MySQL vs PostgreSQL

From WikiVS, the open comparison website

MySQL vs PostgreSQL is a decision many must make when approaching open-source relational database management systems. Both are time-proven solutions that compete strongly with proprietary database software. MySQL has long been assumed to be the faster but less full-featured of the two database systems, while PostgreSQL was assumed to be a more densely featured database system often described as an open-source version of Oracle. MySQL has been popular among various software projects because of its speed and ease of use, while PostgreSQL has had a close following from developers who come from an Oracle or SQL Server background.

MySQL		PostgreSQL
	VS	
http://www.mysql.com/		http://postgresql.org

These assumptions, however, are mostly outdated and incorrect. MySQL has come a long way in adding advanced functionality while PostgreSQL dramatically improved its speed within the last few major releases. Many, however, are unaware of the convergence and still hold on to stereotypes based on MySQL 4.1 and PostgreSQL 7.4. The current versions are MySQL 5.7 and PostgreSQL 9.4.

Contents

Architecture

PostgreSQL is a unified database server with a single storage engine. MySQL has two layers, an upper SQL layer and a set of storage engines. While MySQL 5.6 natively supports 9 storage engines (http://dev.mysql.com/doc/refman/5.6/en/storage-engines.html), the most popular choices are InnoDB, MyISAM and NDB Cluster. Storage engines differ in both functionality and performance characteristics, so it is important to clarify which is being used in any comparisons.

Performance

Database systems can be optimized according to the environment they run in. Thus, it is very difficult to give an accurate comparison in performance without paying attention to configuration and environment. PostgreSQL and MySQL both employ various technologies to improve performance.

Beginnings

MySQL began development with a focus on speed while PostgreSQL began development with a focus on features and standards. Thus, MySQL was often regarded as the faster of the two. The default configuration for both is tuned to run on small systems, and it's common for people performing benchmark tests to either not change the defaults, or properly tune only the one they are most familiar with. Either action will usually give misleading results. Furthermore, both DBMS's will do better in benchmarks related to their original strengths (i.e. MySQL fast in simple operations, PostgreSQL more reliable and faster in complex operations)

Raw Speed

PostgreSQL

PostgreSQL provides significant performance features

- efficient executor for both static SQL or parameterised SQL
- advanced cost-based optimizer, with many plan choices and adaptive statistics collection
- indexing: partial, functional, multiple-index-combining, index-only scans, 5 different kinds of index (btree, hash, gist, gin, spgist and brin)
- TOAST (http://www.postgresql.org/docs/current/static/storage-toast.html) data compression
- improved cache management in versions 8.1 and 8.2
- huge scalability on write intensive workloads from 8.2+
- asynchronous commit ("MyISAM for Postgres")
- asynchronous Replication built-in from 9.0+
- synchronous Replication built-in from 9.1

The 8.x releases have added more than 75 new discrete performance features. These have been added as a result of a multi-year project to improve performance by steadily identifying and removing key bottlenecks in scalability, as well as adding low-level tuning and architectural features. [1] (http://www.2ndquadrant.com/download/Postgres_Performance_Update83.pdf) shows feature set added for the 8.3 release, for example.

PostgreSQL can compress and decompress its data on the fly with a fast compression scheme to fit more data in an allotted disk space. The advantage of compressed data, besides saving disk space, is that reading data takes less IO, resulting in faster data reads.

PostgreSQL supports one storage engine, with tight integration between that storage engine and the rest of the database. Options like asynchronous commit can be set on a per-transaction, per-user or whole system basis, allowing different transaction types to co-exist efficiently without the need to select storage engine types once for each table ahead of time.

By default, PostgreSQL comes tuned to run on a shared server, so has low performance settings. When running on a dedicated server performance can be improved by changes to a few key parameters.

MySQL: All Storage Engines

Core features that MySQL has independent of storage engines:

- A broad subset of ANSI SQL 99, as well as extensions
- A query cache to bypass the SQL parser and return previous results if the data in a table has not changed.
- Table Partitioning using LIST, HASH, RANGE and SET.
- Built in replication via either statements or row-change events.
- Network protocol compression.
- SSL support
- Triggers
- Cursors
- Updateable Views
- Stored Procedures
- Unicode Support

MySQL:MyISAM

MyISAM was the previous default storage engine for MySQL (before MySQL 5.6). It is notably faster than InnoDB in the cases of table-scans, index-scans, insert-only and some single-threaded operations. It is likely to perform better than PostgreSQL in these same scenarios.

MyISAM tables suffer from table-level locking, and do not support ACID features such as data durability, crash recovery, transactions or foreign keys. Previously it has been claimed to perform better in read-only or read-heavy operations, but this is no longer necessarily the case.

System tables currently use the MyISAM storage engine, but Oracle has commented that work is being done to convert to InnoDB [2] (http://www.tocker.ca/2014/05/22/mysql-soon-to-store-system-tables-in-innodb.html). This has been used to criticise it for the potential of losing system table information, but some claim that this doesn't happen "in practice" as it is unlikely that such generally infrequently updated tables will be in the process of being written to at the time of a crash.

MySQL:InnoDB

InnoDB is an ACID (http://en.wikipedia.org/wiki/ACID) compliant, transactional storage engine using MVCC (http://en.wikipedia.org/wiki/Multiversion_concurrency_control) technology. It's the normal choice for most modern applications using MySQL.

The InnoDB storage engine stores the data with the primary key, so primary key lookups are fast. Good choice of primary key for physical optimisation can be very useful; in cases where it's undesirable or where the desired primary key produces poor physical performance a simple integer can be used. An internal integer primary key is the default if no primary key or unique column is present.

The InnoDB engine automatically generates hash index entries when processing SELECTs. This feature can be turned off if necessary; some workloads perform better without it.

The InnoDB engine has an insert buffer that caches updates to secondary index entries and applies them in the background. This can significantly speed up inserts, reducing the number of physical writes required by combining many updates. If a secondary index page has outstanding updates when it is needed for a query the updates will be merged first. As of version 5.5 the insert buffer is also used as a buffer for other types of writes, improving the performance of UPDATE queries as well.

With the InnoDB installed via Plugin MySQL 5.1 supports on-the-fly compression of InnoDB tables.

> Beginning with this release of the InnoDB Plugin, you can use the attributes ROW_FORMAT=COMPRESSED or KEY_BLOCK_SIZE in the CREATE TABLE and ALTER TABLE commands to request InnoDB to compress each page to 1K, 2K, 4K, 8K, or 16K bytes.
>
> — InnoBASE OY , 3.2. Specifying Compression (http://www.innodb.com/doc/innodb_plugin-1.0/innodb-compression.html#innodb-compression-usage)

Despite major changes in ownership in recent years, InnoDB performance has received continuing development attention.

With InnoDB, the secondary key lookup is slow, because the leafs of its secondary indices are the primary keys. So it has to do 2 lookups when using a secondary index: one for the secondary index itself, and one for the primary key.

MySQL:NDB Cluster

NDB is a high performance, highly available, (mostly) in-memory storage engine. Non-indexed attributes may be stored on disk. Data and logs are automatically flushed to disk periodically to mitigate data loss in the event of a cluster failure. NDB is used heavily in Telecommunications applications where uptime and real time performance is critical. NDB transparently organizes records into fragments which are distributed evenly to all nodes in the cluster. NDB uses synchronous replication internally to ensure records are committed to at all nodes within a logical node group before returning a COMMIT. This two phase commit gives NDB the ability to support sub-second failover. Further to this NDB also supports automatic node recovery. NDB supports the online introduction of additional nodes to the cluster without impacting availability of the application.
Significant Limitation: The distributed nature of the tables makes NDB perform poorly for complex JOIN operation as compared to traditional storage engines. This has been resolved in version ndb-7.2.1 and later with Adaptive Query Localization (http://www.clusterdb.com/mysql-cluster/70x-faster-joins-with-aql-in-mysql-cluster-7-2-dmr) (distributed pushed-down join, formerly SPJ)
PostgreSQL: PostgreSQL has no comparable solution.

Multi-Core Scalability

Historically MySQL has focused more on scale-out than scaleup and PostgreSQL was often considered to scale better with large numbers of cores or uncommonly high concurrency levels.

With the increased availability of multi-core systems MySQL has since improved this in recent versions, with MySQL 5.6 scaling to 48 core machines (http://www.computerworld.com/s/article/9236511/MySQL_5.6_tackles_NoSQL_competitors). Some of these improvements are the result of contributors from Google (http://mysqlha.blogspot.com/2008/09/more-patches-than-we-know-what-to-do.html), Percona and Facebook.

Earlier versions of MySQL (particularly prior to 5.5) are known to not scale well on multi-core machines. For this reason, it is particularly important to pay attention to the version of software used in older benchmarks.

IO Device Scalability

PostgreSQL supports a full fledged asynchronous API (http://www.postgresql.org/docs/8.3/static/libpq-async.html) for use by client applications. It is reported to increase performance by up to 40% in some cases (http://oldmoe.blogspot.com/2008/07/faster-io-for-ruby-with-postgres.html). MySQL supports Native Asynchronous I/O for Linux with libaio userspace library since version 5.5 (AIO in InnoDB (http://dev.mysql.com/doc/innodb/1.1/en/innodb-performance-aio-linux.html)) and simulated Asynchronous I/O via IO threads in versions prior.

Contributions by Percona, Google and Facebook allow MySQL to scale better on modern IO devices, which often require operations to be performed in parallel [3] (http://dev.mysql.com/doc/refman/5.6/en/license-percona-io-threads-patch.html). Recent versions of MySQL are also able to make better use of SSDs with MySQL 5.6 allowing control over 'neighbor flushing' and 5.7 supporting Fusion-IO atomic writes.

COUNT(*)

Both InnoDB and PostgreSQL implement MVCC (http://en.wikipedia.org/wiki/Multiversion_concurrency_control), which means that the number of rows visible in a COUNT(*) will be dependent on the MVCC visibility context of a current transaction. By contrast, MyISAM is able to perform an almost instant COUNT(*) for queries that count all rows in a table without a where clause. With a WHERE clause, MyISAM also requires scanning indexes and/or rows.

Because InnoDB indexes contain all MVCC versions, COUNT(*) can be generated by performing an index-only scan operation.

Starting with PostgreSQL 9.2, index-only scans are also supported via the visibility map feature to determine whether a row is visible to the current transaction rather than visiting the page. This means dramatically faster COUNT(*) results. PostgreSQL 9.2 index-only scans (http://rhaas.blogspot.com/2011/10/index-only-scans-weve-got-em.html) PostgreSQL Slow Count() Workaround (http://omega-glory.net/2007/12/12/postgresql-count-workaround/) InnoDB COUNT(*) (http://www.mysqlperformanceblog.com/2006/12/01/count-for-innodb-tables/).

Transaction System

PostgreSQL does not implement UNDO logging, and in the case of UPDATE statements each modification will result in a new version of the row to be written to the tablespace. Once rows are no longer required for older transactions (MVCC) they will be cleaned up by autovacuum.

With InnoDB, UPDATE statements modify rows in place and relocate the earlier version of the row to UNDO space. The exception to this is overflow pages used by large text/blob columns, which will be written as new.

In the case that transactions are short and InnoDB has a sufficiently large buffer pool, many of the UNDO pages will be modified in memory and freed before they need to be written to the tablespace. This can result in a significant IO reduction, as well as simplified house keeping against fragmentation. The downside is that InnoDB can be much slower at rolling back changes.

PostgreSQL 8.3 also introduced Heap-Only Tuples (HOT) as a way to reduce cleanup required for updates to non-indexed columns [4] (http://www.postgresql.org/message-id/200710202249.19KMn0u22815@momjian.us).

Both InnoDB and PostgreSQL have support for group commit, and will efficiently batch changes to be written to the REDO log and WAL respectively. PostgreSQL also offers the ability to specify `commit_delay` and `commit_siblings` to improve batching.

Connection Scalability and Thread Pool

In MySQL each new connection is an OS thread, of which the server maintains a small thread cache by default to reduce creation cost[5] (http://dev.mysql.com/doc/refman/5.6/en/server-system-variables.html#sysvar_thread_cache_size). A thread pool plugin is available as a commercial extension as part of MySQL Enterprise[6] (http://dev.mysql.com/doc/refman/5.6/en/thread-pool-plugin.html).

In PostgreSQL each new connection is an OS process, which will consume approximately 10MB of RAM each[7] (http://hans.io/blog/2014/02/19/postgresql_connection/). In practice many PostgreSQL installations will use PGBouncer as a connection pooler in front of the database to reduce this impact.

Further enhancements to connection scalability will be forthcoming in MySQL 5.7[8] (http://dev.mysql.com/worklog/task/?id=6606).

Benchmarks

When considering any benchmarks it is important to evaluate the workload being tested and how it relates to your application. For example, some trivial micro-benchmarks will test the speed of creating thousands of tables in a loop, yet few applications will have a requirement for this. Industry standard benchmarks tend to try and simulate a hypothetical situation such as entering and delivering orders, recording payments, checking the status of orders, and monitoring the level of stock at the warehouses.[9] (http://www.tpc.org/tpcc/default.asp)

Both MySQL and PostgreSQL remain under heavy development, so comparisons can easily favor one over the other if a development version of one is used in comparison to the current stable version of the other. Worse still, many older benchmarks can be used for comparison long after the versions involved are out of date, for example, tweakers.net (http://tweakers.net/reviews/657/6) and jamonation (http://jamonation.com/node/734) tests that precede many improvements that were implemented in MySQL 5.0 and the subsequent MySQL and Oracle releases (http://dimitrik.free.fr/db_STRESS_MySQL_540_and_others_Apr2009.html#note_5443).

Benchmarking in a best-to-best comparison is also difficult, since few benchmark authors will have equal ability to tune and configure both databases. Both MySQL and PostgreSQL have conservative default configuration, and require changes to perform optimally.

A 2009 benchmark (http://www.randombugs.com/linux/mysql-postgresql-benchmarks.html) for MySQL 5.0.51 and MySQL 5.1.30 with InnoDB 1.0.3 (MySQL 5.4 contains the patches from InnoDB 1.0.3) compared with PostgreSQL 8.3.7 suggested that MySQL and PostgreSQL were almost equal in terms of scalability, by at least one standard of measure.

While benchmarks can be an indicator of performance, MySQL performance in many subquery / join statements is behind PostgreSQL due to the inferior Query Optimizer (a much wider benchmark would show PostgreSQL as a clear winner in this case).. Recent enhancements (http://blogs.oracle.com/MySQL/entry/more_early_access_features_in) to the MySQL optimizer provide large gains in the performance of subquery / join statements.

ACID Compliance

ACID (http://en.wikipedia.org/wiki/ACID) stands for Atomicity, Consistency, Isolation and Durability and is a common set of properties which database management systems strive to implement.

PostgreSQL provides full ACID compliance, and is widely acknowledged as having more rigorous defaults in its approach to robustness and data integrity.

The InnoDB engine is fully ACID compliant, but fails the standard definition of consistency when using a combination of InnoDB, foreign keys with cascading actions and triggers. This is the result of triggers being implemented at MySQL's SQL layer and foreign keys being implemented at the InnoDB Storage Engine level.

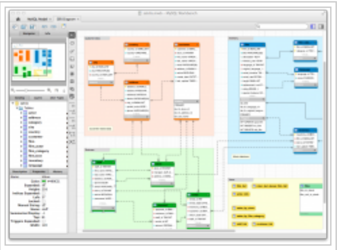
The NDB Cluster storage engine also offers ACID compliance using the definition of durability to mean that changes are guaranteed to be on more than one node, but may still be in memory.

Features

PostgreSQL and MySQL both have an impressive array of features that increase data integrity, functionality, and performance. The features included in a database may help improve performance, ease of use, functionality, or stability.

Graphical Tools

MySQL Workbench is the official GUI tool for MySQL, and ships as a separate download from the MySQL Developer Zone (http://dev.mysql.com/downloads/workbench/6.1.html). It supports features such as Visual Database Design (schema design with ER diagrams), a SQL Editor and a Performance Dashboard.


MySQL Workbench Visual Database Design running on Mac OS X

PostgreSQL can be administered via pgAdmin (http://www.pgadmin.org/), or a number of community developed tools (https://wiki.postgresql.org/wiki/Community_Guide_to_PostgreSQL_GUI_Tools).

Ease of use

PostgreSQL supporters claim that MySQL has more "gotchas"[10] (http://sql-info.de/mysql/gotchas.html) than PostgreSQL[11] (http://sql-info.de/postgresql/postgres-gotchas.html), due to its historical deviation from the SQL standard, and its various functional limitations which may not seem intuitively obvious to a new user. To use an example of a VARCHAR column being truncated from MySQL 5.5, with the insert still occurring:

```
mysql> CREATE TABLE t1 (a VARCHAR(2));
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO t1 VALUES ('too big');
Query OK, 1 row affected, 1 warning (0.01 sec)

mysql> SHOW WARNINGS;
+---------+------+------------------------------------------+
| Level   | Code | Message                                  |
+---------+------+------------------------------------------+
| Warning | 1265 | Data truncated for column 'a' at row 1 |
+---------+------+------------------------------------------+
1 row in set (0.00 sec)
```

On the other hand, PostgreSQL will not allow data to be inserted if it is too long:

```
psql=> CREATE TABLE t1 (a VARCHAR(2));
CREATE TABLE
psql=> INSERT INTO t1 VALUES ('too big');
ERROR:  value too long for type character varying(2)
```

MySQL has stated that they want to "work toward compliance with the SQL standard, but without sacrificing speed or reliability" [12] (http://dev.mysql.com/doc/refman/5.6/en/compatibility.html). In order to provide backwards compatibility, MySQL has introduced a series of SQL modes that can be turned on/off in order to behave more consistently with SQL standards. MySQL 5.6 was the first version of MySQL to enable the `STRICT` mode by default in new configurations, and MySQL 5.7 will offer `STRICT` as a compiled default:

```
mysql> CREATE TABLE t1 (a VARCHAR(2));
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO t1 VALUES ('too big');
ERROR 1406 (22001): Data too long for column 'a' at row 1
```

Insert Ignore / Replace

MySQL supports the statements, '`INSERT IGNORE`', which inserts if a row doesn't exist, and '`REPLACE`', which replaces the current row.

PostgreSQL below 9.5 supports neither of these statements and suggests using stored procedures to get around the lack of these statements. However, there are major shortcomings:

> it can only insert a single value at a time. This is a major performance limitation, and also suffers concurrency issues. INSERT IGNORE and REPLACE handle multi-valued inserted much more gracefully.
>
> — Robin Johnson , PostgreSQL Vs. MySQL for INSERT IGNORE + REPLACE - stored procedures, savepoints and beyond (http://robbat2.livejournal.com/214267.html)

When appropriate, PostgreSQL can also use `RULE`s to produce any of these behaviors on a normal `INSERT` or `UPDATE`; this can also be used on a view.

Starting with PostgreSQL 9.5 there exists a `INSERT ... ON CONFLICT DO (NOTHING | UPDATE)...` SQL instruction [13] (http://www.postgresql.org/docs/devel/static/sql-insert.html), quite similar to MySQL's `INSERT ... ON DUPLICATE UPDATE` one.

Furthermore PostgreSQL intends to support also the MERGE clause in a future version [14] (http://archives.postgresql.org/pgsql-hackers/2010-08/msg00299.php), which follows the SQL:2008 standard. This clause approaches the same as MySQL's non-standard 'INSERT IGNORE','REPLACE' and 'INSERT ... ON DUPLICATE UPDATE' statements but with more granularity.

Constraints

Both PostgreSQL and MySQL support Not-Null, Unique, and Primary Key constraints. Foreign Key constraints are supported by PostgreSQL and by MySQL's InnoDB and NDB storage engines. However MySQL silently ignores the CHECK constraint (http://www.dbforums.com/showthread.php?t=1633783) which PostgreSQL has supported for a long time.

> InnoDB and NDB tables support checking of foreign key constraints.... For other storage engines, MySQL Server parses and ignores the FOREIGN KEY and REFERENCES syntax in CREATE TABLE statements. The CHECK clause is parsed but ignored by all storage engines.
>
> —MySQL AB , MySQL 5.6 Reference Manual :: 13.1.17 CREATE TABLE Syntax (http://dev.mysql.com/doc/refman/5.6/en/create-table.html)

For MySQL engines not supporting foreign key constraints or to implement constraints between tables in differing engines, triggers (http://forge.mysql.com/wiki/ForeignKeySupport#Appendix_A:_Triggers_implementing_foreign_key_constraints) can be used to check the constraint, although this is not a guarantee as it's prone to race conditions. [citation needed]

Also, the support of ForeignKeys in MySQL is not yet complete, as cascade actions are not considered as SQL statements and thus do not fire triggers etc. (they only cascade further)

PostgreSQL also supports deferrable constraints as specified in the SQL standard, allowing individual constraints to be temporarily violated locally inside a transaction, with validation deferred until commit time. Transaction isolation and atomicity (ACID) guarantees that the constraint-violating data is never visible to other users. Currently only Unique, Primary Key and Foreign Key constraints are deferrable, but deferrable Not-Null and Check constraints can be emulated using deferrable constraint triggers. MySQL does not support deferrable constraints.

Default Values

PostgreSQL allows any function (whether marked as IMMUTABLE, STABLE or VOLATILE) to be used as the default value for a column. Currently, NOW() is the only function that can be used as a default value in a MySQL table. Before MySQL 5.6.5, it could only be applied to one column per table, on TIMESTAMP columns only. Starting from 5.6.5 (https://dev.mysql.com/doc/refman/5.6/en/timestamp-initialization.html), it can be applied to several TIMESTAMP or DATETIME columns.

Stored Procedures

MySQL supports stored procedures, per se; PostgreSQL supports stored functions (http://en.wikipedia.org/wiki/Stored_procedures#Comparison_with_functions), which are in practice very similar.

The first query language for PostgreSQL, PL/pgSQL, is similar to Oracle's PL/SQL. PostgreSQL supports SQL:2003 (http://en.wikipedia.org/wiki/SQL2003) PSM stored procedures as well as many other general purpose programming languages such as Perl (PL/Perl), Python (PL/Python), TCL (PL/Tcl), Java (PL/Java), JavaScript (PL/V8) and C (PL/C).

> MySQL follows the SQL:2003 syntax for stored routines, which is also used by IBM's DB2.
>
> —MySQL AB , MySQL 5.1 Reference Manual :: 18 Stored Procedures and Functions (http://dev.mysql.com/doc/refman/5.1/en/stored-procedures.html)

Via the plugin interface MySQL supports external language (http://forge.mysql.com/wiki/ProjectPage_External_Language_Stored_Procedures) stored procedures in Java, Perl, XML-RPC with more language plugins in the works.

Triggers

Both PostgreSQL and MySQL support triggers. A PostgreSQL trigger can execute ANY user-defined function from any of its procedural languages, not just PL/pgsql.

> MySQL triggers are activated by SQL statements only. They are not activated by changes in tables made by APIs that do not transmit SQL statements to the MySQL Server; in particular, they are not activated by updates made using the NDB API.
>
> —MySQL AB , MySQL 5.1 Reference Manual :: 19 Triggers (http://dev.mysql.com/doc/refman/5.1/en/triggers.html)

MySQL triggers are also not activated by cascading updates and deletes even when caused by a SQL statement (this is against the standard), since those are a feature of the InnoDB engine rather than the database as a whole. In these cases, MySQL silently ignores the triggers without issuing a warning (i.e. in other dbms's you must be extra careful to not pop triggers accidentally, in MySQL you must be extra careful that many of your statements will not activate the triggers, such as cascades, etc.).

PostgreSQL also supports "rules," which allow operating on the query syntax tree, and can do some operations more simply that are traditionally done by triggers. However, the rule system is on the way out in favour of (more powerful) triggers.

Syntax for definition of triggers in PostgreSQL isn't as straightforward as in MySQL. PostgreSQL requires separate definition of a function with specific data type returned (this is the general behaviour of PostgreSQL = more strict vs MySQL = less strict. The syntax in itself is as straightforward but the implementation is stricter and much more powerful (any user function), thus a bit harder to learn). On the upside, PostgreSQL supports multiple actions per trigger using OR (e.g. BEFORE INSERT OR UPDATE).

Prior to MySQL 5.7, MySQL does NOT support multiple triggers of the same type (i.e. maximum one ON UPDATE BEFORE and one ON UPDATE AFTER trigger) on the same table, whereas PostgreSQL does (If multiple triggers of the same kind are defined for the same event, they will be fired in alphabetical order by name. > from the manual).

PostgreSQL supports deferrable triggers that fire at transaction commit time (primarily useful for implementing deferred constraint checks). MySQL does NOT support deferrable triggers or deferrable constraint checking.

PostgreSQL, as of 9.0, supports triggers that fire only before or after updates on specific columns (i.e., BEFORE/AFTER UPDATE OF column_name[,...]), or when a user defined condition holds (WHEN condition). MySQL does not directly support either of these features, but they can be emulated in the user-defined trigger code.

PostgreSQL, as of 9.1, supports TRIGGERS on views, MySQL does not.

PostgreSQL, as of 9.3, supports event triggers which can fire on most types of command (CREATE TABLE, DROP INDEX, ALTER TYPE etc.), except for ones affecting the database globally, such as roles and tablespaces.

Replication and High Availability

Replication is a database management system's ability to duplicate its stored data for the purposes of backup safety and is one way to prevent database downtime. PostgreSQL and MySQL both support replication:

PostgreSQL

PostgreSQL has built-in asynchronous replication as of 9.0, consisting of streaming replication [15] (http://www.postgresql.org/docs/9.0/static/warm-standby.html#STREAMING-REPLICATION) and hot standby [16] (http://www.postgresql.org/docs/9.0/static/hot-standby.html). PostgreSQL streams the write-ahead log data which replays activity on the database guaranteeing identical results. MySQL's older form of statement based replication could introduce slave inconsistency when executing non-deterministic statements. MySQL now warns when using potentially unsafe statements, switching to a ROW image based log format.

There are several packages (http://edoceo.com/liber/db-postgresql-replication) that also provide replication in PostgreSQL:

- PGCluster (http://pgcluster.projects.postgresql.org/)
- Slony-I (http://slony.info/)
- DBBalancer (http://sourceforge.net/projects/dbbalancer)
- pgpool (http://pgpool.projects.postgresql.org/)
- PostgreSQL table comparator (http://pg-comparator.projects.postgresql.org/)
- SkyTools (https://developer.skype.com/SkypeGarage/DbProjects/SkyTools)
- Sequoia (http://sequoia.continuent.org/HomePage)
- Bucardo (http://bucardo.org/)
- Mammoth Replicator (http://www.commandprompt.com/products/mammothreplicator/)
- Cybercluster (http://www.postgresql.at/english/pr_cybercluster_e.html)
- GridSQL (shared-nothing) (http://www.enterprisedb.com/community/projects/gridsql.do)
- rubyrep (asynchronous, master-master) (http://www.rubyrep.org/)
- bondreplicate (asynchronous, master-master) (http://www.treshna.com/replicate/)

It is a common misconception that these "third-party packages" are somehow less well integrated. Slony, for example, was designed and built by Jan Wieck, a PostgreSQL core team member, and has a number of other members of the PostgreSQL community involved in its ongoing design and maintenance. However, Slony is considerably slower and uses more resources than built-in replication, as it uses SQL and triggers rather than binary log shipping to replicate the data across servers. That may make it less suitable for larger cluster deployments with high performance demands.

Slony-I Replication Weakness

Slony-I, the most widely used PostgreSQL replication tool, is inherently inferior to MySQL's built in replication for a number of reasons. First, it uses SQL and triggers to replicate the data across servers. This is considerably slower than MySQL's binary log shipping and makes the communication costs much higher. Second, Slony-I's communication costs grow quadratically in relation to the number of servers in the replication pool (Order(n^2)). This makes it inherently unusable for larger clusters. If we conservatively figure that Slony-I's SQL/trigger method takes twice as much communication as MySQL's binary log shipping, we can easily see how poorly this would work for larger clusters in the real world.

With two servers: MySQL: 2 = 2 PostgreSQL: 2*2^2 = 8

With 4 servers: MySQL: 4 = 4 PostgreSQL: 2*4^2 = 32

With 12 servers: MySQL: 12 = 12 PostgreSQL: 2*12^2 = 288.

While Slony-I is adequate for high availability with two servers, its communication costs are simply prohibitive for scaling out.

Note from Jan Wieck: I don't quite understand how the author of the above came up with those numbers. It seems to refer to the SYNC events and their confirmations. Slony does create these internal housekeeping messages on all nodes and they are transported to all other nodes. But it doesn't transport all user data changes back and forth in that fashion. These are small (maybe 30-100 byte messages) that occur once every couple of seconds. Claiming that with 2 nodes it has 4 times MySQL's communication volume or with 12 nodes 24 times that is outright wrong. Also, Slony does allow cascading of nodes, where one replica will act as a multiplier, serving more replicas in a star like cluster configuration. There is no need to pull directly from the origin.

Slony-I is also difficult to administer.

PGCluster is not useful for situations where high-performance and a decent amount of writes are to be expected. This is because it is a synchronous replication system which waits until a write has happened on all machines in the cluster rather. However, in situations that have very few writes and require data to be absolutely consistent across each database, PGCluster can be a good tool to use.

PostgreSQL Synchronous Replication

PostgreSQL 9.1 comes with support for synchronous replication. It thus allows for the popular star-topology configurations with one write and multiple read-only slaves with slaves capable of stepping up in case the primary goes down.

MySQL

MySQL ships with support for both asynchronous and semi-synchronous replication. In this form of replication a log of events is transmitted to the slaves. These slaves must apply the statements or rows in this log to each of the slave servers independently. This has the limitation that each slave may have a slightly different view of the data depending upon the length of their lag in transferring and applying this log. Prior to 5.1 non-deterministic statements in this log can cause records to be inserted or updated differently on each slave [17] (http://dev.mysql.com/doc/refman/5.1/en/replication.html)

Starting with version 5.1, MySQL supports two forms of replication; statement based replication (SBR) and row based replication (RBR). SBR, used prior to 5.1, collects SQL queries which affect changes to the database in a binary log which the slave servers subscribe to for their changes. RBR instead records the incremental row changes themselves in the binary log that are then applied to the slave. RBR is used automatically when non-deterministic queries are executed on the master. The storage engines NDB and InnoDB, in certain cases, only support replication using this new row based binlog format. [18] (http://dev.mysql.com/doc/refman/5.6/en/replication-formats.html)

semi-synchronous [19] (http://dev.mysql.com/doc/refman/5.6/en/replication-semisync.html) replication requires that at-least one slave confirms receipt of the log by at least one slave before a client receives acknowledgement from a COMMIT. This allows for greater data integrity and simplified slave promotion in the event of a catastrophic failure of the master.

Within the NDB storage engine MySQL supports scalable synchronous replication between ndbd nodes using a two phase commit protocol which does not rely upon a binary log. The two phase commit protocol still exhibits excellent performance due to the in-memory nature of NDB. Once an update has been committed to memory on two nodes it is considered durable. [20] (http://lists.mysql.com/cluster/5729) Replication between two NDB clusters or between NDB and tables in another engine are possible via the asynchronous replication that comes standard with MySQL.

In addition to the built-in asynchronous and semi-synchronous replication methods provided by MySQL Server, additional 3rd party solutions exists to provide virtually-synchronous replication.

- Tungsten Replicator (http://code.google.com/p/tungsten-replicator/)
- Galera Replication (http://codership.com/products/galera_replication)

DataTypes

PostgreSQL does not have an unsigned integer data type (which is not a standard one and can be replaced by a SQL DOMAIN in PG), but it has a much richer data type support in several aspects: standards compliance, the logically fundamental data type BOOLEAN, IP address and networks, user-defined data types mechanism, built-in and contributed data types.

PostgreSQL allows columns of a table to be defined as variable-length multidimensional arrays. Arrays of any built-in or user-defined base type, enum type, or composite type can be created. Arrays of domains are not yet supported. [21] (http://www.postgresql.org/docs/8.3/interactive/arrays.html)

PostgreSQL (through the hstore module) offers a data type for storing sets of JSON-like key/value pairs within a single value. It supports GIN and GiST indexing, and may be used in GROUP BY, ORDER BY and DISTINCT expressions. [22] (http://www.postgresql.org/docs/current/static/hstore.html)

MySQL has set and enum types. Postgres has enum and array types as of 9.0.

MySQL does not have arrays or key/value pairs functionality.

MySQL does not have network IP address data types that PostgreSQL has but does provide INET_ATON() and INET_NTOA() functions to convert IPv4 addresses to and from integers, which are easily stored.

Also, PostgreSQL supports a range of IP-related functions like checking whether a range is part of another range, etc.

Subqueries

Both MySQL and PostgreSQL support subqueries. Support for them is more recent in MySQL and performance is still being improved for some types of subquery (http://forge.mysql.com/wiki/6.0_Subquery_Optimization_Cheatsheet) that may have already been optimised in PostgreSQL. It should also be noted that MySQL does not allow for a subquery in a view. This feature was removed after 4.1 and as of 5.7 was still not available. Workarounds include nested views, joins and hacking the source code.
However, it must be extremely clear that there remains a lot of basic subqueries with major optimization problems in current MySQL versions (5.5.x), which HAVE to be worked around..
Also, be careful as the way "NOT IN" works is not the same across DBMS's (i.e. you need inner/outer value null checks in mysql, they're integrated in postgresql, etc.).

Joins

Both MySQL and PostgreSQL support SQL join syntax, with the notable exception that MySQL does not support FULL OUTER JOIN [23] (http://dev.mysql.com/doc/refman/5.7/en/join.html).

MySQL's query optimizer is able to execute join queries via a modified nested loop join which supports some block-based access, but does not support a classic hash-join or sort-merge join [24] (http://www.xaprb.com/blog/2013/10/01/mysql-isnt-limited-to-nested-loop-joins/). This leaves some edge cases where some more complex queries will execute less efficiently.

PostgreSQL's query planner supports nested loop, hash, and sort merge joins. This makes it able to handle a wider range of complex queries efficiently.

Advanced Indexing

Advanced indexing methods allow database systems to optimize queries to achieve greater performance.

Index Type	MySQL	PostgreSQL
Hash indexes	InnoDB, NDB and MEMORY engines support Hash indexes	PostgreSQL supports Hash indexes, though as of 8.1 they are never faster than b-tree indexes [25] (http://www.postgresql.org/docs/8.1/static/indexes-types.html)
Multiple Indexes	MySQL supports multiple indexes per query.[26] (http://dev.mysql.com/doc/refman/5.6/en/index-merge-optimization.html)	PostgreSQL supports multiple indexes per query.
Change Buffering (also called insert buffer)	InnoDB has a feature to delay building index pages of non unique secondary indexes when the index is too large to fit in memory. This allows it to merge modifications, and in many cases reduce IO considerably.	Postgresql does not support this feature.
	MySQL comes with full-text search for InnoDB and MyISAM storage engines. Prior to version 5.6 only the MyISAM storage engine supported this feature. [27] (http://dev.mysql.com/doc/refman/5.6/en/fulltext-search.html)	PostgreSQL 8.2 has full text search in the tsearch2 (http://www.sai.msu.su/~megera/postgres/gist/tsearch/V2/) module.

Full-Text Indexes	A 3rd party add-on to MySQL, Sphinx Fulltext Search Engine (http://www.sphinxsearch.com/features.html) allows it to support full-text searches on storage engines which do not natively support it.	PostgreSQL 8.3 integrates tsearch2 into the core: "TSearch2, our cutting-edge full text search tool, has been fully integrated into the core code, and also has a cleaner API." [28] (http://www.postgresql.org/about/press/features83.html)
Partial Indexes	MySQL does not support partial indexes.	PostgreSQL supports partial indexes: A partial index is an index built over a subset of a table; the subset is defined by a conditional expression (called the predicate of the partial index). The index contains entries for only those table rows that satisfy the predicate. Partial indexes are a specialized feature, but there are several situations in which they are useful. One major reason for using a partial index is to avoid indexing common values. Since a query searching for a common value (one that accounts for more than a few percent of all the table rows) will not use the index anyway, there is no point in keeping those rows in the index at all. This reduces the size of the index, which will speed up queries that do use the index. It will also speed up many table update operations because the index does not need to be updated in all cases. —PostgreSQL , PostgreSQL 8.2.6 Documentation: Chapter 11. Indexes (http://www.postgresql.org/docs/8.2/static/indexes-partial.html)
Prefix Indexes	MySQL supports prefix indexes. Prefix indexes cover the first N characters of a string column, making the index much smaller than one that covers the entire width of the column, yet still provide good performance characteristics.	With PostgreSQL, prefix indexes are a particular case of Expression Indexes (see below).
Multi-column Indexes	MySQL is limited to 16 columns per index. [29] (http://dev.mysql.com/doc/refman/5.1/en/myisam-storage-engine.html) And not all storage engines provide multi-column indexes.	PostgreSQL is limited to 32 columns per index. [30] (http://www.postgresql.org/docs/8.3/interactive/indexes-multicolumn.html)
Bitmap Indexes	MySQL has no bitmap indexes but achieves similar functionality using its "index_merge" feature.	PostgreSQL supports the ability to combine multiple indexes at query time using bitmap indexes.
Expression Indexes	Expression Indexes can be emulated in MySQL by adding a precomputed column and using a trigger to maintain it.	PostgreSQL allows you to create indexes based on expressions (which may include calls to immutable functions). This is very handy in case there is a table with relatively stable data (not a lot of inserts / updates) and will often be running a query which involves an expensive calculation - the expression itself can be indexed thus eliminating the need of computing it at query runtime.
Non-blocking CREATE INDEX	Dependent on the storage engine. Some engines (such as NDB Cluster and InnoDB Plugin) support online add/drop index (no locks taken). If the engine doesn't support online add/drop index, a write exclusive lock is required and the table copied.	PostgreSQL supports the ability to create indexes without locking the table for writes.
Covering Indexes	MySQL supports covering indexes, which allow data to be selected by scanning the index alone without touching the table data. This is advantageous with large tables that have many millions of rows.	Covering indexes were added to PostgreSQL 9.2

Partitioning

MySQL Supports several forms of horizontal partitioning.

- RANGE
- LIST
- HASH
- KEY
- Composite partitioning using a combination of RANGE or LIST with HASH or KEY subpartitions
- MySQL supports a total of 1024 partitions + subpartitions per table.

PostgreSQL only supports RANGE and LIST partitioning[31] (http://www.postgresql.org/docs/current/static/ddl-partitioning.html).

- HASH partitioning is supported via immutable functions.
- Composite partitioning is also supported.
- PostgreSQL partitions are tables that inherit from a master table.
- I have (thus far) been unable to find a specific technical or practical upper limit to the number of partitions supported in PostgreSQL. Anecdotally, the practical limit is less than the technical limit.

Common Table Expressions

A very useful way to define one or more temporary tables and refer to them in a query, all in one SQL statement. MySQL does not provide CTEs. PostgreSQL does provide CTEs. As of version 9.1, PostgreSQL also support writable CTEs, which allow INSERT, DELETE and UPDATE statements to be used, which works using the RETURNING clause.

Analytic functions

PostgreSQL supports oracle-like analytic function. MySQL does not.

Sequences

PostgreSQL supports sequences. MySQL does not.

Diagnostics and Performance Management

MySQL 5.6 ships with Performance Schema as a way to monitor MySQL Server level execution at a low level[32] (http://dev.mysql.com/doc/refman/5.6/en/performance-schema.html). It is designed based on similar concepts to the Oracle Wait Interface, and automated tools and DBAs to write SQL queries to extract runtime information.

PostgreSQL does not offer native comparable feature, and instead relies on using profiles (gprof and oprofile are popular) and dtrace.[33] (http://stackoverflow.com/questions/2080451/what-is-the-most-critical-feature-postgresql-lacks-compared-with-oracle-and-db2)

Other features

In PostgreSQL, there is no built-in mechanism for limiting database size, mostly due to the risk (http://postgresql.1045698.n5.nabble.com/Best-way-to-limit-database-sizes-td2089238.html) it implies. This is another reason, after popularity, why the most of the web hosting companies are using MySQL. Also, PgAgent (http://www.postgresonline.com/journal/archives/19-Setting-up-PgAgent-and-Doing-Scheduled-Backups.html) a scheduling agent for PostgreSQL allows for scheduled processes.

Licensing

PostgreSQL comes with an MIT-style license, which fits the Free Software Definition (http://www.gnu.org/philosophy/free-sw.html) and Open Source Definition (http://www.opensource.org/docs/osd), and conforms to both the Debian Free Software Guidelines (http://www.debian.org/social_contract#guidelines) and the Copyfree Standard (http://copyfree.org/standard).

MySQL's source code is available under terms of the GNU General Public License, which also fits the Free Software and Open Source definitions and conforms to the Debian Free Software Guidelines (but not to the Copyfree Standard). It is also available under a proprietary license agreement, which is typically intended for use by those who wish to release software incorporating MySQL code without having to release the source code for the entire application. In practical terms, this means that MySQL can be distributed with or without source code, as can PostgreSQL, but to distribute without source code in the case of MySQL requires paying Oracle for a MySQL Commercial License.

Even the MySQL client library is GPL (not LGPL: see GPL vs LGPL for more discussion of these licenses), which means that to use (and therefore link to) the MySQL client library the program must either itself be GPL, must use one of a broad range of FOSS licenses (http://www.mysql.com/company/legal/licensing/foss-exception.html) including BSD and LGPL, or must have a commercial license from Oracle.

See Copyfree vs Copyleft for more about the differences in licensing styles.

Development

MySQL is owned and sponsored by a single for-profit firm, Oracle. MySQL AB holds copyrights to most of the codebase.

By contrast, PostgreSQL is not controlled by any single company, but relies on a global community of developers and companies to develop it. It does, however, enjoy both software development help and resource contributions from businesses who make use of PostgreSQL database technologies, such as EnterpriseDB. Corporate sponsors are considered contributors roughly like any other, however, within PostgreSQL's community-driven development model.

> MySQL is an open-source PRODUCT.
>
> Postgres is an open-source PROJECT.
>
> — Greg Sabino Mullane (http://people.planetpostgresql.org/greg/index.php?/authors/1-Greg-Sabino-Mullane) , Postgres is not for sale (http://www.jiaozhoujob.com/newsprint-995.html) (reprint of original blog post)

One criticism of the MySQL development model has been the historical reluctance of its corporate development team to accept patches from external sources. This has prompted some to say MySQL is not a "true" open source project. Nontrivial improvements from Google and Percona have been accepted into the main codebase recently, though how significant a change in external development policy this represents is yet to be seen.

Furthermore, PostgreSQL's development team is much more accessible than that of MySQL, and they will go as far as to provide you with a patch if there really is a problem with the engine.

Culture

Community

MySQL's community is supported in part by the company's Community Relations Team (http://dev.mysql.com/community/). MySQL AB also sponsored the annual User's Conference and Expo which was run by O'Reilly until 2011. It has since been replaced by MySQL Central @ OpenWorld (https://www.oracle.com/openworld/mysql/index.html).

PostgreSQL is a community supported open source project, with no singular corporate sponsorship. Instead, companies whose business models depend on PostgreSQL are accepted as members of the community, and code from corporate contributors is accepted under the same terms as from any other external contributor.

Both also have large numbers of enthusiastic supporters who are willing to assist on a voluntary basis.

Support Services

As a business product, MySQL's corporate sponsor provides its own official support (http://www.mysql.com/support/) for the server, and there are independent support providers available as well. A MySQL blog consolidator aggregates information about independent support providers, and many are invited to MySQL's Users Conference. One even holds its own sub-conference in association with the main event.

PostgreSQL is a project with many sponsors and developers, and is not controlled by any one company. A realistic choice of support is available from a range of Professional Support Companies (http://www.postgresql.org/support/professional_support).

Name

When the ANSI SQL standard was written, its author explained that the official standards-compliant pronunciation of SQL is "ess queue ell". The names of both MySQL and PostgreSQL reflect the pronunciation specified by the SQL standard's author.

MySQL is officially pronounced "my ess queue ell", though those unfamiliar with this often call it "my sequel" instead -- especially if their previous DBMS experience centered around Microsoft SQL Server (pronounced either "sequel server" or "ess queue ell server").

Because MySQL is a corporate software product, MySQL AB has complete control over the name of the project. As a result of this, and the desire for a consistent brand identity, the MySQL name is likely to remain static.

PostgreSQL is pronounced "post gress queue ell", formed by combining Postgres (the name of the original database management system from which PostgreSQL is descended) with SQL. PostgreSQL is a true portmanteau, in that it not only combines the spellings and pronunciations of two words, but also their meanings: it is the Postgres DBMS updated to use SQL. Some people refer it as "pgsql". The mispronunciation "post gree sequel", and related abbreviation "post gree", may be largely due to MS SQL Server DBA influence as well, though it is a very rare error amongst PostgreSQL users.

There has been talk of going back to the original Postgres name, though how much traction the idea has had is debatable. The PostgreSQL wiki provides an overview of the debate (http://wiki.postgresql.org/wiki/Postgres), including pros and cons for such a name change and alternatives.

Popularity

MySQL is widely popular among various open-source web development packages. The MyISAM engine is often the only database engine offered by webhosting providers. Many web developers use MySQL. (http://articles.techrepublic.com.com/5100-22-1045125.html?tag=rbxccnbtr1) Thus, MySQL became widely popular in web development, and MySQL calls itself "The world's most popular open source database," a grounded claim.

Part of the reason for this popularity is a common perception that MySQL is "easier" to use than other databases -- particularly PostgreSQL. That perception arose years ago, and has fed itself by word of mouth, such that whether it is still true or not is likely to have little or nothing to do with MySQL's current reputation for being comparatively easy to use. In fact, in recent years PostgreSQL has made significant changes that have now caused the perception that they have "closed the gap", and may even have improved its ease of use beyond that of MySQL, though the validity of such claims is as open to question as those of MySQL.

PostgreSQL was not available on Windows. First windows native version was 8.0. This was an advantage for MySQL.

Links

- PostgreSQL vs. MySQL - Pros & Cons by the CEO of EnterpriseDB Postgres (http://vork.us/go/bm24) Event organized by the NYC MySQL Group at Oracle's NYC headquarters
  - 720p HD-Video of the PostgreSQL vs. MySQL event at Oracle (http://LeadIT.us/hands-on-tech/PostgreSQL-vs-MySQL-Pros-Cons)
- Cross Compare of PostgreSQL 8.4, SQL Server 2008, MySQL 5.1 (http://www.postgresonline.com/journal/index.php?/archives/130-Cross-Compare-of-PostgreSQL-8.4,-SQL-Server-2008,-MySQL-5.1.html)

(Note that many of these links are rather old and may not be accurate)

- Is MySQL really this bad? - Ars Technica (http://episteme.arstechnica.com/eve/forums/a/tpc/f/6330927813/m/377006385931)
- Comparison Matrix (http://www.devx.com/dbzone/Article/29480?trk=DXRSS_DB) (includes Apache Derby and One$DB)
- http://www-css.fnal.gov/dsg/external/freeware/pgsql-vs-mysql.html
- http://www.devx.com/dbzone/Article/20743
- Comparison of MySQL 5.5 and PostgreSQL 9.0 replication (http://www.theserverside.com/feature/Comparing-MySQL-and-Postgres-90-Replication)

Pro PostgreSQL

- Related article on Command Prompt: PostgreSQL vs MySQL @ Oracle NYC Head Quarters NYC (my brain just broke) (http://www.commandprompt.com/blogs/joshua_drake/2011/01/postgresql_vs_mysql__oracle_nyc_head_quarters_nyc_my_brain_just_broke/)
- PostgreSQL Connection strings (http://www.connectionstrings.com/postgre-sql)
- Why PostgreSQL Instead of MySQL in 2009 (http://wiki.postgresql.org/wiki/Why_PostgreSQL_Instead_of_MySQL_2009)
- Transactional DDL in PostgreSQL: A Competitive Analysis (http://wiki.postgresql.org/wiki/Transactional_DDL_in_PostgreSQL:_A_Competitive_Analysis)
- http://article.gmane.org/gmane.comp.lang.ruby.rails/12576
- http://www.sitepoint.com/article/site-mysql-postgresql-1
- http://feedlounge.com/blog/2005/11/20/switched-to-postgresql/

Pro MySQL

- Hi-Def Video of the HuffingtonPost CTO discussing Enterprise MySQL: Heavy-Traffic Management and Ultra-Availability (http://leadit.us/hands-on-tech/HuffingtonPost-CTO-on-Enterprise-MySQL-Heavy-Traffic-Management-and-Ultra-Availability) at the NYC MySQL Group at Sun Microsystems
- A conversation with 5 Facebook MySQL gurus (http://www.mysqlperformanceblog.com/2014/03/27/a-conversation-with-5-facebook-mysql-gurus/) - Yoshinori explains ": I have not been able to find a transactional NoSQL database better than InnoDB"
- Why VividCortex uses MySQL (https://vividcortex.com/blog/2014/04/30/why-mysql/)