The Accidental DBA

a guide for the perplexed

Josh Berkus
PostgreSQL Experts, Inc.
FOSS4G 2014

PGX
POSTGRESQL
EXPERTS, INC.

covered in this talk

- installation
- extensions
- installing PostGIS
- updates and upgrades
- configuration
- connections & security
- backups
- VACUUM
- query management

not covered

- older versions (< 9.1)
  - older PostGIS (< 2.0)
- schema design
- query rewriting
- indexes
- testing
- application stuff

replication

- included in the slides
- included in the exercises
- not covered in this workshop
- covered in the afternoon session

Exercises & Slides

- https://github.com/pgexperts/accidentalDBA
- accidentalDBA/tutorial/exercises.txt
- exercises are Vagrant VM
  - you needed to install this before you got here

“You know Linux, right?
You're in charge of the database now.”

“Efficiency”
“DevOps”
“Cloud”
“Growth Opportunity”

“We're not going to hire a DBA”

Y U no DBA?

1. limited budgets
2. shortage of operational staff
3. cheaper OSS databases

… you are the DBA now.

PostgreSQL 9.2.4 Documentation

The PostgreSQL Global Development Group

Table of Contents


Oh My God,
We're All Going To Die.


Don't Panic

vagrant up now

```
cd /dir/to/accidentalDBA/vagrant/
vagrant up
```

Installation

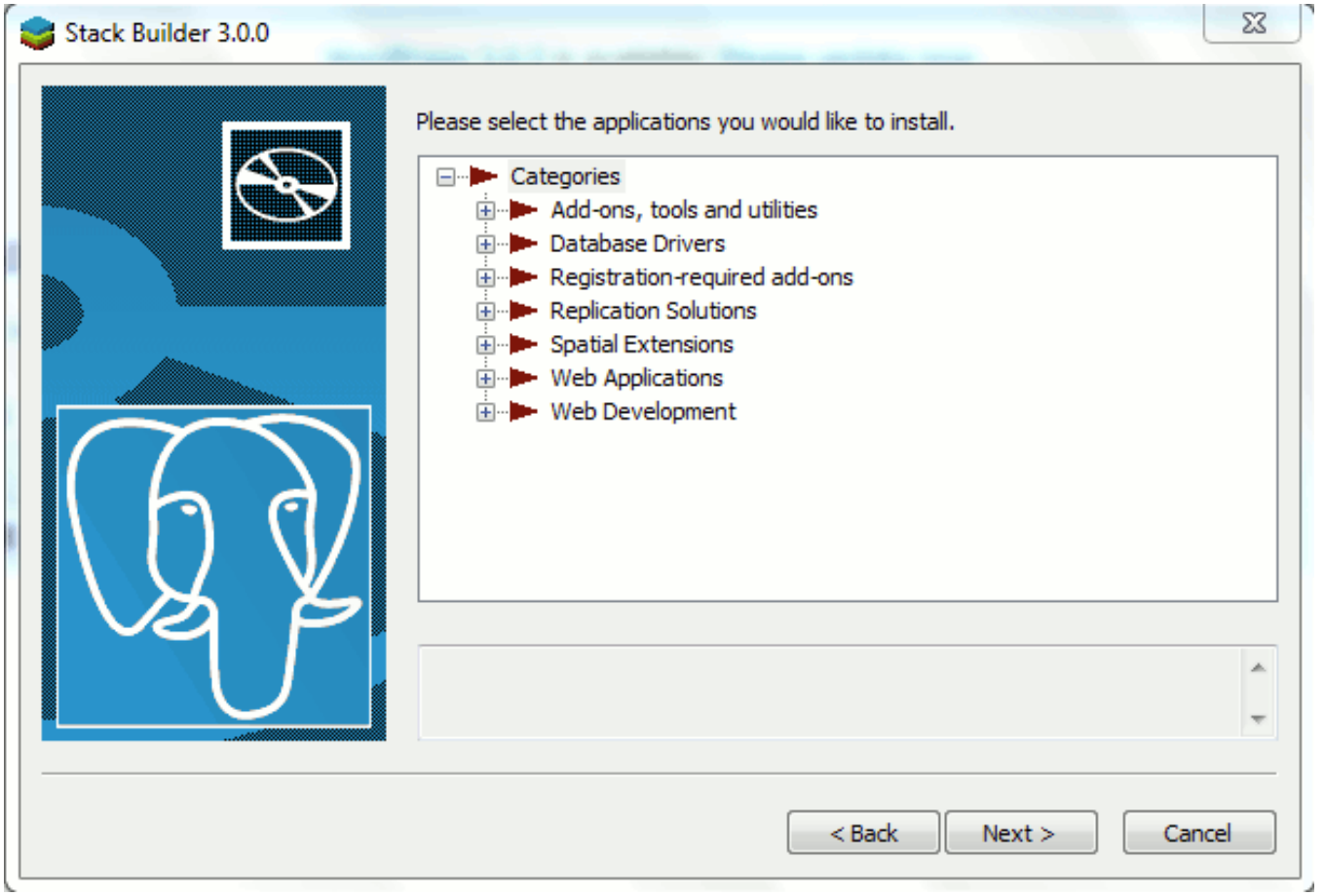
Linux: use packages!

- version not important?
- use the ones that come with your distro
  - Red Hat, Centos, SciLinux
  - Debian, Ubuntu
  - SuSE

Linux: use packages!

- need the latest version?
- alternate packages
  - Red Hat: yum.postgresql.org
  - Ubuntu: apt.postgresql.org
  - SuSE: build service
  - Debian: apt.postgresql.org

Windows/OSX


Stack Builder 3.0.0
Please select the applications you would like to install.
Categories
Add-ons, tools and utilities
Database Drivers
Registration-required add-ons
Replication Solutions
Spatial Extensions
Web Applications
Web Development
< Back
Next >
Cancel

Windows/OSX

- use the graphical installer
  - from EnterpriseDB
  - “wizard” GUI
- also installs optional components
  - pgAdmin
  - Some extensions

other platforms

- Packages available:
  - Solaris 10/11
  - OpenSolaris/ Illumos
  - FreeBSD
  - OpenBSD
  - NetBSD
- No Packages:
  - HP-UX
  - AIX
  - Solaris 9
  - Tablets
  - “Home” Windows

get logged in

```
vagrant ssh
sudo su -
tmux
```

packages exercise

```
less
/etc/apt/sources.list.d/pgdg.list

apt-cache search postgresql

apt-cache search postgis
```

create data directory

- $PGDATA is where the database files live
- most packages create it
- if not, use “initdb” to create it
  - pick a suitable location!

```
initdb -D /db/9.3/main
```

initdb exercise

```
su - postgres
mkdir test
initdb -D test
cd test
ls -l
exit
```

starting & stopping

3 commands

service / initdb scripts

- normal use, recommended

pg_ctl

- for custom configurations
- must be run as “postgres”

postgres

- only for control script-writers


kill

4 states

1. start
2. stop
3. restart
4. reload

start & stop

- what it says on the tin
- except …
  - “smart” stop: wait for all connections
  - “fast” stop: force disconnect
  - “immediate” stop: just like “kill”

restart

- shuts down and restarts postgres
- breaks all connections
- required for:
  - changes to memory, connections
  - changes to archive_mode
  - changes to logging_collector

reload

- signals the postmaster to reload files
- does not break connections
- works for
  - changes to security (pg_hba.conf)
  - most changes to logging
  - changes to defaults

start/stop exercise

```
service postgresql start
service postgresql restart
service postgresql reload
service postgresql stop
```

start/stop exercise

```
su - postgres

pg_ctl -D /etc/postgresql/9.3/main
start

pg_ctl -D /etc/postgresql/9.3/main
           -m fast stop

exit
```

start/stop exercise

```
cd /etc/postgresql/9.3/main/
EDITOR postgresql.conf
service postgresql start
less
/var/log/postgresql/postgresq
l-{DOW}.log
```

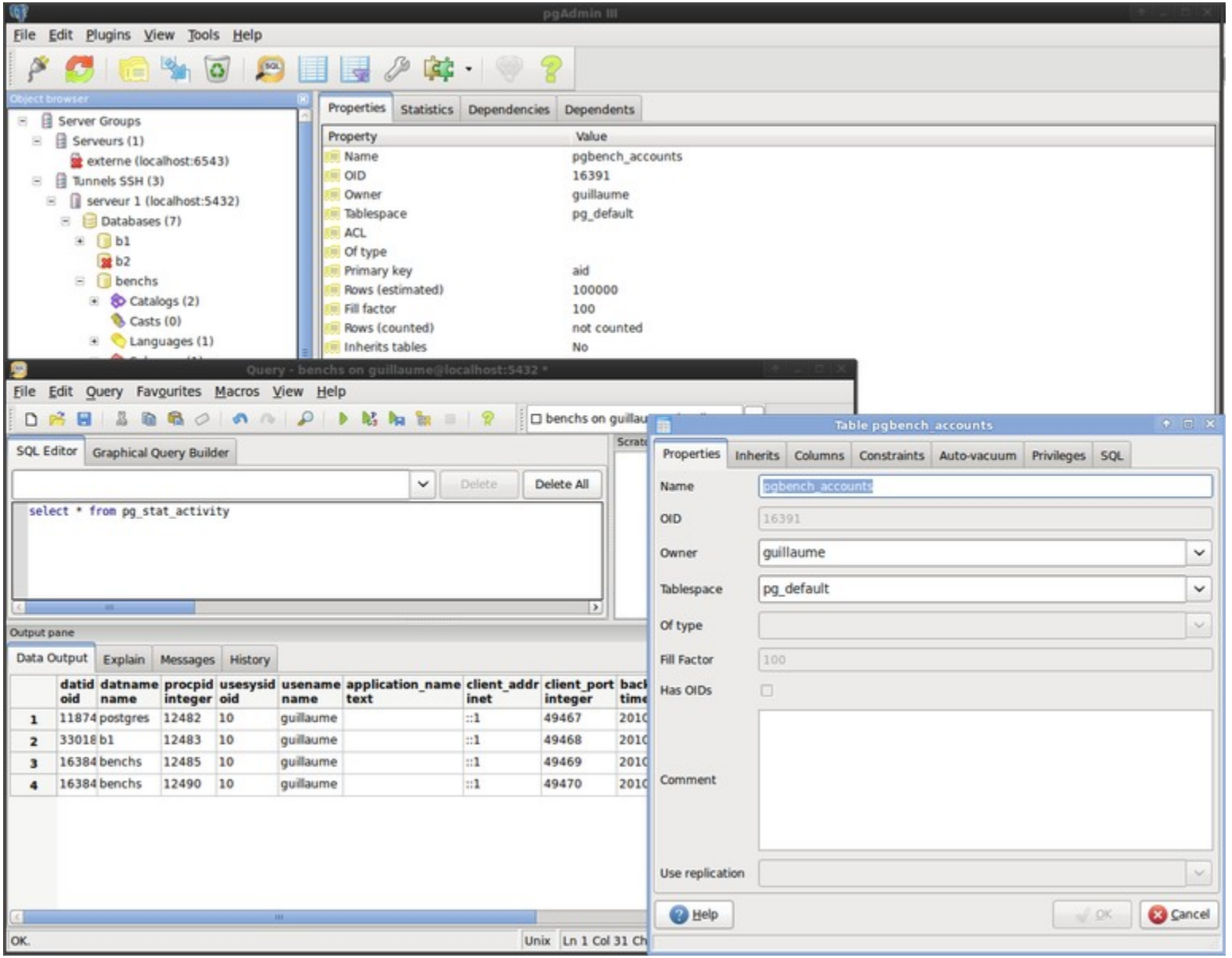
start/stop exercise

```
EDITOR postgresql.conf

service postgresql start

less
/var/log/postgresql/postgresq
l-{DOW}.log
```

```
Sidney-Stratton:~ josh$ psql libdata
psql (9.1.1)
Type "help" for help.

libdata=# \dt
             List of relations
 Schema |      Name      | Type  |  Owner
--------+----------------+-------+---------
 public | books          | table | libdata
 public | branches       | table | libdata
 public | copies         | table | libdata
 public | copy_history   | table | libdata
 public | copy_status    | table | libdata
```

the psql command line

pgAdmin


pgAdmin III
File Edit Plugins View Tools Help
Object browser
Server Groups
Serveurs (1)
externe (localhost:6543)
Tunnels SSH (3)
serveur 1 (localhost:5432)
Databases (7)
b1
b2
benchs
Catalogs (2)
Casts (0)
Languages (1)
Properties Statistics Dependencies Dependents
Property Value
Name pgbench_accounts
OID 16391
Owner guillaume
Tablespace pg_default
ACL
Of type
Primary key aid
Rows (estimated) 100000
Fill factor 100
Rows (counted) not counted
Inherits tables No
Query - benchs on guillaume@localhost:5432 *
File Edit Query Favourites Macros View Help
SQL Editor Graphical Query Builder
Delete Delete All
select * from pg_stat_activity
Output pane
Data Output Explain Messages History
datid oid datname name procpid integer usesysid oid usename name application_name text client_addr inet client_port integer
1 11874 postgres 12482 10 guillaume ::1 49467
2 33018 b1 12483 10 guillaume ::1 49468
3 16384 benchs 12485 10 guillaume ::1 49469
4 16384 benchs 12490 10 guillaume ::1 49470
OK.
Unix Ln 1 Col 31
Table pgbench_accounts
Properties Inherits Columns Constraints Auto-vacuum Privileges SQL
Name pgbench_accounts
OID 16391
Owner guillaume
Tablespace pg_default
Of type
Fill Factor 100
Has OIDs
Comment
Use replication
Help OK Cancel

psql exercise

```
su - postgres
psql libdata
\?
\h create table
\dt
\d+ copies
```

extensions

extensions?

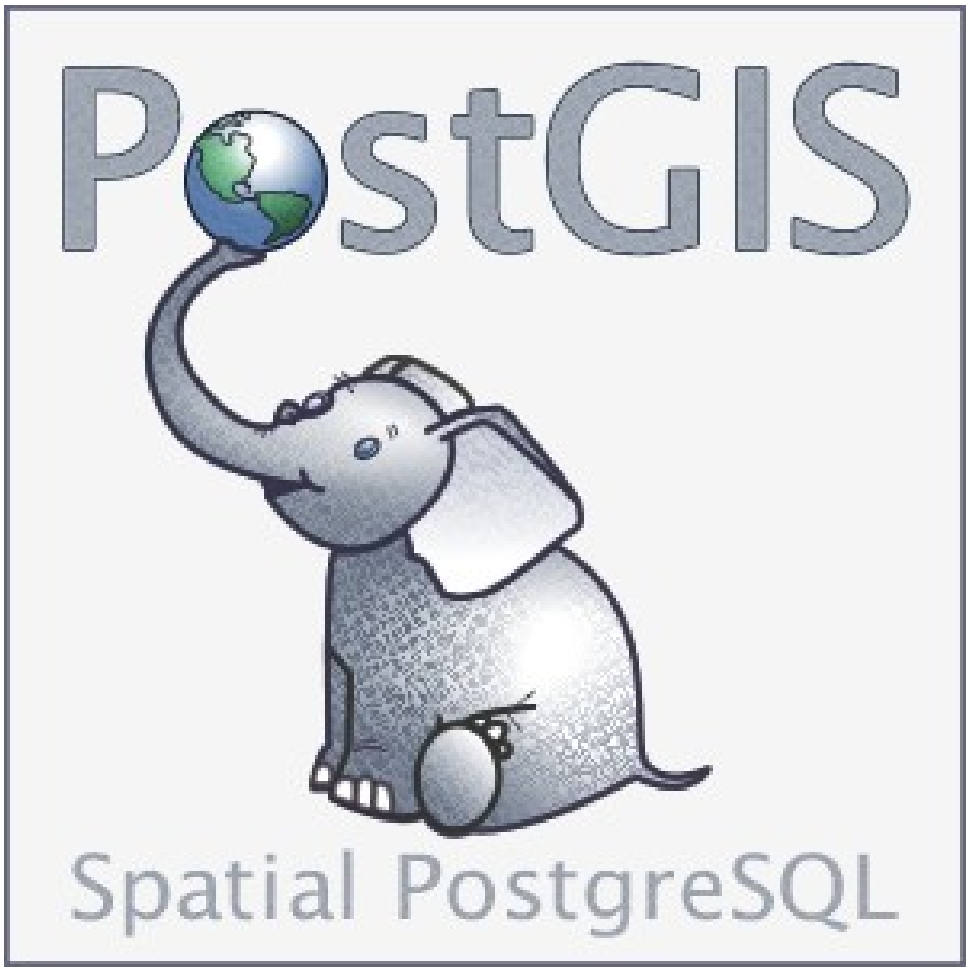
- extensions add extra functionality
  - like Python/Perl modules, Ruby Gems, etc.
- need to be installed separately
  - some come with PostgreSQL packages
  - some need to be installed from PGXN or source
- handled very differently before 9.1

libdata extensions

Extension	Purpose	Source
plpgsql	*Procedural language for triggers and functions. Automatically included 9.0 and later.*	core
citext	*Case-insensitive text data type*	contrib
isn	*Product scan code data type. Used in libdata for ISBNs*	contrib

installing extensions

1. install the binary files
   - using packages, PGXN, or source
   - installs to postgres “share” directory
   - a few extensions don't have binaries
2. install the extension in each database where it's used


PostGIS
Spatial PostgreSQL

Installing PostGIS

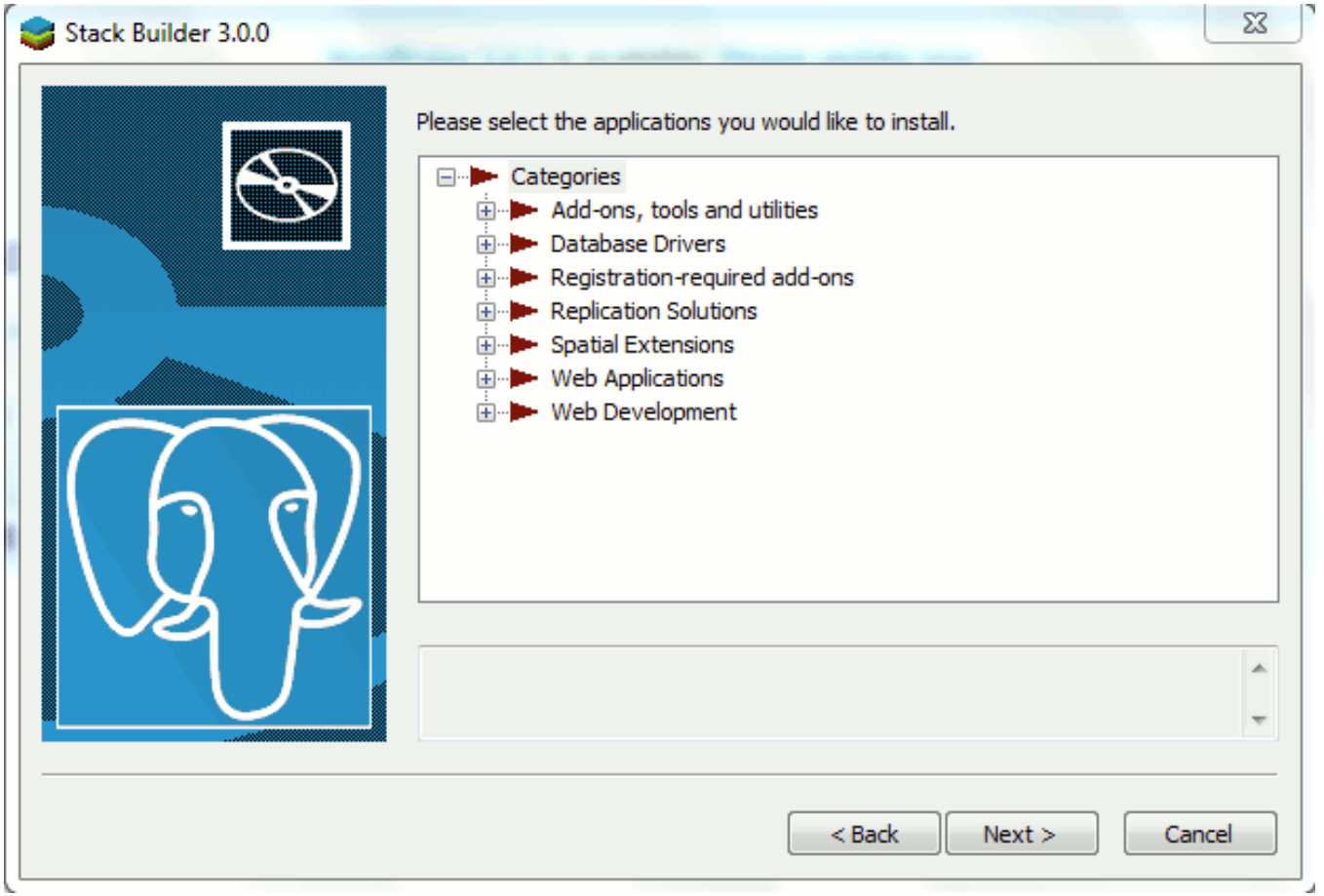
1. Install PostGIS packages / binaries
2. Install the PostGIS extension into each database
3. Load databases
4. Load geographic data

Packages for PostGIS

- Most updated Postgres:
  - apt.postgresql.org / yum.postgresql.org
- Best lib/tool support:
  - OSGeo / ubuntugis / etc.
- Too old:
  - OS packages
- Too hard:
  - source code compile

Mac/Windows

- EnterpriseDB installer



extensions exercise

```
\dx
create extension hstore;
select * from pg_extension;
select 'fname => josh,
  lname => berkus'::hstore;
```

extensions exercise

```
\c earth

\dx

alter extension postgis
update;
```

updates
&
upgrades

APRIL 05, 2013

PostgreSQL updates address high-risk vulnerability, other issues

VMware also releases fixes for its PostgreSQL-based vFabric Postgres database product

By Lucian Constantin | IDG News Service

The PostgreSQL developers released updates for all major branches of the popular open-source database system on Thursday in order to address several vulnerabilities, including a high-risk one that could allow attackers to crash the server, modify

major vs. minor

9.3 == a major version

- requires an *upgrade* from 9.2.8
- contains features not in 9.2
- requires testing and planned downtime

9.3.5 == a minor version

- is a minor *update* from 9.3.4.
- can (and should) be applied immediately

minor updates

- come out ~~ every 2 months
- contain only bugfixes
  - security hole patches
  - data loss prevention
  - fix server crashes
- no new or changed features
  - *occasional* documented breakage

update promptly
update often

update procedure

1. schedule 5 minute downtime
2. download packages
3. shut down postgresql
4. install packages
5. restart postgresql
6. restart application

major upgrades

- come out once per year
- have many new features
  - and sometimes break stuff which used to work
- require extensive testing with your application
- require significant downtime to upgrade

upgrade procedures

- dump & reload
  - use pg_dump & pg_restore on database
  - most reliable way
  - “cleans up” database in process
  - best with small databases
  - can take a long, long time

upgrade procedures

- pg_upgrade
  - upgrade “in place”
  - much faster
  - does not “clean up” database
  - sometimes doesn't work
    - issues with extensions

EOL after 5 years

upgrading extensions

not possible before 9.1

1. upgrade Postgres (or not)
2. install new extension binaries
3. run upgrade script in each DB

```
ALTER EXTENSION postgis
UPDATE;
```

configuration

configuration

1. Hardware
2. OS/FS
3. PostgreSQL

use good hardware

- databases use *all* the hardware
  - RAM, CPU, IO
  - disk can be very important
    - DB larger than RAM
    - write-heavy database
  - PostGIS requires lots of RAM + CPU

the database cannot outperform bad hardware

put the database on its own server (or virtual server)

cloud servers

- cloud server performance sucks
  - especially IO
- AWS tip: make sure you have enough RAM to cache the *whole* database

Linux configuration

1. turn the OOM killer off
2. turn reclaim_zone_files off
3. use XFS or Ext4 for database files
    1. use “noatime,nodiratime”
4. increase shmmax, shmall
    1. so that you can raise shared_buffers
    2. *only required before 9.3*

tmux new window

```
<ctrl>b,c
<ctrl>b,n
<ctrl>b,n
```

kernel params

```
less
/etc/sysctl.d/30-postgresql-shm.conf
```

- Q: what would it take for 1GB?

not required for 9.3 and later

BSD/Solaris Config.

- Use ZFS
  - decrease block size to 8K
- increase shmmax/shmmall on BSD
- may need to mess with ulimits
  - on very busy systems
  - Postgres will give you errors

Windows/OSX

optimization not possible

the xlog

- xlog == WAL
  - where transactions are recorded
- best on its own drive/volume
  - write-only
  - writes synchronously
  - response time paramount
- create volume, then link pg_xlog dir

xlog dir

```
<ctrl>b,n
cd 9.3/main/pg_xlog
ls -lh
```

postgresql.conf

parameters you care about:	parameters you don't care about:
10 to 25	206 to 207

postgresql.conf

```
cd
/etc/postgresql/9.3/main
$EDITOR postgresql.conf
```

- editors availble:
  - joe, jmacs, vi, nano

postgresql.conf

```
psql
show max_connections;
show all;
\x
select * from
pg_settings;
```

connections & security

network

- local connections: unix sockets
  - faster than TCP/IP
- other servers: port 5432
  - make sure it's open on the firewall!
- on the cloud? use SSL
  - secure your connections
  - PITA to set up, though

max_connections

"ERROR: connection limit exceeded for non-superusers"

- postgresql.conf
- increase number of connections
- good up to about 20 + 10 x cores
- keep needing to increase it? something is wrong

security

```
"FATAL:  password authentication
failed for user "wwwuser""
```

- Postgres users & passwords
  - CREATE/ALTER USER
  - “group” ROLEs
- Or: use LDAP, GSSAPI or PAM

create user

```
psql
create user 'bench'
password 'benchmark';
\du
\q
```

host-based access

```
“FATAL: no pg_hba.conf entry for host
"192.168.0.1", user "chaos", database
"chaosLRdb", SSL off”
```

- pg_hba.conf
- access control list:
  - database/user/host address
  - like iptables for Postgres
- change config and reload

pg_hba.conf

```
<ctrl>b,n
$EDITOR /etc/postgresql
  /9.3/main/pg_hba.conf
service postgresql
reload
```

pg_hba.conf

```
<ctrl>b,n
psql
\q
psql -U bench
\q
```

.pgpass

```
cd ~
cp /setup/postgres/.pgpass .
chmod 700 .pgpass
less .pgpass
psql -U bench
```

three tips for security

1. don't expose the postgres server/port to the internet
2. don't allow users to connect as the superuser (postgres)
3. use the strongest authentication which is practical

connection pooling

- pgbouncer
  - event-based pooler
  - separate package
  - on DB server, or
  - app server, or
  - 3rd “bouncer” server

pgbench

- simple “benchmark” utility
  - based on “Wiconsin” benchmark
  - ships in postgresql-contrib
  - useful for testing for really bad OS/hardware issues
  - also for demos

setting up pgbench

```
createdb bench
pgbench -U bench -i -s 10
```

pgbouncer

```
<ctrl>b,n
less /etc/pgbouncer/pgbouncer.ini
less /etc/pgbouncer/userlist.txt
service pgbouncer start
<ctrl>b,n
psql -U bench -p 6432
\q
```

pgbouncer

```
cd /setup/pgbench
less runbench_pool.sh
./runbench_pool.sh
<ctrl>b,n
su - postgres
psql
```

pgbouncer

```
select count(*) from
pg_stat_activity;
\q
<ctrl>b,n
<ctrl>c
```

backups

```
2012-01-27 18:00:44 MSK FATAL:
    invalid page header in block
    311757 of relation
    base/26976/27977
2012-01-27 18:00:44 MSK CONTEXT:
    xlog redo insert:
    rel 1663/26976/27977;
    tid 311757/44
2012-01-27 18:00:44 MSK LOG:
    startup process (PID 392)
    exited with exit code 1
2012-01-27 18:00:44 MSK LOG:
    aborting startup due
    to startup process failure
```


Our Disaster Recovery Plan Goes Something Like This...
HELP!
HELP!
DILBERT
By Scott Adams

three methods

A) pg_dump
B) PITR
C) filesystem snapshot

pg_dump

- “logical” backup
  - portable
  - compressed
  - works for upgrades
- good for small databases
- use -Fc
  - custom binary format

pg_restore

- used to reload pg_dumps
  - parallel mode for fast loading
  - extract specific tables, schema
  - extract to an alternate database

pgdump

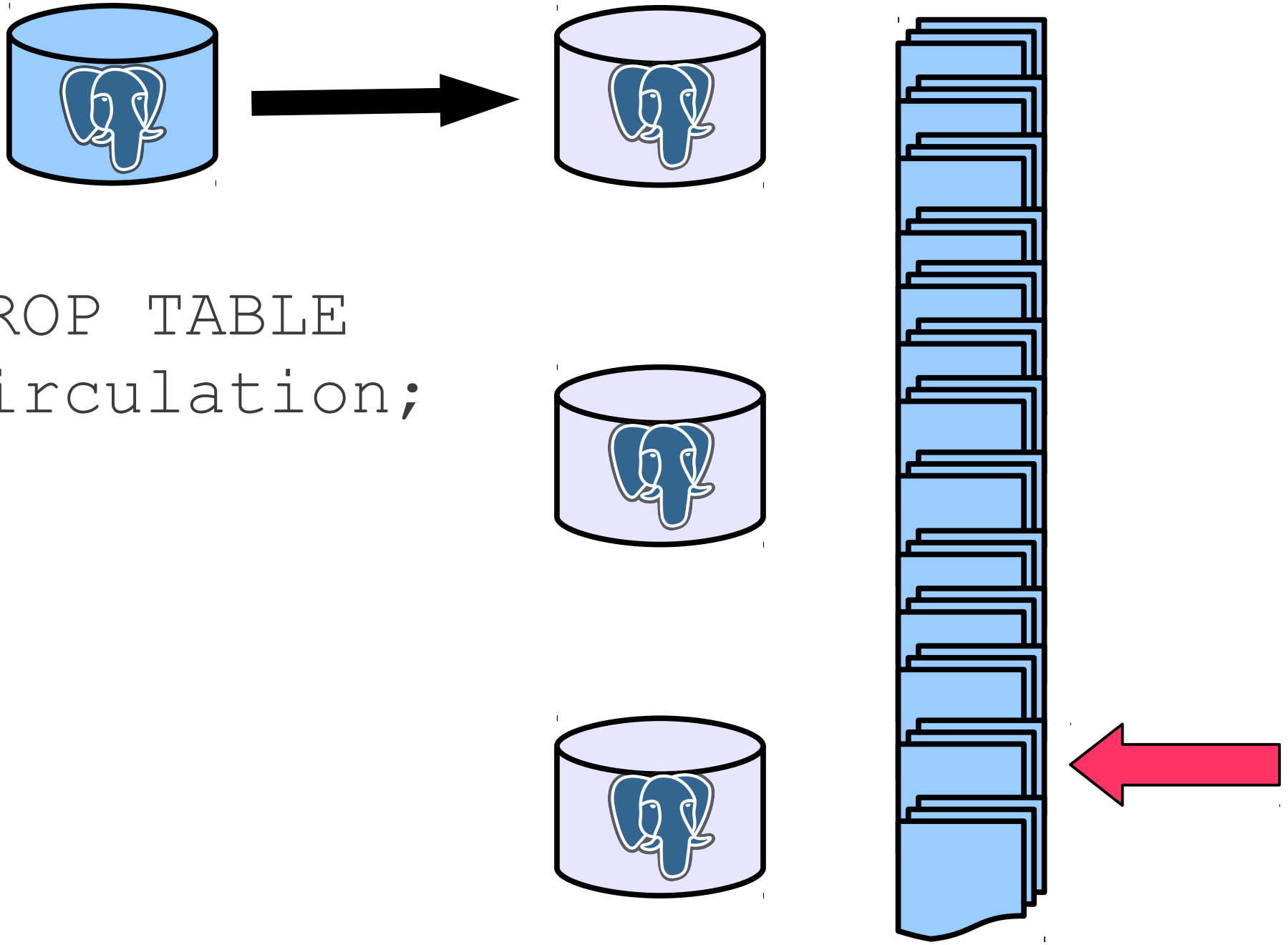
```
cd ~
pg_dump -Fc -v
   -f backup/libdata.dump
   libdata
ls -lh backup/libdata.dump
```

pg_restore

```
pg_restore -l backup/libdata.dump

createdb libdata2

pg_restore -v -d libdata2
    backup/libdata.dump

psql libdata2

\dt

\q
```

PITR

- “Point-In-Time Recovery”
- “binary” and “continuous” backup
  - take snapshot of DB files
  - accumulate logfile copies
- good for large databases
- can combine with replication


DROP TABLE
circulation;

PITR - PITA

- can be difficult to set up & monitor
- use tools:
  - Barman
  - OmniPITR
  - WAL-E (for AWS)

PITR

```
<ctrl>b,n
exit
$EDITOR /etc/postgresql/9.3/main
   /postgresql.conf
$EDITOR /etc/postgresql/9.3/main
   /pg_hba.conf
service postgresql restart
less /setup/postgres/archive_logs.sh
```

PITR

```
<ctrl>b,n
cd ~
pg_basebackup -x -P -D 9.3/replica
cd wal_archive
ls -lh
```

filesystem snapshot

- works with snapshotting filesystems
  - ZFS, BTRFS
  - LVM
  - Some SANs
- must be real snapshot
  - coherent
  - point-in-time

have a DR plan

- ways you can lose data
  - recovery time
  - acceptable data loss
- how to recover from lost data
  - detailed steps
  - verification

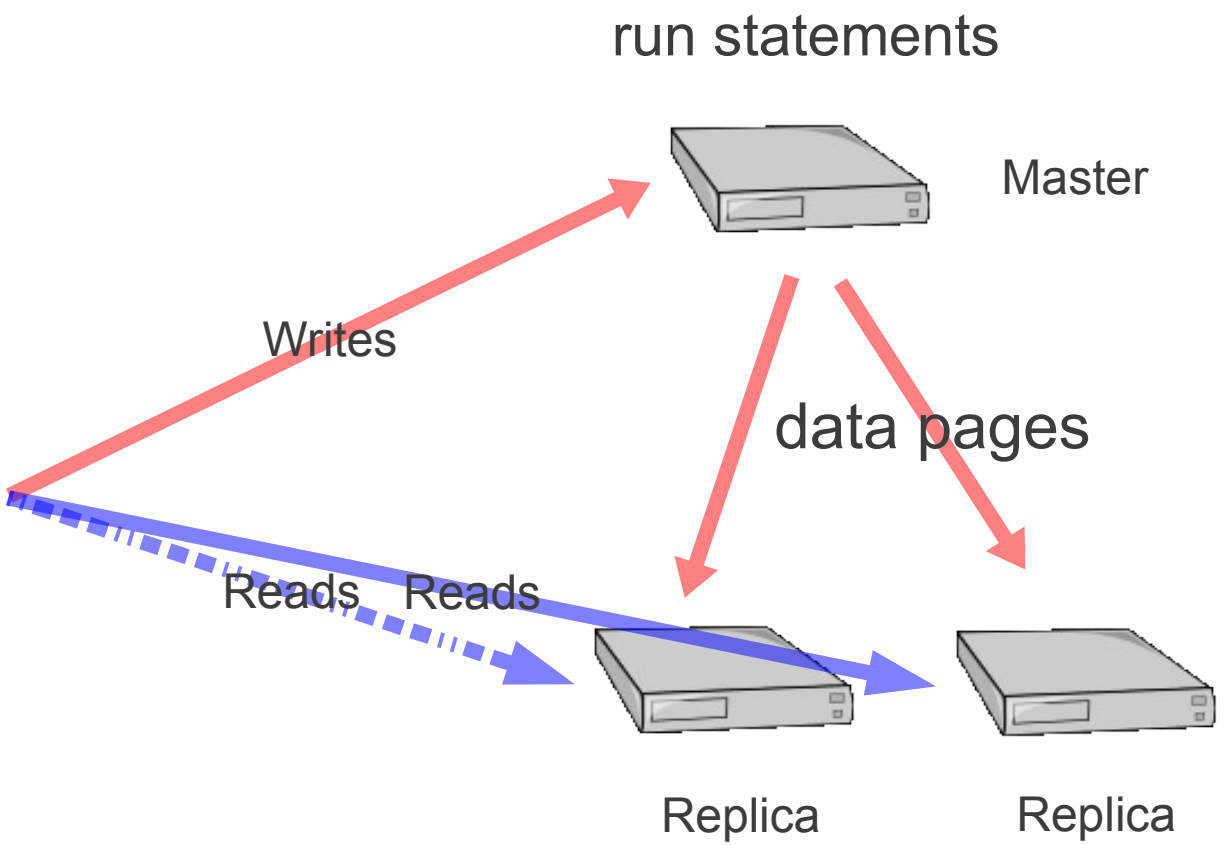
replication

Not Appearing in this Version

use replication for ...

- **availability:** have an “always on” failover server
- **load-balancing:** offload traffic
  - especially reporting workloads
- **security:** provide users read-only access

binary replication



DRBD for PostgreSQL

(only much much faster)

also called ...

- **streaming replication**
  - refers to the ability to stream new data pages over a network connection
- **hot standby**
  - refers to the ability of standbys to run read-only queries while in standby mode

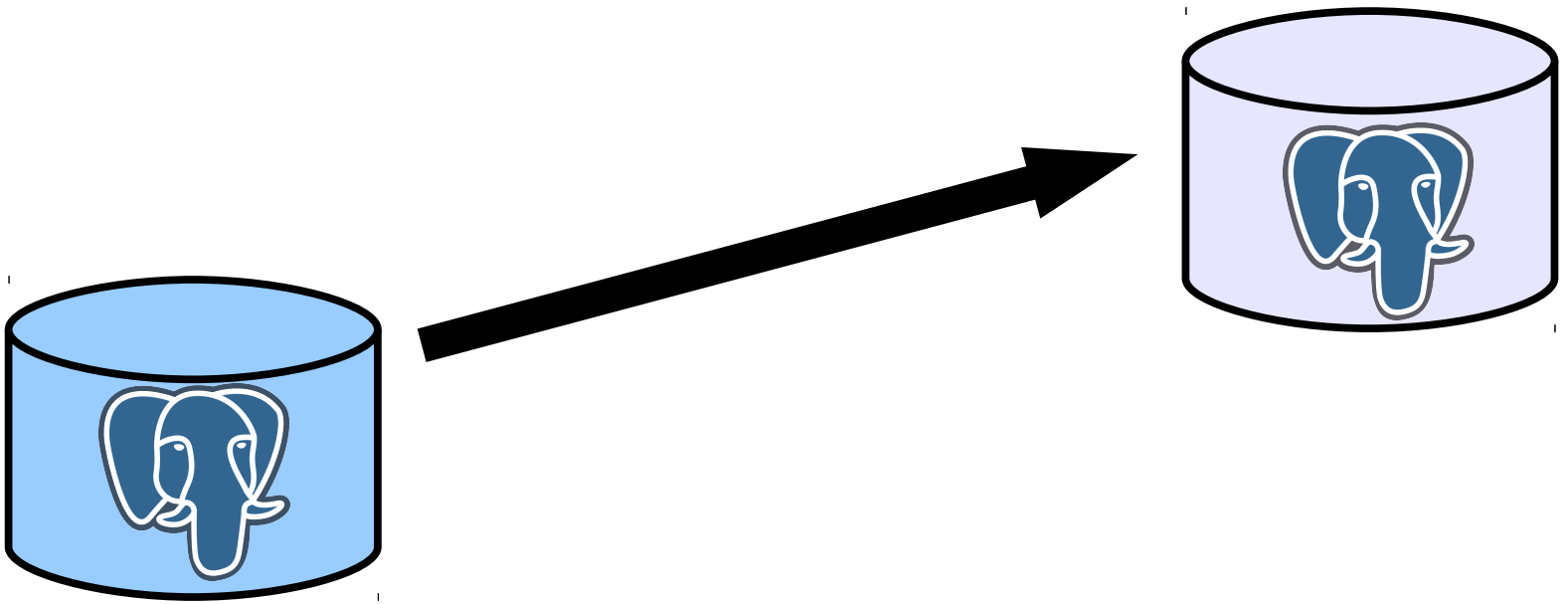
advantages

- low administration
- low overhead on master
  - not much incremental cost
- non-invasive
  - no extra tables/triggers
  - no primary key requirements
  - no limitations on statements

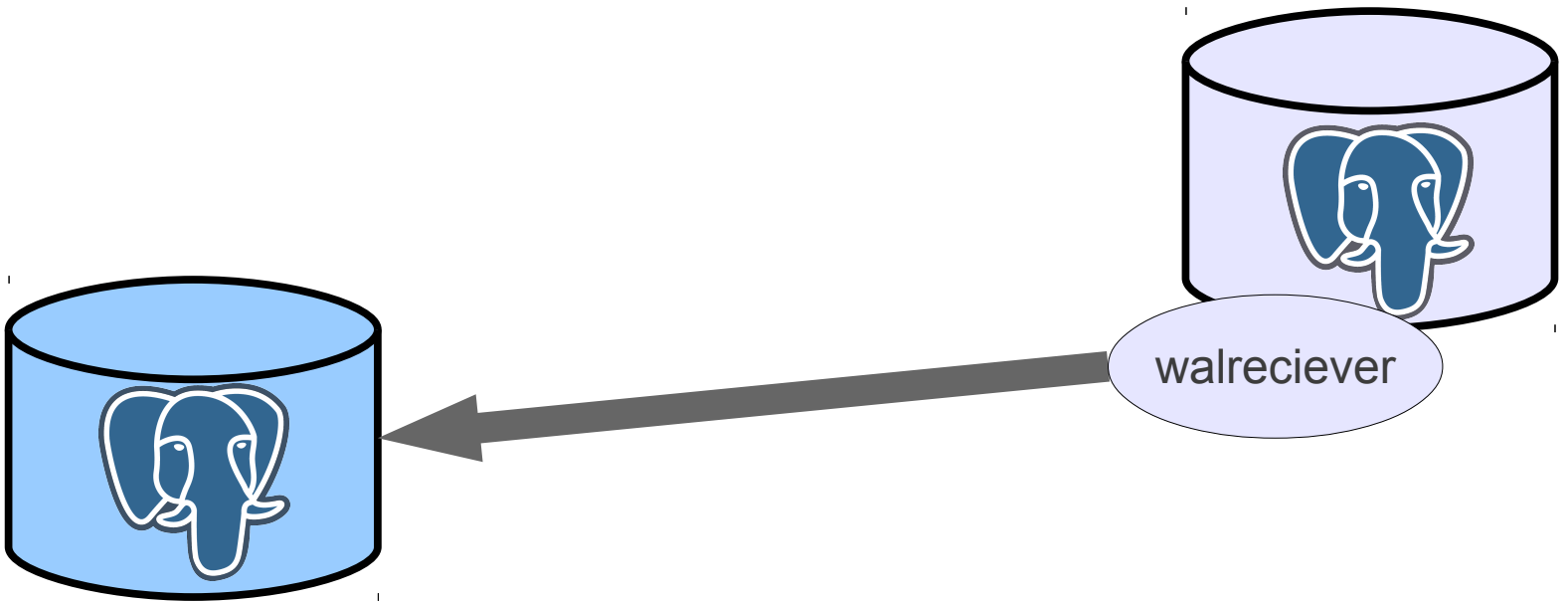
disadvantages

- need to replicate the whole server
  - not individual databases or tables
- no writes of any kind on replicas
- some things not replicated
- query cancel (described later)

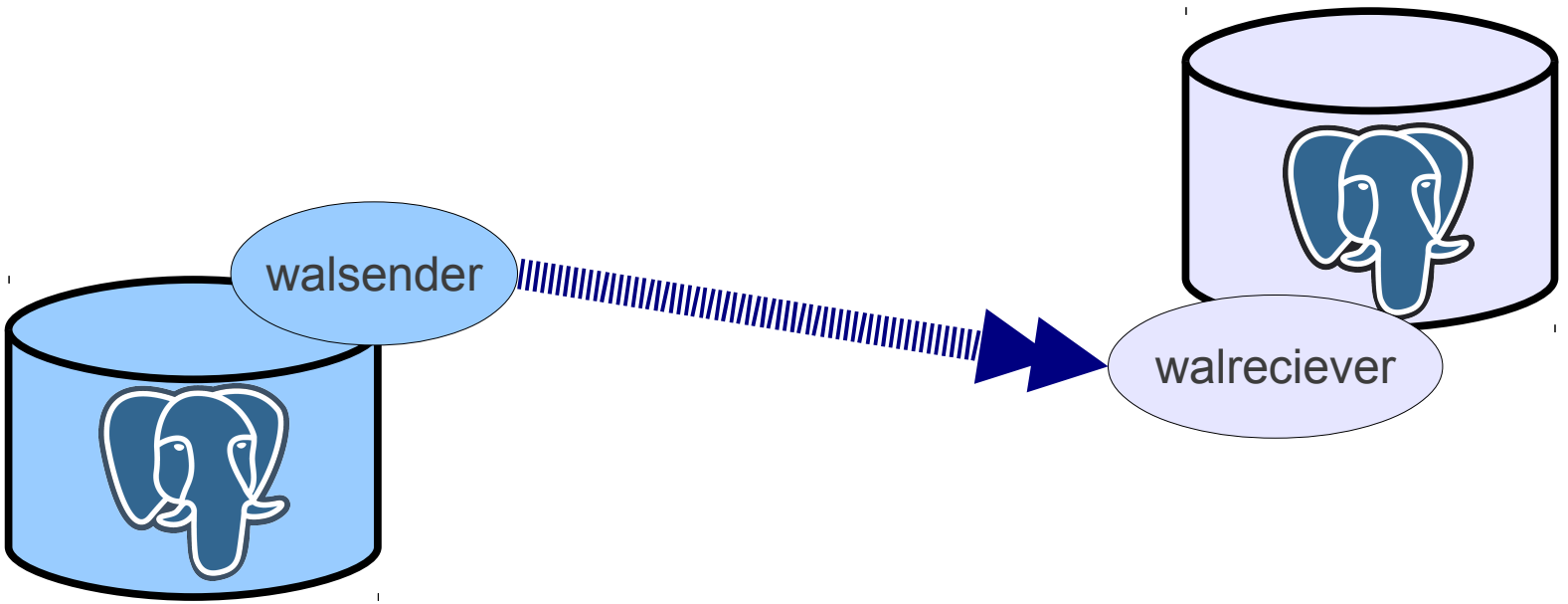
how it works



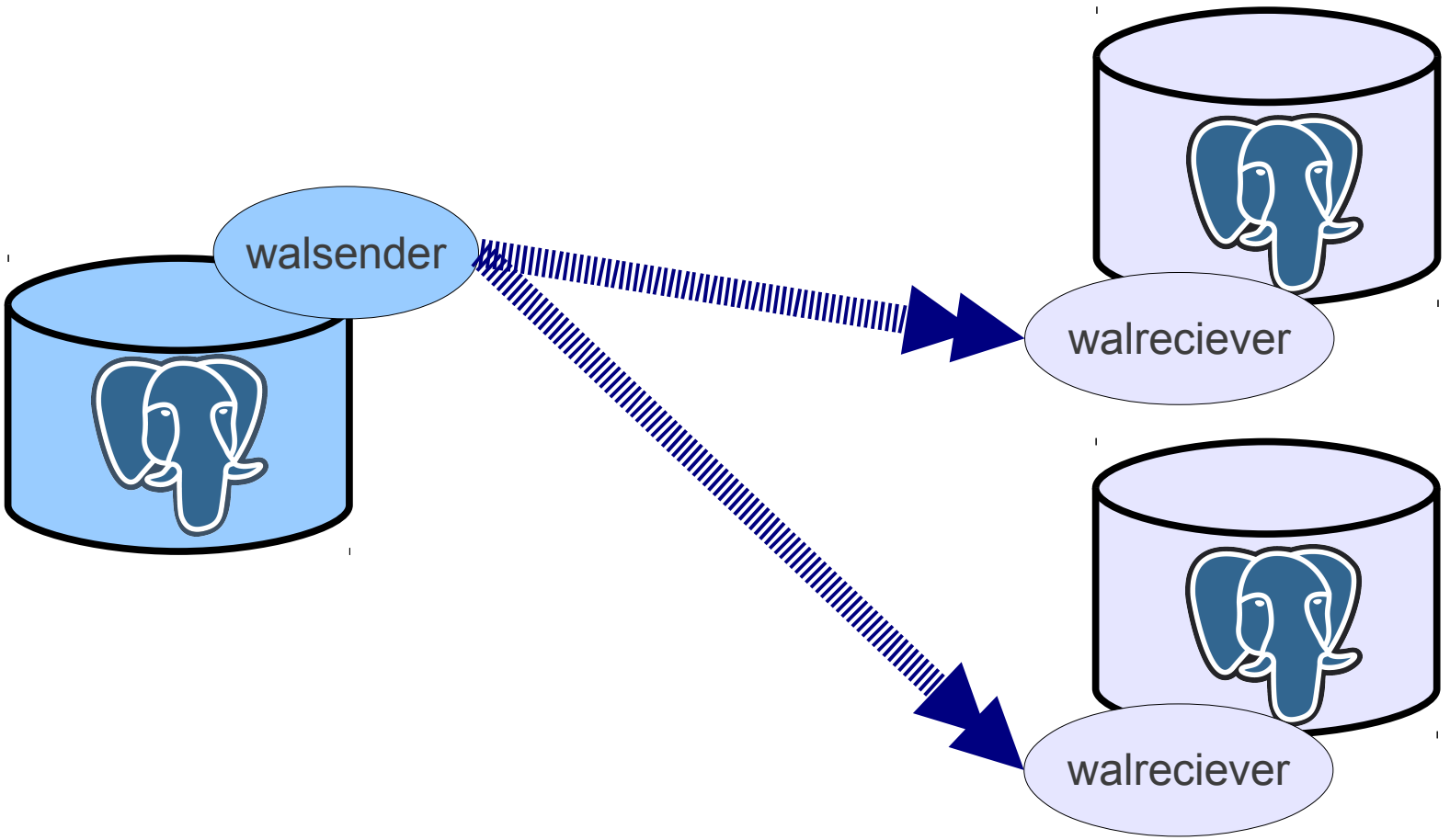
how it works



how it works



how it works



“recovery”

- historical term
  - because replication grew out of PITR
- used to refer to the replica being “in recovery mode”, i.e no write queries
- all over parameter and file names
  - recovery.conf
  - recovery_target
  - pg_is_in_recovery()

replication

```
<ctrl>b,n
cd ~/9.3/replica
cp -p -r
/setup/postgres/archive/* .
less recovery.conf
pg_ctl -D . start
```

replication

```
psql -p 5433
select pg_is_in_recovery();
\q
psql -p 5432
select * from
     pg_stat_replication;
\q
```

failover

```
pg_ctl -D . promote
psql -p 5433
select * from pg_is_in_recovery();
\q
pg_ctl -D . stop
```

other cloning methods

- rsync
- filesystem snapshot
  - (on shared storage)
- VM cloning
  - (if rapid)

creating a snapshot

1. make sure you are replicating transaction logs to the replica
2. pg_start_backup('replication')
3. rsync / snapshot / clone
4. pg_stop_backup()

failover vs. load-balancing

load causes replicas to fall behind

your replica can work for:

- rapid failover
- offloading work

but not both!

other replication

- synchronous replication
- Slony-I
- Londiste
- Bucardo

monitoring

use your favorite tool

ganglia, collectd, Hyperic, OpenNMS, OpenView, whatever ....

- nagios check_postgres.pl
  - broad list of checks
  - mine it for queries and techniques

many useful checks

- disk space
- caching RAM
- response time
- connections
- idle transacts
- table growth
- waiting queries
- long queries
- database size
- table bloat
- system load
- replication lag
- XID wraparound
- execution time

OS checks

- disk space (per volume!)
- system load
- memory usage
- IO activity
- network activity

database checks

- connections: active, idle, idle xtns
- blocked queries (number, time)
- query times (pg_stat_statements)
- table size & growth
- table & index bloat
- XID wraparound
- replication lag

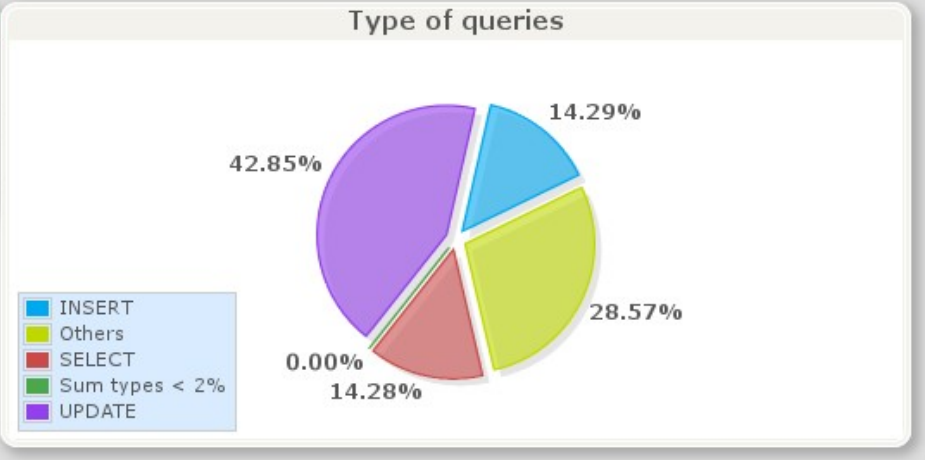
activity log

- connections & disconnections
- slow queries
- DB swap usage
- schema changes
- lock waits & deadlocks

pgbadger

Queries by type ^

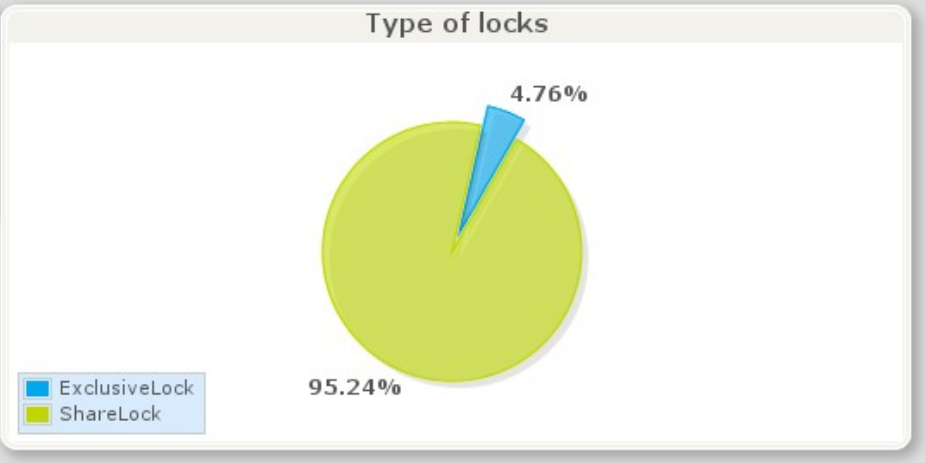
Type	Count	Percentage
SELECT	48,568	14.28%
INSERT	48,578	14.29%
UPDATE	145,701	42.85%
DELETE	0	0.00%
OTHERS	97,154	28.57%



To Image Download Reset

Locks by type ^

Type	Object	Count	Total Duration	Av. duration (s)
ExclusiveLock		1	1.14s	1.14s
	tuple	1	1.14s	1.14s
ShareLock		20	25.46s	1.27s
	transaction	20	25.46s	1.27s
Total		21	26.60s	1.27s



DB activity

```
<ctrl>b,n
$EDITOR /etc/postgresql/9.3/main
   /postgresql.conf
service postgresql reload
<ctrl>b,n
cd /setup/pgbench
runbench_log.sh
```

DB activity

```
<ctrl>b,n
su - postgres
psql
select * from pg_stat_activity;
\x
select * from pg_stat_activty;
```

DB activity

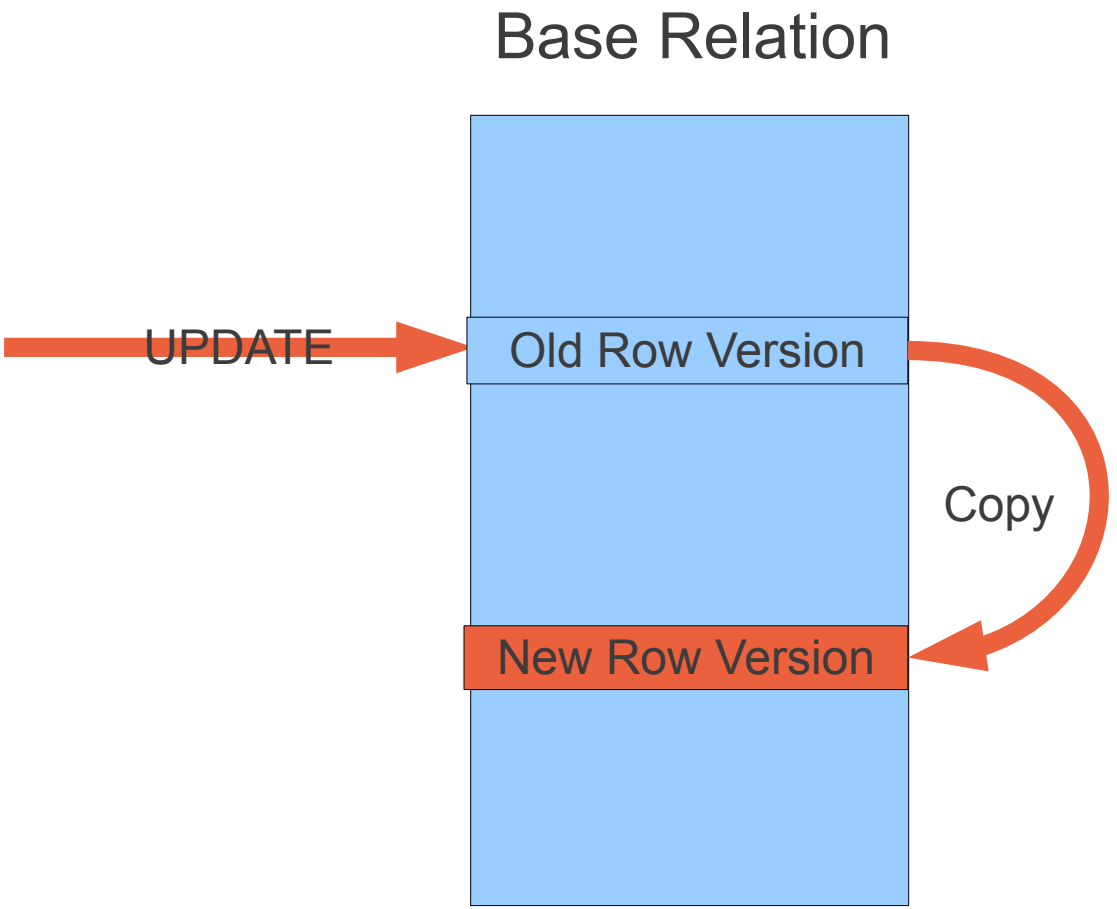
```
\c bench
select * from pg_stat_user_tables;
select pg_size_pretty(
  pg_total_relation_size(
    'pgbench_history'));
select * from pg_stat{TAB}
```

DB logs

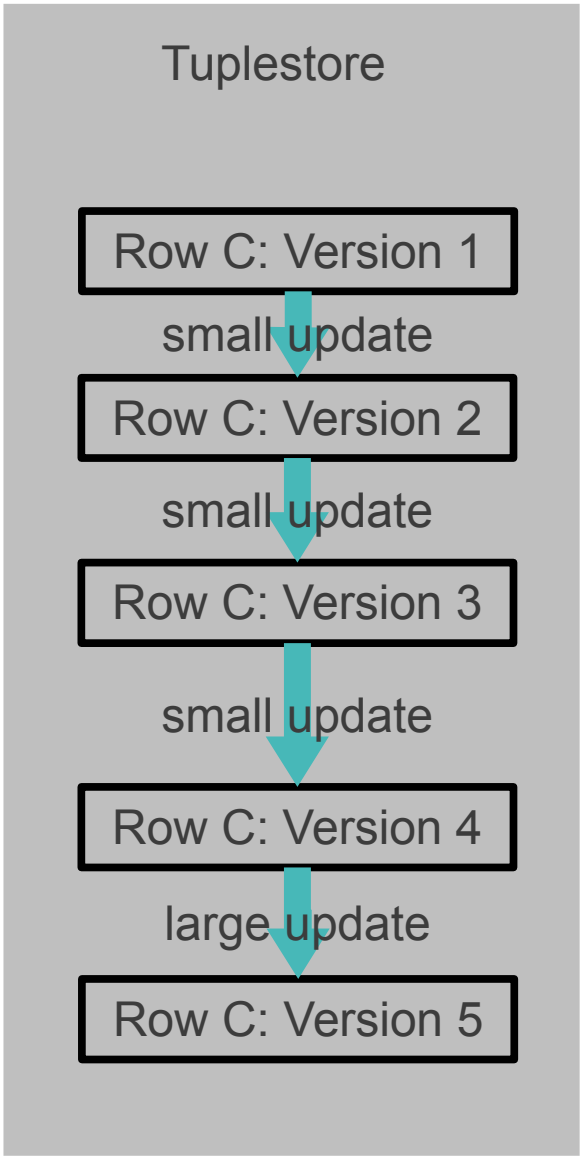
```
exit

cd /var/log/postgresql

less activitylog-Mon.csv

pgbadger --format csv --out
   /setup/postgres/badger.html
   activitylog-Mon.csv
```

VACUUM & ANALYZE

non-overwriting



garbage collection

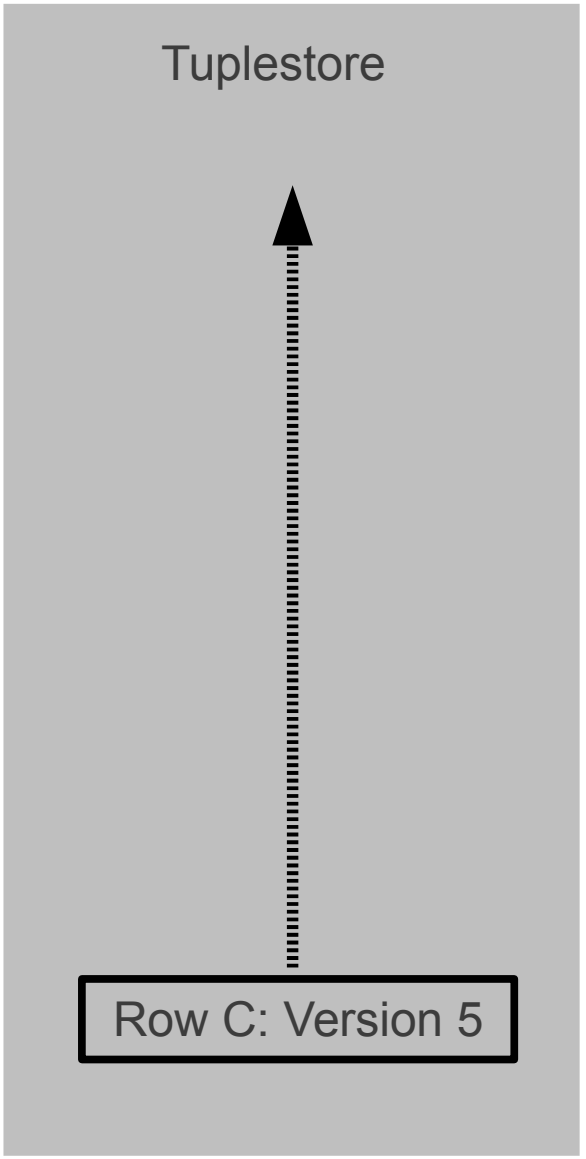


garbage collection

Tuplestore

Row C: Version 5

garbage collection



garbage collection

Tuplestore

Row C: Version 5

garbage collection

Tuplestore

Row C: Version 5

Autovacuum

falling behind

bloat check

- pg_stat_user_tables: dead_rows/live_rows
- nagios bloat query
  - https://gist.github.com/jberkus/9923948
- trend table/database size
- scans very slow to return the first row

fixing bloat

- manual VACUUMs
  - even VACUUM FULL
- pg_repack
- tune autovacuum
  - increase frequency for busy tables
  - increase workload size (vacuum_cost_limit)
  - more autovacuum workers

vacuum

```
<ctrl>b,n
psql bench
\x
select * from pg_stat_user_tables;
select pg_total_relation_size(
   'pgbench_tellers');
```

vacuum

```
vacuum analyze pgbench_tellers;

select pg_total_relation_size(
   'pgbench_tellers');

vacuum full pgbench_tellers;

select pg_total_relation_size(
   'pgbench_tellers');
```

XID wraparound

preempt wraparound

- track age(datfrozenxid) for each table
  - nagios has probe for this
- VACUUM FREEZE tables preemptively
  - link_to_flexible_freeze_here
- lower vacuum_freeze_min_age
  - maybe to 250,000

xid freezing

```
select relname,
  age(relfrozenxid) as xid_age
from pg_class JOIN
pg_stat_user_tables USING (relname)
order by xid_age desc;

VACUUM FREEZE;

select relname ...
```

ANALYZE

- keeps DB statistics up to date
- non-intrusive
  - uses sampling
  - low IO
- should be run frequently (hourly?)
  - autovacuum takes care of this
- when not run, query plans get really bad

ANALYZE

- manual ANALYZE when:
  - you create & populate a brand new table
  - you do a massive DELETE or TRUNCATE
  - you're seeing bad query plans due to bad stats
- ANALYZE individual tables

stats

```
\a
select * from pg_stats where
schemaname = 'public';
```

managing queries

pg_stat_activity

```
-[ RECORD 2 ]----+------------------------------
datid            | 16422
datname          | libdata
procpid          | 46295
usesysid         | 10
username         | dataentry
application_name | psql
client_addr      | 192.168.101.114
client_port      | 5432
backend_start    | 2012-08-26 15:09:05.233-07
xact_start       | 2012-08-26 15:09:06.113-07
query_start      | 2012-08-26 15:11:53.521-07
waiting          | f
current_query    | <IDLE> in transaction
```

locks

- write queries can block on other write queries
  - as can table schema changes
  - queries can wait forever on locks
- look for old “idle in transaction”
  - that's a zombie database connection

kill it kill it kill it

killing zombies

- pg_cancel_backend(pid)
  - kills running queries with sigINT
  - like CTRL-C
- pg_terminate_backend(pid)
  - kills bad connections, idle transactions
  - can cause DB to reload in some cases

zombie killing

```
cd /setup/pgbench
runbench_locks.sh
<ctrl>b,n
su - postgres
psql bench
select * from pg_stat_activity;
```

zombie killing

```
select pg_cancel_backend(pid)
from pg_stat_activity
where state = 'idle in transaction'
and state_change <
( now() - interval '10 seconds');


select * from pg_stat_activity;
```

EXPLAIN

```
Nested Loop  (cost=792.00..828.08 rows=1422317 width=99)
  ->  HashAggregate  (cost=792.00..792.00 rows=1 width=4)
        ->  Index Scan using
index_player_summaries_on_player_id on player_summaries
ps  (cost=0.00..791.80 rows=403 width=4)
              Index Cond: (player_id = 21432312)
  ->  Index Scan using index_player_summaries_on_match_id
on player_summaries  (cost=0.00..33.98 rows=600 width=99)
        Index Cond: (match_id = ps.match_id)
```

EXPLAIN ANALYZE

```
Nested Loop  (cost=792.00..828.08 rows=1422317
width=99) (actual time=9928.869..20753.723
rows=13470 loops=1)
  ->  HashAggregate  (cost=792.00..792.00 rows=1 width=4)
(actual time=9895.105..9897.096 rows=1347 loops=1)
        ->  Index Scan using
index_player_summaries_on_player_id on player_summaries
ps  (cost=0.00..791.80 rows=403 width=4) (actual
time=27.413..9890.887 rows=1347 loops=1)
              Index Cond: (player_id = 21432312)
  ->  Index Scan using index_player_summaries_on_match_id
on player_summaries  (cost=0.00..33.98 rows=600 width=99)
(actual time=7.375..8.037 rows=10 loops=1347)
        Index Cond: (match_id = ps.match_id)
Total runtime: 20764.371 ms"
```

explain.depesz.com

HTML | TEXT | STATS

exclusive	inclusive	rows x	rows	loops	node
30.788	20753.723	↑ 105.6	13470	1	→ Nested Loop (cost=792.00..828.08 rows=1422317 width=99) (actual time=9928.869..20753.723 rows=13470 loops=1)
6.209	9897.096	↓ 1347.0	1347	1	→ HashAggregate (cost=792.00..792.00 rows=1 width=4) (actual time=9895.105..9897.096 rows=1347 loops=1)
9890.887	9890.887	↓ 3.3	1347	1	→ Index Scan using index_player_summaries_on_player_id on player_summaries ps (cost=0.00..791.80 rows=403 width=4) (actual time=27.413..9890.887 rows=1347 loops=1) Index Cond: (player_id = 21432312)
10825.839	10825.839	↑ 60.0	10	1347	→ Index Scan using index_player_summaries_on_match_id on player_summaries (cost=0.00..33.98 rows=600 width=99) (actual time=7.375..8.037 rows=10 loops=1347) Index Cond: (match_id = ps.match_id)

explain.depesz.com

HTML | TEXT | STATS

exclusive	inclusive	rows x	rows	loops	node
30.788	20753.723	↑ 105.6	13470	1	→ Nested Loop (cost=792.00..828.08 rows=1422317 width=99) (actual time=9928.869..20753.723 rows=13470 loops=1)
6.209	9897.096	↓ 1347.0	1347	1	→ HashAggregate (cost=792.00..792.00 rows=1 width=4) (actual time=9895.105..9897.096 rows=1347 loops=1)
9890.887	9890.887	↓ 3.3	1347	1	→ Index Scan using index_player_summaries_on_player_id on player_summaries ps (cost=0.00..791.80 rows=403 width=4) (actual time=27.413..9890.887 rows=1347 loops=1) Index Cond: (player_id = 21432312)
10825.839	10825.839	↑ 60.0	10	1347	→ Index Scan using index_player_summaries_on_match_id on player_summaries (cost=0.00..33.98 rows=600 width=99) (actual time=7.375..8.037 rows=10 loops=1347) Index Cond: (match_id = ps.match_id)

what to look for

- “seq scan” on large table
  - maybe index needed
- cartesian joins
- really bad row estimates
  - ANALYZE needed?

explain analyze

```
explain select count(*) from loans
where checkout_date between
'2011-01-01' and '2011-03-31';

explain analyze …

explain ( analyze on, buffers
on )...

explain ( analyze on, format
yaml ) ...
```

explain analyze

```
\i /setup/postgres/
      explain_quarterly_report.sql
```

Learn More

- Main Docs
  - http://www.postgresql.org/docs
- Blogs
  - http://planet.postgresql.org
- User Groups
  - http://www.postgresql.org/community/user-groups/

Learn More

User Groups:

- PDXPUG
- SFPUG
- NYCPUG
- many others!

Conferences:

- PDXpgDay
  Sept. 6, Portland
- PostgresOpen
  Sept. 16,
  Chicago
- pgConf.EU:
  Oct. 23, Madrid

questions?

- www.pgexperts.com/tutorials.html
- Josh Berkus: josh@pgexperts.com
  - PGX: www.pgexperts.com
  - Blog: www.databasesoup.com

PGX
POSTGRESQL
EXPERTS, INC.