

# Percona Server with XtraDB for Software-as-a-Service Application Databases

*A Percona White Paper*

Baron Schwartz and Vadim Tkachenko<sup>1</sup>



## Abstract

The *Software as a Service* (SaaS) deployment model is popular with customers, but SaaS databases are often difficult for providers to manage, due to the unique stresses placed upon them. The open-source MySQL database server can suffer from performance and operational limitations in SaaS deployments. Percona Server with XtraDB is an enhanced version of MySQL that solves many of these problems. It enables SaaS providers to scale effectively and economically.

Software as a Service (SaaS) is a deployment model that eliminates the need for customers to install or host software in their own environment. Instead, the provider hosts the application, and the customer uses it over the Internet, usually through a browser or an API. SaaS is attractive to customers because it is usually purchased through a subscription, which makes it instantly available and reduces expenses. With SaaS, customers can effectively outsource application hosting to a specialist.

SaaS applications are ubiquitous today. Well-known examples include Google Apps, the Salesforce CRM application, and the New Relic application management system. Although SaaS and cloud deployment are similar, the difference is that SaaS providers deliver applications to the user, whereas cloud providers deliver infrastructure services. SaaS applications are frequently built in a so-called *multi-tenant* fashion, where processing and data storage for many customers are co-hosted on a single server, instead of being segregated onto their own servers.

**Percona Server** is an enhanced version of the world's most popular open-source database, MySQL. MySQL is used by many of the world's largest websites, including Facebook, Flickr, and YouTube. MySQL is also deployed widely in industries such as financial services, government, education, pharmaceuticals, and telecommunications. Its simplicity, reliability, and ease of use make it cost-effective to manage, and because it is open-source, it can be used without license fees. Percona Server is derived from the MySQL database, to which it adds features such as enhanced monitoring and configurability.

**Percona XtraDB** is an enhanced version of the InnoDB storage engine. *Storage engines* are a unique feature of the MySQL database architecture. They are the software that stores and retrieves the data, and executes queries at the lowest level. MySQL supports a large variety of storage engines with differing characteristics. The user can choose which storage engine is best suited for each table, based on features such as ACID compliance, full-text indexing, and clustering. InnoDB is the most popular general-purpose OLTP storage engine. It is transactional and ACID compliant, with foreign keys, row-level locking, and an advanced MVCC (multi-version concurrency control) architecture. It is stable and mature, and has been in production use for many years. Percona XtraDB builds on this foundation with improved performance and scalability, and adds a number of useful features.

**Percona Server with XtraDB** is the combination of Percona Server and the XtraDB storage engine. It includes a companion hot-backup tool, Percona XtraBackup. XtraBackup can make non-blocking backups of XtraDB's data<sup>2</sup> while the server is running, without interrupting the database's normal operation at all.

---

<sup>1</sup>Thanks to Mark Callaghan and Dimitri Kravtchuk for reviewing this paper.

<sup>2</sup>XtraBackup also works perfectly for backing up data from standard MySQL and InnoDB; it is not limited to XtraDB.

The rest of this white paper explores how multi-tenant SaaS applications are built at the database level. It explains the areas in which the MySQL database server is not well suited for large-scale SaaS databases. It then shows how Percona Server with XtraDB addresses these shortcomings to enable SaaS providers to build scalable and flexible database solutions for multi-tenant applications.

## Architecting a Data Store for SaaS

A good database architecture for a multi-tenant SaaS application is difficult to design. Textbook approaches rarely work well, because of the challenging requirements. Typical requirements include at least the following:

- The schema might need to be flexible; some users might need to store and query data differently from others.
- The dataset will probably grow much larger than a single server can hold, so it must be *sharded* (partitioned) across multiple machines.
- Stale data usually needs to be aged and purged out of the system.
- Tiered subscription plans often require different data retention policies for different customers.
- A typical application has a mixture of queries that access single rows, ranges of rows, and even full scans that need to execute efficiently.
- The database must share resources amongst users, even when some users attempt to do too much work and consume too many resources.

These requirements are often conflicting. For example, a popular pattern when the schema is not fixed in advance is to build a virtual schema from object-key-value tuples. This is commonly known as the Entity-Attribute-Value model, or EAV for short. The application dynamically constructs queries to join many rows into a single row, resulting in what appears to be a table with the desired structure, but is really the result of a much simpler table joined with itself in complex ways. The EAV model usually causes serious and immediate performance problems, and simply will not work for many types of applications, due to limitations such as the number of tables permitted in a single join. It also requires much more storage space than a purpose-built schema, which bloats an already large dataset and causes more disk I/O.

Variable data retention and purging requirements are difficult to manage. Data must be purged when subscriptions expire or are cancelled, or data is older than the contract permits. Purging is usually an intensive task, so it is most efficient to do it lazily, instead of immediately. However, retaining data longer than needed implies that it will be backed up, which is costly and time-consuming. Some applications might not even offer data backups at lower subscription tiers. If the data is commingled, however, it is difficult to avoid backing up the unwanted data. Recovering a specific customer's data is also difficult.

Finally, data clustering (physical proximity) is important on multi-tenant applications. If a query against one customer's data must traverse unrelated data from other customers, then the database is forced to perform much more work than is necessary to complete the query. Therefore, it is highly beneficial to store each customer's data as close together as possible on disk and in memory, and segregate it as cleanly as possible from other data. Clustering data together physically, and separating it clearly from other data, eases backup, recovery, ageing out, and purging.

Broadly speaking, a successful strategy for building a multi-tenant SaaS data store is to use variations on the following practices:

- Instead of using EAV, the application should support a flexible schema by automatically creating tables with the required structure, and dynamically generating queries against them.
- Store only one user's data in any table. This gives physical and logical locality and segregation of storage, to support operations such as purges, backups, and restores.
- Use a storage engine that supports clustered indexes for optimal data layout, and design tables to take advantage of this feature.

### Flexible Application Architecture

SaaS providers generally need to plan for an unpredictable and rapidly varying workload, because they do not know the load patterns their customers will impose upon them. For example, some customers will have a highly seasonal pattern, such as spikes of usage around major holidays. As a result, the provider must build elasticity into the system. The application must be able to cope with a sudden flood of new database objects, an unexpected query load, or sustained addition of data that is related to other data, causing a combinatorial explosion of complexity.

Sharding (partitioning) is the first step towards the needed flexibility at the database layer. However, sharding alone is not enough. Heavy usage on any shard can add so much load that one user's activity impacts other users. It can even simply run the shard out of storage space. If the shards become unbalanced, then the data must be re-partitioned and moved between shards. Therefore, the application needs to be able to cope with data that moves from place to place. As a result, some type of *directory sharding* is usually needed, where the partitioning is not fixed, but is looked up from a directory server. There are various strategies for accomplishing the move; sometimes it requires no downtime, but ideally the SaaS provider will re-shard during maintenance windows, when a specific user's data is either unavailable or read-only.

If the database is designed with no more than one user per table, then repartitioning is relatively easy. It can be accomplished by marking that user as offline or read-only in the global directory, moving the user's data to another server, updating the directory, and then restoring the user to full functionality. Our experience is that variations on this strategy are very successful.

### Choosing Technologies for SaaS and MySQL

If the architecture follows good design practices, then with a few important exceptions noted later, MySQL is a very good fit for multi-tenant data stores.

In general, the most important consideration is to use a reliable, transactional storage engine with row-level locking, write-ahead logging, and automatic recovery after crashes. Server crashes, hardware problems, and disk corruption are a fact of life, and the database must be able to handle them. At enterprise data sizes, a storage engine that does not support these features is unsuitable. This immediately rules out MyISAM, the default storage engine for all GA (generally available) MySQL releases at the time of writing. It has table-level locking, does not support transactions, and is not ACID compliant. If the server crashes, the recovery process is to rebuild the tables and discard rows that appear to be corrupt. This is painfully slow and unreliable.

The other standard storage engine to consider is InnoDB, which has been MySQL's de facto general-purpose ACID-compliant storage engine for many years. It is reliable and robust, and goes to great lengths to ensure data safety. It recovers automatically from crashes. It detects data corruption that is caused externally, such as a bad block on disk. It has row-level locking to permit high concurrency. It clusters tables by the primary key to create data locality, and supports index-only queries for high performance.

Although there are several other special-purpose or experimental ACID storage engines for MySQL, none of them is proven and production-ready. Some storage engines that once appeared promising, such as Falcon, have been cancelled and will never be useful options.

### Shortcomings of Standard MySQL and InnoDB

Although InnoDB is a very good ACID storage engine, it has a few shortcomings that become problematic in a multi-tenant environment.

The first and most important limitation is the difficulty of moving individual tables between servers. Although it is possible to store each table's data in its own file, this does not mean that the files can simply be moved between servers. A portion of the table is stored in a shared global file, and the files contain internal cross-references that must match the global file. If a table's data file is moved to another server, the other server will reject it and refuse to use it. This makes repartitioning hard: to move tables between servers, one must export the tables, delete them, and re-import them on the destination server.<sup>3</sup> This can be prohibitively slow in comparison with the time needed to copy files between servers. As a result, it is harder to build a dynamically scalable application that can be rebalanced with changes in load.

A related shortcoming is the difficulty of backing up and restoring only selected tables, which is needed for effective data retention policies and tiered subscription plans. If it were possible to restore a table by simply copying its data file into place, this would not be hard, but unfortunately not only must the table's data file match the global file's internal identifiers, but it must also match the transactional state of the entire system. InnoDB enforces data consistency and correctness across all of its data together, as a whole (this is the C in ACID). If a data file from some point in the past is introduced into a running system, InnoDB will reject it. Thus, in standard InnoDB, the only feasible option to back up and restore tables individually is to do it at a logical level by exporting and importing data, rather than at a physical level by copying files into place. Again, this is too burdensome.

One might argue that tiered subscription plans can be supported by segregating users onto different servers: non-paying users on one set of servers, paying users on another. This is rarely possible in practice, because many SaaS applications offer free subscriptions to coax users to try the application, hoping they will upgrade to paid accounts later. If a user does upgrade, then the data either needs to be backed up as required by the subscription tier, or the data needs to be moved to a different server. Neither is easy with standard InnoDB.

Standard InnoDB also has a limitation that prevents it from working effectively on tens or hundreds of thousands of tables, which is common in multi-tenant applications<sup>4</sup>. When InnoDB opens a table, it creates an internal *data dictionary* structure to represent the table in memory. This structure is never deallocated, even after the MySQL server closes the table at higher levels inside the database server. InnoDB keeps all tables

<sup>3</sup>It is also possible to `ALTER` the tables to MyISAM, then move the physical files to the destination server and convert the tables back to InnoDB, but this rebuilds the tables and indexes twice, which is still far too slow for large tables.

<sup>4</sup>Filesystems also have limitations with the number of files in a directory, but these can be avoided by creating many databases—perhaps one database per customer—and then InnoDB's shortcoming becomes the limiting factor.

opened until the server shuts down. This causes the data dictionary to consume large amounts of memory, and prevents standard InnoDB from being useful with more than a few thousand tables, depending on the size of the server.

A related limitation is ironically caused by InnoDB's extreme caution. Although it's usually a good thing for InnoDB to go to any lengths to protect its data, this needs to be relaxed for multi-tenant applications. Standard InnoDB will refuse to run at all when it detects data corruption in any of its data; it will violently crash the entire server to prevent executing on bad data. This becomes expensive when, as inevitably happens, someone makes a small operational error that affects only a portion of the system's data, or a particular file becomes corrupted for some reason. The more databases and tables there are, the more likely that one of the tables will have some small, non-critical problem that will crash the entire server.

Finally, and completely separately from InnoDB, the MySQL server itself lacks some features that are very helpful in multi-tenant applications. Effective multi-tenant operation requires careful balancing of the workload, but MySQL has no monitoring features to show the relative usage of different tables, users, or databases. This is needed to determine which data should be moved to repartition and rebalance the shards. MySQL also does not support flexible resource controls, such as control over the number of queries a user is allowed to run concurrently.

## The Percona Server with XtraDB Solution

The features in Percona Server with XtraDB are targeted towards solving real-world problems such as those encountered by SaaS providers. Percona Server with XtraDB eases many of the operational and functional limitations in MySQL and InnoDB, making it easier to use in demanding, dynamic workloads on large sharded datasets.

Perhaps the first feature a SaaS provider would find useful is the added monitoring. Percona Server has additional tables in the `INFORMATION_SCHEMA` database that give easy access to statistics about objects such as users and tables, and this is a good way to determine what constitutes the majority of the work on a server. Figure 1 on the following page is a sample query to find out which users consume the most time querying the server. Apparently `user_118` and `user_2` cause the majority of this server's work, and perhaps they should be moved off to different shards to reduce the load on this shard.

After discovering which data to move, another set of features included only in Percona Server with XtraDB will become useful. In combination with XtraDB and Percona XtraBackup, it is possible to move InnoDB tables from one server to another, live, without the need to shut down and restart either server. The following process outlines the steps necessary to perform this task:

- Use XtraBackup to back up only the desired tables from the source server, while the server continues to run. XtraBackup can back up selected tables matching a pattern. It can stream the tables directly over the network to the destination server, or you can take the backup locally and move the files afterwards.
- Use XtraBackup to "prepare" the backup, with the `--export` option.
- On the destination server, execute `SET GLOBAL innodb_expand_import=1` inside MySQL. This prepares XtraDB to import the tables.
- On the destination server, create empty tables with the desired structure, and then use the command `ALTER TABLE <table> DISCARD TABLESPACE` to discard their data files.

```
mysql> SELECT      USER,
>                BUSY_TIME * 100 /
>                (SELECT SUM(BUSY_TIME)
>                FROM   INFORMATION_SCHEMA.USER_STATISTICS)
>                AS PCT_LOAD
> FROM            INFORMATION_SCHEMA.USER_STATISTICS
> ORDER BY       PCT_LOAD DESC
> LIMIT          5;
```

USER	PCT_LOAD
user_118	23.8030
user_2	15.3183
user_927	3.9270
user_183	3.1717
user_627	2.8293

Figure 1: Finding out what proportion of server resources each user consumes

- Move the files from the prepared backup into the MySQL server's data directory.
- Use the command `ALTER TABLE <table> IMPORT TABLESPACE` to import the data files.

After this process completes, the tables are imported and ready to query. Neither server needs to be shut down or paused to move the tables. In fact, the source server does not even need to run XtraDB; XtraBackup can export the tables from any version of InnoDB, and only the destination server has to have XtraDB. This functionality provides a great deal of flexibility to users. In addition, this process of moving the tables is much more efficient than exporting with *mysqldump* and importing, because it uses binary file copies.

As mentioned previously, servers with tens or hundreds of thousands of tables become unwieldy with standard InnoDB, because its data dictionary uses large amounts of memory, and a problem in any table crashes the entire server. Percona Server with XtraDB solves both of these problems. The size of the data dictionary is configurable, and after tables are closed at the MySQL server level and removed from its table cache, the corresponding data dictionary entry inside XtraDB can be removed automatically to save memory. And instead of crashing, XtraDB simply reports that a table is corrupt and prevents the user from querying it, so the rest of the application can continue to function.

Percona XtraBackup contains special features to help with backup and restoration of specific tables, providing the flexibility needed to run a tiered subscription model, without the cost of treating all data as equally valuable. It can back up a subset of tables, which you can specify by command-line options or by providing a file with a list of the tables you wish to back up. And in conjunction with XtraDB, as discussed earlier, individual tables can be imported into the server without downtime, making it easy to restore selected tables.

In addition to these features, Percona Server with XtraDB contains many other enhancements that can make any MySQL deployment more manageable, more scalable, and higher performance—not just multi-tenant SaaS applications. Here are a few examples:

- On servers with large amounts of memory, XtraDB can enable higher uptime by placing the buffer pool into shared memory, which persists across restarts.
- For faster warm-up, XtraDB can store the state of the buffer pool across restarts, so it can be filled optimally after restart, cutting warm-up time to minutes instead of hours or days.
- XtraDB has a wide variety of extra monitoring features, and the ability to query the server's internal structures. For example, you can discover which tables are resident in the buffer pool and how much space they consume, by querying the `INFORMATION_SCHEMA.INNODB_BUFFER_POOL_PAGES_INDEX` table.
- A given version of Percona Server with XtraDB is faster than the MySQL version from which it is derived. For example, at the time of writing Percona Server 5.1 offers higher performance than standard MySQL 5.1, and is comparable to the performance gains in MySQL 5.5.

Please refer to the Percona Server documentation for information on these and other features.

## Summary

A multi-tenant SaaS application places extraordinary demands on its database server. The database must support very large numbers of tables, huge datasets, and operational flexibility. The enhancements detailed in this white paper—monitoring, live export and import of tables, resilience to isolated data problems, and flexible backup and recovery—set Percona Server with XtraDB apart from standard MySQL with InnoDB. Percona Server with XtraDB enables SaaS providers to more easily build large, scalable, high-performance applications, and enables business models such as tiered subscriptions and variable data retention policies.

### *About Percona Server with XtraDB*

Percona Server, XtraDB, and Percona XtraBackup are 100% free and open-source software. They are available pre-compiled for popular platforms. You can find downloads, installation instructions, user guides, and documentation at <http://www.percona.com/software/>.

Installation is easy and safe; Percona Server with XtraDB is 100% backwards-compatible by default, and you can simply uninstall and return to your previous version of MySQL if desired. No special configuration is required to enable most of the server's additional features and performance, but the server provides advanced configuration options for expert users.

Percona provides commercial support, consulting, training, and engineering services for Percona Server with XtraDB. You can contact us through our website, and we invite you to call us. In the USA, you can reach us during business hours in Pacific (California) Time, toll-free at 1-888-316-9775. Outside the USA, please dial +1-208-473-2904. You can reach us during business hours in the UK at +44-208-133-0309.