



# Time Zones and MySQL

---

Presented by: Sheeri K. Cabral

Twitter: @sheeri



Pythian  
love your data

# About Pythian

- Recognized Leader:

- Global industry-leader in database infrastructure services for Oracle, Oracle Applications, MySQL and SQL Server
- 150 current multinational companies such as Forbes.com, Fox Sports and Western Union to help manage their complex IT deployments

- Expertise:

- One of the world's largest concentrations of dedicated, full-time DBA expertise.

- Global Reach & Scalability:

- 24/7/365 global remote support for DBA and consulting, systems administration, special projects or emergency response



# ISO SQL:2003 Standard Datetime

- Standard data types (supported by MySQL):
  - DATE
  - TIME(p)
  - TIMESTAMP(p)
- Standard attributes (not supported by MySQL):
  - WITH TIME ZONE
  - WITHOUT TIME ZONE

# MySQL Additional data types

- YEAR(2)
- YEAR(4)
  - If YEAR is specified with no quantifier, or a quantifier other than 2, MySQL will use YEAR(4)
- DATETIME

# MySQL Datetime data types

- DATE – 3 bytes                      1000-01-01 to 9999-12-31
- DATETIME – 8 bytes
  - 1000-01-01 00:00:00 to 9999-12-31 23:59:59
- TIMESTAMP – 4 bytes
  - 1970-01-01 00:00:00 to 2038-01-18 22:14:07
- TIME – 3 bytes                      -838:59:59 to 838:59:58
- YEAR(2) – 1 byte                      00 to 99
- YEAR(4) – 1 byte                      1901 to 2155

# Time Zones in MySQL Data Types

- Not supported
- However, `TIMESTAMP` is stored transparently in UTC.
  - Uses the `time_zone` system variable to convert
  - When retrieved, converts to current `time_zone` value in the server
  - If `'2009-05-08 17:00:00'` is stored when `time_zone` is set to EST, and later the `time_zone` is changed to CST, the value retrieved will be `'2009-05-08 16:00:00'`

# TIMESTAMP stored in UTC

```
CREATE TABLE time_test (  
ts TIMESTAMP,  
dt DATETIME  
) ENGINE=MyISAM;  
  
INSERT INTO time_test (ts,dt)  
VALUES (NOW(),NOW());  
  
SELECT * FROM time_test;  
  
{change time zone, look again}
```



# The mysqld time zone

- When mysqld starts, it finds the OS time zone and sets `system_time_zone` system variable
- By default, the `time_zone` system variable is set to `SYSTEM`, and `system_time_zone` is used.
- If the OS time zone changes, mysql needs to be restarted for `TIMESTAMP` variables to change.
- Only `TIMESTAMP` data type fields change.
  - It bears repeating!



# Getting the current datetime

- `CURRENT_TIMESTAMP()` is the ISO:SQL 2003 standard function, and is supported by MySQL
- `NOW()` is an alias to `CURRENT_TIMESTAMP`

```
mysql> SELECT NOW(), SLEEP(5), NOW() \G
***** 1. row *****
    NOW(): 2009-12-01 21:42:25
SLEEP(5): 0
    NOW(): 2009-12-01 21:42:25
1 row in set (5.00 sec)
```

- `CURRENT_TIMESTAMP()` is replication-safe.
  - It is calculated at the beginning of a statement and used throughout the statement.

# Getting the current datetime

- `UTC_TIMESTAMP()` is replication-safe and based on `CURRENT_TIMESTAMP`

```
mysql> SELECT
UTC_TIMESTAMP(), SLEEP(5), UTC_TIMESTAMP() \G
***** 1. row *****
    NOW(): 2009-12-01 21:43:12
SLEEP(5): 0
    NOW(): 2009-12-01 21:43:12
1 row in set (5.00 sec)
```

- Because it is based on `CURRENT_TIMESTAMP()`, it is calculated at the beginning of a statement and used throughout the statement.

# Getting the current datetime

- **SYSDATE()** is very familiar to Oracle DBA's/dev's.

```
mysql> SELECT SYSDATE(), SLEEP(5), SYSDATE() \G
***** 1. row *****
SYSDATE(): 2009-12-01 21:44:39
SLEEP(5): 0
SYSDATE(): 2009-12-01 21:44:44
1 row in set (5.00 sec)
```

- **SYSDATE()** is, by default, not safe for replication
  - It uses the system date and time
  - It is calculated on an as-needed basis
  - Will produce different values on a master and slave if the slave's time zone is different

# Making SYSDATE() act like NOW()

- sysdate-is-now
  - static system variable, must restart the server
  - Does not show up in SHOW VARIABLES (or SHOW STATUS)
  - SYSDATE() acts like CURRENT\_TIMESTAMP() and NOW()
  - default is off

# Sources of Information

- If the web/application server has a different time zone than the [master] database server, that can cause problems.
- Webserver: GMT
- Database server: EST (GMT-5)
- An order comes in on Dec. 31<sup>st</sup>, 2009 at 10 pm EST
- If the web/application server determines the time, the order will be logged in Jan 2010
- If the database server determines the time, the order will be logged in Dec 2009

# Ways to Convert in MySQL

- `CONVERT_TZ` to convert times
  - `CONVERT_TZ(<time>,<convert_from>,<convert_to>)`
  - `CONVERT_TZ(NOW(),'-5:00','+0:00');`
  - Offset is from UTC
- Daylight Saving Time can wreak havoc
  - The day DST occurs is different for different countries



# “It's all local” approach

- Just store the times and dates as local time.
  - Events that occur at 6 pm PST and 6 pm EST are considered “the same time”
- This can skew reporting, particularly when estimating peak times.
- This is problematic when a user's perspective changes to a different time zone.
  - My cellphone auto-adjusts my time based on time zone in my location, my computer does not.



# “It's all local” conversion

- Example: Storing 2 different events, at the same absolute time, in EST and CST:

```
CREATE TABLE store_times (  
  st datetime,  
  os tinyint,  
  tz varchar(6) ) ENGINE=MyISAM;
```

```
INSERT INTO store_times (dt, os, tz) VALUES  
  (NOW(), -5, 'EST'), (NOW(), -6, 'CST');
```

```
TIMEDIFF(NOW(), UTC_TIMESTAMP()); --offset
```

```
SELECT CONCAT(dt + INTERVAL os HOUR, ' ', tz)  
  FROM store_times;
```

# “It all works out” approach

- Just store the times and dates one way, and if the data is not 100% accurate for “what day/hour did this come in”, it's still precise, relatively accurate.
  - 3 pm PST and 6 pm EST are “the same time”
- For most companies, relative time is important
  - It's often less important to know that “3 – 6 pm is peak time in each time zone” and more important to know that “peak time is 3 pm – 9 pm EST”.
  - Any day or year straddling is consistent – the most important thing is not to change your cutoff once you make it. If it's midnight EST, then a 10 pm PST order will be considered the next day, but it will always be considered such.

# “Store it all in GMT” approach

- Conversion for storing/retrieving events not in GMT
- It is easier to let a user change their display preference
- Application-aware reports may not match application-unaware reports
  - Peak application traffic may be offset with peak network traffic, CPU load, etc.
- Daylight Saving Time can still be an issue
  - When you “fall back”, 2x volume between 2-3 am
  - Not as much of an issue when you “spring ahead”

# “Store it all in UTC” approach

- All time values are converted for storage/retrieval
- Harder to set up properly
- May be the only way to have true unified reporting
  - Most companies do not want nor need to spend the time and effort necessary for this.

# What most companies do

- By default, the “it will all work out approach”
- If they need to re-consider, “Store it all in GMT”

# Problems

- When the server time zone changes
  - Stop MySQL, change time zone, start mysql
- When the application server(s) and web server(s) are different times from each other or the database server(s).
- What do 2 events at the same time mean?
  - Same server time – ie, 6 pm EST = 5 pm CST
  - Same local time – ie, 6 pm EST = 6 pm CST
  - Same time as HQ or “where reports are run from”?



# The mysqld time zone (repeated slide)

- When mysqld starts, it finds the OS time zone and sets `system_time_zone` system variable
- By default, the `time_zone` system variable is set to `SYSTEM`, and `system_time_zone` is used.
- If the OS time zone changes, mysql needs to be restarted for `TIMESTAMP` variables to change.
- Only `TIMESTAMP` data type fields change.
  - It bears repeating!



# Changing the default MySQL time zone

- Set the timezone option to `mysqld_safe`:

```
[mysqld_safe]  
timezone=tz_name
```

- Or set the TZ environment variable before starting MySQL
- Values are system-dependent
- `SET GLOBAL time_zone=timezone`

# Changing a session's MySQL time zone

- Changing the session affects time values:

```
SET SESSION time_zone="-8:00";
```

```
SELECT NOW(), UTC_TIMESTAMP();
```

```
SELECT * FROM time_test;
```

```
SELECT @@global_time_zone, @@session.time_zone;
```

- Changes for the session only
- Affects NOW(), SYSDATE() and TIMESTAMP
- Does not affect UTC\_TIMESTAMP(), DATETIME

# Using Named Time Zones

- Named time zone = “US/Eastern” or “EST”
- Load information into the mysql system database:
  - time\_zone (tz\_id, use\_leap\_seconds)
  - time\_zone\_name (tz\_id, name)
  - time\_zone\_leap\_second (transition\_time, correction)
  - time\_zone\_transition (tz\_id, transition\_time, tt\_id)
  - time\_zone\_transition\_type (tz\_id, tt\_id, offset, is\_dst, abbreviation)

# Loading Time Zone Info

- Some OS have time zone info, in a directory like `/usr/share/zoneinfo`
  - Linux
  - Sun Solaris
  - FreeBSD
  - Mac OS X
- Use the following command:  

```
mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u user -p mysql
```
- Or download MyISAM tables from <http://dev.mysql.com/downloads/timezones.html>
- Reload periodically (in 2007 DST dates changed)

# Loading Time Zone Info

```
$ mysql_tzinfo_to_sql /usr/share/zoneinfo > tz.sql
```

```
Warning: Unable to load '/usr/share/zoneinfo/Asia/Riyadh87' as time zone. Skipping it.
```

```
Warning: Unable to load '/usr/share/zoneinfo/Asia/Riyadh88' as time zone. Skipping it.
```

```
Warning: Unable to load '/usr/share/zoneinfo/Asia/Riyadh89' as time zone. Skipping it.
```

```
$ mysql -u root -p mysql < tz.sql
```

# Testing Time Zone Info

```
SELECT time_zone_id FROM time_zone_name where  
name='US/Eastern'\G
```

```
SELECT offset, is_DST, abbreviation FROM time_zone_transition_type  
where time_zone_id=561;
```

```
+-----+-----+-----+  
| offset | is_DST | Abbreviation |  
+-----+-----+-----+  
| -14400 |      1 | EDT          |  
| -18000 |      0 | EST          |  
| -14400 |      1 | EWT          |  
| -14400 |      1 | EPT          |  
+-----+-----+-----+  
4 rows in set (0.00 sec)
```

```
SELECT -18000/60/60, -14400/60/60;
```

```
SET SESSION time_zone="US/Central";
```

```
SELECT NOW(),TIMEDIFF(NOW(),UTC_TIMESTAMP());
```



# CONVERT\_TZ

- Can use offsets:

```
SELECT CONVERT_TZ(NOW(), '-5:00', '+0:00');
```

- Can use named time zones if the time zone tables are loaded:

- Can mix both:

```
SELECT CONVERT_TZ(NOW(), 'US/Eastern', 'GMT');
```

- Can use session/global variables:

Can mix both:

```
SELECT NOW(), UTC_TIMESTAMP,  
CONVERT_TZ(NOW(), @@session.time_zone, '+0:00');
```



# Most importantly....

- Be careful!
- Do not forget about existing data
- Mass-conversions can be done like:

```
UPDATE tbl SET fld=fld+INTERVAL offset HOUR
```

- Or use `INTERVAL offset SECOND` and the information from `mysql.time_zone_transition_type`
- only replicated properly in MySQL 5.0+:

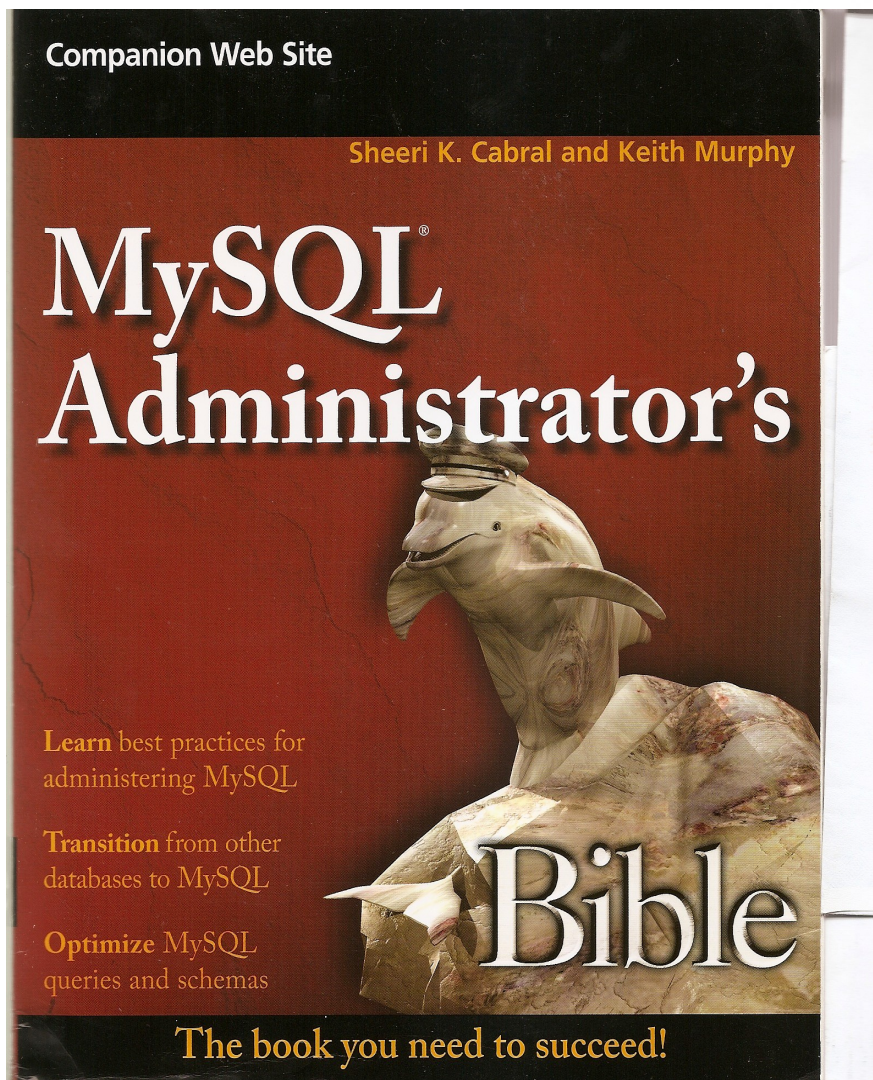
```
CONVERT_TZ(NOW(), @@session.time_zone, '+0:00');
```

# Learn more...

- Experiment and test
- Especially with master/slave and different time zones

<http://dev.mysql.com/doc/refman/5.1/en/time-zone-support.html>

# Thank You.



## Win a signed copy of Sheeri's book.

Leave your business card and you could win a book. We'll invite you to read our blog posts, follow us on twitter, and join our next webinars.

Drawing will be immediately after the talk once all cards are collected.

# Thank You

Questions, Comments, Feedback?

Sheeri Cabral

[cabral@pythian.com](mailto:cabral@pythian.com)

Blog: [www.pythian.com/news/author/sheeri](http://www.pythian.com/news/author/sheeri)

Twitter: @sheeri

Ask me about saving 15% on our Adoption Accelerator for MySQL  
while at MySQL Conference 2010!