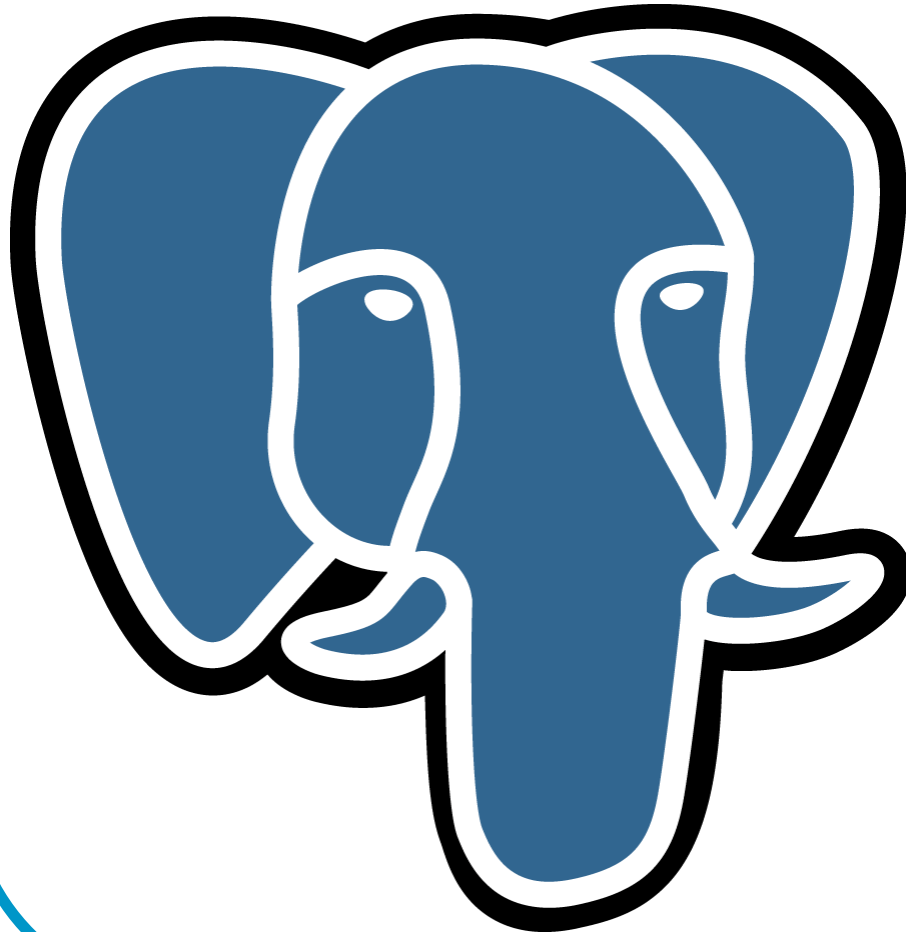


# The PostgreSQL Advantage



Magnus Hagander

PostgreSQL Global  
Development  
Group

# This talk is not about

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

# Agenda

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

# Agenda

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

# Multilanguage stored procedures

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

# Multilanguage stored procedures

- 1 The right tool for the job
  - 1 Database work – PL/pgsql
  - 1 String handling – PL/perl
  - 1 Statistical analysis – PL/R
  - 1 Etc
- 1 Developer consolidation
  - 1 Re-use of skills
  - 1 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 'perl' IMMUTABLE STRICT;
```

# Agenda

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

# Type extensibility

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

# Type extensibility - domains

- 1 Based on standard type such as varchar
- 1 Adds mandatory check constraint
- 1 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

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

# Agenda

- 1 Multilanguage stored procedures
- 1 Type extensibility
- 1 **Partial indexes**
- 1 Expressional indexes
- 1 GiST indexes
- 1 Host based authentication
- 1 Rules
- 1 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

- 1 Example: Web server logs
- 1 99% (at least!) are successes
  - 1 HTTP 200 – 299
- 1 Run regular checks for referrers causing errors
- 1 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

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

# Expressional indexes

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

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

# Agenda

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

# GiST indexes

- 1 Generalized Search Tree
- 1 "Pluggable indexing"
- 1 Indexing of custom operators
- 1 Array indexing (intarray)
- 1 Spatial indexing (PostGIS)
- 1 Full Text Indexing (tsearch2/OpenFTS)
- 1 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

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

# Host Based Authentication

- 1 Does *not* authenticate hosts
  - 1 Still authenticates users!
- 1 Different authentication methods and restrictions for different clients
- 1 Some password-based, some not
  - 1 Trust
  - 1 MD5
  - 1 PAM
  - 1 Kerberos
  - 1 Ident
  - 1 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

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

# Rules

- 1 Custom rewrite of statements
  - 1 ALSO
  - 1 INSTEAD OF
  - 1 NOTHING
- 1 Somewhat like triggers
  - 1 Operates on *queries* instead of *data*
- 1 Advanced view functionality
  - 1 Standard views just a subset
  - 1 *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

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

# Async notifications

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

# Async notifications

Client 1

```
postgres=# LISTEN foo;  
LISTEN
```

```
Postgres=# SELECT 1;  
1
```

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

```
postgres=#
```

Client 2

```
postgres=# NOTIFY foo;
```

# Questions

Questions?