' || groups.name
      END AS login_full,
     (SELECT count(*) FROM logins WHERE users.id = logins.user) AS num_logins
   FROM users, groups
   WHERE users.active AND users.ugroup = groups.id;
```

Изгледи (views) 2/2

- Примери:

```
test=> SELECT * FROM users_active;
  login   |         name          | group_name |   login_full     | num_logins
----------+-----------------------+------------+------------------+------------
 gf       | Георги Чорбаджийски |            | gf               |        250
 gfadmin | Георги Чорбаджийски | admins     | gfadmin@admins |         15
(2 rows)
```

Ограничения (constraints) 1/3

- Ограничения за цялостност на данните
  - PRIMARY KEY
  - FOREIGN KEY (REFERENCES)
  - NOT NULL
  - UNIQUE
  - CHECK
  - defaults
- Примери:

```
CREATE TABLE kinds (
  kind_id     integer PRIMARY KEY,
  kind        varchar(10) UNIQUE
);
test=> INSERT INTO kinds VALUES (1,'Action');
INSERT 160958554 1
test=> INSERT INTO kinds VALUES (1,'Romance');
ERROR:  duplicate key violates unique constraint "kinds_pkey"
```

Ограничения (constraints) 2/3

- Примери:

```
CREATE TABLE films (
  id          serial,
  code        char(5) PRIMARY KEY,
  title       varchar(40) NOT NULL,
  kind        integer REFERENCES kinds,
  dist_id     integer DEFAULT nextval('distributors_serial'),
  cost        real NOT NULL,
  rating      varchar(8) CHECK(rating IN ('PG-12','PG-16','NC-18')) NOT NULL,
  date_prod   date,
  len         int CHECK (len > 0 AND len < 3*3600),
  CHECK(rating = 'PG-12' AND cost < 50000)
);
```

Ограничения (constraints) 3/3

- Примери:

```
test=> INSERT INTO kinds VALUES (1,'Action');
INSERT 160958554 1

test=> INSERT INTO films (code,title,kind,cost,rating,date_prod,len) VALUES
   ('CMND','Commando',1,30000,'PG-12','2004-01-01',195*60);
ERROR:  new row for relation "films" violates check constraint "films_len_check"

test=> INSERT INTO films (code,title,kind,cost,rating,date_prod,len) VALUES
   ('CMND','Commando',1,30000,'PG-12','2004-01-01',7200);
INSERT 160973198 1

test=> INSERT INTO films (code,title,kind,cost,rating,date_prod,len) VALUES
   ('CMND','Commando',1,130000,'PG-12','2004-01-01',7200);
ERROR:  new row for relation "films" violates check constraint "films_check"

test=> SELECT * from films;
 id | code  |  title   | kind | dist_id | cost  | rating | date_prod  | len
----+-------+----------+------+---------+-------+--------+------------+------
  2 | CMND  | Commando |    1 |       2 | 30000 | PG-12  | 2004-01-01 | 7200
(1 row)
```

Правила (rules)

- Преписване на заявка
- Забрани на ниво ред/колона
- Забрана за добавяне/обновяване/изтриване на данни
- Примери:

```
CREATE RULE pseudo_view AS ON SELECT TO t1
  DO INSTEAD SELECT * FROM t2;

CREATE RULE update_payment AS ON UPDATE TO payments
  DO INSTEAD NOTHING;

CREATE RULE delete_payment2 AS
  ON DELETE TO payments WHERE id BETWEEN 50 AND 100
  DO INSTEAD NOTHING;

CREATE RULE user1_view_update AS ON UPDATE TO user1_view
  DO INSTEAD UPDATE user1_table SET data = new.data;
```

Тригери (triggers) 1/2

- Изпълнение на заявки преди и след INSERT / UPDATE / DELETE
- Изпълнение на допълнителни заявки за всеки ред от INSERT / UPDATE / DELETE
- Примери:

```
CREATE TRIGGER if_dist_exists
  BEFORE INSERT OR UPDATE ON films
    FOR EACH ROW
      EXECUTE PROCEDURE
        check_primary_key('did', 'distributors', 'did');
```

Тригери (triggers) 2/2

- Примери:

```
CREATE OR REPLACE FUNCTION delete_payment_func() RETURNS trigger AS '
BEGIN
  UPDATE invoices_items SET
    payed = payed - OLD.payed
  WHERE
    id = OLD.item;
  RETURN OLD;
END;
' LANGUAGE 'plpgsql';

CREATE TRIGGER delete_payment
  BEFORE DELETE ON payments
    FOR EACH ROW
      EXECUTE PROCEDURE delete_payment_func();
```

Полета за данни (domains)

- Специализирани типове данни, разширяващи съществуващите
- Примери:

```
CREATE DOMAIN us_postal_code AS text
  DEFAULT '00000' NOT NULL
  CHECK(VALUE ~ '^\d{5}$' OR VALUE ~ '^\d{5}-\d{4}$'
);

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

Вградени процедури (Stored procedures) 1/2

- Изпълнение на код на сървъра
- Отделяне на логиката на приложението от данните
- Могат да са написани на всеки един от поддържаните езици
- Идентифицират се чрез името и параметрите си
- Възможен е overload на процедури
- "Компилират" се

Вградени процедури (Stored procedures) 2/2

- Пример:

```
CREATE OR REPLACE FUNCTION get_related_objects(int, text, text, text) RETURNS TEXT AS $$
DECLARE
  _id ALIAS FOR $1; _table ALIAS FOR $2; _idfield ALIAS FOR $3;
  _objfield ALIAS FOR $4;
  _query TEXT; _record RECORD; _ret TEXT;
BEGIN
  _ret := '';
  _query := 'SELECT ' || quote_ident(_objfield)
      || '::text AS tmpid FROM ' || quote_ident(_table)
      || ' WHERE ' || quote_ident(_idfield) || '=' || quote_literal(_id);
  FOR _record IN EXECUTE _query LOOP
      _ret := _ret || ',' || _record.tmpid;
  END LOOP;
  RETURN ltrim(_ret,',');
END;
$$ LANGUAGE 'plpgsql';

test=> SELECT get_related_objects(1,'contacts_clients','contact','client');
 get_related_objects
---------------------
 130,244,11
(1 row)
```

Транзакции

- BEGIN, COMMIT, ROLLBACK
- Вложени транзакции

Редовна поддръжка на БД

- VACUUM
- VACUUM ANALYZE
- REINDEX
- CLUSTER
- pg_autovacuum
- vacuumdb, clusterdb

Съвети за подобряване на бързодействието

- Използвайте коректно индексите
- Използвайте EXPLAIN, за да разберете какво върши базата
- Използвайте COPY за зареждане на големи количества данни
- Редовно VACUUM-ирайте и ANALYZE-ирайте базата
- Използвайте вградени процедури

Основни програми

- pg_ctl
  - Стартиране/Спиране/Рестартиране на сървъра
- psql
  - Shell базиран frontend за PostgreSQL
    - tab completion
    - Помощ при въвеждане на SQL заявки (DROP TABLE [tab])
    - Бърз достъп до информация за обектите в базата (\d, \dt, \dv, \ds ....)
    - Изпълнение на команди на сървъра и форматиране на изхода
- pg_dump
  - Архивиране на базата данни
- pg_dumpall
  - Архивиране на целия сървър
- create{db,user}
- drop{db,user}

Програми за администрация и разработка

- pgAdmin III
  - Графична програма за администрация
  - Работи под Unix / Windows
  - http://www.pgadmin.org/
- phpPgAdmin
  - Web базиран
  - http://phppgadmin.sourceforge.net/
- psql
  - Използва се от командния ред
  - В стандартната дистрибуция е
  - Много полезен инструмент

Във версия 8.1 се очаква

- Updateble views
- Интегриране на pg_autovacuum
- По-голяма бързина при големи SMP машини
- Подобряване на бързодействието като цяло
- По-добра съвместимост със SQL стандартите
- Поддръжка на IN и OUT параметри при вложените процедури
- Като цяло, още няма 100% конкретни планове

Връзки

- Основни връзки
  - PostgreSQL - http://www.postgresql.org/
  - PostgreSQL community projects - http://gborg.postgresql.org/
  - Advocacy - http://advocacy.postgresql.org/
  - Developers - http://developers.postgresql.org/
  - List archives - http://archives.postgresql.org/
  - Postgres performance tips - http://PowerPostgreSQL.com/
- PostgreSQL документация
  - http://www.postgresql.org/docs/
  - http://www.archonet.com/pgdocs/pgnotes.html
- PostgreSQL презентации
  - http://candle.pha.pa.us/main/writings/computer.html
  - http://www.linux-mag.com/2004-02/postgresql_01.html
  - http://users.bigpond.net.au/rmoonen/Jason_Godden2/mlug_20030822.html
  - http://tinyurl.com/57l33

Благодаря за вниманието!

Имате ли някакви въпроси?