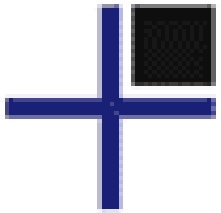
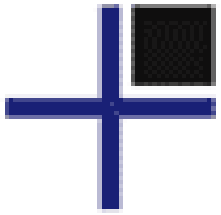


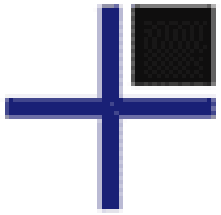
Beyond Query Logging

Greg Smith and Peter Geoghegan
2ndQuadrant



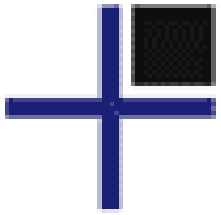
Starter postgresql.conf

```
shared_buffers = 512MB
checkpoint_segments = 64
effective_cache_size = 48GB
work_mem = 8MB
log_line_prefix = '%t [%p]: [%l-1] user=%u,db=%d '
log_min_duration_statement = 1000
log_temp_files = 0
log_checkpoints = on
log_connections = on
log_lock_waits = on
log_autovacuum_min_duration = 1000
```



Standard text logging

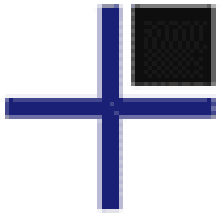
- Crazy naming scheme
- Rotation issues
- Long-term pruning is your problem
- Multi-line statements will split
- Performance issues
- You will hate this forever



Use syslog instead

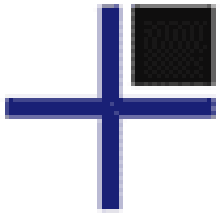
- syslog is a pain to setup
- Or maybe you have rsyslog?
- Perhaps syslog-ng?
- Several variations and defaults here
- Still need to setup naming conventions
- And customize the rotation scheme
- But you'll only hate it for a while
- Make sure there's no write sync:

```
LOCAL0.*    -/var/log/postgresql
```



Query Log Analysis

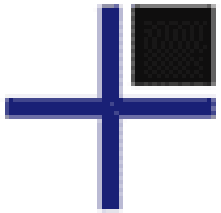
- Wait for a day of log data to finish
- Analyze logs with external tool
  - pgBadger is the latest hotness
    - Analyzes data like temp files too
  - pgFouine for small/medium sized logs
  - pgsi or pg_query_analyser for larger logs
- Process and publish to a web page
- View the history the next day



pgFouine

Queries that took up the most time (N) ^

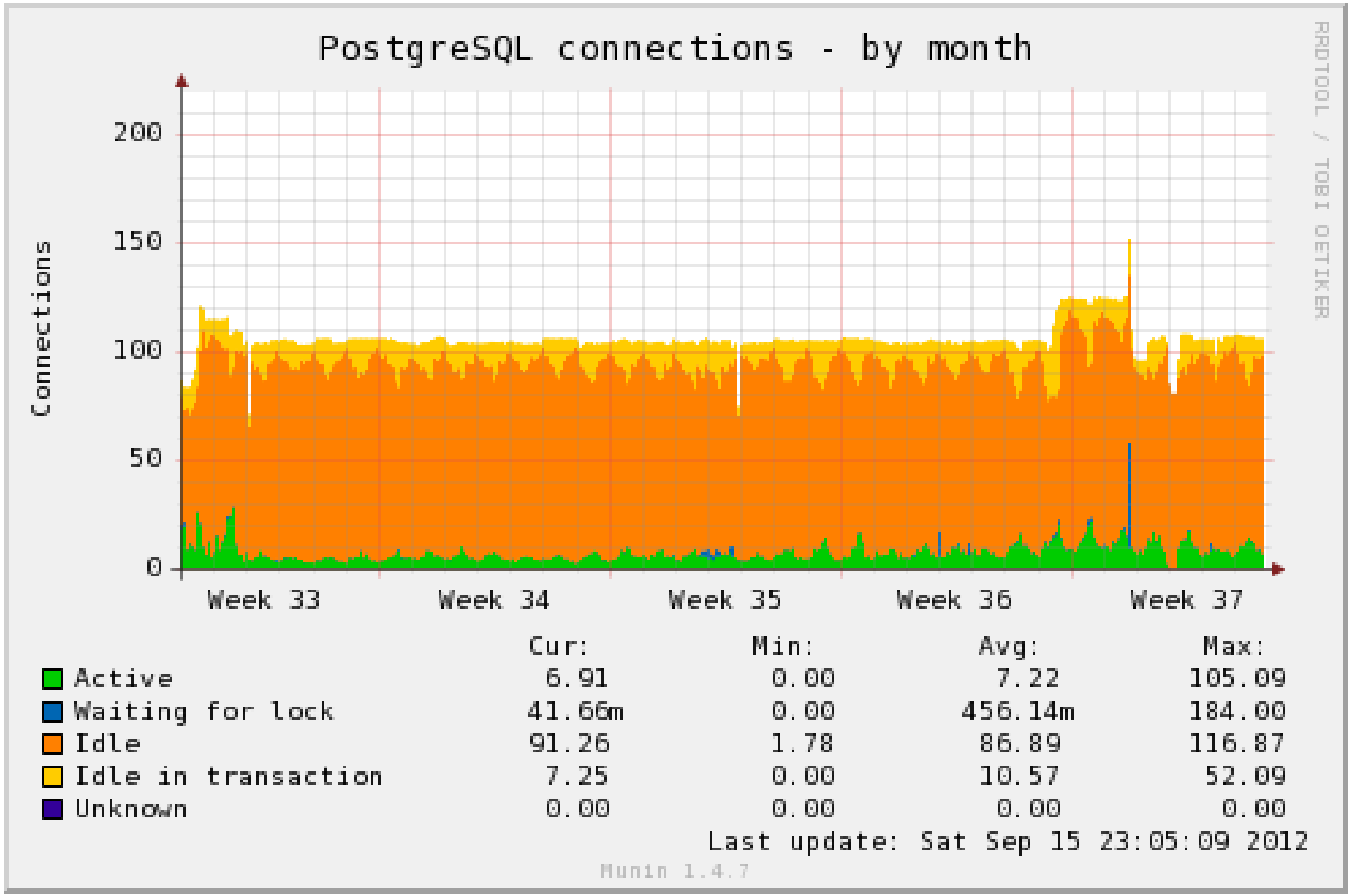
Rank	Total duration	Times executed	Av. duration (s)	Query
1	**1933h26m41s**	23,387	297.62	**UPDATE** accounts **SET** filler=*lower*('') **WHERE** aid < 0; Show examples
2	**17h14m20s**	23,387	2.65	**UPDATE** branches **SET** filler=*upper*(''); Show examples
3	**17m13s**	23,387	0.04	**SELECT** history.* **FROM** accounts, history **WHERE** accounts.aid=0 **AND** accounts.aid=history.aid; Show examples
4	**15m4s**	23,387	0.04	**SELECT** accounts.* **FROM** accounts,history **WHERE** history.aid=0 **AND** accounts.aid=history.aid; Show examples

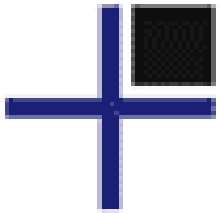


Overhead is high

- The query log is a single resource!
- There will be contention for it as volume increases
- Logs can become quite large
- May not be able to lower resolution enough

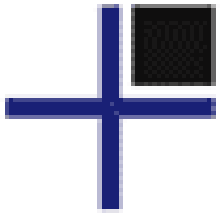
Duration drop: 2s to 500ms





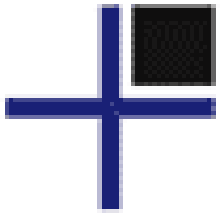
Fundamental problems

- Missing *many* short statements
- Normal trend spotting happens at multiple time scales
  - At best log analysis tools can summarize a longer period (i.e. week or month)
- Storage and processing time become infeasible
- Not even remotely close to real-time
  - Useless for crisis troubleshooting



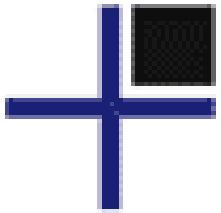
A busy server's logs

```
-rw-r--r--  1 gsmith gsmith  653M Jun 29  2011 postgresql.log-20110621
-rw-r--r--  1 gsmith gsmith  1.8G Jun 29  2011 postgresql.log-20110622
-rw-r--r--  1 gsmith gsmith  2.1G Jun 29  2011 postgresql.log-20110623
-rw-r--r--  1 gsmith gsmith  977M Jun 29  2011 postgresql.log-20110624
-rw-r--r--  1 gsmith gsmith  1.8G Jun 29  2011 postgresql.log-20110625
-rw-r--r--  1 gsmith gsmith  1.2G Jun 29  2011 postgresql.log-20110626
-rw-r--r--  1 gsmith gsmith  1.1G Jun 29  2011 postgresql.log-20110627
-rw-r--r--  1 gsmith gsmith  487M Jun 29  2011 postgresql.log-20110628
```



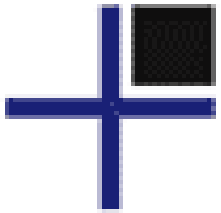
What do we want?

- Real-time data
- Save very frequently
  - 5 minute snapshots are standard for Munin etc.
- Line up in time with other trends
  - Database stats, disk I/O, CPU usage, locks
- Save in a way that aggregates well, too
- Normalized queries



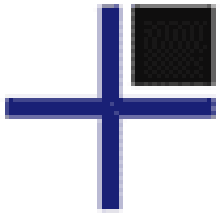
Typical pgbench statemetns

```
UPDATE pgbench_branches SET bbalance = bbalance + -4942 WHERE bid = 1;
UPDATE pgbench_branches SET bbalance = bbalance + -4261 WHERE bid = 1;
UPDATE pgbench_branches SET bbalance = bbalance + 1750 WHERE bid = 1;
INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES (1, 1, 18862, -2137,
  CURRENT_TIMESTAMP);
INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES (2, 1, 10509, 589,
  CURRENT_TIMESTAMP);
INSERT INTO pgbench_history (tid, bid, aid, delta, mtime) VALUES (4, 1, 12216, -3876,
  CURRENT_TIMESTAMP);
SELECT abalance FROM pgbench_accounts WHERE aid = 95405;
SELECT abalance FROM pgbench_accounts WHERE aid = 16069;
```



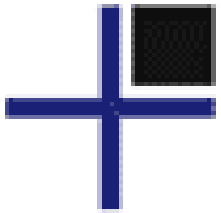
Normalized statements

```
UPDATE pgbench_branches SET bbalance =
  bbalance + ? WHERE bid = ?;

INSERT INTO pgbench_history (tid, bid, aid, delta,
  mtime) VALUES (?, ?, ?, ?, ?);

SELECT abalance FROM pgbench_accounts
  WHERE aid = ?;
```



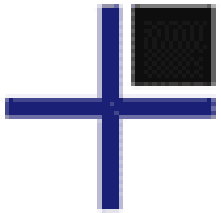
What did we have?

- pg_stat_statements (8.4 to 9.1)
- Only works usefully with prepared statements
- Has the correct workflow
  - Saves results to shared memory
  - Persists to a file across a restart
  - Can quickly take snapshots
  - Only disk access is writing those snapshots



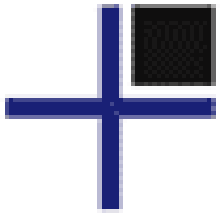
Idea: query normalizing regex

- Apply to statement before it's saved
- Same approach as pgFouine
- Allows real-time query monitoring
  - Not just with prepared statements!
- Avoid log scraping



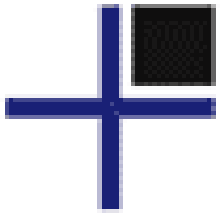
Hashing query trees

- Happens between query parsing and planning
- Quickly identify similar queries
- Output a normalized string



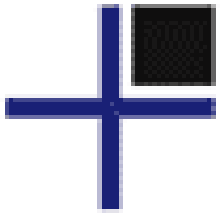
Theoretically sound

- Leveraging the core system in a well principled way
- Changes to PostgreSQL parser will automatically be reflected
  - Tools don't have to be updated for each version



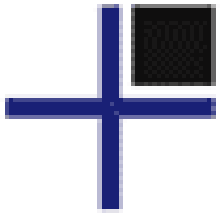
Tricky issues avoided

- Equivalent syntaxes are recognized as such
  - External tools can't know things like search_path
- Canonicalized versions are perfect
  - Constants are replaced with ? strings
  - No risk of accidentally processing the wrong thing
  - “ORDER BY 1,2” will not become “ORDER BY ?,?”
  - If “ORDER BY 1,2” is the same as “ORDER BY a,b”, they will be recognized as the same



pgbench pg_stat_statements

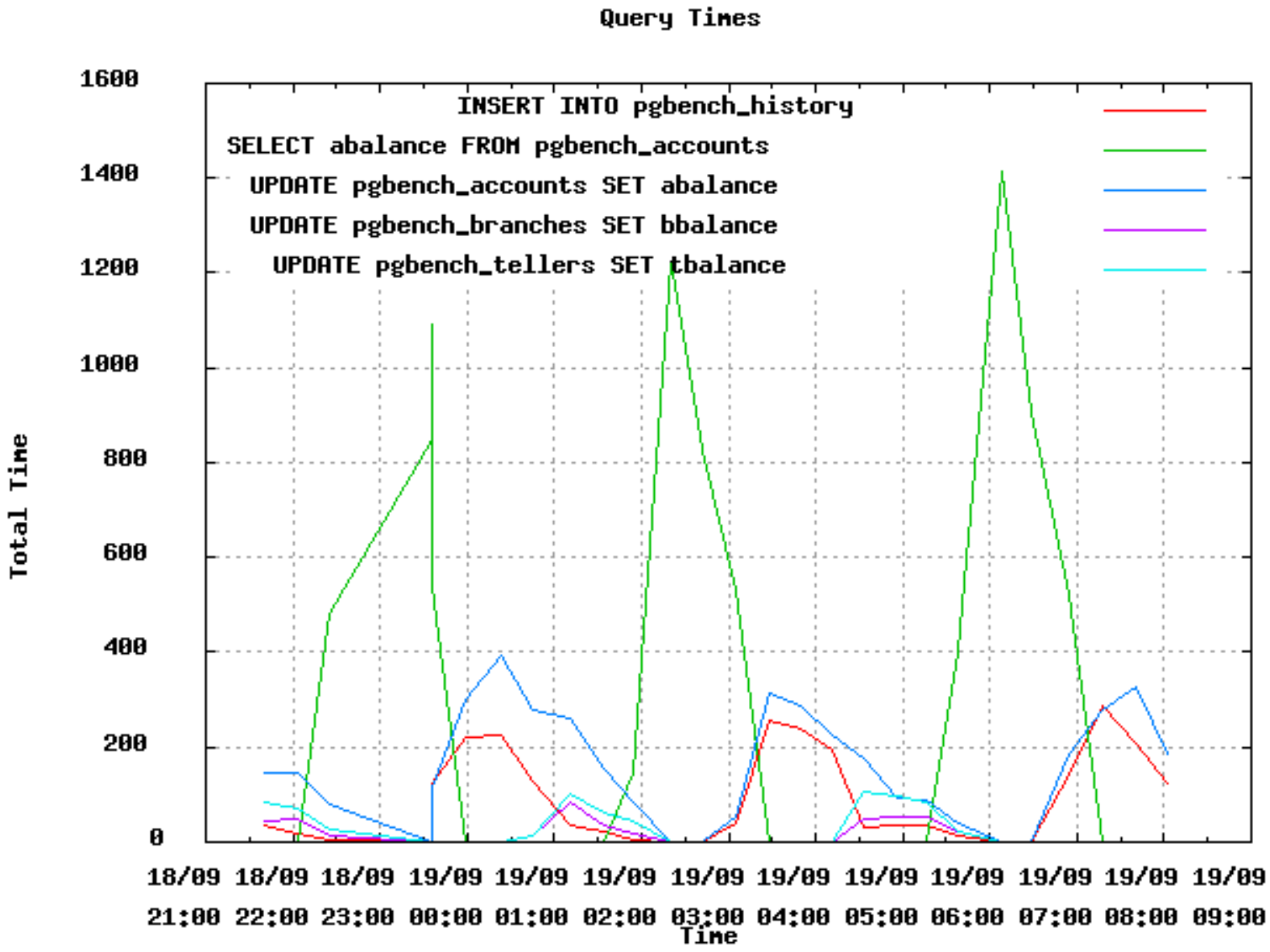
Calls	time	query
3121126	8934	SELECT abalance FROM pgbench_accounts WHERE aid = $1;
228074	4338	UPDATE pgbench_accounts SET abalance = abalance + $1...
228074	2454	INSERT INTO pgbench_history (tid, bid, aid, delta, mtime)...
108150	795	UPDATE pgbench_tellers SET tbalance = tbalance + $1 WHERE...
108150	461	UPDATE pgbench_branches SET bbalance = bbalance + $1 ...



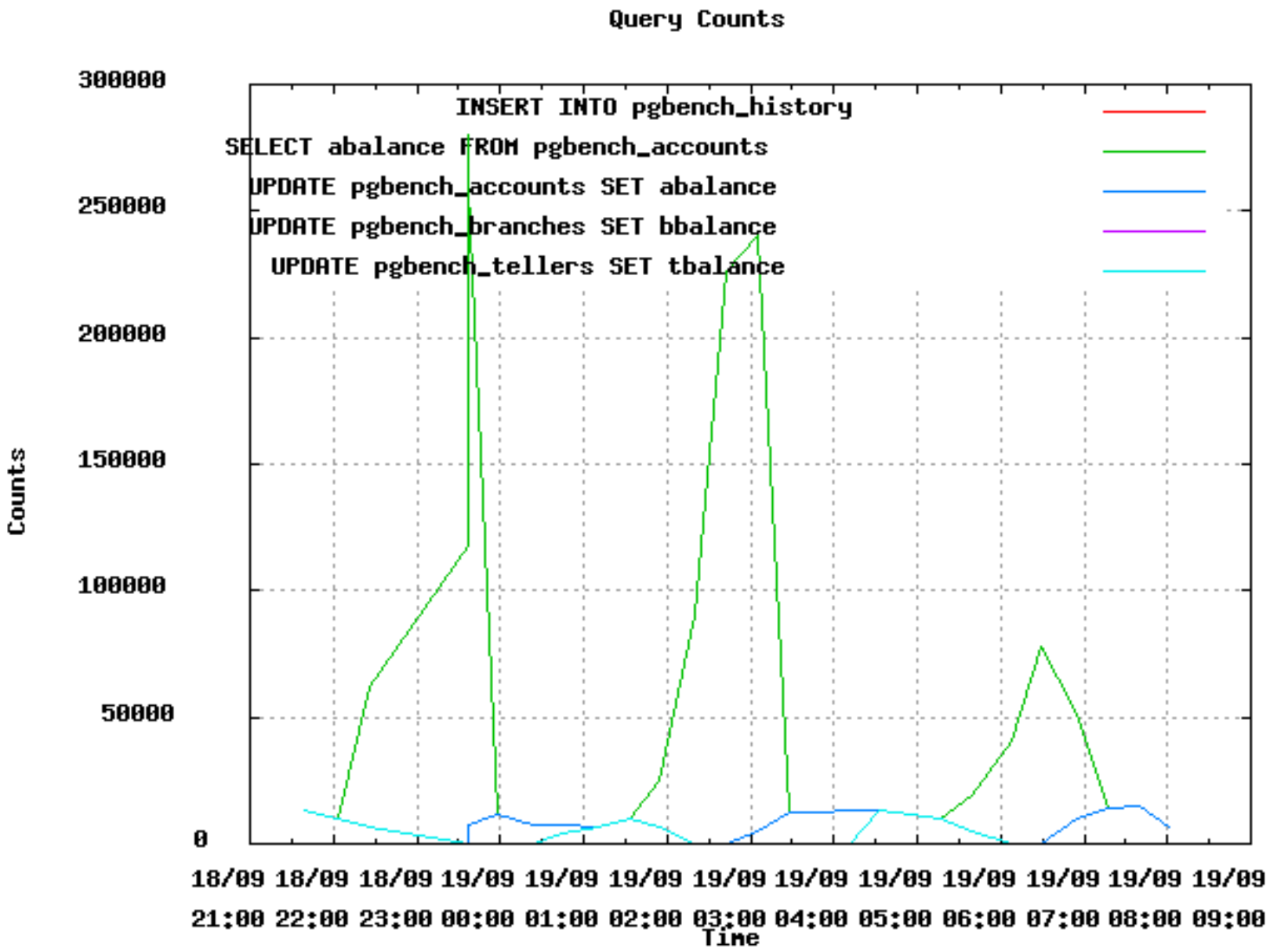
What can we can do with 9.2

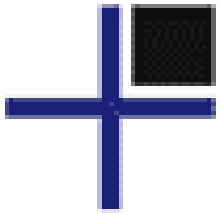
- True Workload Analysis
- Real-time snapshots
- Data on multiple time scales
- Query data graphed right next to database/OS stats

Workload Analysis: Time



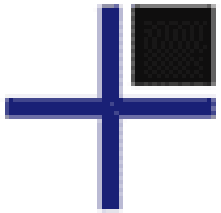
Workload Analysis: Counts





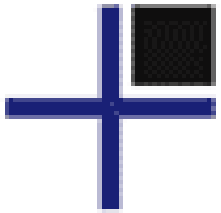
Next generation tools

- Save normalized query data snapshots regularly to an intermediate format
- Existing log parsing/normalization programs could save to that same format
- Display of data happens with another tool
- Tools that don't follow this workflow are dead, they just don't know it yet



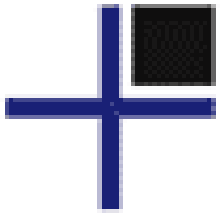
One more thing

- What about when query plans change over time?
- “I must have hints to prevent this from happening!”
- Many downsides to hints
- Query plan stability is a real problem though?
- When a query gets suddenly slow, how did it break?
- Can plan changes be spotted automatically?



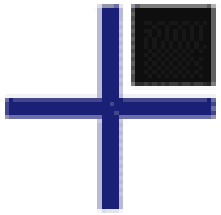
pg_stat_plans

- https://github.com/2ndQuadrant/pg_stat_plans
- Works on PostgreSQL 9.0+
- Working prototype extension, with some known limitations
- Saves the original query text and a hashed identifier of the plan (sort of)
- Clearly shows when a plan for that query text was changed
- Includes a function to grab the current explain plan
    - But doesn't save all of them (far too expensive)
- http://pgeoghegan.blogspot.cz/2012/10/first-release-of-pgstatplans.html



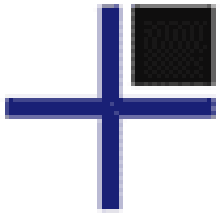
pg_stat_plans

```
postgres=# select
  planid,pg_stat_plans_explain(planid, userid,
  dbid) from pg_stat_plans;

planid                    | 2721250187

pg_stat_plans_explain |

Result  (cost=0.00..0.01 rows=1 width=0)
```



2ndQuadrant R&D

- More core PostgreSQL code contributors than any other company, all dedicated to open source work
- 2ndQuadrant customers are getting the latest tools and features, as we build them to satisfy their needs
- With our advice, you won't spend your time re-solving last year's problem



Questions?