**[Comparing MySQL and Postgres 9.0 Replication](http://blog.netoearth.com/html/201011/comparing-mysql-and-postgres-9-0-replication.htm" \o "Permanent Link to Comparing MySQL and Postgres 9.0 Replication)**

*On 2010年11月22日, in* [*database*](http://blog.netoearth.com/html/category/database)*, by netoearth*

By Robin Schumacher and Gary Carter, [www.enterprisedb.com](http://www.theserverside.com/feature/www.enterprisedb.com)

Replication is one of the most popular features used in RDBMS’s today. Replication is used for disaster recovery purposes (i.e. backup or warm stand-by servers), reporting systems where query activity is offloaded onto another machine to conserve resources on the transactional server, and scale-out architectures that use sharding or other methods to increase overall query performance and data throughput.

Replication is not restricted to only the major proprietary databases; open source databases such as MySQL and PostgreSQL also offer replication as a feature. While MySQL has offered built-in replication for a number of years, PostgreSQL replication used to be accomplished via community software that was an add-on to the core Postgres Server. That all changed with the release of version 9.0 of PostgreSQL, which now offers built-in streaming replication that is based on its proven write ahead log technology.

With the two most popular open source databases now providing built-in replication, questions are being asked about how they differ in their replication technologies. What follows is a brief overview of both MySQL and PostgreSQL replication, with a brief compare and contrast of the implementations being performed immediately afterwards.

**An Overview of MySQL Replication**

Asynchronous replication was introduced into Oracle’s MySQL with version 3.23 and today it remains the primary feature employed by many MySQL users to create scale-out architectures, standby servers, read-only data marts, and more. The various supported MySQL replication topologies include:

• Single master to one slave

• Single master to multiple slaves

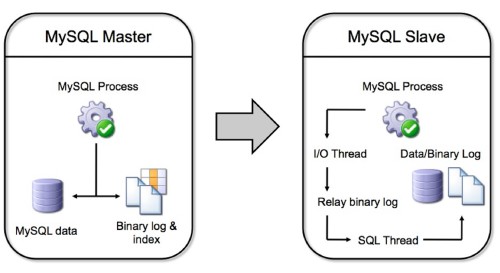
• Single master to one slave to one or more slaves

• Circular replication (A to B to C and back to A)

• Master to master

The major replication topology not currently supported in Oracle’s MySQL today is multi-source replication: having one or more master servers feed a single slave.

A graphical view of how MySQL replication functions can be represented as follows:



Object, data, and security operations run on the master are copied to the master server’s binary log. A user has the option of replicating an entire server, one or more databases, or just selected tables (although filtering by table is only done on the slave). The slave server obtains information from the master’s binary log over the network , copies the commands and/or data, and first applies them to the slave’s relay binary log. That log is then read by another process – the SQL thread – that applies the replicated operations/data to the slave database and its binary log.

Prior to release 5.1, MySQL replication was statement-based, meaning that the actual SQL commands were replicated from the master to one or more slaves. However, certain use cases did not lend themselves to statement-based replication (e.g. non-deterministic function calls) so in MySQL 5.1 row-based replication was introduced. A user now has the option of setting a configuration parameter to use either statement or row-based replication.

The primary bottleneck for busy MySQL replication configurations is the single-threaded nature of its design: replication operations are not multi-threaded at the moment, although MySQL has declared it is coming in a future release. This limitation can cause some slave servers under heavy load to get far behind the master in regards to applying binary log information.

Setting up MySQL replication is a fairly painless process. Although various setup procedures exist, in general, the following is a basic outline of how it is done:

• The master and slave servers are identified

• The master server is modified to include a replication security account

• The master server’s MySQL configuration file is modified to enable binary logging. A few other parameters are included as well (e.g. a unique server ID, type of replication such as statement or row-based, etc.)

• The slave server’s MySQL configuration file is modified to include a unique server ID

• The master server is restarted

• The master server’s log file position is recorded

• The master’s data is copied to the slave to initially seed the slave server. This can be done via a cold backup/restore, using the mysqldump utility, locking the master tables and doing a file copy, etc.

• The slave server is restarted

• The MySQL CHANGE MASTER command is executed on the slave server to set the master host name on the slave server as well as other parameters such as the master account username and password, the log file name, and beginning log file position

Once set up, MySQL replication is quite reliable. Being asynchronous in nature, however, there are use cases that could result in data loss between a master and slave. To help combat these situations, MySQL 5.5 introduced semi-synchronous replication where a pending transaction is sent from a master to a slave, but not committed on the slave; it merely ‘lands’ safely on the slave to be run as soon as possible. Once the master is notified that the transaction is safely recorded on the slave, then the transaction is committed on the master.

In terms of MySQL replication limitations and missing features, besides the already mentioned single threaded nature of the implementation and the inability to perform multi-source replication, other wish-list items include a full synchronous option, conflict detection and resolution, time-delayed replication, changing the binary log to a storage engine, better replication filtering on the master, global statement ID’s, and graphical tools to manage replication functions.

There are third-party providers of MySQL replication solutions that overcome some of the current shortcomings in what is provided out-of-the-box with MySQL. One example is Continuent’s Tungsten product.

For more information about Oracle’s MySQL replication, see: <http://dev.mysql.com/doc/refman/5.5/en/replication.html>.

**An Overview of PostgreSQL Replication**

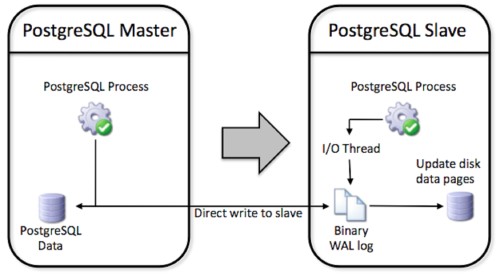
PostgreSQL replication is based on a mature and long used technology called write ahead log (WAL) archiving. WAL technology has been in use since version 7.1 and has been used in features such as backup and restore and warm standby servers (i.e. slave servers offline kept in synch with the master to step in during crash recovery) for high availability.

PostgreSQL 9.0 introduced significant enhancements producing extremely fast WAL processing that results in near real-time replication and hot standby capabilities for slave servers. The supported PostgreSQL replication topologies include:

• Single master to one slave

• Single master to multiple slaves

A graphical view of how PostgreSQL replication functions can be represented as follows:



All objects and data (including schema) and security operation executed on the master are written to the WAL log directly on the slave machine for safety (avoiding complete data loss in the event of a catastrophic master failure). WAL also ensures that no transaction is committed on the master until a successful write of the WAL log has occurred. No filtering is currently possible (although replication with filtering is possible with the xDB Replication Server from EnterpriseDB) so a complete copy of the master is replicated on the slave.

The slave then applies the WAL log by directly rewriting the raw table data on disk, which is much faster than statement based replication. It is also safer since statements such as:

INSERT INTO table (column) VALUES (SELECT function());

may have unexpected and inconsistent results if the function returns different values on different servers – perhaps because it involves a generated timestamp or uuid.

The primary limitations of PostgreSQL replication are topology based. It cannot currently do cascading replication or filter tables by rows for replication. Again, these are capabilities available in a separate replication solution from EnterpriseDB called xDB Replication Server.

Setting up PostgreSQL replication is very straightforward. WAL logging is always enabled with minimal configuration needed by the user to utilize replication. The basic process to get replication going is:

• The master and slave servers are identified

• The postgresql.conf file on the master is edited to turn on streaming replication

• The pg\_hba.conf file on the master is edited in order to let the slave connect

• The recovery.conf and postgresql.conf files on the slave are edited to start up replication and hot standby

• The master is shutdown and the data files are copied to the slave

• The slave is started first

• The master is started

The secret sauce to PostgreSQL 9.0’s extremely reliable WAL based replication is a set of enhancements to efficiently stream very small WAL segments compared to earlier versions. Like MySQL there are cases where data loss could occur – however, depending on how you configure the system, your hardware architecture, and load, its possible the data loss could be as small as a single transaction.

PostgreSQL does not currently have native synchronous replication. However, there are multiple replication options available from other community and third-party software providers. PostgreSQL offers multiple solutions for multi-master replication, including solutions based on two phase commit. Offerings include [Bucardo](http://en.wikipedia.org/wiki/Bucardo_%28PostgreSQL%29), [rubyrep](http://www.rubyrep.org/), [PgPool and PgPool-II](http://pgpool.projects.postgresql.org/) and Tungsten Replicator as well as some proprietary solutions. Another promising approach, implementing eager (synchronous) replication is [Postgres-R](http://www.postgres-r.org/), however it is still in development. Yet another project implementing synchronous replication is [Postgres-XC](http://sourceforge.net/projects/postgres-xc/), which is a shared-nothing, transactional scale-out solution that is still under development.

For more information on PostgreSQL replication see:

PostgreSQL Documentation: <http://www.enterprisedb.com/docs/en/9.0/pg/high-availability.html>

Bucardo: <http://bucardo.org/wiki/Bucardo>

PgPool-II: <http://pgpool.projects.postgresql.org/>

Tungsten Replication: <http://www.continuent.com/community/tungsten-replicator>

**A Brief Compare and Contrast of MySQL and PostgreSQL Replication**

Those wanting to use an open source database for a particular application project that requires replication have two good choices in MySQL and PostgreSQL. But, the question naturally arises, which should be used? Is one just as good as the other?

As demonstrated above, there are both feature and functional differences between how MySQL and PostgreSQL implement replication. However, for many general application use cases, either MySQL or PostgreSQL replication will serve just fine; technically speaking, from a functional and performance perspective, it won’t matter which solution is chosen. That said, there still are some considerations to keep in mind in deciding between the different offerings. Some of these include the following:

• Oracle’s MySQL offers both statement and row-based replication, whereas PostgreSQL only uses the latter based on write ahead log information. There are pro’s and con’s to using statement-based replication, which MySQL has documented here: <http://dev.mysql.com/doc/refman/5.5/en/replication-sbr-rbr.html>. It is generally acknowledged that row or WAL-based replication is the safest and most reliable form of replication. It does, however, result in larger log files for MySQL than the statement-based option does.

• MySQL currently supports more replication topologies than PostgreSQL (e.g. ring, etc.). However PostgreSQL does have a number of community supported replication offerings that help close this gap (e.g. Bucardo’s master-to-master solution).

• In regard to data loss, MySQL 5.5 offers the semi-synchronous option, which helps minimize the risk of master-slave synchronization problems due to a master server going down. For PostgreSQL, a full synchronous replication option is in development and scheduled for release sometime in 2011.

• As to replication filtering, MySQL provides filtering on the slave server, whereas with PostgreSQL, no filtering is available; in other worlds, the entire database from the master is replicated to the slave. With MySQL, all the information is sent, but then options exist to selectively apply the replicated events on the slave. However, as the MySQL binary log is not used for crash recovery purposes in the same way as PostgreSQL’s WAL is, a user can configure a MySQL master so only certain databases are logged and, in that sense, a filter for the master server is available.

• Both MySQL and PostgreSQL replication are single-threaded at the current time.

• With respect to monitoring replication, MySQL provides a number of SHOW commands to understand the state of replication between a master and slave. To date, PostgreSQL offers functions to compute the differences in log positions between the master and slave servers, but that is all that is currently provided in 9.0.

• For failover and load balancing, the PostgreSQL community provides pgPool, which is middleware that provides connection pooling, load balancing, failover, and more between replicated servers. MySQL 5.5 supports connection pooling in the Enterprise edition, but failover and load balancing must be handled via a third-party product or custom development.

**Conclusions**

As was previously stated, for many application use cases, both Oracle’s MySQL and PostgreSQL replication will be an equally good choice. The best way to determine which is right for you is to download both and put each through a comprehensive evaluation.

You can download Oracle’s MySQL at <http://www.mysql.com/downloads/>, while both community and EnterpriseDB’s offerings of PostgreSQL can be found at: <http://www.enterprisedb.com/products/download.do>.

By Robin Schumacher and Gary Carter, [www.enterprisedb.com](http://www.theserverside.com/feature/www.enterprisedb.com)