A PostgreSQL Overview

Joshua Tolley
eggyknap
End Point Corp.

A PostgreSQL Overview

Joshua Tolley
eggyknap
End Point Corp.

September 24, 2010

- Began in 1986 under Dr. Michael Stonebraker at U. C. Berkeley as a follow-up to Ingres
- Open-sourced in 1996 after its QUEL language was replaced with SQL
- Distributed under the PostgreSQL license, similar to the BSD and MIT licenses.
- Supports many platforms and operating systems
- Large and active community, including several support companies and commercial forks
- Designed for extensibility, stability, scalability, and compliance with standards
- Currently maintained by volunteer developers worldwide, and a six-person core team
- Exceptionally well documented
- No single organization owns PostgreSQL

- Process-based, not thread-based. Communication between processes occurs using various OS-level locking structures and shared memory.

Advantages

- Simpler programming. Problems on one backend can't as easily mess up other backends
- Easier to support many architectures

Disadvantages

- Longer startup times for new connections (use a pooler)
- Shared memory applications are particularly slow on Windows; UNIX systems don't have that problem

- ACID-compliant transactions (Atomicity, Consistency, Isolation, Durability)
- Multi-version concurrency control
  - Some systems (Oracle, MySQL's InnoDB) use a redo log; PostgreSQL stores multiple versions of a row within the table itself
  - Each row version is marked with transaction IDs to determine its visibility
  - "Vacuum" cleans out rows that are too old to be visible
  - ROLLBACK becomes very fast this way; vacuuming and table bloat are notable concerns

```
josh@eddie:~/tmp/pgsql$ initdb -D data -U josh
The files belonging to this database system will be owned
by user "josh". This user must also own the server process

The database cluster will be initialized with locale
en_US.UTF-8. The default database encoding has
accordingly been set to UTF8.

...

WARNING: enabling "trust" authentication for local
connections. You can change this by editing pg_hba.conf
or using the -A option the next time you run initdb.

Success. You can now start the database server using:

    postgres -D data
or
    pg_ctl -D data -l logfile start
```

```
josh@eddie:~/tmp/pgsql$ pg_ctl -D data/ -l logfile start
server starting
josh@eddie:~/tmp/pgsql$ cat logfile
LOG:  autovacuum launcher started
LOG:  database system is ready to accept connections
```

The initdb application creates a postgres database. When `pg_ctl` starts PostgreSQL, on UNIX it opens a socket, and on Windows it listens on TCP 5432.

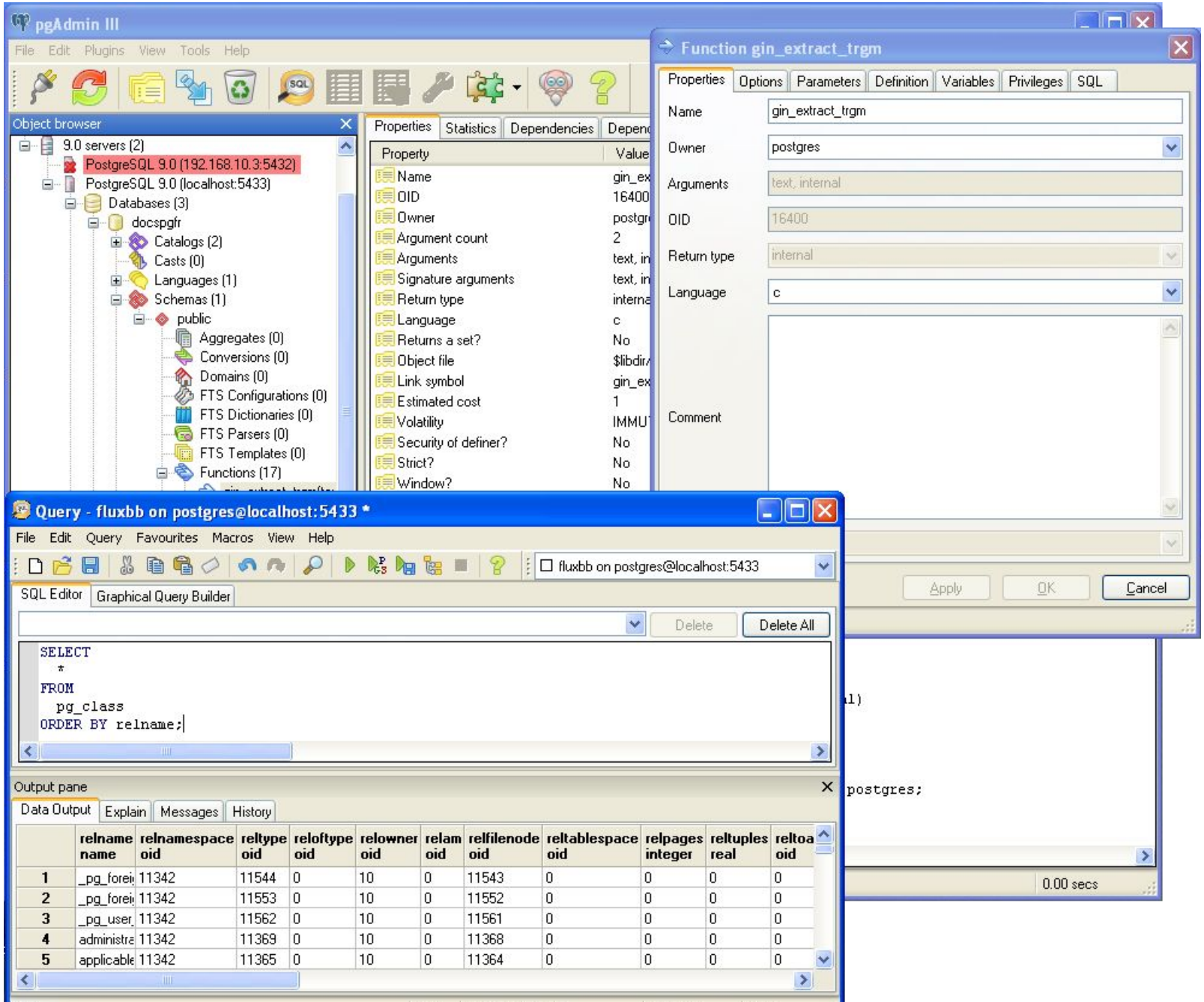
psql is a command-line client application that ships with the PostgreSQL code itself

```
josh@eddie:~/tmp/pgsql$ psql postgres
psql (9.0.0)
Type "help" for help.

5433 josh@postgres#
```

psql features online documentation, tab completion and readline support (are you listening, Oracle?), many simple keyboard commands for manipulating queries and loading instructions from outside the session

For those who prefer a GUI, other options exist, notably PgAdminIII.



PostgreSQL correctly handles input validation, triggers, data integrity constraints

- NOT NULL
- DEFAULT values
- Foreign and unique keys
- Full, mature trigger support
- Correct three-valued logic (i.e. True != False != NULL)

Why this might matter:

```
mysql> SELECT user_id, expire_date FROM users
WHERE user_id = 1710001;
+---------+-------------+
| user_id | expire_date |
+---------+-------------+
| 1710001 | 0390-08-00  |
+---------+-------------+
1 row in set (0.05 sec)
```

PostgreSQL's query parser supports very complex queries without irritating workarounds.

- Subqueries
- Set operations (INTERSECT, UNION)
- Standard SQL operators (COALESCE, CASE, NULLIF)
- Arrays
- Window functions
- Recursive queries

Rank employees in each department by salary

```
SELECT first, last, salary, department,
    RANK() OVER (
        PARTITION BY department
        ORDER BY salary DESC
    )
FROM employee
```

... and get this:

```
 first   |   last    | salary |   department    | rank
---------+-----------+--------+-----------------+------
 john    | doe       |  99000 | administration  |    1
 speedy  | gonzales  |  96000 | development     |    1
 hans    | reiser    |  93000 | development     |    2
 martha  | stewart   |  90000 | development     |    3
 fred    | rogers    |  97000 | sales           |    1
 carly   | fiorina   |  95000 | sales           |    2
 johnny  | carson    |  89000 | sales           |    3
(7 rows)
```

Recursive query to retrieve management hierarchy:

```
# WITH RECURSIVE t (id, managernames) AS (
    SELECT e.id, first || ' ' || last
        AS managernames
    FROM employee e WHERE manager IS NULL
        UNION ALL
    SELECT e.id,
    first || ' ' || last || ', ' || managernames
        AS managernames
    FROM employee e
    JOIN t ON (e.manager = t.id)
    WHERE manager IS NOT NULL
)
SELECT e.id, first || ' ' || last AS name,
    managernames
FROM employee e JOIN t ON (e.id = t.id);
```

...and get this...

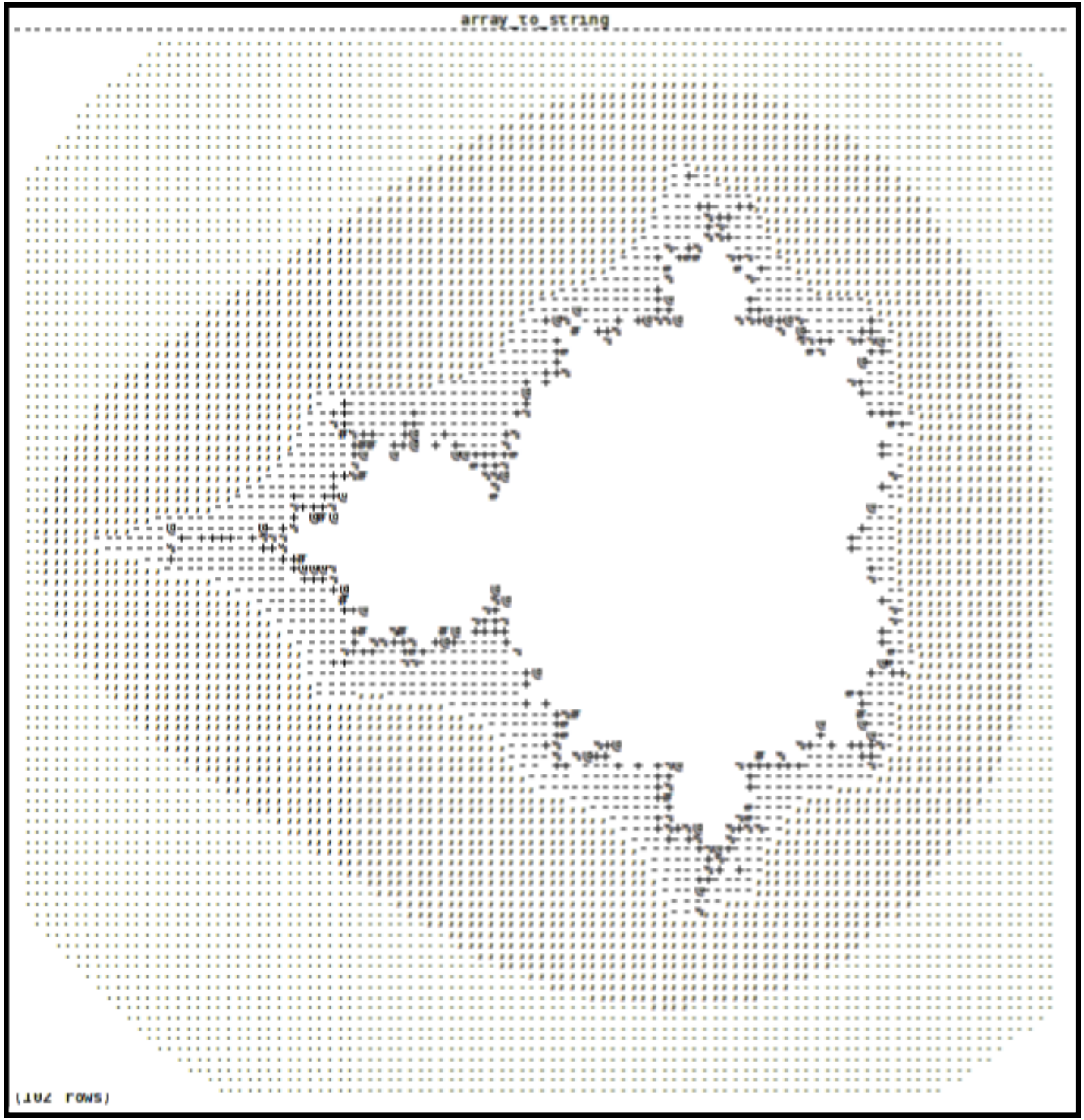
```
 id |      name         |  managernames
----+-------------------+-----------------------------
  1 | john doe          | john doe
  2 | fred rogers       | fred rogers, john doe
  3 | speedy gonzales   | speedy gonzales, john doe
  4 | carly fiorina     | carly fiorina, john doe
  5 | hans reiser       | hans reiser, fred rogers,
    |                   |  john doe
  6 | johnny carson     | johnny carson, hans reiser,
    |                   |  fred rogers, john doe
  7 | martha stewart    | martha stewart, speedy
    |                   |  gonzales, john doe
(7 rows)
```

```
WITH RECURSIVE x(i) AS
(VALUES(0) UNION ALL SELECT i + 1 FROM x WHERE i < 101),
Z(Ix, Iy, Cx, Cy, X, Y, I)
AS (
    SELECT Ix, Iy, X::float, Y::float, X::float, Y::float, 0
    FROM (SELECT -2.2 + 0.031 * i, i FROM x) AS xgen(x,ix)
        CROSS JOIN
    (SELECT -1.5 + 0.031 * i, i FROM x) AS ygen(y,iy)
        UNION ALL
    SELECT
        Ix, Iy, Cx, Cy, X * X - Y * Y + Cx AS X,
        Y * X * 2 + Cy, I + 1
    FROM Z
    WHERE X * X + Y * Y < 16.0 AND I < 27),
Zt (Ix, Iy, I) AS (
    SELECT Ix, Iy, MAX(I) AS I
    FROM Z GROUP BY Iy, Ix
    ORDER BY Iy, Ix
)
SELECT array_to_string(
    array_agg(
        SUBSTRING(' .,,,-----++++%%%%@@@@#### ',
            GREATEST(I,1), 1)
    ),''
)
FROM Zt GROUP BY Iy ORDER BY Iy;
```

(yes, this query is SQL-spec compliant)

A PostgreSQL Overview

Joshua Tolley
eggyknap
End Point Corp.



PostgreSQL allows users to customize the database in many different ways:

- Functions (including custom aggregates and window functions)
- Data types
- Index types
- Text search configurations (dictionaries, parsers, etc.)
- Procedural langauges
- Query planners
- All sorts of miscellaneous hooks within the backend

PostgreSQL offers many "procedural languages" for writing database functions. Four ship with the software by default.

- PL/pgSQL (resembles Oracle's PL/SQL)
- PL/Perl
- PL/Python
- PL/Tcl

Additionally, many others are available from other sources

- PL/Ruby
- PL/Java
- PL/Scheme
- PL/Lua
- PL/PHP
- PL/R
- PL/LOLCODE

```
5433 josh@postgres# DO $$
HAI
    BTW Calculate pi using Gregory-Leibniz series
    BTW This method does not converge particularly quickly...
    I HAS A PIADD ITZ 0.0
    I HAS A PISUB ITZ 0.0
    I HAS A ITR ITZ 0
    I HAS A T1
    I HAS A T2
    I HAS A PI ITZ 0.0
    I HAS A ITERASHUNZ ITZ 1000

    IM IN YR LOOP
        T1 R QUOSHUNT OF 4.0 AN SUM OF 3.0 AN ITR
        T2 R QUOSHUNT OF 4.0 AN SUM OF 5.0 AN ITR
        PISUB R SUM OF PISUB AN T1
        PIADD R SUM OF PIADD AN T2
        ITR R SUM OF ITR AN 4.0
        BOTH SAEM ITR AN BIGGR OF ITR AN ITERASHUNZ, O RLY?
            YA RLY, GTFO
        OIC
    IM OUTTA YR LOOP
    PI R SUM OF 4.0 AN DIFF OF PIADD AN PISUB
    VISIBLE "PI R: "
    VISIBLE PI
    FOUND YR PI
KTHXBYE
$$ LANGUAGE PLLOLCODE;
NOTICE:  3.143589
DO
```

Several useful data types ship with PostgreSQL, or are available from the community.

- Geometry and geography data (PostGIS outclasses all open source and most commercial GIS databases)
- Network data types (cidr, inet, macaddr)
- Array types
- User-definable composite types
- Intervals

Various projects have built very specialized data types using PostgreSQL, for storing such things as probability distributions, mammography images, and gene sequences

PostgreSQL functions very well within mission-critical environments, and is widely used by large enterprises:

- Afilias, the world's second largest internet registrar, uses PostgreSQL to run their domain name database
- Skype uses PostgreSQL as their main database platform
- MyYearbook.com, rated in the top 10 social networking sites worldwide in terms of unique visitors, uses PostgreSQL for most everything
- Nippon Telegraph and Telepone uses PostgreSQL heavily
- Etsy runs their e-commerce site on PostgreSQL
- NASA is working on using PostgreSQL in the space station

Replication comes in many forms:

- Hot standby: built into PostgreSQL, allows read-only slave databases asynchronously updated from a single master
- Bucardo: trigger-based replication system allows asynchronous multi-master replication
- Slony, Londiste: trigger-based master-slave replication

Work is underway to introduce synchronous replication in PostgreSQL 9.1

Questions?