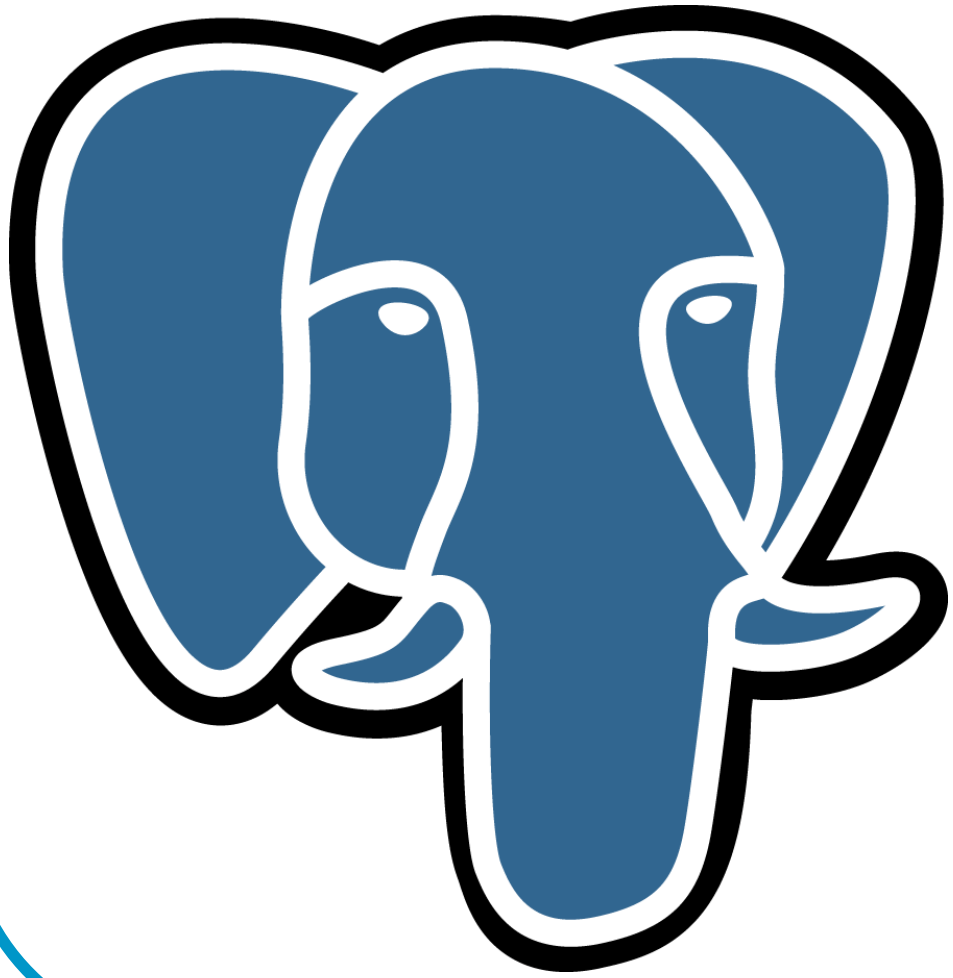
The PostgreSQL Advantage



Magnus Hagander

PostgreSQL Global Development Group

This talk is not about

- Transactions, ACID compliance
- ANSI SQL compliance
- Referential Integrity
- Stored procedures
- Subselects
- Because everybody does this
  - Even MySQL

Agenda

- Multilanguage stored procedures
- Type extensibility
- Partial indexes
- Expressional indexes
- GiST indexes
- Host based authentication
- Rules
- Asynchronous notifications

Agenda

- Multilanguage stored procedures
- Type extensibility
- Partial indexes
- Expressional indexes
- GiST indexes
- Host based authentication
- Rules
- Asynchronous notifications

Multilanguage stored procedures

- Stored procedures in different languages
- Interpreted and compiled
- Many languages supported
  - SQL
  - PL/pgsql
  - PL/perl, PL/python, PL/tcl, PL/PHP, PL/R
  - PL/Java, PL/J
  - C/C++/etc
- Trusted and untrusted

Multilanguage stored procedures

- The right tool for the job
  - Database work – PL/pgsql
  - String handling – PL/perl
  - Statistical analysis – PL/R
  - Etc
- Developer consolidation
  - Re-use of skills
  - Re-use of libraries

Multilanguage stored procedures

```
CREATE OR REPLACE FUNCTION
 validate_pnr(text) RETURNS bool AS $$

return 'f' unless ($_[0] =~
  /^(\d{2})([01]\d)([0123]\d)-(\d|T)(\d|F)\d{2}$/);
return 'f' if ($2>12);
return 'f' if ($3>31);
… … … … …
return ($c == int(chop($_[0])))?'t':'f‘
$$
LANGUAGE 'plperl' IMMUTABLE STRICT;
```

Agenda

- Multilanguage stored procedures
- Type extensibility
- Partial indexes
- Expressional indexes
- GiST indexes
- Host based authentication
- Rules
- Asynchronous notifications

Type extensibility

- Create custom datatypes:
  - Apply rules to content
  - Represent complex datastructures
- Create custom operators

Type extensibility - domains

- Based on standard type such as varchar
- Adds mandatory check constraint
- Can call any stored procedure in any language

```
CREATE DOMAIN personnummer
    AS varchar(11)
    NOT NULL
     CONSTRAINT personnummer_check
      CHECK (validate_pnr(value))
```

CREATE TABLE test (pnr personnummer)

Type extensibility - native

- Write accessor functions in C
  - Input/output functions
  - Send/receive functions
- Can store *any* type of data
- Classic example: complex number
- Custom operators in C

Agenda

- Multilanguage stored procedures
- Type extensibility
- Partial indexes
- Expressional indexes
- GiST indexes
- Host based authentication
- Rules
- Asynchronous notifications

Partial indexes

- What kills performance?
  - I/O
- Indexes help
  - Read only the interesting rows of data
  - B-tree access to index data
- Partial indexes help even more
  - Read only the interesting rows of *indexes*
  - Then only the interesting rows of data
- ”Free updates”

Partial indexes

- Example: Web server logs
- 99% (at least!) are successes
  - HTTP 200 – 299
- Run regular checks for referrers causing errors
- Index on (status,referrer)

```
SELECT referrer,count(*) FROM weblog
  WHERE status<200 OR status>299
  GROUP BY referrer
```

Partial indexes

```
CREATE INDEX fail_ref ON weblog
  (referrer) WHERE (status < 200
  OR status > 299)

SELECT referrer,count(*) FROM
  weblog WHERE status<200 OR
  status>299 GROUP BY referrer
```

Agenda

- Multilanguage stored procedures
- Type extensibility
- Partial indexes
- Expressional indexes
- GiST indexes
- Host based authentication
- Rules
- Asynchronous notifications

Expressional indexes

- Index the *result of a function call or expression*
- Function must be *IMMUTABLE*
  - Depends only on argument, nothing else
- ”Expensive” updates, cheap search
- Example: case-insensitive substring matching

```
CREATE INDEX foo ON bar (lower(str))
SELECT ...
  WHERE lower(str) LIKE 'test%'
```

Agenda

- Multilanguage stored procedures
- Type extensibility
- Partial indexes
- Expressional indexes
- GiST indexes
- Host based authentication
- Rules
- Asynchronous notifications

GiST indexes

- Generalized Search Tree
- ”Pluggable indexing”
- Indexing of custom operators
- Array indexing (intarray)
- Spatial indexing (PostGIS)
- Full Text Indexing (tsearch2/OpenFTS)
- Transactionally safe, ACID compliant, MVCC

GiST indexes – tsearch2

```
CREATE TABLE test (t tsvector not null);
CREATE INDEX test_fti ON test using GIST (t);

INSERT INTO test (t) VALUES
    (to_tsvector('testing full text search'));

BEGIN TRANSACTION;
  INSERT INTO test (t) VALUES
    (to_tsvector('testing one more row'));
  SELECT count(*) FROM test
    WHERE t @@ to_tsquery('test');
ROLLBACK;

SELECT count(*) FROM test
  WHERE t @@ to_tsquery('test');
```

Agenda

- Multilanguage stored procedures
- Type extensibility
- Partial indexes
- Expressional indexes
- GiST indexes
- Host based authentication
- Rules
- Asynchronous notifications

Host Based Authentication

- Does *not* authenticate hosts
  - Still authenticates users!
- Different authentication methods and restrictions for different clients
- Some password-based, some not
  - Trust
  - MD5
  - PAM
  - Kerberos
  - Ldap
  - Ident
  - etc

Host Based Authentication

```
local      all  all                          trust
# Windows clients
hostssl    all  all  10.0.0.0/8               krb5
# Web server
host       one  one  172.16.1.10/32 md5
host       two  two  172.16.1.10/32 md5
# Mac clients
hostssl    @macdb    +macusr
                     11.0.0.0/24    ldap
```

Agenda

- Multilanguage stored procedures
- Type extensibility
- Partial indexes
- Expressional indexes
- GiST indexes
- Host based authentication
- Rules
- Asynchronous notifications

Rules

- Custom rewrite of statements
  - ALSO
  - INSTEAD OF
  - NOTHING
- Somewhat like triggers
  - Operates on *queries* instead of *data*
- Advanced view functionality
  - Standard views just a subset
  - *Any* view is updatable

Rules

```
CREATE TABLE tab1(a int PRIMARY KEY, b
  int, c int);
CREATE VIEW v1 AS SELECT a,b,c FROM tab1

db=# insert into tab1 values (1,1,1);
INSERT 0 1

db=# select * from v1;
 a | b | c
---+---+---
 1 | 1 | 1
```

Rules

```
CREATE RULE v1_ins AS
  ON INSERT TO v1 DO INSTEAD NOTHING;

db=# insert into v1 values (2,2,2);
INSERT 0 0

db=# select * from v1;
 a | b | c
---+---+---
 1 | 1 | 1
```

Rules

```
CREATE RULE v1_upd AS
   ON UPDATE TO v1 DO INSTEAD
    UPDATE tab1 SET b=NEW.b
     WHERE tab1.a=NEW.a;

db=# update v1 set b=2,c=2 where a=1;
UPDATE 1

db=# select * from v1;
 a | b | c
---+---+---
 1 | 2 | 1
```

Agenda

- Multilanguage stored procedures
- Type extensibility
- Partial indexes
- Expressional indexes
- GiST indexes
- Host based authentication
- Rules
- Asynchronous notifications

Async notifications

- Signals between clients
- Transaction aware
- Multiple senders, multiple receivers
- Typical app: Cache invalidation
  - Use a queue table with trigger
  - Use triggers on main table
  - Use single connection daemon process
  - Load throttling?

Async notifications

```
        Client 1                          Client 2
postgres=# LISTEN foo;
LISTEN
                                  postgres=# NOTIFY foo;
Postgres=# SELECT 1;
      1


Asynchronous
   notification "foo"
   received from server
   process with PID 2948.


postgres=#
```

Questions

Questions?