Hidden gems of PostgreSQL

Magnus Hagander

magnus@hagander.net

Redpill Linpro

&

PostgreSQL Global Development Group

Agenda

- Advanced FTI
- Pgcrypto
- Contrib modules in general

Transactional DDL

- Hidden agenda!

Transactional DDL

```
testdb=# select count(*) from testtable;
  1000

testdb=# begin transaction;
BEGIN

testdb=# drop table testtable;
DROP TABLE
```

Transactional DDL

```
proddb=# select count(*) from testtable;
  1000

proddb=# begin transaction;
BEGIN

proddb=# drop table testtable;
DROP TABLE

proddb=# select count(*) from testtable;
ERROR:  relation "testtable" does not exist
```

Transactional DDL

```
proddb=# select count(*) from testtable;
  10

p   ddb=#    gi   ansacti

BE   N

proddb=# drop table testtable;
DROP TABLE

proddb=# select count(*) from testtable;
ERROR:  relation "testtable" does not exist
```

OOPS!

Transactional DDL

```
proddb=# select count(*) from testtable;
  1000

pr
BE

pr
DR

proddb=# select count(*) from testtable;
ERROR:  relation "testtable" does not exist
```

```
proddb=# rollback;
ROLLBACK

proddb=# select count(*) from testtable;
  1000
```

Redpill Linpro

PostgreSQL

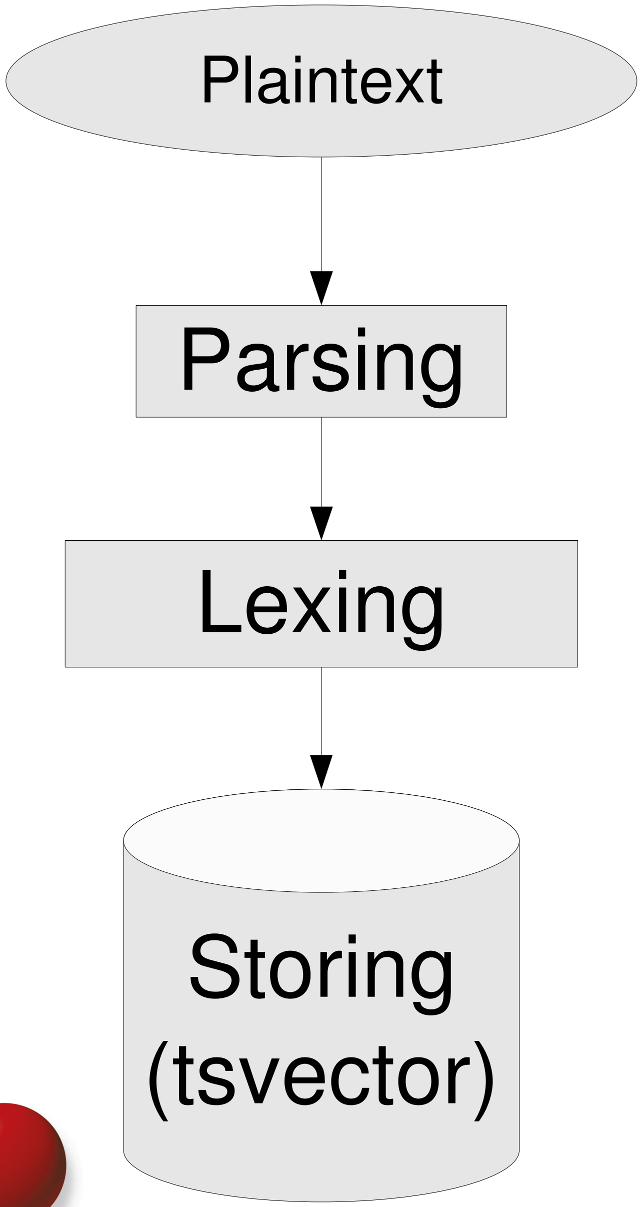
Transactional DDL

- Bottom line: PostgreSQL is always ACID!
  - You don't have to give anything up

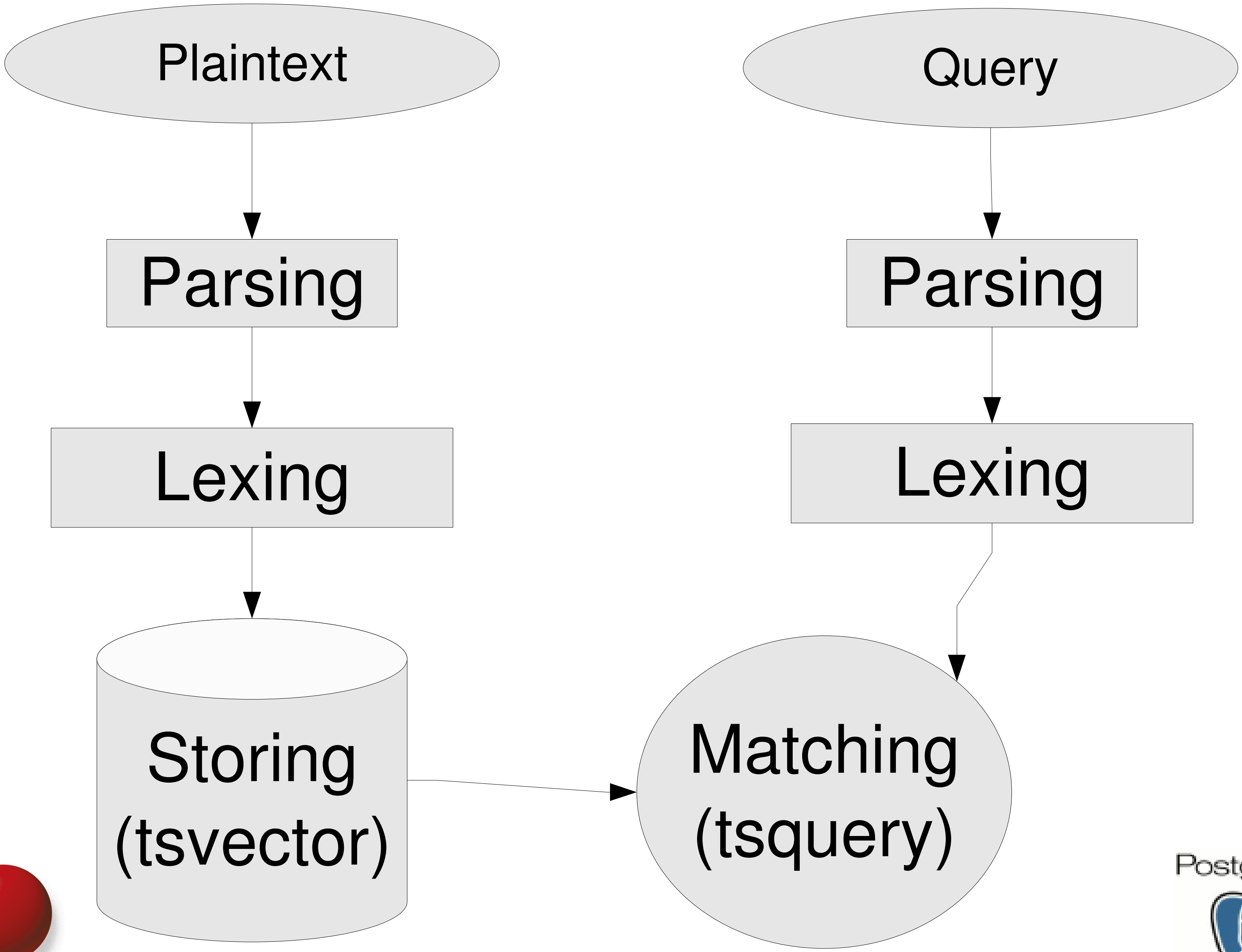
Advanced FTI

- Tsearch2 => Full Text Indexing
- Included in 8.3+
- Always available

FTI concepts



FTI concepts



FTI concepts

```
postgres=> SELECT to_tsvector('postgresql is a database');
        to_tsvector
-----------------------------
 'databas':4 'postgresql':1

postgres=# set default_text_search_config ='swedish';
SET

postgres=> SELECT to_tsvector('postgresql is a database');
               to_tsvector
-----------------------------------------
 'a':3 'databas':4 'is':2 'postgresql':1
```

FTI basics – setting up

```
postgres=# CREATE TABLE t(a SERIAL PRIMARY KEY,
   txt text,
   fti tsvector);

postgres=# CREATE TRIGGER
   t_fti_update_trigger
   BEFORE INSERT OR UPDATE ON t
   FOR EACH ROW EXECUTE PROCEDURE
   tsvector_update_trigger(fti,'pg_catalog.english',txt);
```

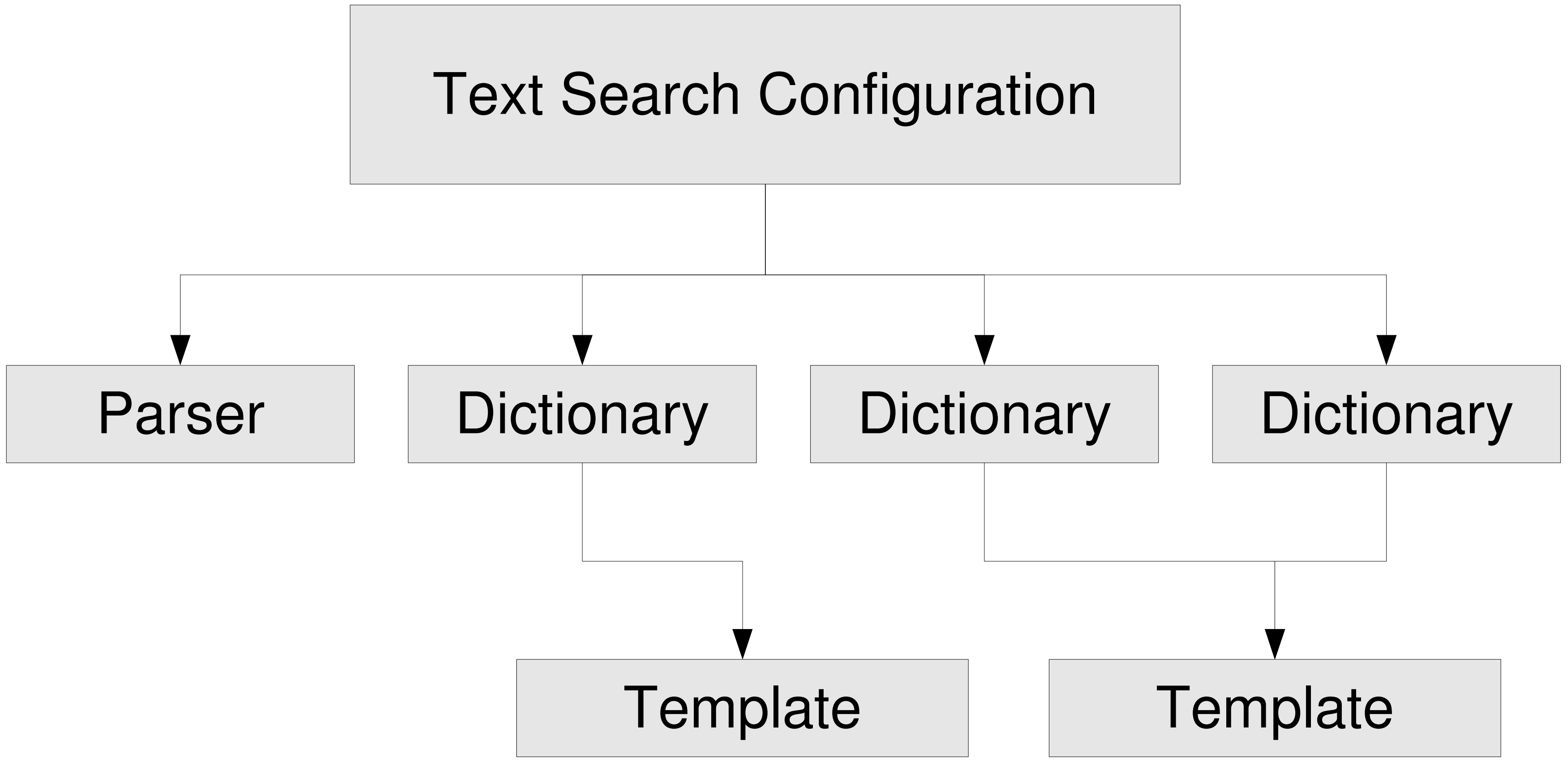
FTI basics - data

```
postgres=# INSERT INTO t(txt) VALUES
   ('postgresql is a database'),
   ('python is a language');
INSERT 0 2


postgres=# select * from t;
-[ RECORD 1 ]--------------------
a   | 2
txt | postgresql is a database
fti  | 'databas':4 'postgresql':1
```

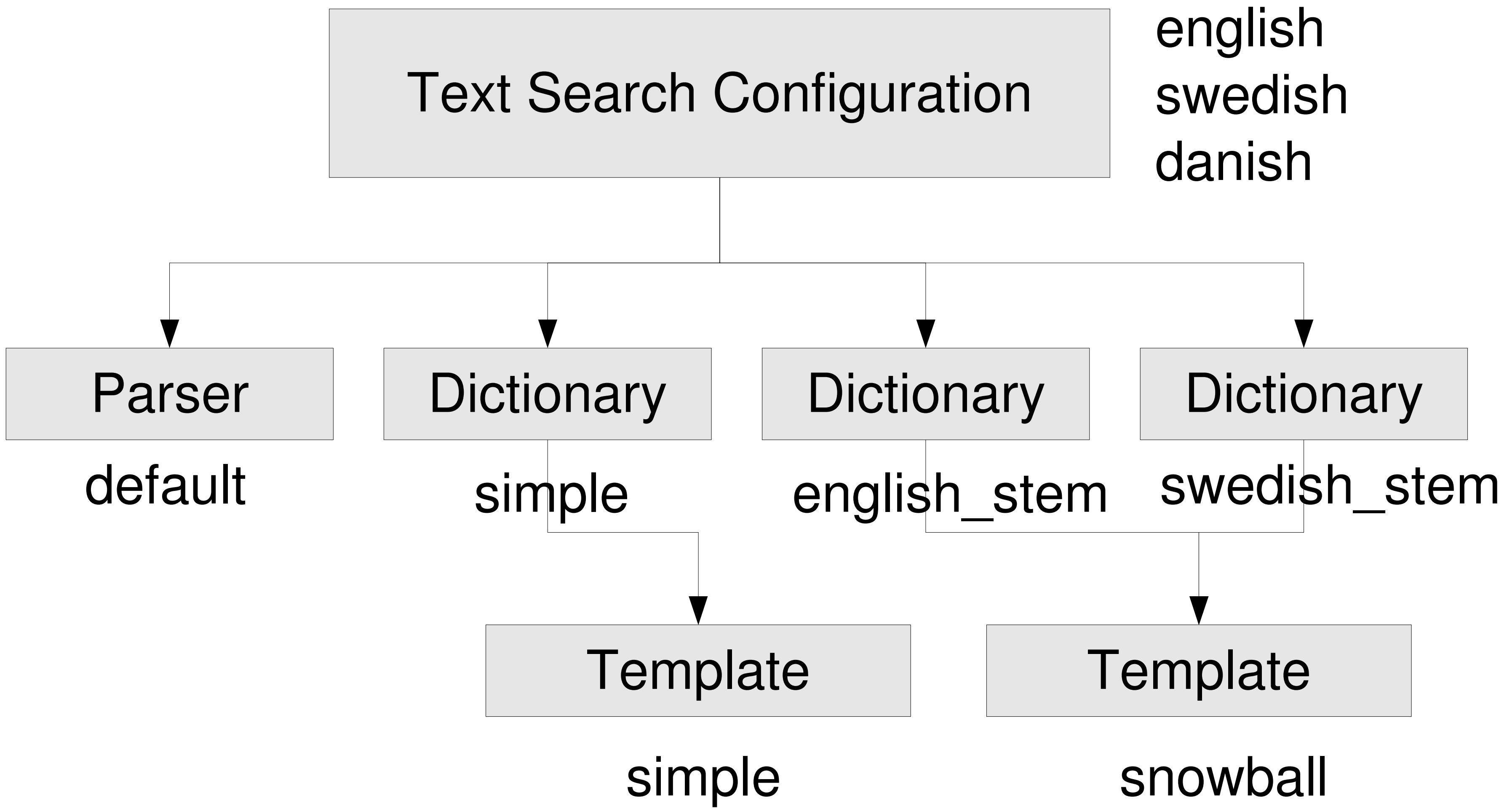
FTI basics - querying

```
postgres=# SELECT * FROM t
   WHERE fti @@ plainto_tsquery('database');
-[ RECORD 1 ]--------------------
a   | 2
txt | postgresql is a database
fti | 'databas':4 'postgresql':1

postgres=# SELECT * FROM t
   WHERE fti @@ plainto_tsquery('is');
NOTICE:  text-search query contains only stop words or
   doesn't contain lexemes, ignored
(No rows)
```

FTI pieces


Text Search Configuration
Parser
Dictionary
Dictionary
Dictionary
Template
Template

FTI pieces


Text Search Configuration
english
swedish
danish
Parser
default
Dictionary
simple
Dictionary
english_stem
Dictionary
swedish_stem
Template
simple
Template
snowball

FTI dictionary templates

- Simple
- Snowball
- Ispell
- Synonym
- <custom>

FTI synonym dictionaries

- Simple textfile (UTF8!)
- $PREFIX/share/tsearch-data/pgdict.sym

```
pgsql       postgresql
postgres    postgresql
postgre     postgresql
```

FTI synonym dictionaries

```
postgres=> CREATE TEXT SEARCH DICTIONARY pgdict
    (template=synonym, synonyms=pgdict);
CREATE TEXT SEARCH DICTIONARY

postgres=# ALTER TEXT SEARCH CONFIGURATION english
 ALTER MAPPING FOR asciiword,word WITH
   pgdict,english_stem;
ALTER TEXT SEARCH CONFIGURATION

postgres=# SELECT to_tsvector('pgsql is a database');
  to_tsvector
-----------------------------
 'databas':4 'postgresql':1
```

FTI - multilanguage

- to_tsvector takes a config name

```
postgres=# SELECT to_tsvector('english','pgsql is a database');
        to_tsvector
------------------------------
 'databas':4 'postgresql':1


postgres=# SELECT to_tsvector('danish','pgsql is a database');
            to_tsvector
---------------------------------------
 'a':3 'databas':4 'is':2 'pgsql':1
```

FTI - multilanguage

- Config name stored in table

```
postgres=# CREATE TABLE t(conf regconfig, t text, fti tsvector);
CREATE TABLE

postgres=# CREATE TRIGGER t_fti_update_trigger
  BEFORE INSERT OR UPDATE ON t
  FOR EACH ROW EXECUTE PROCEDURE
     tsvector_update_trigger_column(fti,conf,t);
CREATE TRIGGER
```

FTI - multilanguage

```
postgres=# insert into t (conf,t) values
  ('english','postgresql is a database'),
  ('swedish','postgresql is a database'),
  ('swedish','postgresql är en databas');
INSERT 0 3

postgres=# select * from t;
  conf   |            t             |            fti
---------+--------------------------+----------------------------
 english | postgresql is a database | 'databas':4 'postgresql':1
 swedish | postgresql is a database | 'a':3 'databas':4 'is':2 'post
 swedish | postgresql är en databas | 'datab':4 'postgresql':1
(3 rows)
```

Redpill Linpro

PostgreSQL

FTI – Two Index Types

- GIN
  - Inverted index
  - Very fast searches, slow updates
- GIST
  - Generalized Indexed Search Trees
  - Fast searches, fast updates
- Combinations!
  - UNION ALL / partitioning

FTI – two index types

- Speed difference?
- Example: ~550,000 emails, PostgreSQL 8.2:
  - No index: 6,000 ms
  - GiST index: 250ms
  - GIN index: 6ms

FTI - summary

- Almost endless configurability
- Very good matches if configured properly
- Always ACID safe!

Pgcrypto

- Advanced cryptography functions
- “Normal” crypto
  - DES, 3DES, AES, Blowfish
- “Normal” hash
  - MD5, SHA1, SHA256, SHA512 etc
- Strong random numbers
- PGP wrappers

Pgcrypto – use-case

- In-database authentication
- DB level API
- No need to duplicate security code
- Example: postgresql.org

Pgcrypto - use-case

- Let's start with a table...

```
CREATE TABLE users (
  userid text NOT NULL PRIMARY KEY,
  pwdhash TEXT NOT NULL
);
```

Salt + hash!

Pgcrypto – creating users

```
CREATE OR REPLACE FUNCTION auth_create(_userid text,_pwd text)
 RETURNS void
AS $$
BEGIN
 INSERT INTO users (userid, pwdhash)
   VALUES (lower(_userid), crypt(_pwd, gen_salt('bf')));
END;
$$ LANGUAGE plpgsql;
```

Pgcrypto – checking login

```
CREATE OR REPLACE FUNCTION auth_login(_userid text, _pwd text)
 RETURNS boolean
AS $$
BEGIN
  PERFORM * FROM users WHERE users.userid=lower(_userid)
    AND pwdhash = crypt(_pwd, users.pwdhash);
  IF FOUND THEN
    RETURN 't';
  END IF;
  RETURN 'f';
END;
$$ LANGUAGE plpgsql;
```

Pgcrypto - testing

```
auth=# select auth_create('mha','secret');

auth=# select * from users;
 userid |                                          pwdhash
--------+-----------------------------------------------
 mha    | $2a$06$fhb1j.yj.lbHvPVugCEAgO$2a$06$fhb1j.yj.lbHv

auth=# select auth_login('mha','secret');
 t

auth=# select auth_login('mha','public');
 f
```

Next step!

- Users should not see the hash!

```
auth=> select * from users;
 userid |                                        pwdhash
--------+-----------------------------------------------------------
 mha    | $2a$06$fhb1j.yj.lbHvPVugCEAgO$2a$06$fhb1j.yj.lb

auth=# revoke all on users from public;
REVOKE

auth=> select auth_login('mha','secret');
ERROR:  permission denied for relation users
```

Pgcrypto – checking login Mk II

```
CREATE OR REPLACE FUNCTION auth_login(_userid text, _pwd text)
 RETURNS boolean
AS $$
BEGIN
  PERFORM * FROM users WHERE users.userid=lower(_userid)
    AND pwdhash = crypt(_pwd, users.pwdhash);
  IF FOUND THEN
    RETURN 't';
  END IF;
  RETURN 'f';
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;
```

Using “setuid” functions

```
auth=> select auth_login('mha','secret');
 auth_login
-------------
 t
(1 row)

auth=> select * from users;
ERROR:  permission denied for relation users
```

Pgcrypto - summary

- Avoid application code duplication
- Isolate security sensitive data
- Uses crypto from OpenSSL when available
- Server-side only – don't forget to use TLS connections!

Contrib modules

- Additional modules and tools
- Some just examples
- Included in source, but not built by default
- Separate RPM/DEB
- Usually need to be enabled manually

Contrib modules - dblink

- Access remote databases inline
- Exposed as SRF, easily wrapped as a view
- Remote can be local
- Consider DBI-Link
  - http://pgfoundry.org/projects/dbi-link

Contrib modules - intarray

- For one-dimensional integer arrays
- Operators/functions
  - SELECT .. WHERE arrayfield && '{1,2}'
  - sort(arrayfield)
  - Idx(), uniq(), subarray() etc
- Indexing support!
  - Both GIST and GIN

Contrib modules - tablefunc

- Example of set returning functions
- Provides some useful functions as well
  - crosstab() - “pivot tables”
  - connectby() - trivial “recursive queries”

Contrib modules - pg_buffercache

- Real-time view of the buffer cache
- Very low-level view
- One row for each 8kb buffer page
- Which page, of which relation, in which database, dirty+usage etc
- **WARNING: locks!**

Contrib modules - pgstattuple

- Statistics about tables
- Tuple sizes
- Free space
- Dead rows/vacuum need

Contrib modules - summary

- A general mix of “stuff”
- Much is very useful – not just examples!
- API stability potential issue

Thank you!

Thank You!

Questions?

magnus@hagander.net