EXPERT TIPS AND TRICKS FOR INTEGRATING LARGE-SCALE DATABASES WITH PERL AND PYTHON

AN ACTIVESTATE BEST PRACTICES WHITEPAPER

ActiveState

EXPERT TIPS AND TRICKS

INTRODUCTION

Organizations depend on large-scale databases to manage large data sets and need well-tested, well-supported solutions to manage and massage their data.

Enterprises are deploying databases in cluster environments that appear to the end users as a single unified computing resource when in actuality they are a set of independent computer systems and network interconnecting them. To pull this ruse off, enterprises rely on dynamic languages such as Perl and Python to glue together these networks and ensure reliable and secure connections to their data stores.

This whitepaper covers details on integrating Perl and Python with commercial relational database management systems (RDBMS) including specifics on tools and resources to utilize. Part I covers Perl, Part II covers Python.

PART I: HOW TO INTEGRATE LARGE-SCALE DATABASES WITH PERL

It's all too tempting today to always look to the latest technologies as a way to solve problems, without looking at tried-and-true methods that have been working for years. There's nothing wrong with adopting new tools, of course, but there's a tendency to throw the technological baby out with the bathwater because of the perception that new equals better. Sometimes it does, but when working with mission-critical systems and data, it's quite likely that the mature solutions are going to do better by you.

What we have learned is that Perl is one of the best ways to tackle integration of large-scale databases. Perl, while no longer the hot new thing, is mature and still thriving. Perl 5 has a long history of successfully working with open-source and commercial RDBMSs such as MySQL, PostgreSQL, Oracle, Microsoft SQL Server, and many others.

Use the right tools

A common mistake is for developers to reinvent the wheel when working with databases. Don't! Use the standard DBI driver modules that come with Perl to connect to your database. They've been tested hard, probably by companies with even larger data sets and more traffic to and from the database.

Perl's DBD::* modules provide a standard database interface that defines methods, variables and conventions that is consistent. This means that working with databases is not only well-documented and tested, it also gives great flexibility down the road. You might build your application on MySQL for testing but deploy on PostgreSQL or Oracle. You might need to migrate away from SQL Server at some point. Perl's DBI module lets you avoid lock-in on the application side.

Another tip for using the right tools is to use package managers—rather than the Comprehensive Perl Archive Network (CPAN) module—to manage your Perl modules. There are package managers such as ActiveState's Perl Package Manager (PPM) available that offer a good way to manage binary modules without having to build from CPAN for updates.

Never trust the client

The customer is always right, but client data input should always be assumed to be wrong. Data can be malformed accidentally or maliciously. But either way, it has the potential to cause problems.

EXPERT TIPS AND TRICKS

Perl provides some excellent tools to sanitize external input data. Make sure that you're stripping "special" characters from input, avoid stored HTML and be careful where you're storing user-supplied data. Use of Perl's "taint" mode will also ensure data generated outside your program as tainted so it cannot accidentally be used as a file name or subprocess command.

"Don't trust data supplied by the browser" should be the foremost rule of thumb.

Your data is yours

You should be very conservative about data that's accepted as input, and even more conservative about data that is sent out. Make use of security features that are available in connecting to your database. Most databases can work with SSL or have other features to ensure that communication between an application server and a database server are encrypted. It's also a good idea to store data in an encrypted state should an attacker actually get so far as gaining access to your data store.

Legacy systems or applications may be constructed in such a way that a native encrypted connector is not possible. That's suboptimal, but not impossible to fix. Use a Secure Shell (SSH) tunnel between systems when SSL is not natively supported by the database connector.

Ensure that session data is encrypted. Any session exchanging personal data between your application and the user over a network should be encrypted, but also look to encrypting session state information when storing session data in a URL. The Crypt::* modules will provide the proper tools to do this and also look to the CGI::EncryptForm module.

Performance improvements

The first thing some programmers do is commit premature optimization. That is, worrying about getting the tightest code but failing to optimize the way the database is used. Many times, the performance bottleneck is the database—so figure out how to optimize its performance before worrying whether you've got the very best algorithms.

Naturally, you're going to have a beefy database server but put it to good use. Don't hit the database server unless it's absolutely necessary. Cache results so that you're not making multiple (and unnecessary) calls for data you've already fetched once.

But you need to get the data, so how can you avoid it? There are a couple of ways: You can use an intermediate local data store to cache data between the main RDBMS and your application, such as memcached or Berkeley DB. Another tip is to avoid sprinkling unnecessary SQL queries throughout your code. Use object-relational mapping (ORM) to convert data between incompatible type systems. Perl's DBIx::Class module can speak with all kinds of traditional RDBMSs to handle just about any type of work you're doing.

SUMMARY

No matter what RDBMS you're using, it goes well with Perl. These guidelines are a good starting point to ensure that your application running on a large-scale database is going to be successful. Take the advice in this whitepaper and you'll be well on the way to a successful deployment or revision.

PART II: HOW TO INTEGRATE LARGE-SCALE DATABASES WITH PYTHON

Python coupled with a commercial-grade enterprise RDBMS is often your best solution in cost, stability, and usability for lowering risks and ensuring enterprise levels of support for your community of users.

Python is a portable, platform-independent, general-purpose language that can perform the same tasks as the database-centric, proprietary fourth-generation language (4GL) tools supplied by database vendors. Like 4GL tools, Python lets you write programs that access, display and update information in a database with minimal effort. Unlike many 4GLs, Python also gives you a variety of other capabilities, such as parsing HTML, making socket connections and encrypting data.

Python Knows Large-Scale Databases

Your average enterprise databases (e.g. Sybase, IBM DB2, Oracle, or more) are all supported in varying degrees by a vibrant Python developer community with a good series of packages to ease integration issues. Importantly, most of these modules are well supported in turn by the enterprise database vendors. Support from both sides, community and vendor, is the key to happiness. A symbiotic, two-way relationship between Python communities and database vendors is a best practice.

Apply Plenty of Python Glue

The Python DB API was defined to encourage standardization between Python modules in their approach to how they access databases. The Python community has done an excellent job developing and applying this standardized way to approach relational databases and SQL databases. This level of API standardization is a sign of the maturity of the community and bodes well for enterprise looking to leverage Python.

For more information see:
http://www.python.org/dev/peps/pep-0249/

Many enterprises have already plunged in and begun using Python as their "glue" and have a head start in integrating Python with their enterprise databases due in large part to development of the following Python packages:

SQLAlchemy
When it comes to enterprise databases, SQLAlchemy is your friend. It is the Python Object Relational Mapper that gives application developers the style and flexibility of Object-Oriented Programming without compromising the power of raw SQL database access. In other words, SQLAlchemy helps to overcome the Object-Relational "impedance mismatch" the same way Java Data Objects (JDO) does. There's a large community of developers supporting it. SQLAlchemy provides a full suite of well-known enterprise-level persistence patterns, designed for efficient and high-performing database access, adapted into a simple and Pythonic domain language. It is utilized by many enterprises.

For more information see:
http://www.sqlalchemy.org/

cx_Oracle
cx_Oracle is the Python interface to Oracle. It is a good example of a module that is community-led project but has great support from Oracle as well.

For more information see:
http://pypi.python.org/pypi/cx_Oracle/5.0.4

Python-sybase
Python-sybase is the Sybase module for Python. It is not as well loved as other enterprise data stores when it comes to community support even though Sybase itself continues to be quite a popular choice in the financial

sector. The Python modules supporting it are not as well cared for and maintained as others.

For more information see:
http://python-sybase.sourceforge.net/download.html

PyDB2
Well supported by and loved by both the community and the vendor. For more information see:
https://github.com/ibmdb/python-ibmdb

Open Source Data Store Options

We would be remiss not to mention the two open source options that are now being widely adopted by enterprises: MySQL and PostgreSQL. Their Python data connectors: mysql-python and psycopg2 have seen rigorous stress testing both with enterprise deployments and on cloud infrastructure offerings such as Amazon's EC2 platform.

Oh, But You've Got Big Data!

NoSQL is hot. The compelling feature of NoSQL data sets is the volume. Think social media. Companies like Hadoop, MongoDB, and Cassandra are providing the foundation for massive repositories of data like Facebook, eBay and others.

But NoSQL is important for large enterprises as well. Under the hood of SAP and ERP systems, what you're dealing with are transactions. Business transactions. The volume of data in those systems might be very high, but they don't approach the volume of data needed to try to keep track of everything anyone on Twitter says about you. So, the projects that people are having to manage now are changing their nature, which is why the NoSQL large databases are really starting to be required within the enterprise.

Good news, the Python community has gotten fully behind supporting big data.

The House that Guido Built

Python's been integrated with enterprise class databases for many years. With newer, technologies like NoSQL it's maybe a little bit more mixed. There are examples already where Python's doing very interesting things on a very large scale, as exemplified by Google's embrace and adoption of Python as one of the key components of Google App Engine's stack. A lot of the interest in Python and development is being driven by Google, and is being donated back into the Open Source community – all of which has lead to a more robust, mature and stable Python. Guido van Rossum, the Dutch programmer best known as the author of Python, is now a Google employee.

For example, MapReduce is the patented software framework that Google brought into being to support distributed computing on large data sets on clusters of computers. It is basically is a NoSQL approach. Python developers already have access to Python module called mincemeat.py that is a Python implementation of the MapReduce distributed computing framework.

For more information see:
https://github.com/michaelfairley/mincemeatpy

Get Your Messaging Queue Here!

Messaging queuing is important functionality for enterprises where performance and speed are essential, especially in financial services. Companies that need to initiate stock trades quickly and do algorithmic training. They want to have the fastest message queuing possible to make trades instantly. There are two excellent Python wrappers are available – RabbitMQ & the newcomer zeromq.

RabbitMQ

RabbitMQ is a complete and highly reliable enterprise messaging system based on the emerging AMQP standard. To use RabbitMQ with Python you need amqplib.

For more information on RabbitMQ see:
https://www.rabbitmq.com/

zeromq

Python is available for zeromq. For more information see:
http://www.zeromq.org/

The Global Interpreter Lock (GIL) Issue

There is a perceived issue with Python arising from the Global Interpreter Lock (GIL). GIL has a global lock and only one process at a time can access memory. It can make things difficult when working with databases when you're making a lot of connections. With a large number of concurrent connections open, suddenly, GIL becomes the bottleneck.

But in many environments this is no issue at all. In fact, it is a distinct advantage. If you have a huge, full database against which you make long-running queries, you will not care about GIL.

The reason why GIL exists is that it provides an extremely high level of security in the concurrency sense. You're prevented from shooting yourself in the foot.

You Need Testing and Support Now!

It's common to set up test environments on MySQL and Python. Then you'll deploy on Oracle or other bigger enterprise class databases. In other words, Python is the language that you prototype in.

The key point when making a decision about what database you're going to use is what support do you have available. For the mission critical systems at large enterprises, large databases have good, strong vendor support. You can count on them.

You can also count on paying a lot.

If you want enterprise support for your Python database connector modules, come to ActiveState. We can provide the same reliable support that you'd expect from a large database vendor at a much lower cost.

For more information see:
http://www.activestate.com/support/commercial

Don't Fight It, Open Source Will Be Smuggled In Anyway

There's another reason you might consider ActiveState support and training. Open source comes in as a grass-roots movement. It starts with one engineer testing out features. When she realizes the advances, she suggests it to her team. This way open source solutions integrate into teams and organizations. This is a common pattern.

Stop Looking Dumb To Engineers

PyPI Index enables you to search for build information and availability of Python packages or modules in the Python community's build repositories. For Perl, there's the CPAN and ActiveState's PPM.

The difference with PyPM Index from all the other Python index pages is the build information. Other indexes just provide a way to find the source code so you can build it yourself. But the PyPM Index provides builds but also great information on cross-platform issues.

EXPERT TIPS AND TRICKS

SUMMARY

Python has you covered. From well-known and trusted relational databases used by enterprises around the world to fancy cutting-edge NoSQL ventures to high demand messaging queuing... whatever your environment, you can utilize Python to your advantage.

ON-DEMAND DYNAMIC LANGUAGE EXPERTISE

ActiveState's top Perl and Python experts have decades of experience developing, troubleshooting and deploying applications with dynamic languages. Our experts will advise you on ways to improve development efficiencies, minimize downtime, improve time-to-market, to get the best return on your Perl or Python investment.

Our Perl and Python experts will work with your team to provide:

- Best practices advice on module usage to save developer time
- Expert how-to guidance on database connectivity
- Customized builds of Perl or Python with the modules you need, on the operating systems you rely on
- Troubleshooting of code problems or leaks, including code audits, to ensure maximum application performance

Talk to our team of dynamic language experts and you'll see why Fortune 1000 companies, like Cisco, CA, Juniper Networks, Credit Suisse, Bank of America, The Boeing Company and NASA, trust ActiveState expertise.

www.activestate.com/support/commercial

ActiveState Software Inc.
sales@activestate.com

Phone: **+1.778.786.1100**
Fax: **+1.778.786.1133**

Toll-free in North America:
1.866.631.4581

ActiveState®

ABOUT ACTIVESTATE

ActiveState believes that enterprises gain a competitive advantage when they are able to quickly create, deploy and efficiently manage software solutions that immediately create business value, but they face many challenges that prevent them from doing so. The company is uniquely positioned to help address these challenges through our experience with enterprises, people and technology. ActiveState is proven for the enterprise: more than two million developers and 97 percent of Fortune 1000 companies use ActiveState's end-to-end solutions to develop, distribute, and manage their software applications written in Java, Perl, Python, Node.js, PHP, Tcl and other dynamic languages. Global customers like Cisco, CA, HP, Bank of America, Siemens and Lockheed Martin trust ActiveState to save time, save money, minimize risk, ensure compliance and reduce time to market.