# PostgreSQL
## The World's Most Advanced Open Source Database

FOSS-STHLM, Feb 2010
Stockholm, Sweden

Magnus Hagander
Redpill Linpro AB

## PostgreSQL is a RDBMS

- Strictly speaking, ORDBMS
- Speaks SQL
  - d'uh!
- Stores your data
  - double d'uh!
- So what now?

## Distribution specific login

- RedHat and Debian use «ident»
  - `su postgres -c psql`
  - `su postgres -c createuser kalle`
- Might want to use «md5» remote?
  - Set a password!
  - `\password kalle`
- pg_hba.conf

# Basic configuration is tiny!

- shared_buffers = 24MB?
  - ¼ of RAM
- work_mem = 1MB?
  - Probably 10
- checkpoint_segments=3?
  - Start at 10?
- effective_cache_size=128MB?

## PostgreSQL *validates* your data

- Encoding
  - Recommended: UTF8
- Datatype
  - Exceeding varchar length throws **error**
  - Invalid data format is **always** error
- Constraints are **enforced**

## Dumps are not backups!

- Slow to take on large databases

## Dumps are not backups!

- Slow to take on large databases
- Only restores to backup time!
- Use PITR!
- With log shipping!

## Dumps are still pretty neat...

- *Transactionally safe snapshots!*
- No locks
  - Well, almost, can't do DDL
- Easy format for export/import
  - Full DDL and all data included
- Always use *custom* format!

## PostgreSQL is not SQLite/MySQL

(or mariadb, or drizzle, or xtradb, or ourdelta, or ....)

- Don't be afraid to throw work at the database
- In most cases, a lot smarter and faster!

## CTEs – the road to Turing Complete

```
WITH RECURSIVE x( s, ind ) AS
( SELECT sud, position( ' ' IN sud )
  FROM  (SELECT '53  7    6  195    98    6 8   6   34  8 3  17   2   6 6
28    419  5    8  79'::text
             AS sud) xx
  UNION ALL
  SELECT substr( s, 1, ind - 1 ) || z || substr( s, ind + 1 )
       , position(' ' IN repeat('x',ind) || substr( s, ind + 1 ) )
  FROM x
     ,  (SELECT gs::text AS z FROM generate_series(1,9) gs) z
  WHERE ind > 0
  AND NOT EXISTS ( SELECT NULL
                   FROM generate_series(1,9) lp
                   WHERE z.z = substr( s, ( (ind - 1 ) / 9 ) * 9 + lp, 1 )
                   OR    z.z = substr( s, mod( ind - 1, 9 ) - 8 + lp * 9, 1 )
                   OR    z.z = substr( s, mod( ( ( ind - 1 ) / 3 ), 3 ) * 3
                                      + ( ( ind - 1 ) / 27 ) * 27 + lp
                                      + ( ( lp - 1 ) / 3 ) * 6
                                   , 1 )
                 )
) SELECT s FROM x WHERE ind = 0;
```

## More useful CTEs

```
WITH RECURSIVE t(id, department, name, manager) AS
(
   SELECT id, department, name, manager
    FROM emp WHERE name='Kalle'
 UNION ALL
   SELECT emp.id,emp.department,emp.name,emp.manager
    FROM emp JOIN t ON t.manager=emp.id
)
SELECT * FROM t;
```

## Replication is for *replication*

- Q: Creating a index on a 500M row table locks my table for a day!
- A: Set up a second server, enable replication, add index there, and do failover

## Replication is for *replication*

- Q: Creating a index on a 500M row table locks my table for a day!
- ~~A: Set up a second server, enable replication, add index there, and do failover~~
- A: Use CREATE INDEX CONCURRENTLY

## Replication is for *replication*

- Q: Adding a column to my 100M row table locks my table for hours!
- ~~A: Set up a second server, enable replication, add column there, and do failover~~
- A: Just add the column, don't set a DEFAULT

## Bottom line

- Don't assume the database can't do it
  - (better than you)
- Assume it can, only workaround when it can't

## Upcoming 9.0 release

- Lots of new features!
  - Would take hours to talk about all...
- I'm just going to pick one....

## Let's do a challenge

- Task: room booking system
- Requirements: support high performance and concurrency
- Problem: conflict detection and resolution?

## Booking system

- Let's define a table

```
CREATE TABLE bookings(
  title text,
  room text,
  start timestamptz,
  end timestamptz
)
```

## Booking system

- The PERIOD datatype
  - Available on pgFoundry
  - Makes dealing with time intervals *much* nicer
  - *Not* a requirement, but easier
- Single datatype for start and end time

## Booking system

- Let's define a table

```
CREATE TABLE bookings(
  title text,
  room text,
  during period
)
```

## Booking system

- How to prevent the same room to be booked twice?
  - Or overlapping?
- Enforced, please!
  - In the system, not in the room!
- *Ideas?*

## Exclusion Constraints!

```
CREATE TABLE bookings(
   title text,
   room text,
   during period,
   EXCLUDE USING gist
      (room WITH =,
       during WITH &&)
)
```

```
NOTICE:  CREATE TABLE / EXCLUDE will create
implicit index
"bookings_room_during_exclusion" for table
"bookings"
```

## Constraint violations

```
INSERT INTO bookings values ('Features
talk', 'AW1.121', period('2010-02-06
17:30', '2010-02-06 18:15'));
```

```
ERROR:  conflicting key value violates exclusion
constraint "bookings_room_during_exclusion"
DETAIL:  Key (room, during)=(AW1.121, [2010-02-06
17:30:00+01, 2010-02-06 18:15:00+01)) conflicts
with existing key (room, during)=(AW1.121, [2010-
02-06 17:15:00+01, 2010-02-06 18:00:00+01)).
```

## Exclusion Constraints

- Any operator that can define differences
- Typically enforces non-overlap
  - Timeframes
  - Geometric (square/circle/line)
  - Geographical regions (PostGIS)
- Enforced in the database!

## Thank You!

*Questions?*

*magnus@hagander.net*
*http://blog.hagander.net/*
*Twitter: magnushagander*
*FreeNode: #postgresql:magnush*