# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>3</td>
</tr>
<tr>
<td>InnoDB is the Default Storage Engine</td>
<td>3</td>
</tr>
<tr>
<td>Improved Performance and Scalability</td>
<td>3</td>
</tr>
<tr>
<td>Performance and Scalability Benchmarks</td>
<td>6</td>
</tr>
<tr>
<td>Improved Availability</td>
<td>9</td>
</tr>
<tr>
<td>Improved Manageability and Efficiency</td>
<td>13</td>
</tr>
<tr>
<td>Improved Usability</td>
<td>13</td>
</tr>
<tr>
<td>Improved Instrumentation and Diagnostics</td>
<td>17</td>
</tr>
<tr>
<td>MySQL 5.5: Production Ready Software and Support</td>
<td>18</td>
</tr>
<tr>
<td>Conclusion</td>
<td>18</td>
</tr>
<tr>
<td>Learn More</td>
<td>19</td>
</tr>
</tbody>
</table>

© 2010, Oracle Corporation and/or its affiliates
Introduction

MySQL is the most trusted and depended-on open source database platform in use today. Many of the most popular and highly-trafficked websites in the world are built on MySQL because of its ubiquity across heterogeneous platforms and application stacks and for its well-known performance, reliability and ease of use. With MySQL Oracle now provides a complete LAMP (Linux, Apache, MySQL, PHP, Perl, Python, etc.) stack that allows users and customers of any size or ambition to build applications and products that leverage the best available technology solutions and support. MySQL 5.5 delivers a better MySQL by providing a feature set that greatly improves performance, scalability, and usability specifically on today’s modern, multi-processing hardware, software, and middleware architectures.

MySQL 5.5 combines the benefits of the MySQL database and InnoDB storage engine to provide a high-powered data management solution that includes:

- InnoDB as the default database storage engine
- Improved Performance and Scalability on Windows
- Improved Performance and Scalability to fully utilize the computing power of modern multi-core architectures across all platforms
- Improved Availability
- Improved Manageability and Efficiency
- Improved Usability
- Improved Instrumentation and Diagnostics

This paper provides an introductory look at the MySQL 5.5 and InnoDB performance enhancements that support these improvements and that provide MySQL users with a better MySQL. Each of the key enhancements and how they are implemented are described below.

InnoDB is the Default Storage Engine

Oracle’s InnoDB is the mostly widely used storage engine for Web, eCommerce, Financial Systems, Health Care, and Retail applications built on MySQL and for good reason; InnoDB provides highly efficient ACID compliant transactional capabilities and includes unique architectural elements that assure high performance and scalability. Further, InnoDB is designed to handle transactional applications that require crash recovery, referential integrity, high levels of user concurrency and SLA exceeding response times.

One of the most notable enhancements in MySQL 5.5 is that InnoDB is now the default storage engine. While MyISAM and the other table types are still readily available users can now create applications built on InnoDB without configuration setting changes. InnoDB has been optimized to take full advantage of modern hardware and operating system resources. The new optimizations, settings and default behaviors for MySQL 5.5 and InnoDB are described below.

Improved Performance and Scalability

MySQL 5.5 introduces a re-architected InnoDB that includes many performance and scalability features. InnoDB extends MySQL 5.5 performance and scalability by adding the following enhancements:

- **Improved Performance and Scalability on Windows** - MySQL has traditionally performed well on UNIX based platforms. With more developers now building and deploying applications on Windows,
MySQL’s footprint has expanded on Windows from the development desktop to the production datacenter. In fact, Windows is now the most commonly downloaded MySQL platform. MySQL 5.5 includes Windows specific improvements that ramp up performance and scalability for systems and applications designed to service high concurrency and user loads. The key MySQL 5.5 improvements for Windows include:

- **MySQL now uses native Windows synchronization primitives to implement mutexes and locking algorithms.**
- **MySQL now uses native Windows atomic operations vs POSIX threads to implement and free read/write specific locks.**
- **MySQL now uses native Windows operating system memory allocators by default.**
- **MySQL on Windows I/O thread handles maximum now equals the maximum on other platforms.**
- **Legacy optimizations made on other platforms have now been ported to MySQL on Windows.**
- **Many lingering Windows specific bugs have been cleaned up.**

**Improved Default Thread Concurrency** – InnoDB now defaults to allow an unlimited number of concurrently executing threads, leveraging the processing power of multi-threaded and multi-core systems. Users can override this default behavior by setting the value of `innodb_thread_concurrency` to the limit that best works for specific implementations.

- **Control of Background I/O Threads** – Users now have two new configuration parameters for all platforms, `innodb_read_io_threads` and `innodb_write_io_threads` that allow for the setting of the number of background threads used for read and write requests. This helps users tune and scale their MySQL applications on high-end, multi-core systems.

- **Control of Master Thread I/O Rate** – Users can now configure the overall I/O capacity and bandwidth available to InnoDB for running background tasks, via the new `innodb_io_capacity` in the `my.cnf` or `my.ini` files.

- **Control of Using Operating System Memory Allocators** – Users can now control whether InnoDB uses its own memory allocator or leverages the more efficient allocators available in the current versions of the most commonly deployed operating systems. This is easily controlled by setting the value of the new system configuration parameter `innodb_use_sys_malloc` in the MySQL 5.5 option file (`my.cnf` or `my.ini`). The default setting is 1, which instructs InnoDB to use the operating system resource.

- **Control of Adaptive Hash Indexing** – Users can now disable the Adaptive Hash Indexing feature, which is enabled by default in the built-in InnoDB. This is useful when tuning applications or systems; too many threads blocked by RW-latch contention caused by the underlying Adaptive Hash Indexing process (as shown in the output from `SHOW ENGINE INNODB STATUS`) is a good indicator this feature can be disabled to improve performance.

- **Control of Insert Buffering** – Users can now disable the InnoDB insert buffer for operations involving working data sets or inserts that fit or almost fit into the InnoDB buffer pool. This keeps InnoDB from unnecessarily merging buffered insert changes into the InnoDB buffer pool retaining more buffer pool space and persistence of the most actively accessed data pages.

- **Improved scalability via Faster Locking algorithm** – For most platforms (UNIX, Linux, Windows), InnoDB now uses native atomic operations vs POSIX threads to implement mutexes and read/write locks. This boosts InnoDB performance and scale, specifically on multi-core systems.

- **Restored Group Commit** – An optimization that allows InnoDB to perform low-level I/O operations (log write) once for a set of commit operations, rather than flushing and syncing separately for each commit, which can significantly improve throughput.

- **Improved Recovery Performance** - InnoDB is known for its ability to reliably recover data after a crash. In previous versions the recovery time needed to scan and apply the redo log prior to the next startup could be exceedingly long depending on the amount of data and time between server restarts. MySQL 5.5 includes a number of default optimizations designed to speed up the scanning and applying of redo logs so the next restart is faster. Users who had previously sized redo logs artificially low because of slow recovery times can now increase the log file size without concern.
MySQL

- **Multiple Buffer Pool Instances** - Today's buffer pools are consistently sized in the multi-gigabyte range, data pages are persisted and are constantly being read and updated by different database threads. MySQL 5.5 removes the bottleneck of waiting threads when one thread is updating the buffer pool. All the structures normally associated with the buffer pool can now be multiplied, such as its protecting mutex, the last recently used (LRU) information, and the flush list. Users can now control and tune how many buffer pool instances are used; however, for backward compatibility the default is still 1. This feature works best with combined buffer pool sizes of several gigabytes, where each buffer pool instance can be a gigabyte or more.

- **Multiple Rollback Segments** - InnoDB can now use multiple rollback segments improving performance and scalability and greatly increasing the number of concurrent transactions that can be serviced. While previous InnoDB versions had a limit of 1023 concurrent transactions, MySQL 5.5 now allows for up to 128K concurrent transactions that create undo data (from insert, update, and delete operations). This improvement reduces the contention on the single rollback segment mutex resulting in higher throughput.

- **Native Asynchronous I/O for Linux** - MySQL 5.5 enables improved concurrency of I/O requests on Linux systems. Previous versions of InnoDB have provided "simulated asynchronous I/O" for Linux by internally arranging I/O calls as if they were asynchronous, while behind the scenes the query thread would block competing threads until the I/O request was complete. MySQL 5.5 now provides true native asynchronous I/O support for Linux and Windows based systems. This feature requires the libaio userspace library to be installed on Linux and comes with a configuration option innodb_use_native_aio that can turned off if the new default setting is not compatible with the host I/O subsystem.

- **Extended Change Buffering: Now with Delete and Purge Buffering** - Like all databases, InnoDB uses indexed columns to make query and primary key lookups more efficient and performant. Secondary indexes, or those on columns other than the primary key, require disk writes to keep them up to date when primary key columns are inserted, deleted, or updated. Previous versions of InnoDB include an optimization that delays disk writes for secondary index maintenance when the changes are due to INSERT operations. Meaning that InnoDB would delay or buffer the changes to secondary indexes until the index contents are read into the buffer pool during normal operations, such as a requesting query. The changes can then be made quickly in memory and then flushed back to disk using the normal schedule for writing dirty blocks. When the changes in the buffer pool affect a group of sequential disk blocks, they can be flushed more efficiently than if the data were written piece by piece at the time of the change. MySQL 5.5 extends this same functionality to include writes caused by deletes (an initial delete marking operation, followed later by a purge operation that collects/purges all the deleted records). The new Delete buffering and legacy Insert buffering features are now controlled using the new innodb_change_buffering configuration option, which has a default of all.

- **Improved Log Sys Mutex and Separate Flush List Mutex** - Operations involving the buffer pool and the flush list previously were protected by a single buffer pool mutex, which could cause contention and unnecessary delays. In MySQL 5.5 the flush list has its own mutex, reducing contention with other buffer pool operations. This is the new default behavior and requires no configuration setting to enable. With the multiple buffer pool improvement this only gets better because each buffer pool instance has a separate flush list mutex reducing contention even further.

- **Improved Purge Scheduling** - The InnoDB purge operation is a type of garbage collection that runs periodically. In previous versions, the purge was part of the master thread, meaning that it could block other database operations when running. In MySQL 5.5 this operation can run in its own thread, allowing for more concurrency. Users can control whether the purge operation is split into its own thread with the innodb_purge_threads configuration option, which can be set to 0 (the default) or 1 for a single separate purge thread.

- **Improved Metadata Locking Within Transactions** - In previous MySQL versions when a transaction acquired a metadata lock for a table used within a statement, it released the lock at the end of the statement. This approach had the disadvantage that if a data definition language ("DDL") statement occurred for a table that was being used by another session in an active transaction, statements could be written to the binary log in the wrong order. MySQL 5.5 ensures transaction
serialization by not permitting one session to perform a DDL statement on a table that is used in an incomplete transaction in another session. This is achieved by acquiring metadata locks on tables used within a transaction and deferring release of those locks until the transaction ends. This metadata locking approach has the implication that a table that is being used by a transaction within one session cannot be used in DDL statements by other sessions until the transaction ends. For example, if a table t1 is in use by a transaction, another session that attempts to execute DROP TABLE t1 will block until the transaction ends.

These changes, along with optimizations made to how MySQL internally manages table locking (LOCK_open) improve performance for OLTP applications, specifically those that require frequent DDL activity.

Learn about all of the new MySQL 5.5 and InnoDB performance and scalability enhancements, including how to enable and implement them here:


Performance and Scalability Benchmarks

Much of the engineering work that has gone into MySQL 5.5 and re-architecting InnoDB has been focused on providing users with “transparent” performance and scalability gains across heterogeneous platforms specifically on multi-CPU/core, hyper-threaded architectures. For quick reference, below are Oracle’s internal SysBench benchmark tests comparing the performance of MySQL 5.5 with version 5.1 on Linux and Windows platforms. These benchmarks include instances of 5.1 configured with the optional InnoDB 1.0 plug-in and with the default InnoDB built-in.

MySQL 5.5 SysBench Benchmarks on Linux

For context, the SysBench benchmarks for Linux were performed on the following server configuration:

Intel Xeon x7460, x86_64
4 CPU x 6 Cores/CPU
2,86 GHz, 32 GB RAM
Fedora 10

MySQL 5.5.6
(Now InnoDB)

MySQL 5.1.50
(InnoDB Plug-in)

MySQL 5.1.50
(InnoDB built-in)
Figure 1 SysBench Benchmarks for Linux – Read Only

Figure 1 shows a 200% performance and scalability gain for MySQL 5.5 over 5.1 for Read Only activity on Linux at higher transaction and connection loads. New concurrency improvements allow MySQL 5.5 to sustain improved performance levels at higher transactions per second and user connection loads so applications remain responsive even when the physical server resources are saturated.

Figure 2 SysBench Benchmarks for Linux – Read/Write

Figure 2 shows a 370% performance and scalability gain for MySQL 5.5 over 5.1 for Read/Write activity on Linux at higher transaction and connection loads. Again, although there is some drop-off at higher transaction and user loads, the new concurrency improvements allow MySQL 5.5 to sustain higher performance levels than previous versions of 5.1 at physical server saturation.

MySQL 5.5.6
(New InnoDB)

MySQL 5.1.50
(InnoDB Plug-in)

MySQL 5.1.50
(InnoDB built-in)

MySQL 5.5 SysBench Benchmarks on Windows

For context, the SysBench benchmarks for Windows were performed on the following server configuration:

Intel x86_64
4 CPU x 2 Cores/CPU
3.166 GHz, 8 GB RAM
Windows Server 2008
Figure 3 shows a 540% performance and scalability gain for MySQL 5.5 over 5.1 for Read Only activity on Windows at higher transaction and connection loads. New Windows specific concurrency improvements allow MySQL 5.5 to sustain improved performance levels at higher transactions per second and user connection loads so applications remain responsive even when the physical server resources are saturated.

Figure 4 shows an eye-popping 1500% performance and scalability gain for MySQL 5.5 over 5.1 for Read/Write activity on Windows at higher transaction and connection loads. Again, new Windows specific concurrency improvements allow MySQL 5.5 to sustain improved performance levels at higher transactions per second and user connection loads so applications remain responsive even when the physical server resources are saturated. It should be noted that MySQL 5.5 on Windows is now optimized for most Windows based application deployment use cases that have traditionally been serviced by Microsoft’s SQL Server.
Improved Availability

MySQL replication is the most popular and widely used database feature because it enables scalability and provides an easy to use, robust solution for data redundancy and availability. In MySQL 5.5 replication has been enhanced in response to user requests that MySQL provide high availability features that:

- Ensure data consistency between master and slave servers
- Immediately detect if replication is not working
- Allow a crashed slave to automatically recover from the master relay log
- Allow users to filter events for specific servers
- Correctly convert data types between master and slave servers

MySQL 5.5 replication includes the following enhancements that support users in these key areas:

**Semi-synchronous Replication**

MySQL replication is asynchronous by default meaning that a master and its slave/slaves are autonomous when it comes to data consistency. Asynchronous replication provides optimal performance because a master is free to service other inbound transactional requests after writing updates to its Binlog without waiting to verify that updates have been replicated to at least one slave in the topology. While fast, this comes with a high risk of master/slave data inconsistency or even data loss at recovery if there is a failure on either end.

MySQL 5.5 introduces semi-synchronous replication to ensure data consistency and redundancy between master and at least one slave in the calling chain. In this configuration, a master and any number of its replicant slaves are configured so that at least one slave in the replication topology must acknowledge updates have been received and written to its relay log before the parent master commits the transaction. In the event of a time-out, the originating master will temporarily switch to asynchronous replication until at least one of the slaves set up for semi-synchronous replication catches up.

![Figure 5 Semi-Synchronous Replication](image-url)
Semi-synchronous replication must be enabled on both the master and slave servers, otherwise the master defaults to asynchronous replication. MySQL 5.5 uses a new plug-in architecture to enable semi-synchronous replication. To this end, the following commands and variable settings are used to enable 5.5 masters and slaves. Static settings can also be added to the my.* configuration files:

To enable the semi-synchronous replicator on the master:

```
INSTALL PLUGIN 'rpl_semi_sync_master' SONAME 'semisync_master.so';
SET rpl_semi_sync_master_enabled=1;
SET rpl_semi_sync_master_timeout=1000; (1s, default 10s)
```

To enable the semi-synchronous replicator on a slave or slaves:

```
INSTALL PLUGIN 'rpl_semi_sync_slave' SONAME 'semisync_slave.so';
SET rpl_semi_sync_slave_enabled=1;
START SLAVE;
```

Once enabled semi-synchronous replication exposes new system and status variables that can be used to check on configuration and operational status. The values for each are exposed using SHOW VARIABLES and SHOW STATUS. These include:

On master:

- `rpl_semi_sync_master_status` – indicates status of when master is using asynchronous or semi-synchronous replication.
- `rpl_semi_sync_master_clients` – shows how many slaves are configured for semi-synchronous replication.
- `rpl_semi_sync_master_yes_tx` – shows number of successfully acknowledged commits by slaves.
- `rpl_semi_sync_master_no_tx` – shows number of unsuccessfully acknowledged commits by slaves.

On Slave:

- `rpl_semi_sync_slave_status` – indicates if semi-synchronous replication is enabled on slave.

**Replication Heartbeat**

MySQL 5.5 provides a new replication heartbeat option that helps users know immediately when replication stops working. The heartbeat is a message sent at regular intervals from a master node to slave nodes. The slave can be configured to automatically check connection and message status; if the message is not received by the slave the slave knows that a connection to the master node has failed in some way.
Figure 6 Replication Heartbeat

Replication heartbeat is an optional configuration and is enabled on the 5.5 slave using:

```sql
STOP SLAVE;
CHANGE MASTER TO master_heartbeat_period= milliseconds;
START SLAVE;
```

The following status variables can then be monitored to easily detect when a master is idle and to get a finer-grained estimate on slave seconds behind master for recovery purposes:

```sql
SHOW STATUS like 'slave_heartbeat_period'
SHOW STATUS like 'slave_received_heartbeats'
```

**Automatic Relay Log Recovery**

MySQL 5.5 ensures master/slave consistency on a restart by allowing replication users to optionally configure slaves to automatically discard its own unprocessed relay logs and then recover pending transactions from the originating master. This can be used after a slave crash to ensure that potentially corrupted relay logs are not processed. For compatibility the default for this is disabled, but can be set using the new `relay_log_recovery=value` to 1 on the slave to be recovered.

**Replication Per Server Filtering**

Circular, or multi-master replication, provides a highly available deployment that ensures redundancy of data in the case any of the servers in a topology ring fails or is removed. In this configuration master servers are configured so that each is also a slave of another server in the topology. Updates written to any of the masters are then replicated around the ring until the transaction reaches the originating server which acts as the terminator of its own events. In the event of a node failure the affected server is removed from the topology and its slave is simply redirected to another master in the ring and processing then continues. This is depicted here:
In previous versions when a server is removed from the ring due to failure, maintenance, etc. users needed to manually ensure that all of its updates were terminated from the new calling chain. MySQL 5.5 provides a new set of time-saving commands that allow users to easily filter out any events related to a removed server. In the above case, when Server A is removed from the topology, users can now easily filter any Server A related events by entering the following command on the next server in the calling chain:

```
Server B> CHANGE MASTER TO MASTER_HOST=D ...
IGNORE_SERVER_IDS=(A)
```

**Replication Slave Side Data Type Conversions**

In MySQL 5.1 precise data type conversions between master and slave are supported for statement-based replication only. In this configuration column types could be different in the master and slave tables as long as the underlying data had high level compatibility (INT to TINYINT for example). MySQL 5.5 now provides precise data type conversions between master and slave for both statement-based and row-based operations. Conversions within integer, decimal, string, binary, BIT, ENUM and SET domains are supported.

A new SET variable in 5.5 enables the conversion, and requires that the slave be restarted to take effect. The settings and what they enable are:

- `SET SLAVE_TYPE_CONVERSIONS=“ALL_LOSSY”` – enables conversions to types with smaller domain (INT to TINY for example)
- `SET SLAVE_TYPE_CONVERSION=“ALL_NON_LOSSY”` – enables conversions to types with larger domain (TINY to INT for example)

Learn more about all of the new MySQL 5.5 replication features, including how to enable, monitor and tune them, here:

Improved Manageability and Efficiency

MySQL 5.5 provides better control over internal InnoDB settings so things like performance, scale and storage can easily be monitored, tuned and optimized for specific use cases and application loads. The key advances and features available in MySQL 5.5 and InnoDB are:

- **Faster Index Creation** – MySQL 5.5 can now add or drop indexes without copying the underlying data of the entire target table. This improves the efficiency and speed of creating and dropping indexes on InnoDB tables.

- **Efficient Data Compression** – New InnoDB table compression options significantly reduce the storage requirements for the MySQL database and improve application throughput by reducing I/O workload, all with minimal overhead and operating expense. Users can specify new `ROW_FORMAT` and `KEY_BLOCK_SIZE` parameters in the `CREATE TABLE` and `ALTER TABLE` commands to store data pages in 1K, 2K, 4K, 8K or the standard 16K byte compression levels. InnoDB also provides new `INFORMATION_SCHEMA` tables around compression so users can monitor and tune their implementations for optimal efficiency.

- **Efficient Storage Options for Large Objects and Variable-Length Columns** – MySQL 5.5 improves storage efficiency of large objects by allowing users to create tables using `ROW_FORMAT=DYNAMIC` or `ROW_FORMAT=COMPRESSED`. With these options, long column values are stored fully off-page, and the associated clustered index record contains only a 20-byte pointer to the overflow page.

- **New INFORMATION_SCHEMA tables** – 7 new tables provide information specific to InnoDB compression and transaction locking.
  - `INNODB_CMP` – Contains status information on the efficiency and operations related to compressed tables.
  - `INNODB_CMP_RESET` – Same as above, but reading from this table resets the reported statistics.
  - `INNODB_CMPMEM` – Contains status information on the compressed pages that reside in the buffer pool.
  - `INNODB_CMPMEM_RESET` – Same as above, but reading from this table resets the reported statistics.
  - `INNODB_TRX` - Contains information about every transaction currently executing inside InnoDB, including whether the transaction is waiting for a lock, when the transaction started, and the particular SQL statement the transaction is executing.
  - `INNODB_LOCKS` – Contains a row for each blocked transaction that describes each lock the transaction has requested and what the transaction is waiting on.
  - `INNODB_LOCK_WAITS` – Contains information about transactions that are waiting for a specific lock.

Learn about all of the new MySQL 5.5 and InnoDB performance and scalability enhancements, including how to enable and implement them here:


Improved Usability

MySQL 5.5 extends the usability of the stored objects and table/index partitioning features that were introduced in earlier versions. The key advances and features available in MySQL 5.5 are:
New SQL Syntax for SIGNAL/RESIGNAL

MySQL 5.0 introduced stored procedures, triggers and views so developers can encapsulate and re-use common application code components. These popular features have been extended in MySQL 5.5 to include support for the ANSI/ISO standard SIGNAL/RESIGNAL syntax. Developers can now use the SIGNAL/RESIGNAL syntax inside of stored objects, including stored procedures, functions, and triggers for easier development and debugging and to raise error conditions which can then be used to invoke specific error handling in the host application. An example of how the new SIGNAL syntax works within a stored procedure would be:

```
CREATE PROCEDURE p (pval INT)
BEGIN
  DECLARE specialty CONDITION FOR SQLSTATE '45000';
  IF pval = 0 THEN
    SIGNAL SQLSTATE '01000';
  ELSEIF pval = 1 THEN
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'An error occurred';
  ELSEIF pval = 2 THEN
    SIGNAL specialty
    SET MESSAGE_TEXT = 'An error occurred';
  ELSE
    SIGNAL SQLSTATE '01000'
    SET MESSAGE_TEXT = 'A warning occurred', MYSQL_ERRNO = 1000;
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = 'An error occurred', MYSQL_ERRNO = 1001;
  END IF;
END;
```

In this simple example, if pval is 0, p() signals a warning because SQLSTATE values that begin with '01' are signals in the warning class. The warning does not terminate the procedure, and can be “seen” by the host application using `SHOW WARNINGS` after the procedure returns.

If pval is 1, p() signals an error and sets the MESSAGE_TEXT condition information item. The error terminates the procedure, and the text is returned with the error information.

If pval is 2, the same error is signaled, although the SQLSTATE value is specified using a named condition in this case.

If pval is anything else, p() first signals a warning and sets the message text and error number condition information items. This warning does not terminate the procedure, so execution continues and p() then signals an error. The error does terminate the procedure. The message text and error number set by the warning are replaced by the values set by the error, which are returned with the error information.

The new RESIGNAL syntax passes on the error condition information that is available during execution of a condition handler within a compound statement inside a stored procedure or function, trigger, or event. RESIGNAL may change some or all information before passing it on and makes it possible to both handle an error and return the error information. Otherwise, by executing an SQL statement within the handler, information that caused the handler's activation is destroyed. RESIGNAL also can make some procedures shorter if a given handler could handle part of a situation, then pass the condition “up the line” to another handler. A simple example of using the RESIGNAL and SIGNAL syntax to raise an error condition would be:
DROP TABLE IF EXISTS xx;
delimiter //
CREATE PROCEDURE p ()
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        SET @error_count = @error_count + 1;
        IF @a = 0 THEN RESIGNAL SET MYSQL_ERRNO = 5; END IF;
    END;
    DROP TABLE xx;
END//
delimiter ;
SET @error_count = 0;
SET @a = 0;
CALL p();

Learn about the new MySQL 5.5 SIGNAL/RESIGNAL syntax options here:

New Table/Index Partitioning Options

MySQL 5.5 extends table and index RANGE and LIST partitioning to non-integer columns and dates and adds the ability to partition on multiple columns. These options are supported by a new COLUMNS keyword. A simple example for how this applies to non-integer columns would be:

CREATE TABLE expenses (  expense_date DATE NOT NULL,
    category VARCHAR(30),
    amount DECIMAL (10,3)
);

Previously this table could be partitioned using only integer values, so partitioning by category for quick lookups would require converting the category column values to a cross-referenced integer and creating an additional lookup table. In MySQL 5.5, users can now do this by simply using the new partition options:

ALTER TABLE expenses
PARTITION BY LIST COLUMNS (category)
(  PARTITION p01 VALUES IN ( 'lodging', 'food'),
  PARTITION p02 VALUES IN ( 'flights', 'ground transportation'),
  PARTITION p03 VALUES IN ( 'leisure', 'customer entertainment'),
  PARTITION p04 VALUES IN ( 'communications'),
  PARTITION p05 VALUES IN ( 'fees')
);

A similar example for how this applies to dates would be:

CREATE TABLE t2
(  dt DATE
)
PARTITION BY RANGE COLUMNS (dt)
(}
PARTITION p01 VALUES LESS THAN ('2007-01-01'),
PARTITION p02 VALUES LESS THAN ('2008-01-01'),
PARTITION p03 VALUES LESS THAN ('2009-01-01'),
PARTITION p04 VALUES LESS THAN (MAXVALUE));

Finally, an example of multiple column partitioning would be:

CREATE TABLE employees (  
    first_name varchar(14) NOT NULL,  
    last_name varchar(16) NOT NULL,  
    gender char(1) DEFAULT NULL,  
    hire_date date NOT NULL  
)  
PARTITION BY RANGE COLUMNS(gender,hire_date)  
(PARTITION p01 VALUES LESS THAN ('F','1990-01-01'),  
PARTITION p02 VALUES LESS THAN ('F','2000-01-01'),  
PARTITION p03 VALUES LESS THAN ('F',MAXVALUE),  
PARTITION p04 VALUES LESS THAN ('M','1990-01-01'),  
PARTITION p05 VALUES LESS THAN ('M','2000-01-01'),  
PARTITION p06 VALUES LESS THAN ('M',MAXVALUE),  
PARTITION p07 VALUES LESS THAN (MAXVALUE,MAXVALUE)  
)

As in previous versions partition distribution can be checked via the INFORMATION_SCHEMA as follows:

SELECT  
    partition_name part,  
    partition_expression expr,  
    partition_description descr,  
    table_rows  
FROM  
    INFORMATION_SCHEMA.partitions  
WHERE  
    TABLE_SCHEMA = schema();

MySQL 5.5 also improves partitioning ease-of-use with the following new usability features:

TRUNCATE PARTITION – Works like the legacy DROP PARTITION, but it leaves an existing partition in place so that it can quickly be refilled with new or redefined partitioning values. This extends the available pruning options so that partitions can easily be dropped or simply emptied for re-use.
TO_SECONDS – Allows users to convert \texttt{DATE} and \texttt{DATETIME} columns into seconds so partitions can be built on time intervals smaller than one day.

Learn about MySQL Partitioning and the new options available in MySQL 5.5 here:

\url{http://dev.mysql.com/doc/refman/5.5/en/partitioning.html}

\section*{Improved Instrumentation and Diagnostics}

\subsection*{New \texttt{PERFORMANCE_SCHEMA}}

MySQL 5.5 introduces a new \texttt{PERFORMANCE_SCHEMA} ("P_S") for monitoring MySQL server run-time performance. Unlike the legacy \texttt{INFORMATION_SCHEMA}, which is used for the recording and analysis of MySQL database metadata, the new \texttt{P_S} is implemented as a new storage engine and underlying database that is used to catalog performance related data and metrics for current and historical analysis.

The new \texttt{P_S} monitors low-level server events, or any server operation that takes time or resources. In 5.5 \texttt{P_S} event collection and monitoring is characterized by the following:

\begin{itemize}
\item The \texttt{P_S} storage engine collects event data using “instrumentation points” users set in the server source code.
\item Collected events are stored in tables in the new \texttt{performance_schema} database. These tables can be queried using SELECT statements like other tables. \texttt{P_S} configuration can be modified dynamically by updating tables in the \texttt{performance_schema} database through SQL statements. Configuration changes affect data collection immediately. Also, tables in the \texttt{performance_schema} database are views or temporary tables that use no persistent on-disk storage.
\item \texttt{P_S} events are distinct from events written to the server's binary log (which describe data modifications) and Event Scheduler events.
\item Current \texttt{P_S} events are available, as well as event histories and summaries. This enables users to determine how many times instrumented activities were performed and how much time they took. Event information is available to show the activities of specific threads, locks/\texttt{mutexes} or activity associated with specific files and objects.
\item \texttt{P_S} instrumentation is available on all platforms supported by MySQL, but the types of timers and instrumentation may not be fully supported or implemented for all platforms and for all storage engines.
\end{itemize}

Activating the new \texttt{P_S} causes no changes in server behavior, performance or throughput. Unlike other server features there are no separate threads or changes to thread scheduling or execution plans associated with \texttt{P_S}. Further, no memory allocation is done beyond that which occurs during server startup. For reliability, execution of server code proceeds normally even if \texttt{P_S} collection fails and when there is a choice between performing processing during event collection initially or during event retrieval later, priority is given to making collection faster. This is because collection is ongoing whereas retrieval is on demand and might never happen at all. Finally, instrumentation is versioned for backward compatibility, so when the instrumentation implementation changes, previously instrumented code will continue to work.

Learn about the new MySQL Performance Schema including how to enable and instrument MySQL server code here:

\url{http://dev.mysql.com/doc/refman/5.5/en/performance-schema.html}
MySQL 5.5: Production Ready Software and Support

MySQL is the most popular open source database in use today and remains available under the GNU General Public License ("GPL"). For those new to MySQL and for advanced users building business-critical applications, MySQL 5.5 is fully available for development, QA and production use and can be deployed with confidence under Oracle’s world-class software and support services. MySQL 5.5 is available under several commercial licensing options that allow users to develop, deploy, monitor and backup the most secure, up to date versions of MySQL, all with the backing of Oracle’s 24x7x365 technical support. The MySQL commercially available software and services include:

- **MySQL Database** – The most secure and up to date version of MySQL is used to power the most demanding online, web and OLTP applications. The MySQL commercial server is a fully integrated transaction-safe, ACID compliant database with full commit, rollback, crash-recovery and row level locking capabilities. MySQL is known for its performance, reliability and ease-of-use across all Linux, UNIX, Mac OS X and Windows platforms.

- **MySQL Enterprise Backup** – Performs non-blocking online “hot” backups of MySQL databases. MySQL Enterprise Backup also supports full, incremental and partial backups with compression as well as point-in-time recovery.

- **MySQL Enterprise Monitor and Query Analyzer** – Continuously monitors MySQL databases and proactively alerts DBA to potential problems, queries and tuning opportunities before they impact key systems or applications. The Monitor provides a set of MySQL expert advisors that provide insight and detailed guidance on fixing and tuning MySQL configurations and variables for optimal security, performance and availability. The built-in, advanced Query Analyzer allows developers to visually find and tune expensive query code without the need for Slow Query Log, SHOW PROCESS LIST or other labor intensive methods.

- **MySQL Workbench** – Provides GUI-based data modeling, SQL development, deployment, and comprehensive administrative tools (server configuration, user administration, object management) for database architects, developers, and DBAs.

- **Oracle Lifetime Support for MySQL** – MySQL Technical Support Services provide direct access to expert MySQL Support engineers who are ready to assist users in the development, deployment, and management of MySQL applications. The MySQL Support team is composed of seasoned MySQL developers and database experts who understand the issues and challenges users face because they’ve overcome these same challenges themselves.

Learn about the commercially available MySQL software and support services here:

http://mysql.com/products
http://mysql.com/support/
http://mysql.com/training/
http://mysql.com/consulting/

**Conclusion**

MySQL is the most popular and widely used open source database in the world because of its performance, reliability and ease of use. MySQL 5.5 builds on this momentum by introducing “transparent” performance and scalability gains specifically designed to fully utilize the computing power of modern, multi-processing architectures across all platforms. MySQL 5.5 performance and scalability gains are most notable on the Windows platform, which now is on par (or better) with the other supported platforms.
platforms. Key to the 5.5 improvements is a re-architected InnoDB, which becomes the default MySQL database storage engine. MySQL 5.5 also provides new replication monitoring and manageability features that enable users to immediately know when replication stops working and, when there is a problem, to easily recover and synchronize masters and slaves. New InnoDB table storage and compression options help to reduce the MySQL data footprint and make data retrieval much more efficient. For the Developer, MySQL 5.5 provides new SQL syntax and partitioning options that make application development, debugging and tuning much easier. Finally, the new Performance Schema provides DBAs and Developers with low-level insight into MySQL database performance metrics so they have an understanding of where their applications are spending precious time and resources down to the MySQL source code level. These features, along with Oracle’s commitment to providing hardware, software and support services that are engineered to work together, make MySQL 5.5 the best release of the database ever.

Learn More

For more information on Oracle’s MySQL products and services, please visit:

http://www.mysql.com/products/

MySQL Customers and Case Studies
http://www.mysql.com/customers

MySQL Enterprise Edition
http://mysql.com/products/enterprise/

MySQL Enterprise Backup
http://www.mysql.com/products/enterprise/backup.html

MySQL Enterprise Monitor and Advisors
http://www.mysql.com/products/enterprise/monitor.html

MySQL Query Analyzer

MySQL High Availability
http://www.mysql.com/products/enterprise/high_availability.html

MySQL Professional Services and Consulting
http://mysql.com/consulting/