Software Developer's JOURNAL

new ideas & solutions for professional programmers

Vol.2 No.13 Monthly Issue 13/2013 (19) ISSN 1734-3933

Building Your Own WordPress Sites from Scratch

Most Exciting Tips and Instructions Written by Best Experts

Table of Contents

Software Developer's JOURNAL
new ideas & solutions for professional programmers

EXTRA +

Expert MySQL Design Practices

Ronald Bradford

All businesses, organizations, websites and governments use data to present information to users. This ranges from contact information of your company, products for sale, where you are and where you have been, and the transactions for your bank account. The right architectural practices for data management ensures successful scalability of your website, application or product managing data. There is no one way to architect a technology solution, however, there are many ways poor design can seriously impact the need for more hardware and resources, and most importantly, take significant time to re-architect causing development of new features to be affected.

Why is there a need for design practices?

Consider this question. How do you describe the need and role of a software architect to a non technical person? This is a generic description for the layperson.

You want to build a new home on a vacant block of land. In order to build a lasting home with all the expectations for today's living, a lot of planning, design, and development occurs long before you can move in, paint the walls, add furniture, cook, entertain and reside.

Your home is built on a foundation that is generally a solid and rigid block of concrete that was carefully planned on a level surface. The plumbing and electrical wiring was carefully prepared before the concrete was set. If the foundation is not level, solid, and meeting the design plan then no matter how much work you put into your home, you will spend more time compensating for the deficiencies of the foundation. Do you spend more and more time and expense trying to compensate, or do you have to give up, tear down the failed attempt and build again?

Designing and developing a website or a software product uses the same approach. With software the amount of investment, the time, and expense is far less than building a home, however the lack of a plan and design can easily lead to a poor product that nobody wants to use.

The availability of open source software, free hosting and ample online resources enables anybody to take an idea and create a software solution with very little effort. If you are building a shack on the beach, you do not invest a lot, you accept the cracks in the walls, the leaky roof, the lack of plumbing. Your first software application will likely be exactly the same. Do you continue to add on, or is there a time you decide you want to start over for your family to live at the beach over the summer with all the expectations of a holiday resort?

Not all plans are the same. If you live in the Amazon, your home is not built with concrete, and has additional different needs for building higher off the ground to avoid tidal flows and to have room for food storage and livestock to have shelter. One size does not suit all. Just like there are shanty towns to million dollar penthouses in large cities, the diversity of websites and software solutions are analogous.

Understanding how to design and develop a successful and scalable software solution takes practice, trial and error, and a good amount of training. Working in a team that includes experienced resources that have completed projects ensures you can learn from the mistakes of others and cultivate a healthy environment for accelerated learning.

It is unfortunate that many startups and open source projects, even the most popular LAMP applications suffer from common design mistakes. These practices misuse MySQL and therefore can shine a poor light on the software and the misconception that MySQL does not scale. Many of today's the largest websites, all the top telecommunication companies, online gaming sites and numerous other companies all rely heavily on your data stored in and managed with MySQL.

Today, many organizations have developers, IT managers and C level executives where the most senior resource has a handful of years of experience or less. They have little or no formal experience in design, data structures, software development, production deployment and testing methodologies. Combined with having never worked in a team environment with more skilled professionals, many software engineers today have a very limited world view of how large and successful products have been developed in the past. This lack of depth in skills reflects how the entire team learns and grows and without the right historical influences too many common beginner mistakes are repeated.

About Expert MySQL Design Practices

This new series by leading MySQL Expert Ronald Bradford helps the software engineer understand, appreciate and develop the right skills and techniques to build scalable software solutions. These proven and reproducible design practices will ensure your use of MySQL to improve performance, scalability and reliability.

These expert design practices are from 25 years of professional experience following formal university qualifications in computer science. All of these practices are written for use with a MySQL based data system however most of the content in these practices predate the existence of the MySQL product and have stood the test of time with emerging technologies and software development approaches. Many practices apply directly to other data stores, whether relational or the new NoSQL products and include working with persistent and non-persistent data storage products.

More information about the series can be found at *http://j.mp/MySQLDP*

DP#4 The importance of using *sql_mode*

What if the data you retrieved from the database did not match the data the application claimed to had successfully stored? How comfortable would your organization feel about your skills and the products that are being used to store important information if data integrity was not guaranteed?

MySQL employs a terrible default technique known as silent truncation where the product determines that it knows about your data better than you. Never has the saying "do not assume" because it makes an "ass" out of "u" and "me" been more applicable.

Customer Example

A HTML form for new customers provide input fields for the customer first and last name. Good design was considered with the HTML form client validation to ensure that each field could not exceed 20 characters in length. However, the database design is different, where the first name is only defined as 10 characters. In most cases this is sufficient, however for first names longer than 10 characters, the data retrieved does not match the data that was apparently successfully stored because there was no SQL error. The following SQL reproduces this situation.

```
DROP TABLE IF EXISTS dp13;
CREATE TABLE dp13 (
 customer_id  INT UNSIGNED NOT NULL AUTO_INCREMENT,
 first_name  VARCHAR(10) NOT NULL,
 last_name  VARCHAR(20) NOT NULL,
 PRIMARY KEY (customer_id)
) ENGINE=InnoDB DEFAULT CHARSET utf8;

INSERT INTO dp13 (customer_id, first_name, last_name)
   VALUES (NULL,'Evangeline','Jones');
INSERT INTO dp13 (customer_id, first_name, last_name)
   VALUES (NULL,'Christopher','Smith');
INSERT INTO dp13 (customer_id, first_name, last_name)
   VALUES (NULL,'Alexander','Bell');

SELECT * FROM dp13;
+-------------+------------+-----------+
| customer_id | first_name | last_name |
+-------------+------------+-----------+
|   1 | Evangeline | Jones  |
|   2 | Christophe | Smith  |
|   3 | Alexander  | Bell   |
+-------------+------------+-----------+
3 rows in set (0.00 sec)
```

As you can see, the first name of Christopher Smith is not actually correctly stored in the database. MySQL DID NOT produce an error message, rather it performed a silent truncation of the data.

Defining *sql_mode*

To demonstrate what level of data integrity you should expect with MySQL, you must define the *sql_mode* configuration option. The following example demonstrates the dynamic syntax for a given connection and the error you should expect.

```
SET SESSION sql_mode='STRICT_ALL_TABLES,NO_ZERO_DATE,NO_ZERO_IN_DATE';

TRUNCATE TABLE dp13;
INSERT INTO dp13 (customer_id, first_name, last_name)
   VALUES (NULL,'Christopher','James');
ERROR 1406 (22001): Data too long for column 'first_name' at row 1
SELECT * FROM dp13;
Empty set (0.00 sec)
```

When MySQL is first installed the following configuration option should always be added to all environments as a default.

```
$ cat /etc/my/cnf
[mysqld]
sql_mode=STRICT_ALL_TABLES,NO_ZERO_DATE,NO_ZERO_IN_DATE,NO_ENGINE_
SUBSTITUTION
```

For more information, refer to the MySQL Reference Manual for sql_mode at http://dev.mysql.com/doc/refman/5.6/en/server-sql-mode.html

NOTE: MySQL provides many different options with *sql_mode*. Careful consideration is needed to determine which options are best for your application. Some options help in providing syntax and compatibility with other database products however these can affect and even break existing products written specifically for MySQL.

MySQL Warnings

The underlying cause of this loss of data integrity is how MySQL handles success and error conditions with SQL Statements. There are the obvious success and failure states, however MySQL has a third state known was warnings, or more specifically success with warnings. As the use of warnings is uncommon with other data store products, many applications, developers and programming languages ignore checking for warnings, or are simply unaware of this inbuilt feature.

Using the MySQL command line client, you can get a visual indication of warnings following an SQL statement which then help the need for reviewing what warnings occurred.

```
INSERT INTO dp13 (customer_id, first_name, last_name)
VALUES (NULL,'Christopher','Smith');
Query OK, 1 row affected, 1 warning (0.00 sec)

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

When using PHP there is no indication of SQL warnings unless you specifically check after every SQL statement. For example:

```
<?php
 $con = mysqli_connect('localhost', 'scott', 'sakila', 'design');
 if (mysqli_connect_errno()) {
 print 'Failed to connect to MySQL: ' . mysqli_connect_error() .
"\n";
 exit(1);
 }
 if (!mysqli_query($con, 'INSERT INTO dp13 (customer_id, first_name,
last_name) ' .
       'VALUES (NULL,"Christopher","Holt") ')) {
 print 'Failed to insert data: ' . mysqli_error($con) . "\n";
 }
 if (($warnings = mysqli_warning_count($con)) > 0) {
 if ($rs = mysqli_query($con, "SHOW WARNINGS")) {
  $row = mysqli_fetch_row($rs);
  printf("%s (%d): %s\n", $row[0], $row[1], $row[2]);
  mysqli_free_result($rs);
 }
 }
 mysqli_close($con);
?>
```

The best recommendation is to avoid all situations where MySQL can produce a warning and does not provide the best possible data integrity.

Refer to the MySQL Reference Manual for more information on SHOW WARNINGS at

http://dev.mysql.com/doc/refman/5.6/en/show-warnings.html

The Larger Problem

This underlying problem is actually more difficult to correct for an existing production system than enabling the correct *sql_mode* configuration option. Using the customer example, the identification of any rows that are 10 characters in length could be valid, or may have been truncated. There is no easy way to obtain the actual value that was originally submitted. The use of the correct numerical data type (DP #14) can provide a check constraint for values, however it can also suffer from the same truncation problem. You especially hope that this does not affect your payroll, your frequent flyer points balance or your accumulated score from your favorite online game.

The solution is to avoid the problem of producing incorrect data.

Review

While this example is using a character data type, field truncation can also occur with numeric and date data types. The use of applicable *sql_mode* configuration settings is a critical MySQL design practice to ensure adequate data integrity that all systems need to implement.

More References

- *http://ronaldbradford.com/blog/why-sql_mode-is-important-part-i-2008-07-17/*
- *http://ronaldbradford.com/blog/why-sql_mode-is-important-2011-06-01/*
- *http://ronaldbradford.com/blog/why-sql_mode-is-essentialeven-when-not-perfect-2012-02-16/*

- *http://ronaldbradford.com/blog/best-practices-additional-usersecurity-2010-06-03/*
- *http://ronaldbradford.com/blog/dont-assume-dataintegrity-2010-03-06/*
- *http://effectivemysql.com/presentation/mysql-idiosyncrasiesthat-bite/*

DP#8 The disadvantages of row at a time processing

It can be hard for software engineers to understand the following principle, however it is very important for improving performance and obtaining immediate scalability options. The principle is **"Do Less Work"**. That is, run less SQL statements.

Just one method to achieving the execution of less SQL statements is to eliminate Row At a Time (RAT) processing. In simple terms, do not perform identical repeating SQL statements in a loop. Relational algebra, and the Structure Query Language (SQL) specification is specifically designed to work with sets of data, or as I describe, Chunk At a Time (CAT) processing.

Customer Example

Your online social media website lets you send messages to multiple friends at one time. You enter the message, select the friends you want to receive the message and click send. While the user waits a moment and gets a success message, behind the scenes the application runs the following SQL statements to record your request.

```
START TRANSACTION;
INSERT INTO dp8_message_sent(message_id, user_id, message, created)
VALUES(NULL, 42, 'Hey guys. Just a reminder. The poker game will start on Friday at
8pm.',NOW());
SELECT @message_id :=LAST_INSERT_ID();
INSERT INTO dp8_message_recipient(message_id, from_user_id, to_user_id, status)
VALUES (@message_id,42,16,'New');
UPDATE dp8_user_notification
SET new_message = 'Y',
  new_message_count = new_message_count + 1
WHERE user_id = 16;
INSERT INTO dp8_message_recipient(message_id, from_user_id, to_user_id, status)
VALUES (@message_id,42,18,'New');
UPDATE dp8_user_notification
SET new_message = 'Y',
  new_message_count = new_message_count + 1
WHERE user_id = 18;
INSERT INTO dp8_message_recipient(message_id, from_user_id, to_user_id, status)
VALUES (@message_id,42,99,'New');
UPDATE dp8_user_notification
SET new_message = 'Y',
  new_message_count = new_message_count + 1
WHERE user_id = 99;
INSERT INTO dp8_message_recipient(message_id, from_user_id, to_user_id, status)
VALUES (@message_id,42,21,'New');
UPDATE dp8_user_notification
SET new_message = 'Y',
  new_message_count = new_message_count + 1
WHERE user_id = 21;
INSERT INTO dp8_message_recipient(message_id, from_user_id, to_user_id, status)
VALUES (@message_id,42,62,'New');
UPDATE dp8_user_notification
SET new_message = 'Y',
  new_message_count = new_message_count + 1
WHERE user_id = 62;
COMMIT;
```

You can define the table structures used in this example with:

```
DROP TABLE IF EXISTS dp8_message_sent;
CREATE TABLE dp8_message_sent(
 message_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
 user_id  INT UNSIGNED NOT NULL,
 message VARCHAR(500) NOT NULL,
 created  DATETIME NOT NULL,
 PRIMARY KEY(message_id),
 KEY(user_id)
) ENGINE=InnoDB CHARSET utf8;
DROP TABLE IF EXISTS dp8_message_recipient;
CREATE TABLE dp8_message_recipient(
 message_id  INT UNSIGNED NOT NULL,
 from_user_id  INT UNSIGNED NOT NULL,
 to_user_id  INT UNSIGNED NOT NULL,
 status ENUM('New','Read','Deleted') NOT NULL,
 PRIMARY KEY(message_id,to_user_id),
 KEY(from_user_id)
) ENGINE=InnoDB CHARSET utf8;
DROP TABLE IF EXISTS dp8_user_notification;
CREATE TABLE dp8_user_notification(
 user_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
 new_message ENUM ('Y','N') NOT NULL DEFAULT 'N',
 new_message_count INT UNSIGNED NOT NULL DEFAULT '0',
 PRIMARY KEY(user_id)
) ENGINE=InnoDB CHARSET utf8;
```

The average software developer may not see the problem here. In your test environment you executed 12 SQL statements and the code worked fine, i.e. it met the requirements for the function. However, while producing the correct result, this is a poor code approach.

This example shows not one repeating query, but two. Lucky you only sent the message to a few friends. If you sent it to 200 friends, you have a significant number more SQL statements to execute. This time the code executes 402 SQL statements for the same feature. The response time to the user is longer, the application connection has to remain open longer and the database has more work to do.

This popular site is sending thousands of messages per second, so the problem is compounded to produce an excess of unnecessary work, not just for the database, but the application web server connections as their are longer open requests.

The solution is straightforward. **Remove repeating queries**. It's not rocket science. This is a simple design practice I teach as the problem is evident on most consulting engagements. Popular products including Drupal and WordPress also implement this poor practice and developers that extend these products propagate this poor practice excessively. If this development approach can be easily found in a few common functions, in it generally a clear indicator this problem can be found throughout the code.

Here is the same operation performed efficiently.

```
START TRANSACTION;
INSERT INTO dp8_message_sent(message_id, user_id, message, created)
VALUES(NULL, 42, 'Hey guys. Just a better reminder. The poker game will start on
Friday at 8pm.',NOW());
INSERT INTO dp8_message_recipient(message_id, from_user_id, to_user_id, status)
VALUES
(LAST_INSERT_ID(),42,16,'New'),
(LAST_INSERT_ID(),42,18,'New'),
(LAST_INSERT_ID(),42,99,'New'),
(LAST_INSERT_ID(),42,21,'New'),
(LAST_INSERT_ID(),42,62,'New');
UPDATE dp8_user_notification
SET new_message = 'Y',
  new_message_count = new_message_count + 1
WHERE user_id IN (16,18,99,21,62);
COMMIT;
```

No matter how many friends you send a message to, only 3 SQL statements are executed. In these queries we see two different examples of leveraging the set capabilities of SQL to perform chunk at a time processing. We discuss the benefits of the multi-values INSERT in more detail with DP#10.

Customer Example 2

The following is a simple example for an online store processing function. Your shipping provider provides an update of all packages that were processed by them for a given date. For each packing tracking code that you have recorded with orders they provide a last known status. For example if the package were successfully delivered, is in transit, or has been returned.

A typical and very common developer process is to open the file, read each line looping through all the rows, and for each row perform a single update without using transactions like:

```
open file
for each line
do
 UPDATE dp8_order
 SET last_shipping_status=?, last_shipping_update=?
 WHERE tracking_code=?;
done
close file
```

As the size of data increases so does the processing time because you execute one statement per row. When there are are 10 packages, 10 SQL statements, when there are 300,000 packages, there are 300,000 SQL statements.

This batch process does not have a user response time requirement like online applications where performance is key to retaining your users. However, while eliminating row at a time processing is critical for providing a better user experience it is also just as important for batch processing.

```
stmt = 'INSERT INTO dp8_batch_tracking (batch_id, tracking_code, status, last_
update) VALUES'
sep = ''

open file
for each line
do
 stmt = stmt + sep + '(42, ?, ?, ?)'
 sep = ''
done
close file

START TRANSACTION;
EXECUTE IMMEDIATE stmt;
UPDATE dp8_order o, dp8_batch_tracking bt
SET o.last_shipping_status=bt.status, o.last_shipping_update=bt.last_update
WHERE bt.batch_id = 42
AND bt.tracking_code = o.tracking_code;
--DELETE FROM batch_tracking WHERE batch_id=42;
COMMIT;
```

This example removes the one query per row problem, and results in just 2 SQL queries for processing the file regardless of size.

NOTE: In MySQL there is a limit to the length of the SQL statement (i.e. The INSERT). This can be adjusted with the max_allowed_packet variable which can be set per SQL statement. If you are processing very large files, the following code would be modified to perform the INSERT for 'n' records, however only a single UPDATE is still required. See DP#10 for an example of using max_allowed_packet.

This example shows just one way to optimize this operation with the least amount of code changes to the existing application. An even better approach is to use the LOAD DATA INFILE syntax to populate the batch table directly. This requires additional SQL privileges and file system access and hence is a more complex solution.

Why is the impact of removing these repeating queries so significant? To answer that question we need to look at the anatomy of the execution of an SQL statement.

SQL statement workflow

To the end user viewing your website with a browser, the result of clicking send on a webpage is a [short] delay before the expected results are displayed or the applicable action occurs. Behind the scenes an extensive amount of work is performed. For anybody that has looked at a waterfall chart showing the response from a web server, there is a far greater complexity for rendering the page you are looking at. The following article gives a good introduction to browser waterfall graphs -- *http://www.webperformancetoday.com/2010/07/09/waterfalls-101/*. While the browser may render 100s of files, it is generally the first response, the actual page that is involved in executing the necessary SQL statements, and the focus of this design practice.

When a HTTP request is made to a web container the application performs a number of operations to satisfy the request and produce a response. With your application, regardless of the programming language, access to the MySQL database is performed by SQL statements. Each statement is passed to the language specific MySQL connector required with your web container. For example, when using the Apache HTTP server and the PHP programming language, the MySQL Native Driver (mysqlnd) is the necessary MySQL Connector. There are connectors for the popular languages including C, C++, Java, .Net, Python, Ruby etc.

Here is a short summarized list of what occurs with all SQL statements.

The application executes an SQL statement.

1. The MySQL client connector accepts the SQL statement then connects across the network to the specified MySQL server and passes the SQL statement to the MySQL server.
2. The MySQL server processes all incoming SQL statements in individual threads, so many SQL statements can be executed concurrently.
3. The MySQL server first parses the SQL statement for valid SQL syntax, and produces a summarized structure of the tables and columns used in the SQL statement.
4. The MySQL server performs a security check to ensure the user that is requesting this SQL statement has the necessary privileges to be able to access/modify the information requested in the SQL statement.
5. The MySQL server then passes the parsed SQL statement to the MySQL query optimizer. This is heart of the decision making process where the cost-based optimizer creates a decision tree, evaluates the various options by pruning the expensive paths to produce the optimal Query Execution Plan (QEP).
6. The MySQL server then passes the QEP to the applicable MySQL storage engine(s) to perform the physical work of storing and/or retrieving the data for the given SQL statement.
7. Depending on the type of query, the MySQL server may have to do additional work, for example to join multiple tables, sort results etc.
8. When the MySQL server has produced the results for the SQL statement, these results are send back across the network to the application server.

NOTE: This is a simplified representation of the execution path of an SQL statement in MySQL. The use of the MySQL Query Cache discussed in QP#9 introduces additional steps and can also produce a significantly simplified and faster execution path.

To summarize, every SQL statement is passed to the MySQL server, the network overhead of points 2 and 9 are the most expensive amount of time in a well tuned MySQL application. This alone is the greatest reason to run less SQL statements.

Every SQL statement is parsed, checked for application permissions and optimized before execution. This is most applicable for example when combining INSERT statements with multiple VALUES clauses. In addition to saving the network round trip, this overhead is also eliminated by combining SQL statements.

Universal Application

This same principle can be applied to other products that process data. For example, memcache is a popular product to improve performance and scalability of your application by providing a memory caching layer. The following figures are for an example benchmark with 28 objects in memcache using two cloud servers in Rackspace Cloud.

Using an individual get call 28 times sequentially in a single PHP file, simulating a client example, the total response time of the benchmarked ranged from 24 to 56 milliseconds. Using the same configuration with a single multi-get call for the same 28 objects the results ranged from 4 to 7 milliseconds.

It does not require a graph to see the **6x-10x** improvement in performance by eliminating row at a time processing. The saving of 20-50 milliseconds may seem small, however when multiplied in environments with thousands of concurrent users, thousands of times per second, has a large impact on resources.

Recap

This principle shows a simple technique for reducing the number of SQL statements by eliminate repeating queries. As a goal of **"Do Less Work",** this is only one case. DP#16 discusses several other query saving techniques that can eliminate repeating and unwanted queries providing improved performance.

More References

- *http://ronaldbradford.com/blog/the-rat-and-thecat-2006-08-24/*
- *http://ronaldbradford.com/blog/optimizing-sql-performance-the-art-of-elimination-2010-07-08/*
- *http://ronaldbradford.com/blog/we-need-morecats-2009-08-22/*
- *http://ronaldbradford.com/blog/simple-lessons-in-improving-scalability-2011-02-16/*