PostgreSQL *Magazine*
News and stories from the Postgres Community

Performance Tuning :
To infinity... and beyond !

Special Edition

Interview : Bruce Momjian
Case Study : French Social Security System
Waiting for 9.1 : Extensions
Opinion : PostgreSQL vs. NoSQL
Tips & Tricks : SQL Query Analysis with pgFouine

May 2011
#00

About PG Mag

PostgreSQL Magazine is edited by and for the PostgreSQL Community.

Editor: Damien Clochard

Writers: Greg Smith, Frank Wiles, Hubert Lubaczewski, Jim Mlodgenski, Matt Tescher, Josh Berkus

Reviewers: Alvarro Herrera, Thom Brown, Greg Smith, Raymond O'Donnell, Ian Bailey-Leung, Osvaldo Tulini, Sivakumar Krishnamurthy, Evan Bennett, Belvedere Editorial

Tools : Scribus 1.4 / Gimp 2.6

Editorial

This is a test. A release candidate. What you are reading right now is the very first issue of a print magazine entirely dedicated to PostgreSQL, the world's most advanced open source database.

Launching a print magazine in 2011 may look like a weird idea... But PostgreSQL is now 25 years old and although it is one of the oldest open source projects alive, there are no print media for its large user base...

One of the goals of this project is to show you who are "the people behind the elephant", like Bruce Momjian (one of the PostgreSQL pioneers) who talks about the past and future of the PostgreSQL project on page 8.

And the future is bright for PostgreSQL with the new 9.1 version coming in a few weeks! This major release will include may stunning features (synchronous replication, unlogged tabled, SQL/MED, etc.). In this issue, we will focus on extensions (p. 20) which are a very easy way to package modules and add-ons for your databases.

For the first issue, we have tried to answer one of the questions most asked to DBAs: "Can you deliver the data faster?" Performance tuning is indeed a vast topic! You will find general advice (p. 12), OS oriented tips (p. 16) and a query analysis method (p. 24). We hope these articles will help you in your day-to-day use of PostgreSQL.

With this magazine, we want to create an open media aimed at the PostgreSQL user base. If you like PostgreSQL, we hope you'll like this magazine too!

And of course if you have any comments or ideas, just send us an email at contact@pgmag.org! We can't wait to read your feedback...

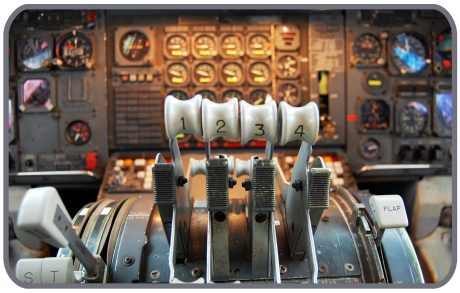
Damien Clochard

PostgreSQL Magazine #00

Agenda 4
Conferences & Events

Announcements 6
Product News

Interview 8
Bruce Momjian: 15 years of PostgreSQL Hacking

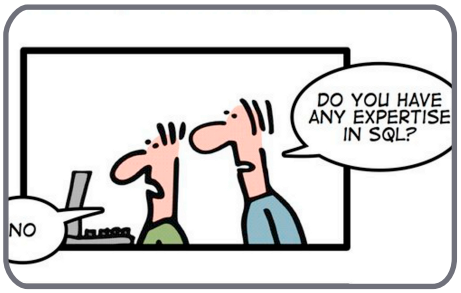


How To 12
Performance Tuning

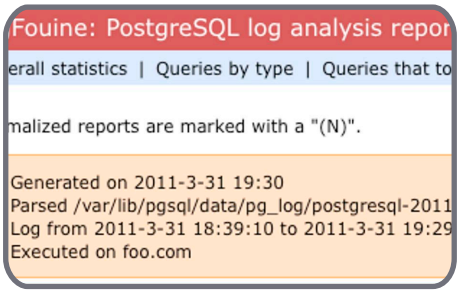
How To 16
Tuning Linux for PostgreSQL

Use Case 18
PostgreSQL at CNAF

Waiting for 9.1 20
Extensions



Opinion 22
One Database To Rule Them All



Tips & Tricks 24
Query Analysis with pgFouine

How To Contribute ?

This magazine is an open project. You can contribute in various ways: writing, editing, translating, proofreading, etc. To participate, send us a message at contact@pgmag.org

Upcoming Events

PGCon 2011
May 17-18 @ Ottawa, Canada

PGCon is the annual international conference for developers and users of PostgreSQL. PGCon is the place to meet, discuss, build relationships, learn valuable insights, and generally chat about the work you are doing with PostgreSQL.

http://www.pgcon.org/2011/

PGBR 2011
Nov. 3-4 @ São Paulo, Brazil

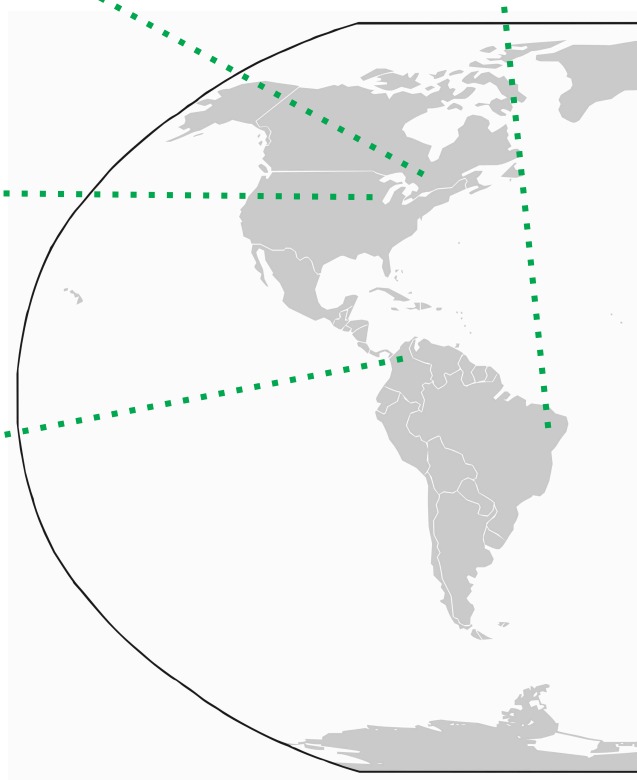
PGBR (formerly known as PGCon Brazil) is the biggest event on PostgreSQL in the Americas: in 2009 and 2008, the event brought more than 300 IT professionals and, in 2007, more than 200. In 2011, three rooms will be concurrent with tutorials, lectures and high level talks.

http://pgbr.postgresql.org.br/

Postgres Open
Sep. 2011 @ Chicago, USA

A summit and conference about the business of PostgreSQL in North America, this new conference welcomes users, vendors, and investors in PostgreSQL database technology.

http://postgresopen.org



Pg Conf Colombia 2011
Aug. 4-5 @ Bucaramanga, Colombia

PG Conf Colombia 2011 will be held from 4 to August 5, 2011 at the Universidad Industrial de Santander UIS, with private lessons on 2 and 3 August 2011.

http://www.pgconf.org/

For more information on local and international PostgreSQL user conferences, check out the wiki !

http://wiki.postgresql.org/wiki/Events

CHAR(11)
Jul. 11-12 @ Cambridge, UK

CHAR(11) is a conference on Clustering, High Availability and Replication that will be held in England this summer on 11-12 July 2011. Early bird bookings are available now by card or Paypal only for £345 plus VAT.

http://www.char11.org/

PostgreSQL Session #2
Jun. 23 @ Paris, France

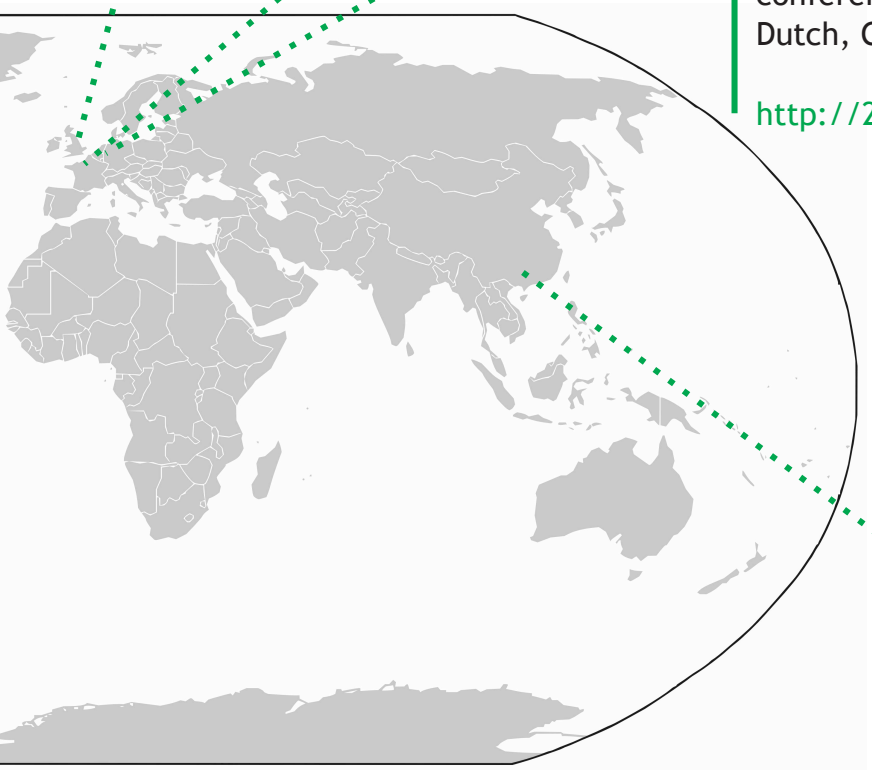
The PostgreSQL Sessions are free-as-beer conferences in Paris. Each session is a day consisting of lectures and workshops, organized around a specific theme and a guest. This second edition will be about PostGIS.

http://www.postgresql-sessions.org/en/

PostgreSQL Conference Europe 2011
Oct. 18-21 @ Amsterdam, Netherlands.

The European PostgreSQL Conference (formerly known as PGDay.EU) will be held at the Casa400 Hotel in Amsterdam. The first day of the conference will be a training day, and the following three days will be regular conference tracks. The conference will accept talks in English, Dutch, German and French.

http://2011.pgconf.eu/



China PostgreSQL Conference 2011
Jul. 16-17 @ Guangzhou, China

The Chinese PostgreSQL User Group (CPUG) is organizing a two day conference in July. Speakers include David Fetter and Tatsuo Ishii, among others.
For information and registration, contact CPUG2011@gmail.com

Announcements

pgwatch, a new PostgreSQL Monitor

In February, Cybertec Schönig & Schönig GmbH released pgwatch, a simple tool to monitor PostgreSQL 9.0 or higher. The goal of the project is to have a professional monitoring tool for PostgreSQL which can be used out of the box.
It offers a rich set of pre-defined graphs and functionality, such as: easy configuration, large number of ready-to-use charts, dashboard for fast system checking included, automated collection of statistics, interactive flash-charts, integrated SQL worksheet, etc.
pgwatch is Open Source and can be used freely under the terms of the Creative Commons License.

http://www.cybertec.at/en/pgwatch/

repmgr 1.1.0 is out

repmgr is a project developed by 2ndQuadrant to build large numbers of replicated PostgreSQL 9.0 standby servers. By providing a system daemon outside of the database to help coordinate multi-node actions, repmgr makes putting many nodes into a cluster easier to setup and manage. repmgr currently targets UNIX-like systems, requiring access to the PGXS interface for compiling PostgreSQL related code and the rsync utility for efficient node synchronization.

http://projects.2ndquadrant.com/repmgr

Dumbo 0.50, a new terminal client

Dumbo is a new "fullscreen" pgAdmin-style editor for the console. It is written in python with the urwid package, and runs on Linux.

https://bitbucket.org/mixmastamyk/dumbo/

PostgreSQL Query Cache released

In March, Satoshi NAGAYASU released a software package that improves query performance (10x-100x) by caching query results in front of backends. The PostgreSQL Query Cache waits for connections from clients on a different port, delegates the queries to a backend (like a proxy) and caches SELECT query results. It also manages the lifecycle of the query cache.

http://code.google.com/p/pqc/

PL/Proxy 2.2 is out

PL/Proxy is a database partitioning system implemented as procedural language. The new version includes a new TARGET statement to specify different functions to call on the remote side.

http://pgfoundry.org/projects/plproxy/

German doc translation project started

The project to translate the PostgreSQL documentation into German has started. A small number of volunteers has gathered to work on tools for better maintenance and to make translations easier. For now, what is not translated yet will be provided in English. The German translations will appear on the website little by little.

http://doc.postgres.de

Google Summer of Code 2011

The PostgreSQL project has been accepted into the Google Summer of Code 2011. Students submitted proposals between March 28 and April 8. Development work will run from May 23 through August 15.

http://wiki.postgresql.org/wiki/GSoC_2011

Helping out Japan and JPUG

PostgreSQL Users,

Our project has a special relationship with Japan, and right now the whole country is hurting. Japan PostgreSQL User Group (JPUG) represents the largest national group of PostgreSQL users in the world, possibly up to 20% of our users worldwide. Around 1/4 of the code contributors to PostgreSQL come from Japan as well, including both major and minor contributors, several of whom we have still not heard from.

As such, we are urging more fortunate supporters of the PostgreSQL project to support Japan in this disaster. The more aid Japan gets right now, the faster they will be able to recover from the current catastrophe. The Japanese on the coast need food, medical care, housing and rebuilding in excess of what the Japanese government can quickly supply.

We recommend the following organizations for donations to Japan relief:

International Federation of The Red Cross/Crescent:
http://pgmag.org/0007a

Doctors Without Borders:
http://pgmag.org/0007b

Josh Berkus
PostgreSQL Core Team

Bruce Momjian: 15 years of PostgreSQL Hacking

A true pioneer. Since 1996, Bruce Momjian has been an active member of the PostgreSQL Global Development Group. His experience gives him a unique standpoint inside the Postgres community, so we've decided to ask him about the past, present and future of the most advanced open source database.

PostgreSQL Magazine: Please tell us a little about yourself...

Bruce Momjian: I am of Armenian ancestry and have lived in the Philadelphia area all my life. I went from majoring in history in college to teaching computer science in high school to developing database applications for law firms. That's kind of an odd mix, and working on open source full-time is an even odder occupation. But it is a very rewarding one, and I thoroughly enjoy it.

PGM: When did you hear of PostgreSQL for the first time and why did you choose to contribute to the project?

BM: As a database applications developer, I used Ingres and Informix at work and wanted a database for my Unix machine at home. All the commercial database systems were too expensive, and I wanted to know how the database worked inside. Postgres was the ideal database because it was like commercial database systems, but I could see the code.

PGM: PostgreSQL has changed a lot since then! How do you see the success of Postgres over the years? Is it something that surprised you or did you expect it from the beginning?

BM: I really had no idea it would succeed. Postgres was just an interesting project and I was learning a lot by working with some very smart people (smarter than me). I never expected it to go anywhere because, at the time, there were few examples of open source software succeeding in the way Postgres has.

In the early years it was a struggle to justify the many hours I poured into Postgres while working at a job where I was paid as an hourly consultant — every hour spent on Postgres was an hour I wasn't paid, but it seemed like a good idea at the time, so I did it. I thought the skills I was learning might be useful someday.

PGM: PostgreSQL is now one of the oldest open source projects still actively developed. What is the secret of this longevity?

BM : Wow, it does surprise me how Postgres has taken off. I think one secret is how a small group of people whose heart is invested in producing top-quality software can motivate and influence many volunteers scattered around the world.

There are many reasons why our community should not be as healthy as it is, but it is healthy, growing, and productive. Humility, utilizing people's strengths, helping volunteers feel their contributions are appreciated — these are all secrets to our success.

PGM: What is your current job?

BM: I tend to do stuff no one wants to do. I used to apply a lot of patches, but now that is handled. I am constantly looking for things that need doing, and doing them, if only imperfectly. Practically, I create patches, track down bugs, improve documentation, write presentations, and speak at conferences.

PGM: How many hours do you work a day?

BM: Sometimes twelve hours, sometimes six. It all depends on what is happening, and, of course, if I am at a conference I am busy all the time.

If I work all day I try to put down my laptop around 9 or 10pm and do some leisure activity, but recently I have been traveling so much I am too backed up on email to do that. Many days I do some activity with my family, and because I work from home, that is easy to do, and it breaks up the day.

This job is a good fit for me because what the community (and my employer EnterpriseDB) need me to do is pretty much what I enjoy doing. Plus I like doing things that are useful (even if they are boring), so that works well for me.

And what is work, really? Sometimes I spend all morning on instant message with community members, going over open items, bugs, and tasks. Is that work? I think so, but it is also fun and interesting — so is improving documentation or cleaning up code. Is doing this interview work? I guess it is, but it is also interesting.

PGM: What is your current participation on the development group of PostgreSQL?

BM: I am a Postgres core member and primary maintainer of the pg_upgrade utility, which has become popular recently. I also know most of the Postgres people personally so I often act as a community liaison.

PGM: According to you, what's the biggest features in the forthcoming PostgreSQL 9.1 version?

BM: Here is my short list:

* Unlogged tables to better handle some NoSQL workloads
* Synchronous replication for greater reliability
* SQL/MED (Management of External Data) (flat files, other databases)
* Per-column collation support
* Security label, including SE-Linux integration
* True serializable isolation with predicate locking (already had snapshot isolation)
* Non-SELECT statement recursive queries
* Nearest-Neighbor (order-by-operator) Indexes
* Extensions (plugin) commands
* PL/Python overhaul (PL/Perl was done in 9.0)

PGM: PostgreSQL is often mentioned as being "the most advanced *open source* database". Do you think that in the near future it could overcome proprietary RDBMS and become the most advanced database of the whole market?

BM: Yes, I think Postgres is clearly on track for that. One of the things I recognized early on is that it doesn't matter how you currently rate against competitors, but rather how quickly you are improving. If you are improving faster than everyone else, you will eventually overtake them, even if it takes decades. Postgres is clearly increasing its user base and adding advanced features faster than any of the proprietary databases, so eventually Postgres will be the most advanced

database, period. We already excel in several areas.

PGM: What do you think of the NoSQL movement? Is it a threat for Postgres?

I think the best ideas from NoSQL will be incorporated into existing database systems over time, and NoSQL databases will lose some of their attraction. We saw that with object databases and XML databases in the past, and I think we are already seeing some of that happening with NoSQL.

PGM: Is there anything you haven't done yet but would like to see happening in the Postgres community?

BM: We had a lot of missing stuff early on, and we just kept chipping away at those. At this point, there isn't much left that is missing. I think we are going to see more advanced stuff being done with Postgres in the coming years, rather than just adding missing features. The good news is that we are even better at designing and implementing advanced, cutting-edge stuff than we are with adding missing features. You can see some of that in 9.1, and we already have some cool stuff lined up for 9.2, such as native JSON support.

PGM: What other things are you interested in outside of open source and PostgreSQL?

BM: I like to read, travel, and I enjoy doing things with my family.

More Momjian...

You can find more about Bruce on his website : blog, conference schedule, presentations, webcasts, résumé, press book, etc.

http://momjian.us

Performance Tuning PostgreSQL

PostgreSQL is the most advanced and flexible open source SQL database today. With this power and flexibility comes a problem. How do the PostgreSQL developers tune the default configuration for everyone? Unfortunately the answer is they can't.

The problem is that every database is not only different in its design, but also in its requirements. Some systems are used to logging mountains of data that is almost never queried. Others have essentially static data that is queried constantly, sometimes feverishly. Most systems however have some, usually unequal, level of reads and writes to the database. Add this little complexity on top of your totally unique table structure, data, and hardware configuration and hopefully you begin to see why tuning can be difficult.

The default configuration PostgreSQL ships with is a very solid configuration aimed at everyone's best guess as to how an "average" database on "average" hardware should be setup. This article aims to help PostgreSQL users of all levels better understand PostgreSQL performance tuning.

Understanding the process

The first step to learning how to tune your PostgreSQL database is to understand the life cycle of a query. Here are the steps of a query:

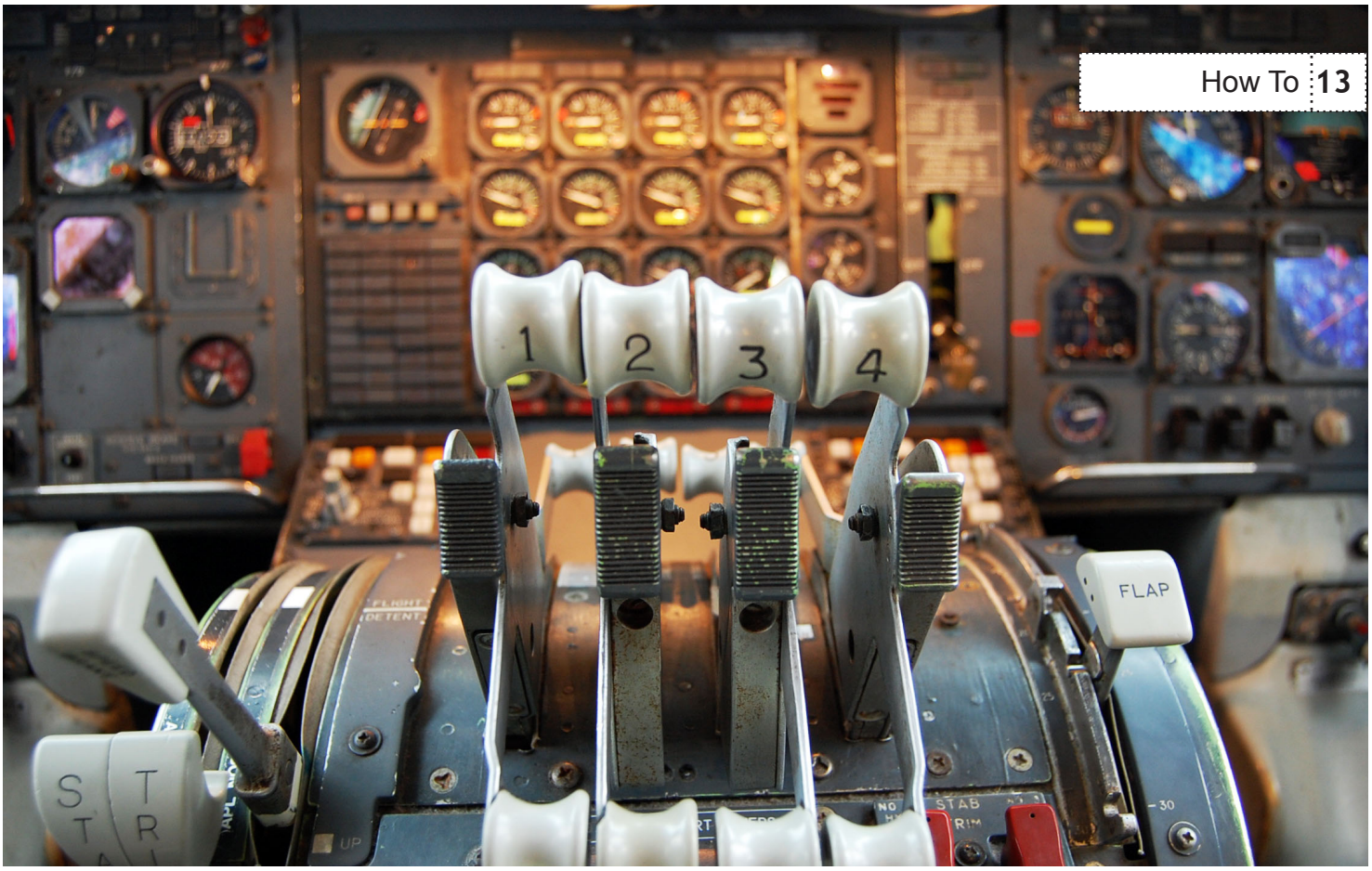
1. Transmission of query string to database
2. Parsing of query string
3. Planning of query
4. Retrieval of data from hardware
5. Transmission of results to client

The first step is the sending of the query string (the actual SQL command you type in or your application uses) to the database backend. There isn't much you can tune about this step; however, if you have very large queries that cannot be prepared in advance it may help to put them into the database as a stored procedure and cut the data transfer down to a minimum.

Once the SQL query is inside the database server it is parsed into tokens. This step can also be minimized by using stored procedures.

The planning of the query is where PostgreSQL really starts to do some work. This stage checks to see if the query is already prepared and if your version of PostgreSQL and client library support this feature. It also analyzes your SQL to determine what the most efficient way of retrieving your data is. Should we use an index and if so which one? Maybe a hash join on those two tables is appropriate? These are some of the decisions the database makes at this point of the process. This step can be eliminated if the query is previously prepared.

Now that PostgreSQL has a plan of what it believes to be the best way to retrieve the data, it is time to actually get it. While there are some tuning options that help here, this step is mostly affected by your hardware configuration.



And finally the last step is to transmit the results to the client. While there aren't any real tuning options for this step, you should be aware that all of the data that you are returning is pulled from the disk and sent over the wire to your client. Minimizing the number of rows and columns to only those that are necessary can often increase your performance.

General Tuning

There are several postmaster options that can be set that drastically affect performance. Below is a list of the most commonly used and how they affect performance:

max_connections: This option sets the maximum number of database backends to have at any one time. Use this feature to ensure that you do not launch so many backends that you begin swapping to disk and kill the performance of all the children. Depending on your application it may be better to deny the connection entirely rather than degrade the performance of all of the other children.

shared_buffers: Editing this option is the simplest way to improve the performance of your database server. The default is pretty low for most modern hardware. General wisdom says that this should be set to roughly 25% of available RAM on the system. Like most of the options I will outline here you will simply need to try them at different levels (both up and down) and see how well it works on your particular system. Most people find that setting it larger than a third starts to degrade performance.

effective_cache_size: This value tells PostgreSQL's optimizer how much memory PostgreSQL has available for caching data and helps in determing whether or not it uses an index. A large value increases the likelihood of using an index. This should be set to the amount

of memory allocated to shared_buffers plus the amount of OS cache available. Often this is more than 50% of the total system memory.

work_mem: This option is used to control the amount of memory usage in sort operations and hash tables. While you may need to increase the amount of memory if you do a ton of sorting in your application, care needs to be taken. This isn't a system wide parameter, but a per operation one. So if a complex query has several sort operations in it, it will use multiple work_mem units of memory. Not to mention that multiple backends could be doing this at once. This query can often lead your database server to swap if the value is too large. This option was previously called sort_mem in older versions of PostgreSQL.

max_fsm_pages: This option helps to control the free space map. When something is deleted from a table it isn't removed from the disk immediately, it is simply marked as "free" in the free space map. The space can then be reused for any new INSERTs that you do on the table. If your setup has a high rate of DELETEs and INSERTs it may be necessary to increase this value to avoid table bloat. Note that max_fsm_pages is removed as of PostgreSQL 8.4 and later.

fsync: This option determines if all your WAL pages are fsync'ed to disk before a transaction is committed. Having this on is safer, but can reduce write performance. If fsync is not enabled there is the chance of unrecoverable data corruption. Turn this off at your own risk.

commit_delay and commit_siblings: These options are used in concert to help improve performance by writing out multiple transactions that are committing at once. If there are commit_siblings

Check out the wiki

The official wiki of the PostgreSQL project (http:/wiki.postgresql.org) has a very nice article titled "Tuning Your PostgreSQL Server". You'll find there most answers to your performance questions.

http://pgmag.org/0014

number of backends active at the instant your transaction is committing then the server waits commit_delay microseconds to try and commit multiple transactions at once.

random_page_cost: This option controls the way PostgreSQL views non-sequential disk reads. A higher value makes it more likely that a sequential scan will be used over an index scan indicating that your server has very fast disks.

Note : Many of these options consume shared memory and it will probably be necessary to increase the amount of shared memory allowed on your system to get the most out of these options.

Hardware Issues

Obviously the type and quality of the hardware you use for your database server drastically impacts the performance of your database. Here are a few tips to use when purchasing hardware for your database server (in order of importance):

RAM: The more RAM you have the more disk cache you will have. This greatly impacts performance, considering memory I/O is thousands of times faster than disk I/O.

Disk types: Obviously fast SAS disks are your best option; however, high end SATA drives are also very good. With SATA each disk is substantially cheaper and with that

you can afford more spindles than with SAS on the same budget.

Disk configuration: The best configuration is RAID 1+0 with as many disks as possible and with your transaction log (pg_xlog) on a separate disk (or stripe) all by itself. RAID 5 is not a very good option for databases unless you have more than 6 disks in your volume. With newer versions of PostgreSQL you can also use the tablespaces option to put different tables, databases, and indexes on different disks to help optimize performance. Such as putting your often used tables on a fast SAS disk and the less used ones slower IDE or SATA drives.

CPUs: The more CPUs the better, however if your database does not use many complex functions your money is best spent on more RAM or a better disk subsystem.

In general the more RAM and disk spindles you have in your system the better it will perform. This is because with the extra RAM you will access your disks less. And the extra spindles help spread the reads and writes over multiple disks to increase throughput and to reduce drive head congestion.

Another good idea is to separate your application code and your database server onto different hardware. Not only does this provide more hardware dedicated to the database server, but the operating system's disk cache will contain more PostgreSQL data and not other various application or system data this way.

For example, if you have one web server and one database server you can use a cross-over cable on a separate ethernet interface to handle just the web server to database network traffic to ensure you reduce any possible bottlenecks there. You can also obviously create an entirely different physical network for database traffic if you have multiple servers that access the same database server.

Application Issues

These issues typically affect both stateful and stateless applications in the same fashion. One good technique is to use server side prepared queries for any queries you execute often. This reduces the overall query time by caching the query plan for later use.

It should be noted however if you prepare a query in advance using placeholder values (such as 'column_name = ?') then the planner will not always be able to choose the best plan. For example, if your query has a placeholder for the boolean column 'active' and you have a partial index on false values the planner won't use it because it cannot be sure the value passed in on execution will be true or false.

You can also obviously utilize stored procedures here to reduce the transmit, parse, and plan portions of the typical query life cycle. It is best to profile your application and find commonly used queries and data manipulations and put them into a stored procedure.

About the article

The original presentation is available here : http://pgmag.org/0015

About the author

Frank Wiles is an open source consultant specializing in scaling and performance . He founded Revolution Systems (revsys.com) to help businesses take full advantage of all the benefits of the Open Source Software revolution.

Tuning Linux for PostgreSQL

Linux is one of the most popular server operating systems for PostgreSQL. One reason for that is that it generally gives good performance. But there are a few choices and optimizations to take care of if you want to get the best performance it's capable of.

Filesystems

Older Linux distributions default to using the ext3 filesystem, which is reliable but not the fastest around. The newer ext4 is faster but still a bit immature; make sure you load test heavily before deploying it. A good middle ground is XFS, which is very stable at this point while also being quite fast. All these filesystems benefit from using the "noatime" mounting parameter. That eliminates updating information about when files were last accessed, something that's normally not important for databases.

Read-ahead

PostgreSQL relies on the operating system to do some amount of read-ahead, which is essential to good read performance on sequential scans of large tables. Linux has an excellent read-ahead implementation, but it's set to a low number of blocks (256) to grab by default. 4096 blocks is a good starting point to use for that setting instead, and read-ahead is set on block devices like this:

```
/sbin/blockdev --setra 4096 /dev/sda
```

Reliable & Fast Writes

Getting the best performance from PostgreSQL can require selecting the right hardware for that job. And choosing the wrong hardware or setting it up incorrectly can put your data at risk. You can find articles on hardware selection and operating system options needed for both good performance and reliable database writes at http://pgmag.org/0017
For example, if you have a system with a battery-backed write controller, that greatly improves write performance. But XFS and ext4 filesystems will need the "nobarrier" mount option to work well with it.

Write caching

On the write side, PostgreSQL expects the operating system to buffer normal writes, only forcing them out to disk when the background checkpoints complete. The size of this write cache was much larger than is normally preferred on Linux kernels before 2.6.22, which includes the still popular RedHat Enterprise 5. You can get the better new defaults on older kernels like this:

```
echo 10 > /proc/sys/vm/dirty_ratio
echo 5 > /proc/sys/vm/dirty_background_ratio
```

These settings allow 10% of RAM to be used as a write cache, preemptively stepping up the rate it's written once it reaches 5%. These numbers may still be too high on systems with many gigabytes of memory. Starting in Linux kernel 2.6.29, these values can be set even smaller using the new "dirty_background_bytes" and "dirty_bytes" settings. They replace the percentage based values when you use them. Note that some bulk operations in PostgreSQL, particularly VACUUM, may slow significantly if you reduce Linux's write cache by too much.

I/O Scheduling

Particularly when running a workload with a heavy mix of read and write operations, Linux's kernel I/O scheduler can impact performance when the system gets very busy. It's hard to see a useful difference here if you just do simple performance testing, and the options already mentioned normally have a bigger impact. But on a complicated database workload, the scheduler can make a significant difference too.

The default scheduler, Completely Fair Queuing (CFQ), is a good starting point but not optimal in every case. The primary alternative to consider is the deadline scheduler, which can give better response times with a read/write mix where response time latency is important.

If you have a disk array with some intelligence in it, such as a good RAID controller or external SAN storage, I/O scheduling may just add overhead without any benefit. In those cases, using the NOOP scheduler may give best performance. It does a minimal amount of work, assuming the disks or their controller will make good decisions instead.

About the author

Greg Smith is the principal consultant of 2ndQuadrant US and the author of "PostgreSQL 9.0 High Performance": http://www.2ndquadrant.com/books
His blog at http://blog.2ndquadrant.com regularly covers Linux and PostgreSQL tuning strategies.

PostgreSQL at CNAF : 1 billion SQL queries per day

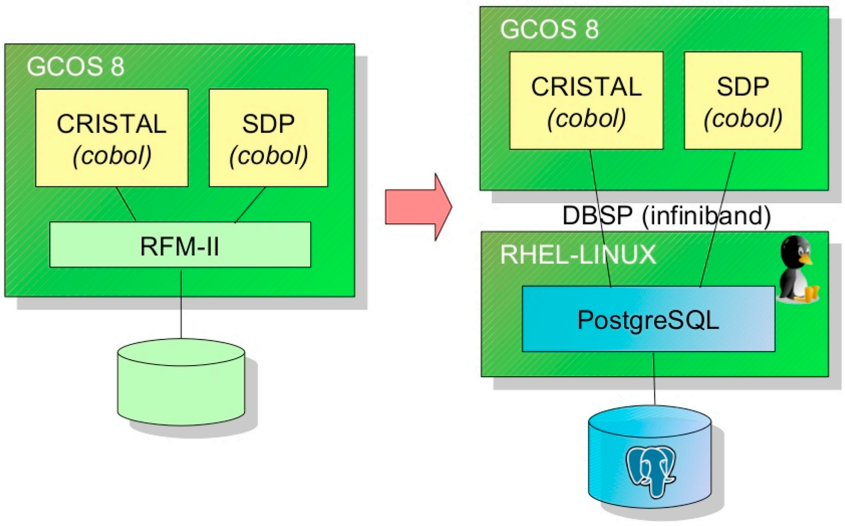
In 2010 the CNAF, one of the biggest parts of the French Social Security System, moved its core business applications to PostgreSQL. This use case is quite a success story, as it shows how PostgreSQL is perfectly fitted for big administrations, high workloads and large scale databases.

The CNAF (see p.19) moved its legacy Bull Interel-RFM2 database to PostgreSQL. That decision was a reaffirmation of CNAF's commitment to move towards open source solutions. As a result of this, every month PostgreSQL is now involved in the payment of €3 billion in benefits to CNAF claimants. The project, which is fully up and running, is an integral part of the program to transform the information systems of this highly sensitive organization.

To improve the quality of its services against a backdrop of cost constraints, CNAF has put the emphasis on the search for better performance and more powerful systems. To achieve this, it decided to modernize two of its strategic and rapidly evolving applications, CRISTAL and SDP, which together account for 20 million lines of application software code and are central to its core business of calculating people's entitlements and paying benefits. The choice of a new database that meets CNAF's criteria in terms of performance and openness was an obvious step, with PostgreSQL proving to be the best option.

As Marc Pavie, Deputy Director of Information Systems at CNAF, explained: "*The choice of PostgreSQL reiterates our commitment to open source. With Bull as our prime contractor, we migrated without any difficulty. With PostgreSQL we are now benefiting from new functionality which is contributing to the optimization of our information system. But most importantly the solution offers levels of robustness and performance largely in line with our production imperatives and challenges at a national level relating to the quality of our services.*"

The migration project started in October 2008, following a feasibility study carried



CNAF ?

The Caisse Nationale d'Allocations Familiales (CNAF) is the family branch of the French social security system, paying financial aid to families and providing further local, collective social action primarily through technical assistance and grants to local social life (town halls, nurseries, etc.). Every year, the CNAF provides some €69 billion of benefits to 11 million claimants and overall 30 million persons concerned. Its information systems hold highly sensitive data, such as the French government's minimum income guarantee for low-income households.

out by Bull and a prototype built in collaboration with CNAF's teams. An extensive testing phase established a sound basis for the implementation, which took place over the nine months to April 2010.

« With PostgreSQL we are now benefiting from new functionality which is contributing to the optimization of our information system »

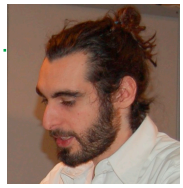
In total, it took just 18 months to migrate the 168 databases involved, representing a total of 4 Terabytes of data. Since then, almost a billion SQL queries are being run every day. Regular administrative tasks have been automated and system supervision is carried out using an open software solution, Nagios. In addition, the architecture that has been implemented features dual high-availability, with a remote business recovery site and locally based service continuity facilities.

One year after the implementation was completed, the PostgreSQL database is clearly proving its robustness and levels of performance. As a result, the reduction in batch processing times and transaction response times has led to improvements in the quality of the service provided.

Jean-Pierre Barbéris, CEO of Bull France, commented: "*We are very proud to have supported CNAF in this project, which is unprecedented in Europe. The resulting benefits bear witness not only to the extraordinary power and robustness of Open Source tools, but also to their ability to support organizations' most innovative projects while at the same time helping them to achieve greater room for maneuver financially.*"

CNAF is now considering some improvements to take advantage of PostgreSQL features such as migrating to PostgreSQL 9.x in order to reach even higher availability for the data servers.

About the author



Damien Clochard is one of the directors of Dalibo. He is also webmaster of the postgresql.fr website and involved in several PostgreSQL advocacy projects in France.

Extensions coming in PostgreSQL 9.1

The forthcoming version of PostgreSQL will bring us a shipload of exciting new features. Among them Extensions are a key improvment that will make it much easier for software vendors to supply additional functionality for PostgreSQL.

On 8th of February, Tom Lane committed this patch:

```
Core support for "extensions", which are
packages of SQL objects.

This patch adds the server infrastructure to
support extensions. There is still one
significant loose end, namely how to make it
play nice with pg_upgrade, so I am not yet
committing the changes that would make all the
contrib modules depend on this feature.

In passing, fix a disturbingly large amount of
breakage in AlterObjectNamespace() and
callers.

Dimitri Fontaine, reviewed by Anssi
Kääriäinen, Itagaki Takahiro, Tom Lane, and
numerous others
```

Extensions are a way to group various database objects together to allow easy manipulation. For starters - all contrib modules became extensions.

This means that installing and changing them is now much simpler.

For example, adding the ltree contrib module to a database is now as simple as:

```
$ select '1.1'::ltree;
ERROR: type "ltree" does not exist
LINE 1: select '1.1'::ltree;
                       ^
$ create extension ltree;
CREATE EXTENSION
$ select '1.1'::ltree;
ltree
-------
1.1
(1 row)
```

Removing it is also trivial:

```
$ drop EXTENSION ltree;
DROP EXTENSION
```

But the great stuff is that you can easily load the extension to a different schema, and after loading you can easily alter it.

Postgres 9.1 Beta1

PostgreSQL beta versions are feature-frozen snapshots of the forthcoming 9.1 major release. They are not meant for production use. They are intended to preview and test upcoming features and to provide the possibility for early feedback. If you are a developer or a DBA, testing PostgreSQL betas is one of the best things you can do to help get the release out faster, with more features and less bugs.

http://www.postgresql.org/developer/beta

Let's see how it works:

```
$ create extension ltree;
CREATE EXTENSION
$ create table test (z ltree);
CREATE TABLE
```

OK. So I have a table with a column of ltree datatype. But now I've decided I don't want ltree-related functions "polluting" the public schema, and I want to move it to the designated schema just for it:

```
$ create schema ltree;
CREATE SCHEMA
$ alter extension ltree set schema ltree;
ALTER EXTENSION
```

The table got modified too:

```
$ \d test
Table "public.test"
Column  | Type         | Modifiers
--------+--------------+-----------
z       | ltree.ltree |
```

About the article

The original article is available at : http://postgr.es/p/-6

While this might not seem like a big deal, in fact it is.

Having all objects grouped together means that you can upgrade them, and dump/restore is easier. And instead of dumping all functions/tables/types definitions, the dump contains only:

```
# pg_dump | grep ltree
-- Name: ltree; Type: SCHEMA; Schema: -;
Owner: depesz
CREATE SCHEMA ltree;
ALTER SCHEMA ltree OWNER TO depesz;
-- Name: ltree; Type: EXTENSION; Schema: -;
Owner:
CREATE EXTENSION ltree WITH SCHEMA ltree;
-- Name: EXTENSION ltree; Type: COMMENT;
Schema: -; Owner:
COMMENT ON EXTENSION ltree IS 'data type for
hierarchical tree-like structures';
z ltree.ltree
```

The benefit of this will be appreciated by anyone who ever tried to load a dump made on some PostgreSQL server to another, when there were loaded extensions (modules), and paths changed. Or the new version of a contrib module didn't provide the same functions (some new, some old disappeared).

Writing extensions looks trivial, so I assume that we will see migration of pgFoundry projects to extensions approach soon.

About the author

Hubert "Depesz" Lubaczewski is a Database Architect at OmniTI. His blog (http://www.depesz.com/) is dedicated to the new features of the forthcoming version of PostgreSQL.

One Database to Rule Them All

The big new topic in databases today is the use of NoSQL implementations such as MongoDB, Cassandra and Couchbase.

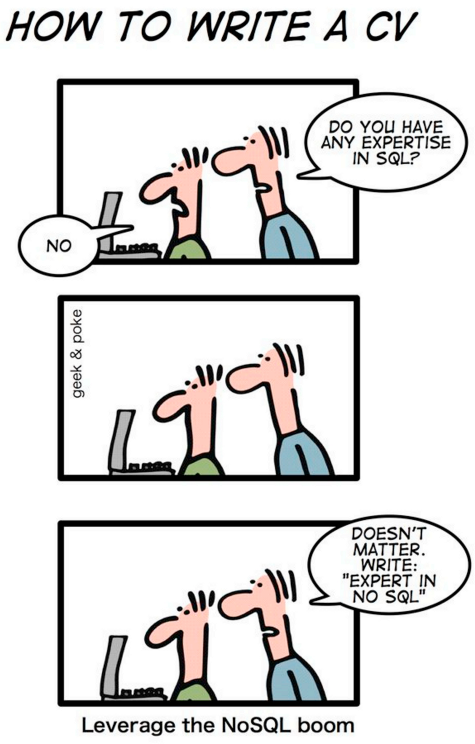
Everywhere I turn, I'm seeing articles about how relational databases are dead and all data will be stored in these flexible eventually consistent data stores. Recently, at PG East, there was a MongoDB track where I had the opportunity to talk to many people either using NoSQL or considering using NoSQL and their reasons for using it were extremely varied.

Some people had very legitimate reasons for using a NoSQL option over a traditional relational database like PostgreSQL. There are just some inheritant limitations in relational theory that will never compete with a schema-less document store when it comes to performance. Inserting a parent child relationship such as a new user of a web application who has 2 addresses and 3 phone numbers is much faster in MongoDB, than in PostgreSQL. In PostgreSQL, we would split the user into a row in the user table, 2 rows in the address table and 3 rows in the phone table and let's just ignore those pesky foreign key constraints slowing things down, but in MongoDB, we just insert a single BSON document containing all of the information. For applications with extremely high throughput rates that fit a NoSQL model, it is hard to argue with the performance advantages of this non relational method.

Many people, however, did not fit into these extreme usage scenarios and used NoSQL as an easy to use persistent data store. For many of these users, PostgreSQL may actually be a better choice, but unfortunately SQL is becoming more of a foreign language to web developers.

Most applications do need to relate information across tables and that is where PostgreSQL outperforms the NoSQL options. To display in an application a list of all of the sales last month from users in New York, a NoSQL database like MongoDB would need to first find all of the users from New York and then use that list as the input into the next query to find the sales.

A developer using PostgreSQL would just simply join the sales table with the user



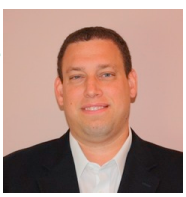
Leverage the NoSQL boom

and address tables in a single query. PostgreSQL would then handle the optimization of the query such as dealing with the case of having 100,000 users in New York, but only 10 total sales. PostgreSQL would know, based on the statistics of the tables, that it is much more efficient to pull all of the sales and use that as the criteria to pull the final result set. In the MongoDB case, the developer needs to know which order is most efficient, and can frequently be wrong.

The manner in which application architecture is evolving implies that we are long past the days where relational databases were a one-size-fits-all solution, but relational databases are nowhere near dead.

The key to the continuing dominance of relational databases and expansion of PostgreSQL use is how relational databases interact and coexist with new data stores. The improvements in the SQL/MED functionality in 9.1 are a great first step. With a MongoDB foreign data wrapper, it will allow MongoDB documents to appear as PostgreSQL rows allowing developers to join data in NoSQL data with relational data. Imagine being able to run a single query that will join data in your MongoDB instance with your financial information sitting in an Oracle database and inserting that result into your Teradata Data Warehouse. PostgreSQL with SQL/MED is building toward that and is putting itself into the position to be the one database to rule them all and bind them.

About the author



Jim Mlodgenski is the founder of Cirrus Technologies, Inc. a consulting and product development organization, dedicated to providing scalable data architectures.

What is NoSQL ?

NoSQL is an umbrella term for a loosely defined class of non-relational data stores that break with a long history of relational databases and ACID guarantees. Data stores that fall under this term may not require fixed table schemas, and usually avoid join operations. The term was first popularised in early 2009.

NoSQL databases are intended to perform at internet scale and reject the relational model in favor of other (key-value, document, graph) models. They often achieve performance by having far fewer features than SQL databases. Rather than being a "jack of all trades, master of none", NoSQL databases tend to focus on a subset of use cases.

Get Ready for SQL/MED !

SQL/MED (Management of External Data) is an extension to the SQL standard that defines foreign-data wrappers and datalink types to allow a DBMS to integrate date stored outside the database.
The implementation of this specification has begun in PostgreSQL 8.4 and will be available in the forthcoming 9.1 version with the very first foreign-data wrapper : file_fdw, which can be used to access data files in the server's filesystem.

http://wiki.postgresql.org/wiki/SQL/MED

SQL Query Analysis with pgFouine

pgFouine is a really nice Postgres log analyzer that I have been using. Like pqa it can generate html reports from a Postgres log file which will display slow running queries and most run queries, among other statistics.

Here are the basic steps to set up and use pgFouine:

1. If you don't already have it, you will need to install php (via yum, manually, etc.):

```
yum install php
```

2. Get the latest version of pgFouine:

```
wget http://.../pgfouine-1.2.tar.gz
tar -zxvf pgfouine-1.2.tar.gz
```

3. In postgresql.conf, you will have to change the default log line prefix. For this example, the log destination is set to stderr rather than syslog :

```
log_destination = 'stderr'
logging_collector = true
log_line_prefix = '%t [%p]: [%l-1] '
log_min_duration_statement = 0
```

Setting log_min_duration_statement to 0 will log all SQL statements.

4. Reload the postgres configuration:

```
sudo -u postgres psql
select pg_reload_conf();
```

5. You might also force the pg log to roll over (this will allow you to generate a report off of a fresh log file):

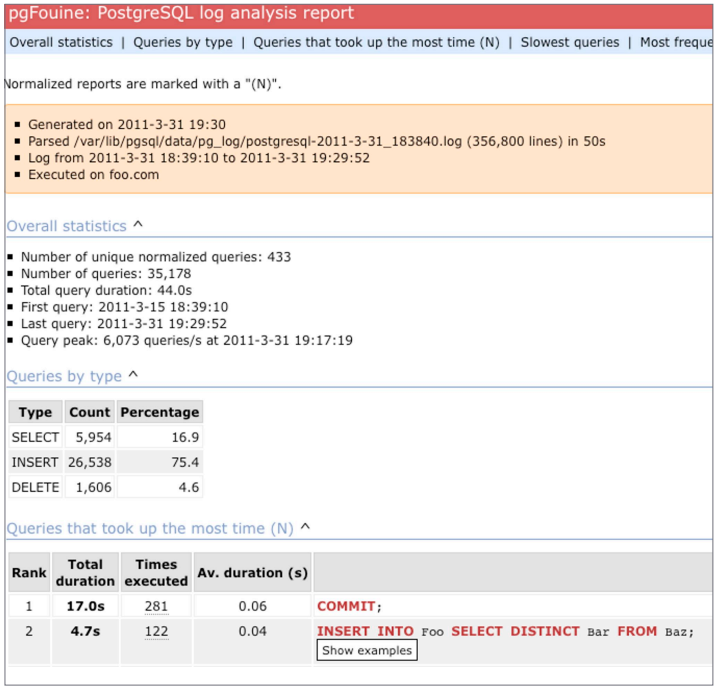
```
select pg_rotate_logfile();
```

6. Now go do some things in your application (i.e. perform a sequence of activities that a user of your application might do), to capture some of the SQL statements that get run by the application.

7. To generate a report, run pgfouine.php on the command line, passing it the latest postgres log to use:

```
./pgfouine.php -logtype stderr  \
               -file postgresql.log \
               > report.html
```

The resulting report will look something like:



pgFouine: PostgreSQL log analysis report

Overall statistics | Queries by type | Queries that took up the most time (N) | Slowest queries | Most freque

Normalized reports are marked with a "(N)".

- Generated on 2011-3-31 19:30
- Parsed /var/lib/pgsql/data/pg_log/postgresql-2011-3-31_183840.log (356,800 lines) in 50s
- Log from 2011-3-31 18:39:10 to 2011-3-31 19:29:52
- Executed on foo.com

Overall statistics ^

- Number of unique normalized queries: 433
- Number of queries: 35,178
- Total query duration: 44.0s
- First query: 2011-3-15 18:39:10
- Last query: 2011-3-31 19:29:52
- Query peak: 6,073 queries/s at 2011-3-31 19:17:19

Queries by type ^

Type	Count	Percentage
SELECT	5,954	16.9
INSERT	26,538	75.4
DELETE	1,606	4.6

Queries that took up the most time (N) ^

Rank	Total duration	Times executed	Av. duration (s)	
1	17.0s	281	0.06	COMMIT;
2	4.7s	122	0.04	INSERT INTO Foo SELECT DISTINCT Bar FROM Baz; Show examples

About the article

The original article is available at :
http://pgmag.org/0024

About the author

Matt Tescher is a Software Engineer at Elastra, in San Francisco, USA