



Integrated Cloud Applications & Platform Services

## MySQL Performance Tuning

Activity Guide

D61820GC40

Edition 4.0 | September 2018 | D102562

Learn more from Oracle University at [education.oracle.com](https://education.oracle.com)



ORACLE®

## **Author**

Mark Lewin

## **Technical Contributors and Reviewers**

John Kehoe  
Jesper Wisborg Krogh  
Kathy Forte  
Kimseong Loh  
Bill Millar  
Ron Soltani  
Megha Singhvi  
Travis Rupe  
Margaret Fisher  
Pedro Pinheiro  
Malika Agarwal

## **Editors**

Arijit Ghosh  
Aju Kumar  
Aishwarya Menon  
Vijayalakshmi Narasimhan

## **Graphic Editors**

Kavya Bellur  
Seema Bopaiyah  
Prakash Dharmalingam

## **Publishers**

Pavithran Adka  
Asief Baig  
Srividya Rameshkumar  
Jobi Varghese

**Copyright © 2018, Oracle and/or its affiliates. All rights reserved.**

## **Disclaimer**

This document contains proprietary information and is protected by copyright and other intellectual property laws. You may copy and print this document solely for your own use in an Oracle training course. The document may not be modified or altered in any way. Except where your use constitutes "fair use" under copyright law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.

The information contained in this document is subject to change without notice. If you find any problems in the document, please report them in writing to: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. This document is not warranted to be error-free.

## **Restricted Rights Notice**

If this documentation is delivered to the United States Government or anyone using the documentation on behalf of the United States Government, the following notice is applicable:

### **U.S. GOVERNMENT RIGHTS**

The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

## **Trademark Notice**

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

# Table of Contents

---

<b>Practices for Lesson 1: Introduction.....</b>	<b>7</b>
Practices for Lesson 1: Overview .....	8
Practice 1-1: Course Environment Overview .....	9
<b>Practices for Lesson 2: Performance Tuning Concepts.....</b>	<b>15</b>
Practices for Lesson 2: Overview .....	16
Practice 2-1: Quiz – Performance Tuning Concepts .....	17
Solution 2-1: Quiz – Performance Tuning Concepts .....	19
<b>Practices for Lesson 3: Performance Tuning Tools .....</b>	<b>21</b>
Practices for Lesson 3 .....	22
Practice 3-1: Using MySQL Command-Line Monitoring Tools.....	23
Solution 3-1: Using MySQL Command-Line Monitoring Tools.....	26
Practice 3-2: Using MySQL Enterprise Monitor.....	36
Solution 3-2: Using MySQL Enterprise Monitor.....	44
Practice 3-3: Using Benchmark Tools .....	45
Solution 3-3: Using Benchmark Tools .....	47
Practice 3-4: Using Linux System Monitoring Tools.....	51
Solution 3-4: Using Linux System Monitoring Tools .....	53
<b>Practices for Lesson 4: Performance Schema.....</b>	<b>59</b>
Practices for Lesson 4: Overview .....	60
Practice 4-1: Examining the Performance Schema Configuration .....	61
Solution 4-1: Examining the Performance Schema Configuration .....	62
Practice 4-2: Querying the Performance Schema .....	67
Solution 4-2: Querying the Performance Schema .....	69
Practice 4-3: Using sys to Work with the Performance Schema.....	73
Solution 4-3: Using sys to Work with the Performance Schema .....	74
Practice 4-4: Using MySQL Workbench for Performance Schema Configuration, Monitoring, and Reporting .....	82
Solution 4-4: Using MySQL Workbench for Performance Schema Configuration, Monitoring, and Reporting .....	85
<b>Practices for Lesson 5: General Server Tuning.....</b>	<b>87</b>
Practices for Lesson 5: Overview .....	88
Practice 5-1: Estimating Total Thread Memory Usage .....	89
Solution 5-1: Estimating Total Thread Memory Usage .....	90
Practice 5-2: Managing the Number of Client Connections .....	92
Solution 5-2: Managing the Number of Client Connections .....	94
Practice 5-3: Investigating the Effects of Multiple Simultaneous Connections .....	102

Solution 5-3: Investigating the Effects of Multiple Simultaneous Connections .....	104
Practice 5-4: Investigating the Effects of Thread Caching .....	107
Solution 5-4: Investigating the Effects of Thread Caching .....	109
<b>Practices for Lesson 6: Tuning Tables, Files, and Logs .....</b>	<b>115</b>
Practices for Lesson 6: Overview .....	116
Practice 6-1: Sizing the Table Cache .....	117
Solution 6-1: Sizing the Table Cache .....	120
Practice 6-2: Tuning the Open Files Limit.....	127
Solution 6-2: Tuning the Open Files Limit.....	129
Practice 6-3: Sizing the Binary Log Cache.....	134
Solution 6-3: Sizing the Binary Log Cache.....	137
Practice 6-4: Identifying I/O Hotspots with MySQL Workbench .....	144
Solution 6-4: Identifying I/O Hotspots with MySQL Workbench .....	146
<b>Practices for Lesson 7: Tuning InnoDB .....</b>	<b>147</b>
Practices for Lesson 7 .....	148
Practice 7-1: Investigating the Effects of Log Files on Transactions .....	149
Solution 7-1: Investigating the Effects of Log Files on Transactions .....	151
Practice 7-2: Using SHOW ENGINE INNODB STATUS.....	156
Solution 7-2: Using SHOW ENGINE INNODB STATUS.....	160
Practice 7-3: Monitoring InnoDB Metrics in the Information Schema.....	170
Solution 7-3: Monitoring InnoDB Metrics in the Information Schema.....	172
Practice 7-4: Evaluating InnoDB Buffer Pool Size.....	179
Solution 7-4: Evaluating InnoDB Buffer Pool Size.....	181
<b>Practices for Lesson 8: Optimizing Your Schema.....</b>	<b>187</b>
Practices for Lesson 8 .....	188
Practice 8-1: Comparing the Effects of Table Normalization on Query Performance .....	189
Solution 8-1: Comparing the Effects of Table Normalization on Query Performance .....	192
Practice 8-2: Choosing the Correct Data Type.....	199
Solution 8-2: Choosing the Correct Data Type.....	201
Practice 8-3: Compressing Tables.....	208
Solution 8-3: Compressing Tables .....	210
Practice 8-4: Partitioning.....	217
Solution 8-4: Partitioning .....	219
<b>Practices for Lesson 9: Monitoring Queries .....</b>	<b>227</b>
Practices for Lesson 9: Overview .....	228
Practice 9-1: Monitoring Statements.....	229
Solution 9-1: Monitoring Statements.....	230
Practice 9-2: Using the Slow Query Log .....	236
Solution 9-2: Using the Slow Query Log .....	237

Practice 9-3: Identifying Slow Queries with MySQL Enterprise Monitor Query Analyzer .....	241
Solution 9-3: Identifying Slow Queries with MySQL Enterprise Monitor Query Analyzer .....	251
Practice 9-4: Identifying Slow Queries with sys Views .....	252
Solution 9-4: Identifying Slow Queries with sys Views .....	253
Practice 9-5: Using MySQL Workbench Query Statistics .....	256
Solution 9-5: Using MySQL Workbench Query Statistics .....	261
<b>Practices for Lesson 10: Optimizing Queries .....</b>	<b>263</b>
Practices for Lesson 10: Overview .....	264
Practice 10-1: Understanding the Query Execution Plan with the EXPLAIN Statement .....	265
Solution 10-1: Understanding the Query Execution Plan with the EXPLAIN Statement .....	268
Practice 10-2: Improving the Performance of a Query .....	282
Solution 10-2: Improving the Performance of a Query .....	284
Practice 10-3: Tracing the Optimizer .....	289
Solution 10-3: Tracing the Optimizer .....	293
<b>Practices for Lesson 11: Optimizing Locking Operations.....</b>	<b>295</b>
Practices for Lesson 11: Overview .....	296
Practice 11-1: Troubleshooting Blocked Queries .....	297
Solution 11-1: Troubleshooting Blocked Queries .....	299
Practice 11-2: Investigating Metadata Locks .....	312
Solution 11-2: Investigating Metadata Locks .....	314
<b>Practices for Lesson 12: Tuning Replication .....</b>	<b>319</b>
Practices for Lesson 12: Overview .....	320
Practice 12-1: Diagnosing Replication Lag by Using Binary Log File Name and Position .....	321
Solution 12-1: Diagnosing Replication Lag by Using Binary Log File Name and Position .....	323
Practice 12-2: Diagnosing Replication Lag by Using GTID Sets .....	330
Solution 12-2: Diagnosing Replication Lag by Using GTID Sets .....	332
<b>Practices for Lesson 13: Conclusion .....</b>	<b>339</b>
Practices for Lesson 13: Overview .....	340
<b>Appendix A: Performance Enhancements in MySQL 5.7 .....</b>	<b>341</b>
Appendix A: Overview.....	342
Features Added in MySQL 5.7 .....	343
Features Deprecated or Removed in MySQL 5.7 .....	345
<b>Appendix B: Server Tuning Checklist .....</b>	<b>347</b>
Appendix B: Overview.....	348
<b>Appendix C: Using MySQL with Oracle Tools .....</b>	<b>351</b>
Appendix C: Overview .....	352
Using Oracle Enterprise Manager with the MySQL Plug-In.....	353
Using Oracle SQL Developer to Connect to MySQL Databases .....	368

Unauthorized reproduction or distribution prohibited. Copyright© 2019, Oracle and/or its affiliates.

GANG LIU (gangli@baylorhealth.edu) has a non-transferable license  
to use this Student Guide.

## **Practices for Lesson 1: Introduction**

## Practices for Lesson 1: Overview

---

### Overview

This practice introduces you to the tools, sample databases, and other resources that you will use in this course.

#### Assumptions

- The operating system is Oracle Linux 7.x.
- You can log in as the `root` user (password: `oracle`).
- The following software products are installed:
  - MySQL Enterprise Edition server (version 5.7.20 or later)
  - MySQL Enterprise Monitor Service Manager (version 4.0.1 or later)
  - MySQL Workbench (version 6.3 or later)
  - MySQL Enterprise Backup (version 4.1 or later)
  - SysBench (version 0.4.12)
- The following databases are installed:
  - `employees`
  - `perf`
  - `sakila`
  - `world`
  - `world_NonNorm`
- The `/root/scripts` directory contains test scripts that create a load against the MySQL server for the practices.
- The `/root/labs` directory contains other resources such as spreadsheets and configuration files, and `*.lab` files that contain the solution steps for each activity.
- The `/root/labs` directory also contains the JSON-DataView Firefox add-on (`json_dataview-1.18-fx.xpi`).

## Practice 1-1: Course Environment Overview

### Overview

This practice guides you through an evaluation of the course environment. It assumes that you are using the Linux operating system environment that is provided in Oracle classrooms. For non-Oracle classrooms, you might need to make some adjustments to file locations.

### Duration

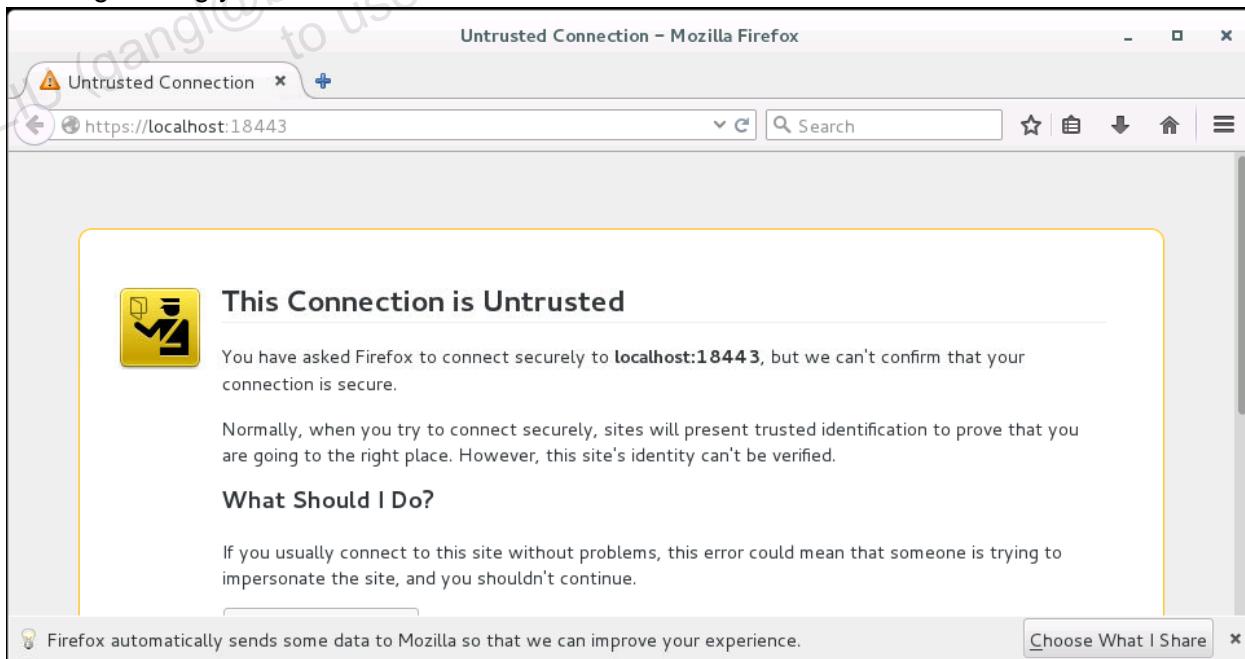
This practice should take you approximately 15 minutes to complete.

### Tasks

1. Open a terminal window as the Linux `root` user.
2. This course uses the MySQL Enterprise Server version 5.7.20 or later, which is installed for you. Confirm the installation by executing the following command at a Linux terminal prompt and receiving the results shown:

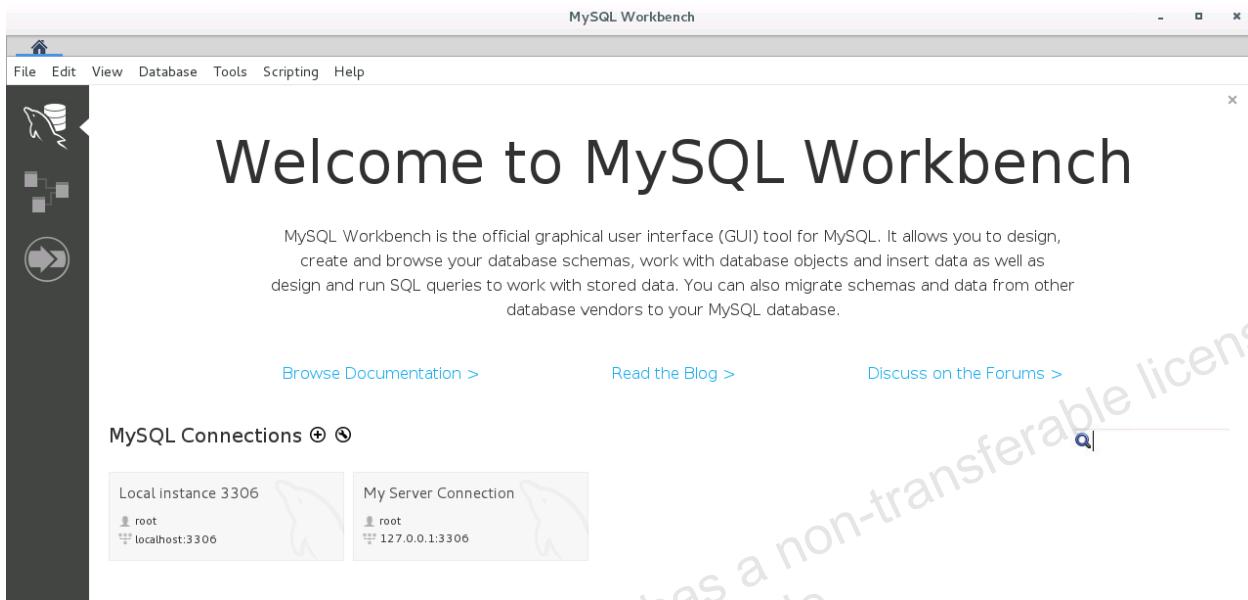
```
# mysql --version
mysql Ver 14.14 Distrib 5.7.20, for linux-glibc2.12 (x86_64)
using EditLine wrapper
```

3. You will use MySQL Enterprise Monitor for various server-monitoring activities during the course. Verify that MySQL Enterprise Monitor is installed by selecting the Applications > Programming > MySQL Enterprise Monitor menu option on the Linux desktop. The Firefox browser should display and attempt to connect to `http://localhost:18443`, with a message telling you that the connection is untrusted:



Close Firefox for now: you will configure MySQL Enterprise Monitor in the practice titled “Using MySQL Enterprise Monitor” for the lesson titled “Performance Tuning Tools.”

- This course uses MySQL Workbench version 6.3 or later, which is installed for you. Confirm that MySQL Workbench is installed correctly by selecting the Applications > Programming > MySQL Workbench menu option on the Linux desktop. The MySQL Workbench application opens and displays the Welcome to MySQL Workbench page:



- Close MySQL Workbench.
- This course does not use MySQL Enterprise Backup, but it is available if you or your instructor wants to explore its capabilities. Confirm the installation of MySQL Enterprise Backup by entering the following command at the Linux terminal prompt and receiving the results shown:

```
# mysqlbackup --version
MySQL Enterprise Backup version 4.1.0 Linux-2.6.39-
400.215.10.el5uek-x86_64 [2017/03/01]
Copyright (c) 2003, 2017, Oracle and/or its affiliates. All
Rights Reserved.

171116 15:28:05 MAIN      INFO: A thread created with Id
'139628084918080'

Run mysqlbackup --help for help information.
```

- The Sysbench tool enables you to benchmark the MySQL server. You will learn more about Sysbench in the activities for the lesson titled “Performance Tuning Tools.” Confirm that Sysbench is installed by executing the following command at the Linux terminal prompt and receiving the results shown:

```
# sysbench --version
sysbench 0.4.12
```

- You will use the mysqlslap command-line tool throughout the course to simulate activity on the MySQL server from multiple clients. Verify that you can execute mysqlslap by

issuing the following command at the Linux terminal prompt and receiving the results shown:

```
# mysqlslap -V
mysqlslap Ver 1.0 Distrib 5.7.20, for linux-glibc2.12 (x86_64)
```

9. You will use several Linux command-line tools that the Oracle Linux 7 distribution includes by default. Verify that you can run the `iostat`, `vmstat`, `sar`, and `top` tools by entering the following commands at the Linux terminal prompt and receiving the results shown:

```
# iostat -V
sysstat version 10.1.5
(C) Sebastien Godard (sysstat <at> orange.fr)
# vmstat -V
vmstat from procps-ng 3.3.10
# sar -V
sysstat version 10.1.5
# top -V
procps-ng version 3.3.10
Usage:
    top -hv | -bcHiOSs -d secs -n max -u|U user -p pid(s) -o field
    -w [cols]
```

10. The following databases are installed for the practices:

- employees
- perf
- sakila
- world
- *world\_NonNorm*

Verify that these databases exist by executing the following command at a Linux terminal prompt that lists the contents of the MySQL data directory. Confirm the presence of the items in bold italics:

```
# ls /var/lib/mysql
auto.cnf          ib_buffer_pool   mysql.sock.lock      server-key.pem
ca-key.pem        ibdata1          performance_schema  sys
ca.pem            ib_logfile0     perf                world
client-cert.pem  ib_logfile1     private_key.pem    world_NonNorm
client-key.pem   ibtmp1          public_key.pem
host-name.pid    mysql           sakila
employees       mysql.sock      server-cert.pem
```

11. The course requires you to use a number of pre-written scripts to test your server's performance. These include shell, Python, and SQL scripts. Verify that the following scripts are present in the `/root/scripts` directory:

```
# ls /root/scripts
10min.sql          perf-sysbench-prepare.sh
```

binlog-cache-test-create.sql	perf-t-sysbench-run.sh
binlog-cache-test-run.sql	query_many_tables.sql
create_city_no_partition.sql	random-lang-denorm.py
create_city_partition.sql	random-lang-norm.py
create_compression_table.sql	random-pop-denorm.py
create_many_tables.sql	random-pop-norm.py
dept_employees.sql	repl-insert.sql
innodb_buffer.sql	schema_test_db.sql
innodb_query.sql	slap-test.sh
perf-t-sysbench-clean.sh	thread_cache_hit_rate.sql

12. The /root/labs directory should contain the following files, which are required for specific practices:

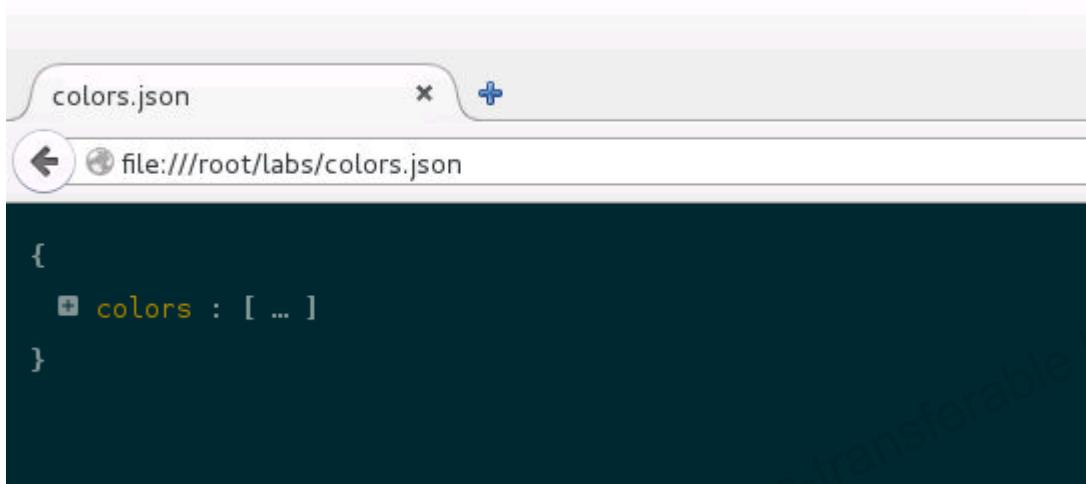
```
# ls -p | grep -v /
colors.json
json_dataview-1.18-fx.xpi
Lesson05Practice.ods
Lesson06Practice.ods
Lesson08Practice.ods
repl.cnf
```

13. This Activity Guide contains practices and solutions for each lesson. The course provides a “catch-up script” (\*.lab) file for every lesson that requires you to type command-line commands and SQL statements. You can use these lab scripts to copy and paste the necessary commands and statements to a terminal window if you are running out of time to complete a practice. Confirm the presence of the catch-up scripts in the /root/labs/catchup directory:

```
# ls /root/labs/catchup
practice_01-1.lab
practice_03-1.lab
practice_03-2.lab
practice_03-3.lab
...
```

14. In the lesson titled “Optimizing Queries,” you examine the MySQL optimizer trace information in JavaScript object notation (JSON) format. Install the JSON-DataView extension for Mozilla Firefox to make this data easier to work with.
- Open Mozilla Firefox.
  - Press the ALT key to display the Firefox menu.
  - Select the File > Open File menu option, browse to the /root/labs/json\_dataview-1.18-fx.xpi file, and click Open. The Software Installation dialog box appears.
  - Verify that the item to install is JSON-DataView and click Install Now.

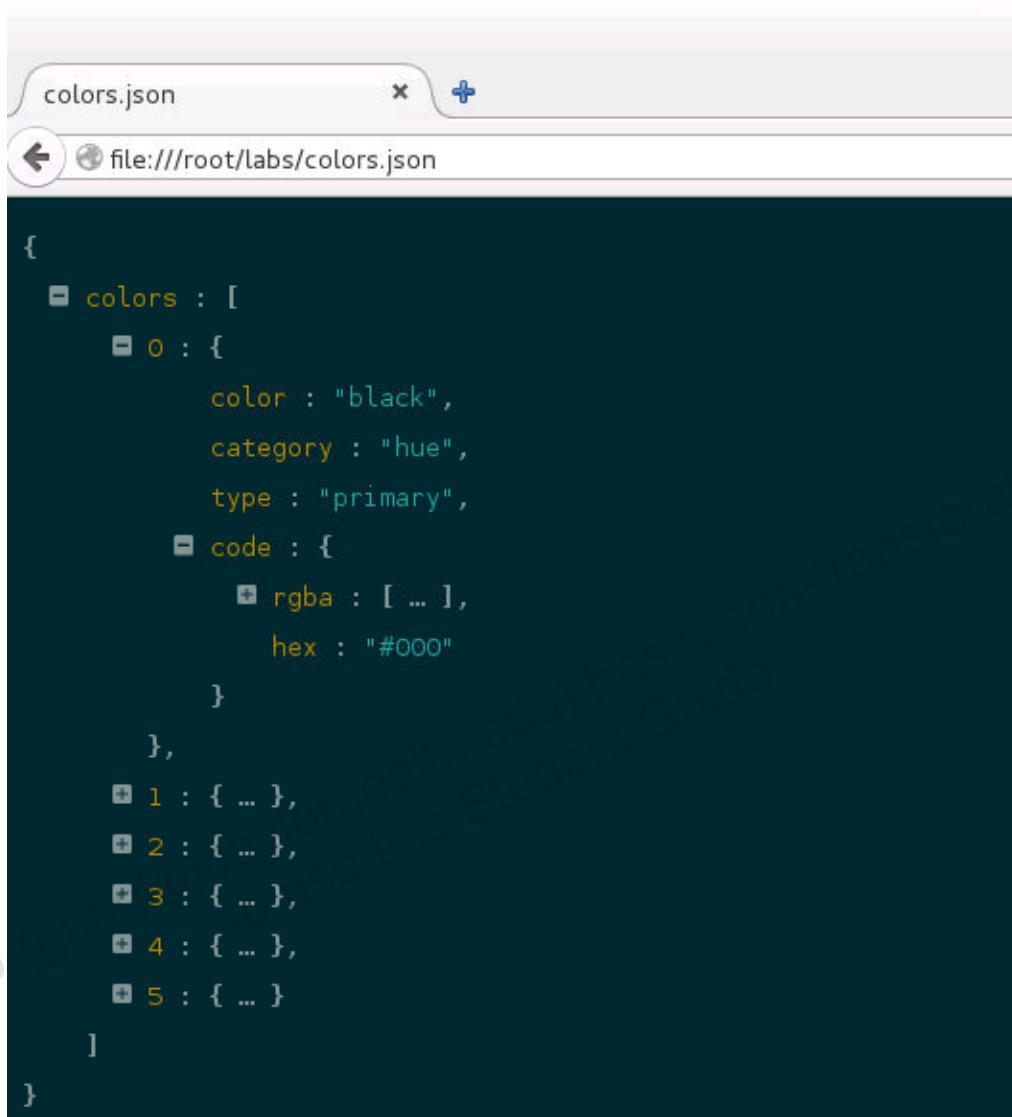
- e. When prompted to restart Firefox, click the Restart Now button.
- f. When Firefox relaunches, press the ALT key to display the menu.
- g. Select the File > Open File menu option, browse to the `/root/labs/colors.json` file, and click Open. The file displays in the browser as follows:



A screenshot of a Firefox browser window. The title bar says "colors.json". The address bar shows "file:///root/labs/colors.json". The main content area displays the JSON file's contents:

```
{  
  "colors" : [ ... ]  
}
```

- h. Click the plus symbol next to the `colors` entry on the browser page and verify that you can expand this and the subsequent nodes in the JSON hierarchy:



The screenshot shows a Mozilla Firefox browser window with the title bar "colors.json". The address bar shows "file:///root/labs/colors.json". The main content area displays a JSON object with the following structure:

```
{  
  "colors": [  
    {"o": {  
      "color": "black",  
      "category": "hue",  
      "type": "primary",  
      "code": {  
        "rgba": [...],  
        "hex": "#000"  
      }  
    },  
    {"1": {...},  
     "2": {...},  
     "3": {...},  
     "4": {...},  
     "5": {...}  
    ]  
  ]  
}
```

15. Close Mozilla Firefox.
16. Close the Linux terminal window by entering the following command at the terminal prompt:

```
# exit
```

## **Practices for Lesson 2: Performance Tuning Concepts**

## Practices for Lesson 2: Overview

---

### Overview

This practice tests your knowledge of basic performance tuning concepts.

## Practice 2-1: Quiz – Performance Tuning Concepts

### Overview

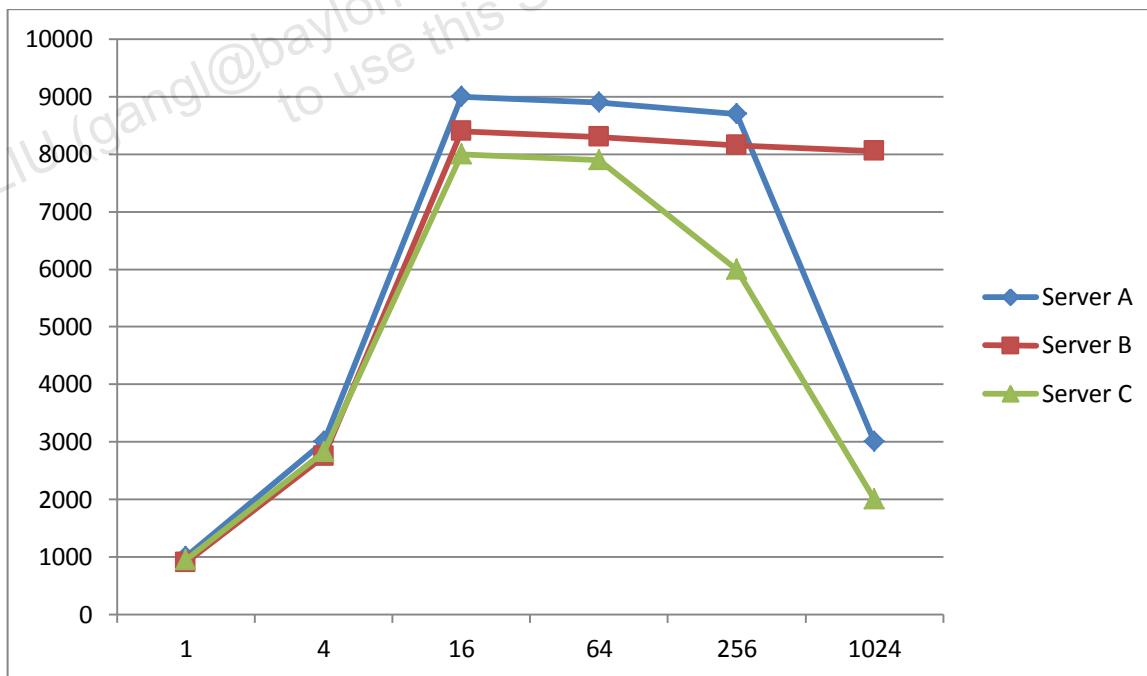
In this practice, you test your knowledge of the performance tuning concepts covered in this lesson.

### Duration

This practice should take you approximately five minutes to complete.

### Quiz Questions

1. Which of the following metrics is the best indicator to use when evaluating performance?
  - a. Throughput
  - b. CPU utilization
  - c. Response time
  - d. Memory usage
2. Which of the following is the best description of response time?
  - a. The product of throughput and wait time
  - b. The sum of wait time and service time
  - c. Tasks per second
  - d. Number of queries in the 95<sup>th</sup> percentile
3. Which of the following servers is the most scalable?
  - a. Server A
  - b. Server B
  - c. Server C



- a. Server A
- b. Server B
- c. Server C

4. Which of the following are *not* good strategies to use for benchmarking a system? (Select all that apply.)
  - a. Using a similar size and similar distribution of data as that used in your production system
  - b. Simulating a single-user environment
  - c. Considering user “think time”
  - d. Using a different server from that used in your production system
5. Which of the following statements are true about the ideal amount of RAM for your system? (Select all that apply.)
  - a. It must be considerably larger than the amount of memory available for caching data.
  - b. It must be at least twice the size of the maximum number of connections.
  - c. If you enable continuous monitoring, you should increase the amount of available RAM.
  - d. You can use less RAM if it is ECC RAM.
6. Which of the following must you consider to properly maintain your MySQL server?
  - a. Upgrade the MySQL server regularly.
  - b. Monitor both network and server.
  - c. Use binary logging to enable point-in-time recovery.
  - d. Conduct regular health checks.
  - e. Implement a good backup schedule.
  - f. All of the above

## Solution 2-1: Quiz – Performance Tuning Concepts

---

### Quiz Solutions

1. **c.** The best indication of performance is how long the server takes to execute queries, and not what resources it consumes.
2. **b.** The sum of wait (or “queue”) time and service time
3. **b.** Server B. The other servers show impressive results initially, but these drop dramatically as the number of concurrent connections increases. Server B maintains good response times as more users compete for resources.
4. **b and d.** You must replicate your production server environment as closely as possible. So executing the benchmark on a different machine is pointless, as is simulating a single-user environment if your application serves multiple users.
5. **a and c.** You must ensure that you have enough RAM available for any caches in addition to the requirements of all the other system processes.
6. **d**

GANG LIU (gangli@baylorhealth.edu) has a non-transferable license  
to use this Student Guide.

## **Practices for Lesson 3: Performance Tuning Tools**

## Practices for Lesson 3

---

### Overview

In these practices, you will:

- Use built-in MySQL command-line monitoring tools and databases, including:
  - SHOW statements
  - The INFORMATION\_SCHEMA and performance\_schema databases
- Evaluate the MySQL Enterprise Monitor GUI tool
- Execute simple benchmarks by using the following tools:
  - mysqlslap
  - sysbench
- Use Linux monitoring tools

**Note:** The outputs of the various Linux commands and SQL statements that are shown in the solution steps for these practices are likely to be different on your system.

### Assumptions

- You are logged in to Linux as the root user.
- The following files are accessible in the /root/scripts directory:
  - 10min.sql
  - dept\_employees.sql
  - slap-test.sh

## Practice 3-1: Using MySQL Command-Line Monitoring Tools

### Overview

In this practice, you evaluate the MySQL monitoring commands, tools, and system databases that are included in the standard MySQL server distribution. These include:

- The `SHOW [GLOBAL|SESSION] STATUS` command
- The `SHOW ENGINE INNODB STATUS` command
- The `INFORMATION_SCHEMA` and `performance_schema` databases
- The `mysqladmin` command-line client

### Duration

This practice should take you approximately 15 minutes to complete.

### Tasks

**Note:** This practice uses two Linux terminal windows. The instructions refer to the terminal windows as `t1` and `t2`. To make it easier to identify each terminal, change its title by using the Terminal > Set Title menu option in each terminal window.

1. Open a terminal prompt (`t1`) and log in to the `mysql` client.
2. Execute the `SHOW STATUS` command at the `t1` prompt that lists all the server status variables that start with the word “Threads%.” How many thread-related server status variables does this query display?
3. Issue a command to reset the server status variables.
4. Execute the `SHOW STATUS` command at the `mysql` prompt to list the server status variables for this session that store metrics about temporary tables and files.
5. Change the current database to `world` and execute the following query:

```
mysql> SELECT Name, Code, Population FROM Country
    -> WHERE Code IN (
    ->     SELECT CountryCode FROM CountryLanguage
    ->     WHERE Percentage > 50
    -> )
    -> ORDER BY Population DESC;
```

6. Repeat the `SHOW STATUS` command that you issued in step 4, and use the output to answer the following questions:
  - How many temporary tables did the MySQL server create to execute the query in the previous step?
  - How many of those temporary tables were stored on disk?
7. Using the `mysql` client, list the tables in the `INFORMATION_SCHEMA` database that start with the letter “S.” How many `INFORMATION_SCHEMA` tables start with the letter “S”?

8. Issue a DESCRIBE statement on the SESSION\_STATUS table in the INFORMATION\_SCHEMA database. What information does the SESSION\_STATUS table of the INFORMATION\_SCHEMA database contain?
9. Query the SESSION\_STATUS table to retrieve all the server status variables that start with the word “Threads%.”
10. List all the tables in the performance\_schema database.
11. List the tables in the performance\_schema database that end in “\_status.”
12. Issue a DESCRIBE statement on the session\_status table in the performance\_schema database.
13. Query the Performance Schema’s SESSION\_STATUS table to retrieve the values of the server session status variables that store metrics about storage engine reads.
  - Is this information also available by using the SHOW SESSION STATUS command?
  - What does the value of the HANDLER\_READ\_RND\_NEXT status variable tell you?
  - What is the benefit of retrieving this data from the Performance Schema?
14. Issue a SHOW ENGINE INNODB STATUS statement to list detailed metrics about the working of the InnoDB storage engine.
15. Terminate the interactive mysql session.
16. Use the mysqladmin command-line tool to display an extended list of server status variables.
17. Open a new Linux terminal window. This activity refers to this new terminal window as t2. In t2, instruct the mysql client to run the 10min.sql script in the /root/scripts directory.

**Note:** The 10min.sql script executes a query in the employees database that takes approximately 10 minutes to complete.
18. In the t1 terminal window, use the mysqladmin command-line tool to display a brief status message every 10 seconds, showing the changes since the last status report. Allow the command to run for a minute or two, and then exit with Ctrl+C.
  - How many active threads are connected to the MySQL server?
  - How many tables are currently open?
  - How many queries were sent to the server at the last status update?
19. In the t1 terminal window, log in to the mysql client and execute a command that lists all the currently running processes.
  - How many MySQL server processes are currently running?
  - What is the connection ID for the query that you executed in step 17?
20. Exit the MySQL client in the t1 terminal window.
21. With the information from the SHOW PROCESSLIST statement that you issued in the previous step, use the mysqladmin client in terminal window t1 to kill the thread that is executing the query issued in step 17.

22. In the t2 terminal window, what error message was displayed after you terminated the query in the preceding step?
23. Close the t2 terminal window.
24. Keep the t1 terminal window open for the next practice.

## Solution 3-1: Using MySQL Command-Line Monitoring Tools

---

### Solution Steps

**Note:** These solutions show sample outputs. Your outputs might differ from those shown.

1. Open a terminal prompt (`t1`) and log in to the `mysql` client.

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# mysql -uroot -p
Enter password: oracle
Welcome to the MySQL monitor. Commands end with ; or \g.
...
mysql>
```

2. Execute a `SHOW STATUS` command at the `t1` prompt that lists all the server status variables that start with the word “Threads%.”

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> SHOW GLOBAL STATUS LIKE 'Threads%';
+-----+-----+
| Variable_name      | Value |
+-----+-----+
| Threads_cached     | 0      |
| Threads_connected  | 1      |
| Threads_created    | 1      |
| Threads_running    | 1      |
+-----+-----+
4 rows in set (#.## sec)
```

- How many thread-related server status variables does this query display?

**Answer:** 4

3. Issue a command to reset the server status variables.

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> FLUSH STATUS;
Query OK, 0 rows affected (#.## sec)
```

4. Execute a `SHOW STATUS` command at the `mysql` prompt to list the server status variables for this session that store metrics about temporary tables and files.

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> SHOW SESSION STATUS LIKE 'Created\_\tmp%';
+-----+-----+
| Variable_name          | Value |
+-----+-----+
| Created_tmp_disk_tables | 0      |
| Created_tmp_files      | 0      |
| Created_tmp_tables     | 0      |
+-----+-----+
```

```
+-----+-----+
3 rows in set (#.## sec)
```

5. Change the current database to `world` and execute the following query:

```
mysql> SELECT Name, Code, Population FROM Country
-> WHERE Code IN (
->     SELECT CountryCode FROM CountryLanguage
->     WHERE Percentage > 50
-> )
-> ORDER BY Population DESC;
```

Enter the following statements at the `mysql` prompt and receive the results shown:

```
mysql> USE world;
mysql> SELECT Name, Code, Population FROM country
-> WHERE Code IN (
->     SELECT CountryCode FROM countrylanguage
->     WHERE Percentage > 50
-> )
-> ORDER BY Population DESC;
+-----+-----+-----+
| Name          | Code | Population |
+-----+-----+-----+
| China         | CHN  | 1277558000 |
| United States | USA   | 278357000  |
...
| Nauru         | NRU  | 12000    |
| Tuvalu        | TUV  | 12000    |
+-----+-----+-----+
168 rows in set (#.## sec)
```

6. Repeat the `SHOW STATUS` command that you issued in step 4.

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> SHOW SESSION STATUS LIKE 'Created\_\tmp%';
+-----+-----+
| Variable_name      | Value |
+-----+-----+
| Created_tmp_disk_tables | 0    |
| Created_tmp_files   | 0    |
| Created_tmp_tables  | 3    |
+-----+-----+
3 rows in set (#.## sec)
```

- How many temporary tables did the MySQL server create to execute the query in the previous step? **Answer:** 3 (`Created_tmp_tables`)

- How many of those temporary tables were stored on disk? **Answer:** None (`Created_tmp_disk_tables`)
7. Using the `mysql` client, list the tables in the `INFORMATION_SCHEMA` database that start with the letter “S.”

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> SHOW TABLES FROM INFORMATION_SCHEMA LIKE 'S%';
+-----+
| Tables_in_INFORMATION_SCHEMA (S%) |
+-----+
| SCHEMATA                           |
| SCHEMA_PRIVILEGES                  |
| SESSION_STATUS                     |
| SESSION_VARIABLES                 |
| STATISTICS                         |
+-----+
5 rows in set (#.# sec)
```

- How many `INFORMATION_SCHEMA` tables start with the letter “S”? **Answer:** 5
8. Issue a `DESCRIBE` statement on the `SESSION_STATUS` table in the `INFORMATION_SCHEMA` database.

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> DESCRIBE INFORMATION_SCHEMA.SESSION_STATUS;
+-----+-----+-----+-----+-----+-----+
| Field          | Type           | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| VARIABLE_NAME | varchar(64)   | NO   |     |         |       |
| VARIABLE_VALUE | varchar(1024) | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (#.# sec)
```

- What information does the `SESSION_STATUS` table of the `INFORMATION_SCHEMA` database contain? **Answer:** The name of each server status variable and its value
9. Query the `SESSION_STATUS` table to retrieve all the server status variables that start with the word “Threads%.”

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.SESSION_STATUS
      -> WHERE VARIABLE_NAME LIKE 'Threads%';
ERROR 3167 (HY000): The 'INFORMATION_SCHEMA.SESSION_STATUS'
feature is disabled; see the documentation for
'show_compatibility_56'
```

- This information is not available by default. The `Information Schema GLOBAL_STATUS` and `SESSION_STATUS` tables are deprecated.

- If you want to query this table, you must enable the `show_compatibility_56` system variable.

10. List all the tables in the `performance_schema` database.

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> SHOW TABLES FROM performance_schema;
+-----+
| Tables_in_performance_schema |
+-----+
| accounts
| cond_instances
| events_stages_current
| events_stages_history
| events_stages_history_long
...
| table_io_waits_summary_by_index_usage
| table_io_waits_summary_by_table
| table_lock_waits_summary_by_table
| threads
| user_variables_by_thread
| users
| variables_by_thread
+-----+
87 rows in set (#.## sec)
```

**Note:** This is only to make you aware that the Performance Schema contains a lot of information about the status of the MySQL server. You will do more with the Performance Schema in later practices.

11. List the tables in the `performance_schema` database that end in “`_status`.”

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> SHOW TABLES FROM performance_schema Like '%_status';
+-----+
| Tables_in_performance_schema (%_status) |
+-----+
| global_status
| replication_applier_status
| replication_connection_status
| session_status
+-----+
4 rows in set (#.## sec)
```

- The Performance Schema contains the `global_status` and `session_status` tables, and provides an alternative method of accessing this information to the Information Schema.

12. Issue a DESCRIBE statement on the session\_status table in the performance\_schema database.

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> DESCRIBE performance_schema.session_status;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| VARIABLE_NAME | varchar(64) | NO | | NULL | |
| VARIABLE_VALUE | varchar(1024) | YES | | NULL | |
+-----+-----+-----+-----+
2 rows in set (#.## sec)
```

13. Query the Performance Schema's SESSION\_STATUS table to retrieve the values of the server session status variables that store metrics about storage engine reads.

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> SELECT * FROM performance_schema.session_status
-> WHERE VARIABLE_NAME LIKE 'Handler\read%';
+-----+-----+
| VARIABLE_NAME | VARIABLE_VALUE |
+-----+-----+
| Handler_read_first | 4 |
| Handler_read_key | 243 |
| Handler_read_last | 0 |
| Handler_read_next | 0 |
| Handler_read_prev | 0 |
| Handler_read_rnd | 7 |
| Handler_read_rnd_next | 1553 |
+-----+
7 rows in set (#.## sec)
```

**Note:** Your session status values might be different from those shown.

- Is this information also available by using the SHOW SESSION STATUS command? **Answer:** Yes. You retrieve the same information by issuing the following SHOW SESSION STATUS command:

```
mysql> SHOW SESSION STATUS LIKE 'Handler\read%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Handler_read_first | 4 |
| Handler_read_key | 243 |
| Handler_read_last | 0 |
| Handler_read_next | 0 |
| Handler_read_prev | 0 |
| Handler_read_rnd | 21 |
+-----+
```

```
| Handler_read_rnd_next | 2186 |
+-----+-----+
7 rows in set (#.## sec)
```

- What does the value of the Handler\_Read\_Rnd\_Next status variable tell you?  
**Answer:** The Handler\_Read\_Rnd\_Next variable shows the number of requests to read a row based on a fixed position. It shows that the query that you executed in step 5 resulted in several table scans.
- What is the benefit of retrieving this data from the Performance Schema? **Answer:** Because the data is stored in database tables, you can query it by using SQL.

14. Issue a SHOW ENGINE INNODB STATUS statement to list detailed metrics about the working of the InnoDB storage engine.

**Note:** This course covers what the output of SHOW ENGINE INNODB STATUS means in the lesson titled “Tuning InnoDB.”

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> SHOW ENGINE INNODB STATUS\G
***** 1. row *****
Type: InnoDB
Name:
Status:
=====
<date and time> 0x7f7dfc122700 INNODB MONITOR OUTPUT
=====
Per second averages calculated from the last 21 seconds
-----
BACKGROUND THREAD
-----
...
-----
SEMAPHORES
-----
...
-----
TRANSACTIONS
-----
...
-----
FILE I/O
-----
...
-----
INSERT BUFFER AND ADAPTIVE HASH INDEX
-----
```

```
...
-----
LOG
-----
...
-----
BUFFER POOL AND MEMORY
-----
...
-----
ROW OPERATIONS
-----
...
-----
END OF INNODB MONITOR OUTPUT
=====
1 row in set (#.## sec)
```

- You can use the output of the `SHOW ENGINE INNODB STATUS` statement to diagnose crashes, bottlenecks, and so on, but it is verbose. Most of the information it provides is also available in the `INFORMATION_SCHEMA` and `performance_schema` databases.

15. Terminate the interactive `mysql` session.

Enter the following statement at the `mysql` prompt:

```
mysql> EXIT
Bye
```

16. Use the `mysqladmin` command-line tool to display an extended list of server status variables.

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# mysqladmin -uroot -p extended-status
Enter password: oracle
+-----+-----+
| Variable_name          | Value   |
+-----+-----+
| Aborted_clients        | 0       |
| Aborted_connects       | 0       |
| Binlog_cache_disk_use  | 0       |
| Binlog_cache_use        | 0       |
|
| Threads_cached          | 0       |
| Threads_connected       | 1       |
| Threads_created         | 1       |
| Threads_running         | 1       |
```

Uptime	607429
Uptime_since_flush_status	1235

**Note:** The mysqladmin command-line tool enables you to retrieve information about the status of the MySQL server without starting an interactive mysql session.

17. Open a new Linux terminal window. This activity refers to this new terminal window as t2. In t2, instruct the mysql client to run the 10min.sql script in the /root/scripts directory.

Enter the following command at the Linux terminal prompt:

```
# mysql -uroot -p < /root/scripts/10min.sql
Enter password: oracle
```

**Note:** The 10min.sql script executes a query in the employees database that takes approximately 10 minutes to complete.

18. In the t1 terminal window, use the mysqladmin command-line tool to display a brief status message every 10 seconds, showing the changes since the last status report. Allow the command to run for a minute or two, and then exit with Ctrl+C.

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# mysqladmin -uroot -p -i10 --relative status
Enter password: oracle
Uptime: 257164 Threads: 2 Questions: 72 Slow queries: 0
Opens: 140 Flush tables: 1 Open tables: 119 Queries per
second avg: 0.000
Uptime: 257174 Threads: 2 Questions: 73 Slow queries: 0
Opens: 140 Flush tables: 1 Open tables: 119 Queries per
second avg: 0.000
Uptime: 257184 Threads: 2 Questions: 74 Slow queries: 0
Opens: 140 Flush tables: 1 Open tables: 119 Queries per
second avg: 0.000
Uptime: 257194 Threads: 2 Questions: 75 Slow queries: 0
Opens: 140 Flush tables: 1 Open tables: 119 Queries per
second avg: 0.000
Uptime: 257204 Threads: 2 Questions: 76 Slow queries: 0
Opens: 140 Flush tables: 1 Open tables: 119 Queries per
second avg: 0.000...
^C
#
```

**Note:** Your values might differ from those shown in the sample output. Using the sample output, the answers to the questions are as follows.

- How many active threads are connected to the MySQL server? **Answer:** 2 (Threads)
- How many tables are currently open? **Answer:** 119 (Open tables)
- How many queries were sent to the server at the last status update? **Answer:** 76 (Questions)

19. In the `t1` terminal window, log in to the `mysql` client and execute a command that lists all the currently running processes.

Enter the following command at the Linux terminal prompt:

```
# mysql -uroot -p
Enter password: oracle
Welcome to the MySQL monitor. Commands end with ; or \g.
...
```

Then, enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> SHOW PROCESSLIST\G
***** 1. row ****
Id: 8
User: root
Host: localhost
db: NULL
Command: Query
Time: 152
State: Sending data
Info: SELECT dept_emp.from_date, dept_manager.from_date,
salaries.from_date
FROM employees.dept_emp, emp1
***** 2. row ****
Id: 10
User: root
Host: localhost
db: NULL
Command: Query
Time: 0
State: starting
Info: SHOW PROCESSLIST
2 rows in set (0.00 sec)
```

- How many MySQL server processes are currently running? **Answer: 2**
- What is the connection ID for the query that you executed in step 17? **Answer: 8**  
**(Note:** Your connection ID might differ from the one shown in the preceding sample output.)

20. Exit the MySQL client in the `t1` terminal window.

Enter the following statement at the `mysql` prompt in `t1` and receive the results shown:

```
mysql> EXIT
Bye
#
```

21. With the information from the `SHOW PROCESSLIST` statement that you issued in the previous step, use the `mysqladmin` client in terminal window `t1` to kill the thread that is executing the query issued in step 17.

Enter the following command at the Linux terminal prompt in t1:

```
# mysqladmin -uroot -p kill 8  
Enter password: oracle
```

22. In the t2 terminal window, what error message was displayed after you terminated the query in the preceding step?

– **Answer:** The message is:

```
ERROR 2013 (HY000) at line 1: Lost connection to MySQL server  
during query
```

23. Close the t2 terminal window.

Enter the following command at the Linux terminal prompt in terminal window t2.

```
# exit
```

24. Keep the t1 terminal window open for the next practice.

## Practice 3-2: Using MySQL Enterprise Monitor

---

### Overview

In this practice, you:

- Check the status of the MySQL Enterprise Monitor Service Manager service
- Create the Manager and Agent users
- Configure monitoring for your MySQL server instance
- Evaluate some of the monitoring capabilities of MySQL Enterprise Monitor

### Duration

This practice should take you approximately 20 minutes to complete.

### Tasks

#### Start and Configure MySQL Enterprise Monitor

1. In a Linux terminal window, as the Linux `root` user, check whether the Apache Web Server (`httpd`) service is configured to start automatically on boot and is running. If not, enable and start the `httpd` service. Use the following `systemctl` commands to achieve this:
  - `systemctl status httpd.service`: To view the current status of the service
  - `systemctl enable httpd.service`: To configure the service to run automatically at system boot
  - `systemctl start httpd.service`: To start the service

```
# systemctl status httpd.service
● httpd.service - The Apache HTTP Server
  Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled; vendor preset: disabled)
  Active: inactive (dead)
    Docs: man:httpd(8)
          man:apachectl(8)
```

- The `httpd` service is disabled and inactive.

```
# systemctl enable httpd.service
Created symlink from /etc/systemd/system/multi-user.target.wants/httpd.service to
/usr/lib/systemd/system/httpd.service.
```

- Enable the `httpd` service.

```
# systemctl status httpd.service
● httpd.service - The Apache HTTP Server
  Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; vendor preset: disabled)
  Active: inactive (dead)
    Docs: man:httpd(8)
```

```
man:apachectl(8)
```

- The httpd service is enabled, but still inactive.

```
# systemctl start httpd.service
```

- Start the httpd service.

```
# systemctl status httpd.service
```

```
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; vendor preset: disabled)
   Active: active (running) since Mon 2017-06-12 10:48:47 UTC;
            3s ago
     Docs: man:httpd(8)
           man:apachectl(8)

Main PID: 7735 (httpd)
   Status: "Processing requests..."
   CGroup: /system.slice/httpd.service
           ├─7735 /usr/sbin/httpd -DFOREGROUND
           ├─7736 /usr/sbin/httpd -DFOREGROUND
           ├─7737 /usr/sbin/httpd -DFOREGROUND
           ├─7738 /usr/sbin/httpd -DFOREGROUND
           ├─7739 /usr/sbin/httpd -DFOREGROUND
           └─7740 /usr/sbin/httpd -DFOREGROUND
```

```
Jun 12 10:48:47 edddr73p1 systemd[1]: Starting The Apache HTTP
Server...
```

```
Jun 12 10:48:47 edddr73p1 systemd[1]: Started The Apache HTTP
Server.
```

- The httpd service is now running.

2. Check whether the MySQL Enterprise Monitor Service Manager (`mysql-monitor-server`) service is running, and start it if necessary:

```
# systemctl status mysql-monitor-server.service
```

```
● mysql-monitor-server.service - LSB: MySQL Monitoring Server
   Loaded: loaded (/etc/rc.d/init.d/mysql-monitor-server)
   Active: inactive (dead)
     Docs: man:systemd-sysv-generator(8)
```

- The MySQL Enterprise Monitor Service Manager service is loaded, but not running.

```
# systemctl start mysql-monitor-server.service
```

- Start the MySQL Enterprise Monitor Service Manager service.

```
# systemctl status mysql-monitor-server.service
```

```
● mysql-monitor-server.service - LSB: MySQL Monitoring Server
   Loaded: loaded (/etc/rc.d/init.d/mysql-monitor-server)
```

```

Active: active (exited) since Mon 2017-06-12 10:57:25 UTC;
46s ago
Docs: man:systemd-sysv-generator(8)
Process: 8427 ExecStart=/etc/rc.d/init.d/mysql-monitor-server
start (code=exited, status=0/SUCCESS)

Jun 12 10:57:20 edddr73p1 systemd[1]: Starting LSB: MySQL
Monitoring Server...
Jun 12 10:57:20 edddr73p1 mysql-monitor-server[8427]:
/etc/rc.d/init.d/mysql-mon...
Jun 12 10:57:25 edddr73p1 mysql-monitor-server[8427]:
/etc/rc.d/init.d/mysql-mon...
Jun 12 10:57:25 edddr73p1 systemd[1]: Started LSB: MySQL
Monitoring Server.
Hint: Some lines were ellipsized, use -l to show in full.

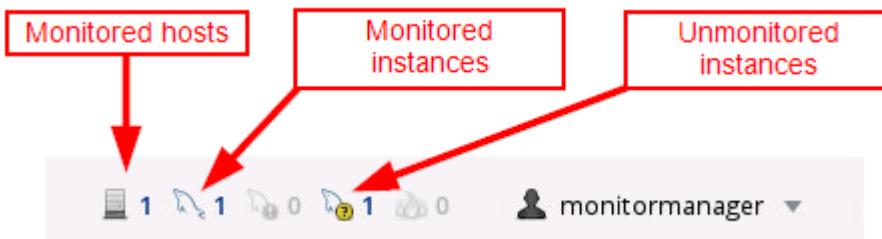
```

- The MySQL Enterprise Monitor Service Manager service is now running.
3. Open the MySQL Enterprise Monitor dashboard by using the Applications > Programming > MySQL Enterprise Monitor menu option. Mozilla Firefox displays a page with the message: "This Connection is Untrusted."
  4. Expand the "I understand the risks" section of the page and click the Add Exception button. The Add Security Exception dialog box appears.
  5. In the Add Security Exception dialog box, ensure that the "Permanently store this exception" option is selected, and then click the Confirm Security Exception button. The MySQL Enterprise Monitor Welcome page appears and prompts you to create two users: one with the manager role, and one with the agent role.
  6. In the "Create user with 'manager' role" form, enter `monitormanager` as the username, with a password of `oracle`.
  7. In the "Create user with 'agent' role" form, enter `monitoragent` as the username, with a password of `oracle`.
  8. Click the Complete Setup button at the bottom of the page. The Welcome! dialog box appears.
  9. In the Welcome! dialog box, make any required Locale and Timezone changes and click the Save button. The What's New page appears.

### **Enabling Monitoring of the MySQL Server Instance**

10. At the top right-hand corner of the page, note that you are logged in as the `monitormanager` user and that the server icons show the number of hosts and instances

that MySQL Enterprise Monitor is monitoring:

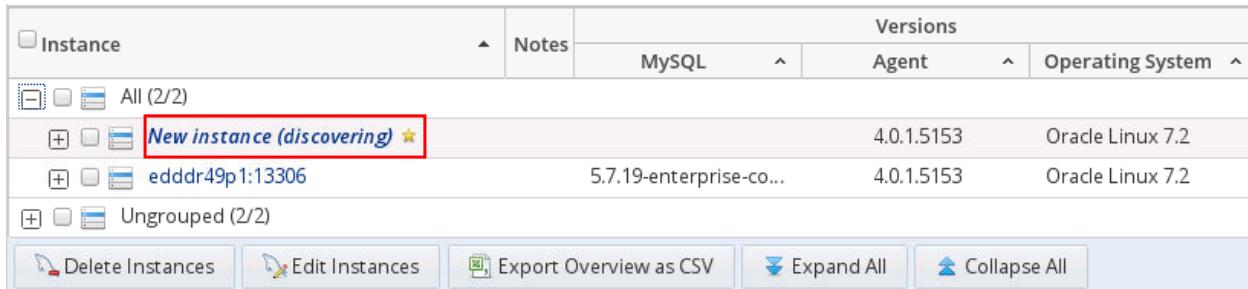


- Click the “1 Unmonitored MySQL Instance” link, which is represented by the Dolphin icon with a question mark. The MySQL Instances page appears and the Unmonitored MySQL Instances panel lists a single unmonitored instance.

The screenshot shows the MySQL Instances page with the 'Unmonitored MySQL Instances' section highlighted. It includes buttons for 'Add MySQL Instance', 'Add Bulk MySQL Instances', 'Monitor Instances', 'Ignore Instances', and 'Cancel Pending Connections'. A dropdown shows 'Show 5 entries'. The main table has a header 'Host' with a checkbox. One row is shown, with the host name 'edddr49p1' highlighted with a red box. The status column shows 'Connecting' and 'No'. A message at the bottom says 'Showing 1 to 1 of 1 entries'.

- Select the check box next to the unmonitored host and click the Monitor Instances button. A pop-up dialog box appears.
- On the Connection Settings tab, perform the following:
  - In the Monitor From drop-down list, select MEM Built-in Agent.
  - In the Connect Using drop-down list, select TCP/IP.
  - Check that the Instance Address field contains 127.0.0.1 and that the Port field contains 3306.
  - Enter the MySQL `root` user (password: `oracle`) authentication details in the Admin User and Admin Password fields.
  - Select Yes from the Auto-Create Less Privileged Users drop-down list.
  - When selected, this option creates less privileged user accounts on the MySQL server for tasks that do not require `root` access.
  - Enter the following details for the General and Limited users:
    - General user.* Username: `memgeneral`, password: `oracle`
    - Limited user.* Username: `memlimited`, password: `oracle`

- h. Click the “Add Instance” button. If Mozilla Firefox offers to remember the `root` user password, decline.
14. Expand the All node in the hierarchy tree at the bottom of the MySQL Instances page to see the new monitored instance in the “discovering” state.



The screenshot shows a table titled "MySQL Instances". The top navigation bar includes "Instance", "Notes", "Versions", "MySQL", "Agent", and "Operating System". Below the header, there are three rows of data:

- All (2/2)**: Contains a link labeled "New instance (discovering) ★". This row is highlighted with a red box.
- edddr49p1:13306**: MySQL version 5.7.19-enterprise-co..., Agent version 4.0.1.5153, Operating System Oracle Linux 7.2.
- Ungrouped (2/2)**: No details shown.

At the bottom of the table are buttons for "Delete Instances", "Edit Instances", "Export Overview as CSV", "Expand All", and "Collapse All".

15. Disable monitoring of the bundled MySQL server in MySQL Enterprise Monitor.
- Select the check box next to the host running on port 13306.
  - Click the Delete Instances button.
  - In the Delete 1 Instance dialog box that appears, confirm that you wish to delete the instance on port 13306.
  - Verify that there is only one server instance being monitored, and that it is the instance running on port 3306.

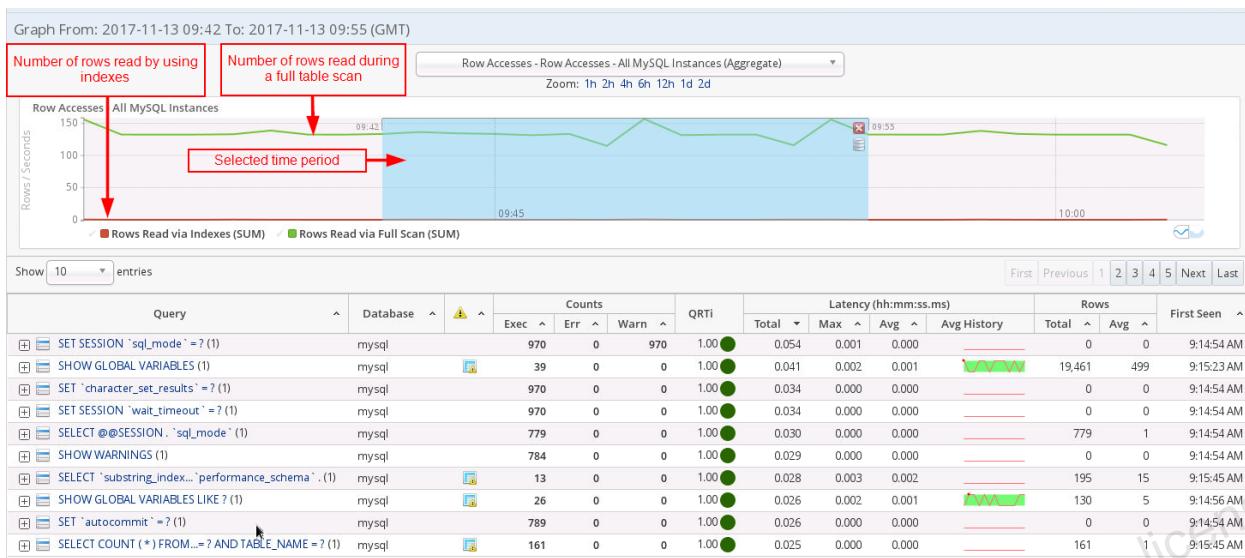
**Note:** You are currently monitoring only one MySQL server instance. In a live deployment with multiple servers, you might choose to monitor many servers at once, or categorize them into “Groups.”

16. On the left-hand side of the page, click the Configuration > Advisors menu option.
- Note:** MySQL Enterprise Monitor has automatically configured advisors for the MySQL instance. To answer questions about the individual advisors:
- Hover the cursor over the Question Mark icon in the Info column for each advisor, and read the information that is provided.
  - Examine the values in the Schedule and Parameters columns.
17. On the Advisors page, click the arrow to the right of the Performance heading to expand the section, and then locate the Excessive Disk Temporary Table Usage Detected advisor. Hover the cursor over the Question Mark icon in the advisor’s Info column and read its associated description.
18. On the left-hand side of the page, click the Events menu option. The page that appears lists all the events that the MySQL Enterprise Monitor has captured. The events range from *Critical* (require immediate attention) to *Info* (issues that do not affect the operation of your server). Click the plus sign next to an event to view its details. Read the problem description and any associated advice.
19. Return to the Advisors page by selecting the Configuration > Advisors menu option on the left-hand side of the page.
20. Click the arrow to the right of the Query Analysis heading, to expand the section.
- Which advisor triggers a warning if a SQL statement takes longer than one second to execute?

- Does a critical event occur if a `CREATE TABLE` operation takes longer than five seconds to execute?
21. Click the arrow to the right of the Operating System heading, to expand the section.
- What information does the CPU Utilization advisor provide?
  - At what CPU I/O Wait Threshold percentage does the CPU Utilization advisor trigger a critical event?

### Evaluating MySQL Enterprise Monitor Capabilities

22. On the left-hand side of the page, click the Overview menu option.
23. Select 30s from the Refresh drop-down list on the right-hand side of the page.
24. Identify the following areas of the Dashboard overview page:
- a. Connections: Shows the number of connections to the server and when they occur. This information helps you to identify peak usage times.
  - b. Database Queries and Row Accesses: Provides an indication of how “busy” your MySQL server is, as measured by the SQL statements it processes
  - c. Database Availability: Displays the uptime statistics for your monitored instances over a day, a week, and a month
  - d. Current Problem Hosts and Current Problem MySQL Instances: Shows at a glance which instances or hosts are not running, or have critical alerts that you need to address
  - e. Current Emergency and Critical Events: Lists all the critical events raised by the monitored instances
25. On the left-hand side of the page, click the Queries menu option. The Query Analyzer page opens. The Query Analyzer feature of MySQL Enterprise Monitor helps you to locate, diagnose, and fix problem queries. The page displays the Query Analysis Reporting – Query Response Time Index (Aggregate) graph. The Query Response Time index is a metric that you can use to assess how optimally your queries are running.
- Note:** This course covers the Query Analyzer in the lesson titled “Monitoring Queries.”
26. Click the drop-down list of graph types above the graph and examine the different graphs that are available.
27. In the drop-down list search box, enter “Row.” The list filters to show the three row-based graphs that are available. Select Row Accesses.
- A new graph is displayed, which shows all the queries that have accessed table rows. The red line on the graph represents queries that accessed the rows via indexes and the green line represents queries that required a full table scan.
28. Click the graph and drag the cursor to either the left or the right to highlight a particular time segment. The graph and query list refresh to show detailed information about the queries that occurred in the specified time frame.



29. Click one of the SQL statements in the Query column in the table at the bottom of the page. A dialog box appears, showing further information about the query in the Canonical Query tab. Note the Execution Time Statistics, Row Statistics, and Execution Summary.
30. Under Canonical Form, click the Full link to display the complete SQL statement.
31. Click the Close button to close the dialog box.
32. On the left-hand side of the page, select the Metrics > All Timeseries Graphs menu option. Examine the range of graphs available.
33. At the top of the page, from the drop-down menu that currently has All Targets selected, select the monitored MySQL instance. Note the “more-detailed” graphs that are available for a specific MySQL instance.
34. Log out of MySQL Enterprise Monitor.
  - a. Click the “monitormanager” button at the top right-hand corner of the page.
  - b. Click Logout.
  - c. Close Mozilla Firefox.
35. The MySQL Enterprise Monitor services affect the results of subsequent practices, so stop the MySQL Enterprise Monitor Service Manager by executing the following command at a Linux terminal prompt and receiving the results shown:

```
# sh /opt/mysql/enterprise/monitor/mysqlmonitorctl.sh stop
Stopping tomcat service . [ OK ]
Stopping mysql service .. [ OK ]
```

**Note:** The `mysqlmonitorctl.sh` script is installed in the root of the MySQL Enterprise Monitor installation directory, and provides a convenient method for starting, stopping, and checking the status of MySQL Enterprise Monitor and its bundled Tomcat instance. Execute it with the `help` argument to see the different options:

```
# sh /opt/mysql/enterprise/monitor/mysqlmonitorctl.sh help
usage: /opt/mysql/enterprise/monitor/mysqlmonitorctl.sh help
        /opt/mysql/enterprise/monitor/mysqlmonitorctl.sh
        (start|stop|status|restart)
```

```
/opt/mysql/enterprise/monitor/mysqlmonitorctl.sh  
(start|stop|status|restart) mysql  
      /opt/mysql/enterprise/monitor/mysqlmonitorctl.sh  
(start|stop|status|restart) tomcat  
  
help      - this screen  
start     - start the service(s)  
stop      - stop the service(s)  
restart   - restart or start the service(s)  
status    - report the status of the service(s)
```

36. Leave the Linux terminal window open for the next practice.

## Solution 3-2: Using MySQL Enterprise Monitor

---

### Solution Steps

Follow the practice task instructions.

### Answers to Questions

20. Click the arrow to the right of the Query Analysis heading, to expand the section.
  - Which advisor triggers a warning if a SQL statement takes longer than one second to execute? **Answer:** The Average Statement Execution Time Advisor
  - Does a critical event occur if a `CREATE TABLE` operation takes longer than five seconds to execute? **Answer:** No. The Average Statement Execution Time Advisor only monitors DML statements by default.
21. Click the arrow to the right of the Operating System heading, to expand the section.
  - What information does the CPU Utilization advisor provide? **Answer:** It monitors CPU utilization, and alerts you when the overall CPU usage is very high, or if the CPU usage of one or more servers in a group is much higher than the others in that group.
  - At what CPU I/O Wait Threshold percentage does the CPU Utilization advisor trigger a critical event? **Answer:** 90%

## Practice 3-3: Using Benchmark Tools

### Overview

In this practice, you evaluate the following benchmark tools:

- mysqlslap
- sysbench

### Duration

This practice should take you approximately 15 minutes to complete.

### Tasks

#### Evaluate mysqlslap

1. Examine the `/root/scripts/dept_employees.sql` script. You will use this SQL script for the load test.
2. Use the `mysqlslap` command-line application to create a load on the MySQL server with the following command:

```
# mysqlslap -uroot -p \
> --create-schema=employees
> --query="/root/scripts/dept_employees.sql" \
> --iterations=10 --concurrency=10 \
> -vv
Enter password: oracle
```

- The `--create-schema` option tells `mysqlslap` which database to use for the test.
- The `--query` option tells `mysqlslap` which SQL script to execute for the test.
- The `--iterations` option tells `mysqlslap` how many times to execute the script for each connection (10).
- The `--concurrency` option tells `mysqlslap` how many connections to the MySQL server to simulate (10).
- The `-vv` option makes the output of the operation verbose.

**Note:** The backslash character ('\') followed by a newline causes bash to continue the command on the next line.

What was the average execution time for all the queries?

### Evaluate sysbench

3. Create the sb database. The sysbench program uses this schema to run the tests. Verify that the database exists before continuing.
4. Prepare the sysbench tool for use by issuing the following command in a terminal window:

```
# sysbench \
> --test=oltp \
> --db-driver=mysql \
> --mysql-table-engine=innodb \
> --oltp-table-size=50000 \
> --mysql-socket=/var/lib/mysql/mysql.sock \
> --mysql-user=root \
> --mysql-password=oracle \
> --mysql-db=sb \
> prepare
```

5. Use the sysbench application to execute multiple transactions by using multiple threads against the MySQL server by issuing the following command in a terminal window:

```
# sysbench \
> --test=oltp \
> --db-driver=mysql \
> --oltp-table-size=50000 \
> --mysql-socket=/var/lib/mysql/mysql.sock \
> --mysql-user=root \
> --mysql-password=oracle \
> --oltp-test-mode=simple \
> --oltp-reconnect-mode=query \
> --mysql-db=sb \
> --max-requests=1000 \
> --num-threads=128 \
> run
```

6. Examine the output and answer the following questions:
  - How many transactions completed per second?
  - How long, on an average, did the MySQL server take to complete each request?
7. Drop and re-create the sb database for later practices.
8. Keep the Linux terminal window open for the next practice.

## Solution 3-3: Using Benchmark Tools

---

### Solution Steps

#### Evaluate mysqlslap

1. Examine the /root/scripts/dept\_employees.sql script. You will use this SQL script for the load test.

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# less /root/scripts/dept_employees.sql
SELECT employees.departments.dept_name,
employees.employees.last_name, employees.employees.first_name
FROM employees.departments, employees.employees,
employees.dept_emp WHERE employees.departments.dept_no =
employees.dept_emp.dept_no AND employees.dept_emp.emp_no =
employees.employees.emp_no AND employees.employees.emp_no =
40793;
...
```

– Enter “q” to exit less.

2. Use the mysqlslap command-line application to create a load on the MySQL server with the following command:

```
# mysqlslap -uroot -p \
> --create-schema=employees
> --query="/root/scripts/dept_employees.sql" \
> --iterations=10 --concurrency=10 \
> -vv
Enter password: oracle
```

- The --create-schema option tells mysqlslap which database to use for the test.
- The --query option tells mysqlslap which SQL script to execute for the test. This must be the full path to the script.
- The --iterations option tells mysqlslap how many times to execute the script for each connection (10).
- The --concurrency option tells mysqlslap how many connections to the MySQL server to simulate (10).
- The -vv option makes the output of the operation verbose.

**Note:** The backslash character ('\') followed by a newline causes bash to continue the command on the next line.

Execute the preceding mysqlslap command at the Linux terminal prompt. This results in an output similar to the following:

```
Parsing engines to use.
Enter password: oracle
Starting Concurrency Test
Generating primary key list
```

```
Generating primary key list
Generating stats
Benchmark
    Average number of seconds to run all queries: 0.038 seconds
    Minimum number of seconds to run all queries: 0.035 seconds
    Maximum number of seconds to run all queries: 0.042 seconds
    Number of clients running queries: 10
    Average number of queries per client: 200
```

- What was the average execution time for all the queries? **Answer:** 0.038 seconds in the sample output shown

### Evaluate sysbench

3. Create the `sb` database. The `sysbench` program uses this schema to run the tests. Verify that the database exists before continuing.

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# mysql -uroot -p \
> -e"CREATE DATABASE sb;SHOW DATABASES ;"
Enter password: oracle
+-----+
| Database      |
+-----+
| information_schema |
| employees      |
| mysql          |
| mysqlslap      |
| performance_schema |
| perft          |
| sakila         |
| sb           |
| sys            |
| world          |
| world_NonNorm |
+-----+
```

4. Prepare the sysbench tool for use by issuing the following command in a terminal window:

```
# sysbench \
> --test=oltp \
> --db-driver=mysql \
> --mysql-table-engine=innodb \
> --oltp-table-size=50000 \
> --mysql-socket=/var/lib/mysql/mysql.sock \
> --mysql-user=root \
> --mysql-password=oracle \
> --mysql-db=sb \
> prepare
```

The command results in an output similar to the following:

```
sysbench 0.4.12: multi-threaded system evaluation benchmark

Creating table 'sbtest'...
Creating 50000 records in table 'sbtest'...
#
```

5. Use the sysbench application to execute multiple transactions by using multiple threads against the MySQL server by issuing the following command in a terminal window:

```
# sysbench \
> --test=oltp \
> --db-driver=mysql \
> --oltp-table-size=50000 \
> --mysql-socket=/var/lib/mysql/mysql.sock \
> --mysql-user=root \
> --mysql-password=oracle \
> --oltp-test-mode=simple \
> --oltp-reconnect-mode=query \
> --mysql-db=sb \
> --max-requests=1000 \
> --num-threads=128 \
> run
```

The command results in an output similar to the following:

```
sysbench 0.4.12: multi-threaded system evaluation benchmark

Running the test with following options:
Number of threads: 128

Doing OLTP test.
Running simple OLTP test
Using Special distribution (12 iterations, 1 pct of values are
returned in 75 pct cases)
```

```

Using "BEGIN" for starting transactions
Using auto_inc on the id column
Maximum number of requests for OLTP test is limited to 1000
Threads started!
Done.

OLTP test statistics:
    queries performed:
        read:                      1001
        write:                     0
        other:                     1001
        total:                     2002
    transactions:           1001  (6075.34 per sec.)
    deadlocks:                  0      (0.00 per sec.)
    read/write requests:       1001  (6075.34 per sec.)
    other operations:          1001  (6075.34 per sec.)

Test execution summary:
    total time:                0.1648s
    total number of events:     1001
    total time taken by event execution: 19.6382
    per-request statistics:
        min:                      10.46ms
        avg:                   19.62ms
        max:                      61.97ms
        approx. 95 percentile:     49.61ms

Threads fairness:
    events (avg/stddev):      7.8203/1.07
    execution time (avg/stddev): 0.1534/0.00

```

The `run` step for the `sysbench` application executes 10,000 read requests from 128 threads against the MySQL server.

**Note:** The online transactional processing (OLTP) test is not an approximation of an OLTP system, but is instead a true database-backed benchmark that conducts transactional queries to a MySQL server instance.

6. Examine the output and answer the following questions:
  - How many transactions completed per second? **Answer:** 6075.34
  - How long, on an average, did the MySQL server take to complete each request?  
**Answer:** 19.62 ms
7. Drop and re-create the `sb` database for subsequent practices.  
Enter the following command at the Linux terminal prompt:
 

```
# mysql -uroot -p -e"DROP DATABASE sb;CREATE DATABASE sb;"
```
8. Keep the Linux terminal window open for the next practice.

## Practice 3-4: Using Linux System Monitoring Tools

### Overview

In this practice, you use various Linux command-line tools to monitor your system. Note that the actual metrics produced by these tools is not important for this practice and may be different on your system from those shown in the sample output.

### Duration

This practice should take approximately 15 minutes to complete.

### Tasks

#### Load the System

1. Execute the `slap-test.sh` script in the `/root/scripts` directory.  
**Note:** This test runs for several minutes and gives the system something to do while you work on the practice steps. If the script completes before you have finished the steps in this practice, re-execute it.
2. Leave the current terminal window open, and open a new terminal window for the remaining steps in the practice.

#### Use `iostat` to Determine System Concurrency

3. Execute `iostat` at two-second intervals, five times. Display extended system information.
  - a. Enter the following at a Linux terminal prompt as the `root` user and receive results similar to those shown as follows.
  - b. Use the final output of `iostat` and Little's Law to answer the following questions:
    - What is the throughput (`r/s + w/s`)?
    - What is the response time? (`svctm/1000`)?
    - Given the two preceding answers, how many concurrent requests was the system handling per second (throughput \* response time)?

#### Use `vmstat` to Display System Resource Usage

4. Execute `vmstat -a` to display summary information about system resource usage.
  - What was the CPU idle time?
  - How long did the CPU spend waiting for I/O?
  - How much swap space (virtual memory) was used?
5. Execute `vmstat -s` to display event counters and memory statistics.
  - How much memory is being used?
  - How much swap space is being used?
  - How many pages are being swapped out?

## Use `sar` to Report Linux Subsystem Activity

6. Execute `sar 1 3` to display system CPU statistics three times, at a one-second interval. What was the average I/O wait percentage?
7. Execute `sar -r 1 3` to display memory usage statistics three times, at a one-second interval. What is the average percentage of memory required for the current workload in relation to the total amount of memory available?
8. Execute `sar -S 1 3` to display swap statistics three times, at a one-second interval. Is the system swapping?

## Use `top` to Report General System Activity

9. Execute the `top` command without any options. This displays running processes in order of CPU usage. Which process is using the most CPU?
10. While `top` is running, press Shift + M to display the processes in order of memory usage. What percentage of memory is the most resource-intensive process using?
11. Terminate all running sessions and close all terminal windows.
  - Press “q” to quit `top`.
  - Press Ctrl + C to halt the execution of the `slap-test.sh` script.
12. Close all terminal windows.

## Solution 3-4: Using Linux System Monitoring Tools

---

### Steps

#### Load the System

1. Execute the `slap-test.sh` script in the `/root/scripts` directory.

Enter the following command at the Linux terminal prompt:

```
# sh /root/scripts/slап-test.sh
```

**Note:** This test runs for several minutes and gives the system something to do while you work on the practice steps. If the script completes before you have finished the steps in this practice, re-execute it.

2. Leave the current terminal window open, and open a new terminal window for the remaining steps in the practice.

#### Use iostat to Determine System Concurrency

3. Execute `iostat` at two-second intervals, five times. Display extended system information.

  - a. Enter the following at a Linux terminal prompt as the `root` user and receive results that are similar to those shown:

```
# iostat -x 2 5
...
avg-cpu: %user    %nice   %system %iowait  %steal    %idle
          32.10     0.00     6.07    4.38     0.00    57.45

Device:      rrqm/s    wrqm/s     r/s      w/s      rkB/s    wkB/s
avgqrq-sz avgqu-sz    await r_await w_await svctm %util
sda           0.00      0.00    0.00    82.00      0.00  1458.00
 35.56      1.00    12.78      0.00    12.78  11.59    95.05
loop0         0.00      0.00      0.00      0.00      0.00      0.00
 0.00      0.00      0.00      0.00      0.00      0.00      0.00
```

- b. Use the final output of `iostat` and Little's Law to answer the following questions. The answers shown relate to the sample output.
  - What is the throughput (`r/s + w/s`)? **Answer:** 82
  - What is the response time? (`svctm/1000`)? **Answer:** 0.01159
  - Given the two preceding answers, how many concurrent requests was the system handling per second (throughput \* response time)? **Answer:** 0.95

#### Use vmstat to Display System Resource Usage

4. Execute `vmstat -a` to display summary information about system resource usage.

Enter the following command at the Linux terminal prompt and receive results similar to those shown as follows:

```
# vmstat -a
procs -----memory----- --swap-- -----io---- -system-- -----cpu-----
 r b  swpd   free  inact active   si   so   bi   bo   in   cs us sy id wa st
```

4	0	<b>76</b>	6874096	4320940	4543784	0	0	0	117	1	0	0	<b>0 97</b>	2	0
---	---	-----------	---------	---------	---------	---	---	---	-----	---	---	---	-------------	---	---

- What was the CPU idle time? **Answer:** 0 (id)
  - How long did the CPU spend waiting for I/O? **Answer:** 97 (wa)
  - How much swap space (virtual memory) was used? **Answer:** 76 (swpd)
5. Execute `vmstat -s` to display event counters and memory statistics.

Enter the following command at the Linux terminal prompt and receive results similar to those shown as follows:

```
# vmstat -s
    16337688 K total memory
1201976 K used memory
    4543804 K active memory
    4320968 K inactive memory
    6874608 K free memory
    282252 K buffer memory
    7978852 K swap cache
    8388604 K total swap
76 K used swap
    8388528 K free swap
    15970059 non-nice user cpu ticks
        1879 nice user cpu ticks
        3030069 system cpu ticks
    4970543951 idle cpu ticks
    119788521 IO-wait cpu ticks
        283 IRQ cpu ticks
        321109 softirq cpu ticks
            0 stolen cpu ticks
        8600908 pages paged in
    5992542038 pages paged out
            0 pages swapped in
20 pages swapped out
    1361102140 interrupts
    150190873 CPU context switches
    1490888041 boot time
    6730230 forks
```

- How much memory is being used? **Answer:** 1201976K
- How much swap space is being used? **Answer:** 76K
- How many pages are being swapped out? **Answer:** 20

## Use sar to Report Linux Subsystem Activity

6. Execute `sar 1 3` to display system CPU statistics three times, at a one-second interval.

Enter the following command at the Linux terminal prompt and receive results similar to those shown as follows:

```
# sar 1 3
Linux 4.1.12-32.2.1.el7uek.x86_64 (edddr73p1) <date and time> _x86_64_ (8 CPU)

02:35:35 PM    CPU    %user    %nice   %system   %iowait   %steal    %idle
02:35:36 PM    all     0.00     0.00     0.00     11.14     0.00    88.86
02:35:37 PM    all     0.63     0.00     0.13     11.76     0.00    87.48
02:35:38 PM    all     1.75     0.00     0.38      9.01     0.00    88.86
Average:       all     0.79     0.00     0.17     10.64     0.00    88.40
```

What was the average I/O wait percentage? **Answer:** 10.64% (%iowait)

- Execute `sar -r 1 3` to display memory usage statistics three times, at a one-second interval.

Enter the following command at the Linux terminal prompt and receive results similar to those shown as follows:

```
# sar -r 1 3
Linux 4.1.12-32.2.1.el7uek.x86_64 (edddr73p1) 06/12/2017 _x86_64_ (8 CPU)

02:37:37 PM kbmemfree kbmemused %memused kbbuffers kbcached kbcommit %commit
kbactive kbinact kbdirty
02:37:38 PM 6853716 9483972 58.05 282252 7511460 4969812 20.10
4555876 4330032 80
02:37:39 PM 6852668 9485020 58.06 282252 7511460 4969812 20.10
4556620 4330016 80
02:37:40 PM 6853432 9484256 58.05 282252 7511460 4969812 20.10
4555204 4330016 80
Average: 6853272 9484416 58.05 282252 7511460 4969812 20.10
4555900 4330021 80
```

What is the average percentage of memory required for the current workload in relation to the total amount of memory available? **Answer:** 20.10% (%commit)

- Execute `sar -S 1 3` to display swap statistics three times, at a one-second interval.

Enter the following command at the Linux terminal prompt and receive results similar to those shown as follows:

```
# sar -S 1 3
Linux 4.1.12-32.2.1.el7uek.x86_64 (edddr73p1) <date and time>
_x86_64_ (8 CPU)

02:39:59 PM kbswpfree kbswpused %swpused kbswpcad %swpcad
02:40:00 PM 8388528 76 0.00 0 0.00
02:40:01 PM 8388528 76 0.00 0 0.00
02:40:02 PM 8388528 76 0.00 0 0.00
Average: 8388528 76 0.00 0 0.00
```

Is the system swapping? **Answer:** Not in the output shown in the preceding step, because `%swpused` is zero. (Note that the `kbswpused` figure relates to memory that was once swapped out, has now been swapped back in, but still resides in the swap area. This is to minimize file I/O if the memory needs to be swapped back out again.)

## Use `top` to Report General System Activity

9. Execute the `top` command without any options. This displays running processes in order of CPU usage.

Enter the following command at the Linux terminal prompt and receive results similar to those shown as follows:

```
# top
top - 14:49:02 up 73 days, 23:15, 3 users, load average: 4.86, 2.87, 2.27
Tasks: 266 total, 1 running, 265 sleeping, 0 stopped, 0 zombie
%Cpu(s): 37.9 us, 8.0 sy, 0.0 ni, 48.5 id, 5.6 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 16337688 total, 6862332 free, 1212184 used, 8263172 buff/cache
KiB Swap: 8388604 total, 8388528 free, 76 used. 14146972 avail Mem

      PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
16756 mysql     20   0 4765704 464872 20592 S 274.8  2.8 18:54.04 mysqld
23729 root      20   0 3336096 13100 4836 S 93.0  0.1 0:16.64 mysqlslap
  32 root      20   0      0      0      0 S 0.3  0.0 3:51.34 rcuos/3
  363 root     20   0      0      0      0 D 0.3  0.0 18:15.25 jbd2/sda5-8
1479 gdm       20   0 1361540 112168 54512 S 0.3  0.7 7:23.63 gnome-shell
1489 vncuser   20   0 2018016 308780 70360 S 0.3  1.9 14:13.07 gnome-shell
  1 root      20   0 198988 15424 3748 S 0.0  0.1 1:48.44 systemd
  2 root      20   0      0      0      0 S 0.0  0.0 0:01.08 kthreadd
  3 root      20   0      0      0      0 S 0.0  0.0 0:02.16 ksoftirqd/0
  5 root      0 -20      0      0      0 S 0.0  0.0 0:00.00 kworker/0:0H
  7 root      20   0      0      0      0 S 0.0  0.0 13:34.65 rCU_sched
  8 root      20   0      0      0      0 S 0.0  0.0 0:00.00 rCU_bh
  9 root      20   0      0      0      0 S 0.0  0.0 6:52.88 rCUos/0
 10 root     20   0      0      0      0 S 0.0  0.0 0:00.00 rCUob/0
 11 root     rt  0      0      0      0 S 0.0  0.0 0:00.57 migration/0
 12 root     rt  0      0      0      0 S 0.0  0.0 0:15.50 watchdog/0
 13 root     rt  0      0      0      0 S 0.0  0.0 0:14.47 watchdog/1
```

Which process is using the most CPU? **Answer:** mysqld (%CPU)

10. While `top` is running, press Shift + M to display the processes in order of memory usage.

Your output should be similar to the following:

```
...
      PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
16756 mysql     20   0 4765704 464872 20592 S 274.8  2.8 18:54.04 mysqld
23729 root      20   0 3336096 13100 4836 S 93.0  0.1 0:16.64 mysqlslap
  32 root      20   0      0      0      0 S 0.3  0.0 3:51.34 rcuos/3
  363 root     20   0      0      0      0 D 0.3  0.0 18:15.25 jbd2/sda5-8
```

What percentage of memory is the most resource-intensive process using? **Answer:** 9.2% (mysqld)

11. Terminate all running sessions and close all terminal windows.

- Press “q” to quit `top`.
- Press Ctrl + C to halt the execution of the `slap-test.sh` script.

12. Close all terminal windows.

Enter the following command in both Linux terminal windows:

```
# exit
```

GANG LIU (gangli@baylorhealth.edu) has a non-transferable license  
to use this Student Guide.

## **Practices for Lesson 4: Performance Schema**

## Practices for Lesson 4: Overview

---

### Overview

In these practices, you will:

- Configure and query the Performance Schema
- Use `sys` to provide easy access to the Performance Schema
- Use MySQL Workbench for Performance Schema configuration, monitoring, and reporting

**Note:** The output of the various Linux commands and SQL statements shown in the solution steps for these practices might be different on your system.

### Assumptions

- MySQL Workbench is installed.
- The `slap-test.sh` file is in the `/root/scripts` directory.

## Practice 4-1: Examining the Performance Schema Configuration

### Overview

In this practice, you examine the default configuration of the Performance Schema.

### Duration

This practice should take you approximately 10 minutes to complete.

### Tasks

1. Start an interactive MySQL session by invoking the `mysql` client in a Linux terminal window.
2. List all the schemas hosted by this MySQL server instance. Note the presence of the `performance_schema` and `sys` databases.
3. Display the value of the relevant server variable to confirm that the Performance Schema is enabled.
4. Execute a `SHOW ENGINE performance_schema STATUS` statement. How much memory is `performance_schema` currently using?
5. Change the current database to `performance_schema`. Execute a query to list all the configuration tables. How many configuration tables are there?
6. Execute a query to list all the user accounts for which foreground thread monitoring and logging are enabled. Which user accounts have foreground thread monitoring and logging enabled by default?
7. List the contents of the `threads` table for all foreground threads. Locate the row that relates to the current user (`root`). What does the `NAME` column value refer to?
8. Execute a query to list all configured instruments. How many instruments are enabled by default?
9. Execute a query to list all consumers. Which consumers are enabled by default?
10. Execute a query to list all database objects that Performance Schema is monitoring. Is Performance Schema monitoring the `country` table in the `world` database?
11. Keep the Linux terminal window open and logged in to the `mysql` client for the next practice.

## Solution 4-1: Examining the Performance Schema Configuration

---

### Solution Steps

1. Start an interactive MySQL session by invoking the mysql client in a Linux terminal window.

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# mysql -uroot -p
Enter password: oracle
Welcome to the MySQL monitor. Commands end with ; or \g.
...
mysql>
```

2. List all the schemas hosted by this MySQL server instance. Note the presence of the **performance\_schema** and **sys** databases.

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> SHOW DATABASES;
+-----+
| Database      |
+-----+
| information_schema |
| employees      |
| mysql          |
| mysqlslap      |
| performance_schema |
| perf          |
| sakila         |
| sb             |
| sys           |
| world          |
| world_NonNorm  |
+-----+
11 rows in set (#.## sec)
```

3. Display the value of the relevant server variable to confirm that the Performance Schema is enabled.

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> SHOW GLOBAL VARIABLES LIKE 'performance\_\schema%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| performance_schema | ON  |
| performance_schema_accounts_size | -1   |
| performance_schema_digests_size | 10000|
```

```

...
+-----+
42 rows in set (#.# sec)

```

- The performance\_schema database (first row in the output) is enabled by default.
  - The other variables specify the size of the various performance\_schema tables. Some values derive from other relevant settings such as max\_connections, and are automatically calculated.
4. Execute a SHOW ENGINE performance\_schema STATUS statement. How much memory is performance\_schema currently using?

Enter the following statement at the mysql prompt and receive the results shown:

```

mysql> SHOW ENGINE performance_schema STATUS\G
...
***** 229. row *****
Type: performance_schema
Name: performance_schema.memory
Status: 155182264
229 rows in set (#.# sec)

```

- **Answer:** The performance\_schema database currently uses approximately 155 MB of memory. This value might be different on your system.
  - **Note:** Because all performance\_schema data is stored in-memory, changing the size of the tables affects the overall memory usage.
5. Change the current database to performance\_schema. Execute a query to list all the configuration tables. How many configuration tables are there?

Enter the following statements at the mysql prompt and receive the results shown:

```

mysql> USE performance_schema
...
Database changed
mysql> SHOW TABLES LIKE 'setup\_%';
+-----+
| Tables_in_performance_schema (setup\_) |
+-----+
| setup_actors                         |
| setup_consumers                      |
| setup_instruments                    |
| setup_objects                        |
| setup_timers                         |
+-----+
5 rows in set (#.# sec)

```

- **Answer:** Five

6. Execute a query to list all the user accounts for which foreground thread monitoring and logging are enabled. Which user accounts have foreground thread monitoring and logging enabled by default?

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> SELECT * FROM setup_actors;
+-----+-----+-----+-----+
| HOST | USER | ROLE | ENABLED | HISTORY |
+-----+-----+-----+-----+
| %    | %    | %    | YES   | YES   |
+-----+-----+-----+-----+
1 row in set (#.## sec)
```

- **Answer:** Performance Schema logs and monitors all user threads by default.
  - The rule is that if any row in setup\_actors matches a user account, Performance Schema uses the ENABLED and HISTORY column values of that row to populate the INSTRUMENTED and HISTORY columns of the threads table, respectively.
  - Background threads do not have a user account, so Performance Schema monitors all background threads unless you disable them in the threads table.
7. List the contents of the threads table for all foreground threads. Locate the row that relates to the current user (root). What does the NAME column value refer to?

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> SELECT * FROM threads WHERE TYPE='FOREGROUND' \G
...
***** 2. row *****
THREAD_ID: 6896
      NAME: thread/sql/one_connection
      TYPE: FOREGROUND
PROCESSLIST_ID: 6871
PROCESSLIST_USER: root
PROCESSLIST_HOST: localhost
PROCESSLIST_DB: performance_schema
PROCESSLIST_COMMAND: Query
PROCESSLIST_TIME: 0
PROCESSLIST_STATE: Sending data
PROCESSLIST_INFO: SELECT * FROM threads WHERE
TYPE='FOREGROUND'
PARENT_THREAD_ID: NULL
      ROLE: NULL
INSTRUMENTED: YES
      HISTORY: YES
CONNECTION_TYPE: Socket
THREAD_OS_ID: 16416
```

```
2 rows in set (#.## sec)
```

- **Answer:** The value of the NAME column is the associated instrument. The thread/sql/one\_connection instrument corresponds to the function in the code that is responsible for handling a user connection.

8. Execute a query to list all configured instruments. How many instruments are enabled by default?

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> SELECT * FROM setup_instruments WHERE ENABLED='YES';
+-----+-----+-----+
| NAME           | ENABLED | TIMED |
+-----+-----+-----+
| wait/io/file/sql/map      | YES    | YES   |
| wait/io/file/sql/binlog    | YES    | YES   |
| wait/io/file/sql/binlog_cache | YES    | YES   |
| wait/io/file/sql/binlog_index | YES    | YES   |
| ...             |         |       |
| memory/performance_schema/table_handles | YES    | NO    |
| memory/performance_schema/table_shares   | YES    | NO    |
| memory/performance_schema/table_io_waits... | YES    | NO    |
| memory/performance_schema/table_lock_waits... | YES    | NO    |
| memory/performance_schema/events_statements... | YES    | NO    |
| memory/performance_schema/prepared_statements... | YES    | NO    |
| memory/performance_schema/scalable_buffer    | YES    | NO    |
+-----+-----+-----+
327 rows in set (#.## sec)
```

- **Answer:** 327 instruments are enabled by default.

9. Execute a query to list all consumers. Which consumers are enabled by default?

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> SELECT * FROM setup_consumers;
+-----+-----+
| NAME           | ENABLED |
+-----+-----+
| events_stages_current | NO     |
| events_stages_history | NO     |
| events_stages_history_long | NO     |
| events_statements_current | YES   |
| events_statements_history | YES   |
| events_statements_history_long | NO     |
| events_transactions_current | NO     |
| events_transactions_history | NO     |
| events_transactions_history_long | NO     |
| events_waits_current | NO     |
| events_waits_history | NO     |
| events_waits_history_long | NO     |
```

```

| global_instrumentation | YES   |
| thread_instrumentation | YES   |
| statements_digest     | YES   |
+-----+
15 rows in set (#.## sec)

```

- **Answer:** The events\_statements\_current, events\_statements\_history, global\_instrumentation, thread\_instrumentation, and statements\_digest consumers are enabled by default.

10. Execute a query to list all database objects that Performance Schema is monitoring. Is Performance Schema monitoring the country table in the world database?

Enter the following statement at the mysql prompt and receive the results shown:

```

mysql> SELECT * FROM setup_objects;
+-----+-----+-----+-----+-----+
| OBJECT_TYPE | OBJECT_SCHEMA | OBJECT_NAME | ENABLED | TIMED |
+-----+-----+-----+-----+-----+
| EVENT       | mysql          | %           | NO      | NO    |
| EVENT       | performance_schema | %           | NO      | NO    |
| EVENT       | information_schema | %           | NO      | NO    |
| EVENT       | %               | %           | YES     | YES   |
| FUNCTION    | mysql          | %           | NO      | NO    |
| FUNCTION    | performance_schema | %           | NO      | NO    |
| FUNCTION    | information_schema | %           | NO      | NO    |
| FUNCTION    | %               | %           | YES     | YES   |
| PROCEDURE   | mysql          | %           | NO      | NO    |
| PROCEDURE   | performance_schema | %           | NO      | NO    |
| PROCEDURE   | information_schema | %           | NO      | NO    |
| PROCEDURE   | %               | %           | YES     | YES   |
| TABLE       | mysql          | %           | NO      | NO    |
| TABLE       | performance_schema | %           | NO      | NO    |
| TABLE       | information_schema | %           | NO      | NO    |
| TABLE       | %               | %           | YES     | YES   |
| TRIGGER     | mysql          | %           | NO      | NO    |
| TRIGGER     | performance_schema | %           | NO      | NO    |
| TRIGGER     | information_schema | %           | NO      | NO    |
| TRIGGER     | %               | %           | YES     | YES   |
+-----+-----+-----+-----+-----+
20 rows in set (#.## sec)

```

- Is the country table in the world database being monitored? **Answer:** Yes (All user tables in all user schemas are monitored by default.)

11. Keep the Linux terminal window open and logged in to the mysql client for the next practice.

## Practice 4-2: Querying the Performance Schema

---

### Overview

In this practice, you execute queries against the Performance Schema. But first, you must configure the Performance Schema by enabling the appropriate consumers to gather the event data that you are interested in.

### Duration

This practice should take you approximately 10 minutes to complete.

### Tasks

1. Issue a `TRUNCATE` statement to reset the counters in the `table_io_waits_summary_by_table` table.
2. Execute the following query against the `world.Country` table:

```
SELECT Code, Name, Continent
  FROM world.country
 WHERE NAME='China';
```

3. Issue the following query against the Performance Schema's `table_io_waits_summary_by_table` table:

```
SELECT OBJECT_SCHEMA, OBJECT_NAME,
       COUNT_STAR, SUM_TIMER_WAIT
  FROM table_io_waits_summary_by_table
 WHERE OBJECT_SCHEMA='world'
   AND OBJECT_NAME='country';
```

How many wait events are recorded for the `world.country` table?

4. Issue a query to determine what the timing units are for the value of the `SUM_TIMER_WAIT` column.
5. Examine the summary wait event data for all waits by querying the `events_waits_summary_global_by_event_name` table.
6. To examine detailed wait information, you must enable the `events_waits_current` consumer. To view historical wait data, you must also enable `events_waits_history` (for ten most recent events per thread) and optionally, `events_waits_history_long` (for 10,000 most recent events per thread). Enable these consumers and examine the statistics that they store by performing the following steps:
  - a. Determine if the `events_waits_*` consumers are enabled.
  - b. If necessary, enable the `events_waits_*` consumers in the `setup_consumers` table.
  - c. Query the `events_waits_current` table to display all current wait events. What is the server doing at the moment?
  - d. Query the `events_waits_history` table to display recent wait events.

7. Keep the Linux terminal window open and logged in to the `mysql` client for the next practice.

## Solution 4-2: Querying the Performance Schema

---

### Solution Steps

- Issue a TRUNCATE statement to reset the counters in the `table_io_waits_summary_by_table` table.

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> TRUNCATE table_io_waits_summary_by_table;
Query OK, 0 rows affected (#.## sec)
```

- Execute the following query against the `world.Country` table:

```
SELECT Code, Name, Continent
FROM world.country
WHERE Name='China';
```

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> SELECT Code, Name, Continent
-> FROM world.country
-> WHERE Name='China';
+----+----+-----+
| Code | Name  | Continent |
+----+----+-----+
| CHN | China | Asia   |
+----+----+-----+
1 row in set (#.## sec)
```

– This step generates some events for analysis in `performance_schema`.

- Issue the following query against the Performance Schema's `table_io_waits_summary_by_table` table:

```
SELECT OBJECT_SCHEMA, OBJECT_NAME,
COUNT_STAR, SUM_TIMER_WAIT
FROM table_io_waits_summary_by_table
WHERE OBJECT_SCHEMA='world'
AND OBJECT_NAME='country';
```

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> SELECT OBJECT_SCHEMA, OBJECT_NAME,
-> COUNT_STAR, SUM_TIMER_WAIT
-> FROM table_io_waits_summary_by_table
-> WHERE OBJECT_SCHEMA='world'
-> AND OBJECT_NAME='Country';
+-----+-----+-----+-----+
| OBJECT_SCHEMA | OBJECT_NAME | COUNT_STAR | SUM_TIMER_WAIT |
+-----+-----+-----+-----+
| world        | country      |       239 |     136996525 |
+-----+-----+-----+-----+
```

```
1 row in set (#.## sec)
```

- How many wait events are recorded for the world.country table? **Answer:** 239
- **Note:** Your results might be different from those shown in the sample output.

4. Issue a query to determine what the timing units are for the value of the SUM\_TIMER\_WAIT column.

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> SELECT * FROM setup_timers WHERE NAME='wait';
+-----+-----+
| NAME | TIMER_NAME |
+-----+-----+
| wait | CYCLE      |
+-----+-----+
1 row in set (#.## sec)
```

- The timing unit is CYCLE, which is based on processor speed. To time wait events, the most important criterion is to reduce overhead at the possible expense of timer accuracy, so using the CYCLE timer is the best approach.

5. Examine summary wait event data for all waits by querying the events\_waits\_summary\_global\_by\_event\_name table.

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> SELECT EVENT_NAME, COUNT_STAR, MAX_TIMER_WAIT
    -> FROM events_waits_summary_global_by_event_name
    -> ORDER BY COUNT_STAR DESC LIMIT 10;
+-----+-----+-----+
| EVENT_NAME          | COUNT_STAR | MAX_TIMER_WAIT |
+-----+-----+-----+
| wait/io/table/sql/handler | 17763941 | 224147896025905 |
| idle                | 885483   | 2880010008500000 |
| wait/synch/mutex/innodb/buf_pool_mutex | 253352   | 16775470 |
| wait/synch/mutex/innodb/log_sys_mutex | 249618   | 16284295 |
| wait/synch/mutex/innodb/flush_list_mutex | 165804   | 13275295 |
| wait/synch/mutex/innodb/fil_system_mutex | 86627    | 13909840 |
| wait/synch/mutex/innodb/dict_sys_mutex | 82905    | 13652600 |
| wait/synch/mutex/innodb/log_flush_order_mutex | 82898   | 13271165 |
| wait/synch/mutex/innodb/log_sys_write_mutex | 82897   | 15075385 |
| wait/synch/mutex/innodb/buf dblwr_mutex | 82894   | 17250715 |
+-----+-----+-----+
10 rows in set (#.## sec)
```

- The waits that resulted from querying the world.country table and the maximum wait time are included in the summary information for all waits. The statistics will probably be different on your system.

6. To examine detailed wait information, you must enable the events\_waits\_current consumer. To view historical wait data, you must also enable events\_waits\_history (for ten most recent events per thread) and optionally, events\_waits\_history\_long (for 10,000 most recent events per thread). Enable these consumers and examine the statistics that they store by performing the following steps:

- a. Determine if the `events_waits_*` consumers are enabled.

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> SELECT * FROM setup_consumers
      -> WHERE NAME LIKE 'events\_\waits\_\%';
+-----+-----+
| NAME           | ENABLED |
+-----+-----+
| events_waits_current | NO     |
| events_waits_history | NO     |
| events_waits_history_long | NO    |
+-----+-----+
3 rows in set (#.## sec)
```

- None of the `events_waits_*` consumers are enabled by default.

- b. If necessary, enable the `events_waits_*` consumers in the `setup_consumers` table.

Enter the following statements at the `mysql` prompt and receive the results shown:

```
mysql> UPDATE setup_consumers SET ENABLED='YES'
      -> WHERE NAME LIKE 'events\_\waits\_\%';
Query OK, 3 rows affected (#.## sec)
Rows matched: 3  Changed: 3  Warnings: 0

mysql> SELECT * FROM setup_consumers
      -> WHERE NAME LIKE 'events\_\waits\_\%';
+-----+-----+
| NAME           | ENABLED |
+-----+-----+
| events_waits_current | YES    |
| events_waits_history | YES    |
| events_waits_history_long | YES   |
+-----+-----+
3 rows in set (#.## sec)
```

- c. Query the `events_waits_current` table to display all current wait events.

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> SELECT THREAD_ID, EVENT_NAME, OPERATION, TIMER_WAIT
      -> FROM events_waits_current;
+-----+-----+-----+-----+
| THREAD_ID | EVENT_NAME       | OPERATION | TIMER_WAIT   |
+-----+-----+-----+-----+
|       6897 | idle            | idle      | 38478566000000 |
+-----+-----+-----+-----+
7 rows in set (#.## sec)
```

- What is the server doing at the moment? **Answer:** Nothing. It is idle.

- **Note:** You may also see wait events for mutexes recorded in the `events_waits_current` table.
- d. Query the `events_waits_history` table to display recent wait events.

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> SELECT THREAD_ID, EVENT_NAME, OPERATION, TIMER_WAIT
-> FROM events_waits_history;
+-----+-----+-----+
| THREAD_ID | EVENT_NAME           | OPERATION      | TIMER_WAIT   |
+-----+-----+-----+
|       6897 | idle                 | idle          | 1825135900000 |
|       6897 | idle                 | idle          | 38478566000000 |
|       6897 | idle                 | idle          | 131612551000000 |
|       6897 | idle                 | idle          | 1002055457000000 |
|       6897 | wait/lock/table/sql/handler | read external | 883525      |
|       6897 | wait/io/table/sql/handler | fetch         | 2081832110    |
|       6897 | idle                 | idle          | 65376660000000 |
+-----+-----+-----+
7 rows in set (#.## sec)
```

**Note:** Your results might be different from those shown in the sample output.

7. Keep the Linux terminal window open and logged in to the `mysql` client for the next practice.

## Practice 4-3: Using sys to Work with the Performance Schema

### Overview

In this practice, you use `sys` views, stored functions, and stored procedures to work with the Performance Schema.

### Duration

This practice should take you approximately 15 minutes to complete.

### Tasks

1. Issue a query that lists all the rows in the `sys` schema's `wait_classes_global_by_avg_latency` view.
2. Execute a `SHOW CREATE VIEW` statement on the `sys` schema's `wait_classes_global_by_avg_latency` view and examine the SQL syntax of the query that creates this view.
3. Execute a `SHOW FULL TABLES` statement to list the views that are available in the `sys` schema.
4. Issue the following query to list the stored functions that are available in the `sys` schema.

```
SELECT ROUTINE_NAME  
FROM INFORMATION_SCHEMA.ROUTINES  
WHERE ROUTINE_SCHEMA = 'sys'  
AND ROUTINE_TYPE = 'FUNCTION';
```

5. Modify the query in the preceding step to display all the stored procedures in the `sys` schema.
6. Invoke the MySQL `sys` schema stored procedure `ps_truncate_all_tables` to reset the `performance_schema` metrics. How many rows does the `table_io_waits_summary_by_table` table now contain? Why?
7. Invoke the built-in MySQL function `CONNECTION_ID()` to display the ID of the current connection.
8. Invoke the `sys` schema's `ps_thread_id()` function by using the MySQL connection ID that you determined in the preceding step to display the internal Performance Schema thread ID for the current connection.
9. Query the `processlist` view in the `sys` schema for the thread identified in the preceding step to retrieve extra information about the thread's current state.
10. Use the MySQL `sys` schema procedure `ps_setup_reset_to_default` to restore the default `performance_schema` configuration.
11. Exit all `mysql` client sessions and close all Linux terminal windows.

## Solution 4-3: Using sys to Work with the Performance Schema

### Solution Steps

- Issue a query that lists all the rows in the `sys` schema's `wait_classes_global_by_avg_latency` view.

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> SELECT * FROM SYS.WAIT_CLASSES_GLOBAL_BY_AVG_LATENCY\G
***** 1. row *****
  event_class: wait/io/file
      total: 170480
total_latency: 5.73 m
  min_latency: 0 ps
  avg_latency: 2.02 ms
  max_latency: 500.22 ms
***** 2. row *****
  event_class: wait/io/table
      total: 10282066
total_latency: 4.38 m
  min_latency: 35.40 ns
  avg_latency: 25.56 us
  max_latency: 1.71 s
***** 3. row *****
  event_class: wait/lock/table
      total: 62872
total_latency: 15.25 ms
  min_latency: 38.94 ns
  avg_latency: 242.54 ns
  max_latency: 7.04 us
3 rows in set (#.# sec)
```

- The `sys` schema provides several useful views that present Performance Schema event data in a useful format that is easy to read. This is just one example.
- Execute a `SHOW CREATE VIEW` statement on the `sys` schema's `wait_classes_global_by_avg_latency` view and examine the SQL syntax of the query that creates this view.

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> SHOW CREATE VIEW
-> sys.wait_classes_global_by_avg_latency\G
***** 1. row *****
          View: wait_classes_global_by_avg_latency
Create View: CREATE ALGORITHM=TEMPTABLE
DEFINER='mysql.sys'@'localhost' SQL SECURITY INVOKER VIEW
`sys`.`wait_classes_global_by_avg_latency` AS select
```

```

substring_index(`performance_schema`.`events_waits_summary_global_by_event_name`.`EVENT_NAME`, '/', 3) AS
`event_class`, sum(`performance_schema`.`events_waits_summary_global_by_event_name`.`COUNT_STAR`) AS
`total`, `sys`.`format_time`(cast(sum(`performance_schema`.`events_waits_summary_global_by_event_name`.`SUM_TIMER_WAIT`) as unsigned)) AS
`total_latency`, `sys`.`format_time`(min(`performance_schema`.`events_waits_summary_global_by_event_name`.`MIN_TIMER_WAIT`)) AS
`min_latency`, `sys`.`format_time`(ifnull((sum(`performance_schema`.`events_waits_summary_global_by_event_name`.`SUM_TIMER_WAIT`)/
nullif(sum(`performance_schema`.`events_waits_summary_global_by_event_name`.`COUNT_STAR`),0)),0)) AS
`avg_latency`, `sys`.`format_time`(cast(max(`performance_schema`.`events_waits_summary_global_by_event_name`.`MAX_TIMER_WAIT`) as unsigned)) AS
`max_latency` from
`performance_schema`.`events_waits_summary_global_by_event_name` where
((`performance_schema`.`events_waits_summary_global_by_event_name`.`EVENT_NAME` > 0) and
(`performance_schema`.`events_waits_summary_global_by_event_name`.`EVENT_NAME` <> 'idle')) group by `event_class` order by
ifnull((sum(`performance_schema`.`events_waits_summary_global_by_event_name`.`SUM_TIMER_WAIT`)/
nullif(sum(`performance_schema`.`events_waits_summary_global_by_event_name`.`COUNT_STAR`),0)),0) desc
character_set_client: utf8
collation_connection: utf8_general_ci
1 row in set (#.## sec)

```

- The SQL that is required to create the output of the `sys` view is complex. Check to see if a suitable `sys` view already exists before attempting to write a query yourself.
  - Note that the `sys` schema also includes several functions, such as `format_time()`, which present the raw Performance Schema data in a more user-friendly format. You can use these functions in your own queries.
3. Execute a `SHOW FULL TABLES` statement to list the views that are available in the `sys` schema.

Enter the following statement at the `mysql` prompt and receive the results shown:

Tables_in_sys	Table_type
host_summary	VIEW
host_summary_by_file_io	VIEW
host_summary_by_file_io_type	VIEW
host_summary_by_stages	VIEW
host_summary_by_statement_latency	VIEW
host_summary_by_statement_type	VIEW
innodb_buffer_stats_by_schema	VIEW

```

| innodb_buffer_stats_by_table          | VIEW   |
| innodb_lock_waits                   | VIEW   |
| io_by_thread_latency                | VIEW   |
...
| x$host_summary                      | VIEW   |
| x$host_summary_by_file_io           | VIEW   |
| x$host_summary_by_file_io_type     | VIEW   |
| x$host_summary_by_stages            | VIEW   |
| x$host_summary_by_statement_latency | VIEW   |
| x$host_summary_by_statement_type    | VIEW   |
| x$innodb_buffer_stats_by_schema    | VIEW   |
| x$innodb_buffer_stats_by_table     | VIEW   |
| x$innodb_lock_waits                | VIEW   |
| x$io_by_thread_latency              | VIEW   |
...
+-----+
100 rows in set (#.## sec)

```

- The `sys` schema contains many views that provide useful insights into Performance Schema statistics. In general, you should avoid using views with the `x$` prefix because these contain unformatted data that is best suited for integration with other tools.

4. Issue the following query to list the stored functions that are available in the `sys` schema.

```

SELECT ROUTINE_NAME
FROM INFORMATION_SCHEMA.ROUTINES
WHERE ROUTINE_SCHEMA = 'sys'
AND ROUTINE_TYPE = 'FUNCTION';

```

Enter the following statement at the `mysql` prompt and receive the results shown:

```

mysql> SELECT ROUTINE_NAME
      -> FROM INFORMATION_SCHEMA.ROUTINES
      -> WHERE ROUTINE_SCHEMA = 'sys'
      -> AND ROUTINE_TYPE = 'FUNCTION';
+-----+
| ROUTINE_NAME               |
+-----+
| extract_schema_from_file_name |
| extract_table_from_file_name |
| format_bytes                 |
| format_path                  |
| format_statement              |
| format_time                  |
| list_add                     |
| list_drop                     |
+-----+

```

```

| ps_is_account_enabled          |
| ps_is_consumer_enabled         |
| ps_is_instrument_default_enabled|
| ps_is_instrument_default_timed |
| ps_is_thread_instrumented     |
| ps_thread_account              |
| ps_thread_id                   |
| ps_thread_stack                |
| ps_thread_trx_info             |
| quote_identifier               |
| sys_get_config                 |
| version_major                  |
| version_minor                  |
| version_patch                  |
+-----+
22 rows in set (#.## sec)

```

- The `sys` schema contains stored functions that query the Performance Schema configuration and provide formatting services.
5. Modify the query in the preceding step to display all the stored procedures in the `sys` schema.

Enter the following statement at the `mysql` prompt and receive the results shown:

```

mysql> SELECT ROUTINE_NAME
      -> FROM INFORMATION_SCHEMA.ROUTINES
      -> WHERE ROUTINE_SCHEMA = 'sys'
      -> AND ROUTINE_TYPE = 'PROCEDURE';
+-----+
| ROUTINE_NAME           |
+-----+
| create_synonym_db      |
| diagnostics            |
| execute_prepared_stmt  |
| ps_setup_disable_background_threads |
| ps_setup_disable_consumer |
| ps_setup_disable_instrument |
| ps_setup_disable_thread  |
| ps_setup_enable_background_threads |
| ps_setup_enable_consumer |
| ps_setup_enable_instrument |
| ps_setup_enable_thread   |
| ps_setup_reload_saved    |
| ps_setup_reset_to_default |
| ps_setup_save            |
+-----+

```

```

| ps_setup_show_disabled           |
| ps_setup_show_disabled_consumers|
| ps_setup_show_disabled_instruments|
| ps_setup_show_enabled           |
| ps_setup_show_enabled_consumers|
| ps_setup_show_enabled_instruments|
| ps_statement_avg_latency_histogram|
| ps_trace_statement_digest      |
| ps_trace_thread                |
| ps_truncate_all_tables         |
| statement_performance_analyzer|
| table_exists                   |
+-----+
26 rows in set (#.## sec)

```

- The sys schema contains stored functions that perform operations such as Performance Schema configuration and generation of diagnostic reports.
6. Invoke the MySQL sys schema stored procedure ps\_truncate\_all\_tables to reset the performance\_schema metrics. How many rows does the table\_io\_waits\_summary\_by\_table table now contain? Why?

Enter the following statements at the mysql prompt and receive the results shown:

```

mysql> CALL sys.ps_truncate_all_tables(TRUE);
...
+-----+
| summary          |
+-----+
| Truncated 44 tables |
+-----+
1 row in set (#.## sec)

Query OK, 0 rows affected (#.## sec)

mysql> SELECT COUNT(*) FROM table_io_waits_summary_by_table;
+-----+
| COUNT(*) |
+-----+
|       31 |
+-----+
1 row in set (#.## sec)

mysql> SELECT * FROM table_io_waits_summary_by_table LIMIT 1\G
***** 1. row *****
OBJECT_TYPE: TABLE

```

```

OBJECT_SCHEMA: employees
OBJECT_NAME: departments
COUNT_STAR: 0
SUM_TIMER_WAIT: 0
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 0
MAX_TIMER_WAIT: 0
...
COUNT_DELETE: 0
SUM_TIMER_DELETE: 0
MIN_TIMER_DELETE: 0
AVG_TIMER_DELETE: 0
MAX_TIMER_DELETE: 0
1 row in set (#.## sec)

```

- **Answer:** The number of rows in `table_io_waits_summary_by_table` is non-zero. This is because the `TRUNCATE` statement that the `sys.ps_truncate_all_tables` stored procedure executes does not delete summary table records. Instead, `TRUNCATE` sets their values to zero.
  - **Note:** The `ps_truncate_all_tables` procedure in the MySQL `sys` schema is only one of many stored routines in `sys` that make the Performance Schema easier to work with. The Boolean value that you passed in as an argument determines whether the output of the command is verbose. This course uses other `sys` schema views and stored routines in later lessons.
7. Invoke the built-in MySQL function `CONNECTION_ID()` to display the ID of the current connection.
- Enter the following statement at the `mysql` prompt and receive the results shown:

```

mysql> SELECT CONNECTION_ID();
+-----+
| CONNECTION_ID() |
+-----+
|          6872 |
+-----+
1 row in set (#.## sec)

```

- Your connection ID will probably be different.
8. Invoke the `sys` schema's `ps_thread_id()` function by using the MySQL connection ID that you determined in the preceding step to display the internal Performance Schema thread ID for the current connection.

Enter the following statement at the `mysql` prompt and receive the results shown:

```

mysql> SELECT sys.ps_thread_id(6872);
+-----+
| sys.ps_thread_id(6872) |
+-----+

```

```
|          6897 |
+-----+
1 row in set (#.# sec)
```

- Your thread ID will probably be different.

9. Query the `processlist` view in the `sys` schema for the thread that was identified in the preceding step to retrieve extra information about the thread's current state.

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> SELECT * FROM sys.processlist
-> WHERE thd_id=6897\G
***** 1. row *****
      thd_id: 6897
      conn_id: 6872
        user: root@localhost
        db: performance_schema
      command: Query
        state: Sending data
        time: 0
  current_statement: SELECT * FROM sys.processlist WHERE
thd_id=6897
statement_latency: 1.17 ms
      progress: NULL
  lock_latency: 882.00 us
rows_examined: 0
  rows_sent: 0
rows_affected: 0
  tmp_tables: 4
tmp_disk_tables: 1
      full_scan: YES
last_statement: NULL
last_statement_latency: NULL
  current_memory: 0 bytes
      last_wait: wait/lock/table/sql/handler
last_wait_latency: 956.68 ns
      source: handler.cc:7840
  trx_latency: NULL
  trx_state: NULL
  trx_autocommit: NULL
      pid: 9951
program_name: mysql
1 row in set (#.# sec)
```

- Replace the thread ID with the appropriate value that was derived in the preceding step.

- **Note:** The sys schema processlist view is based on the Performance Schema threads table. It contains information that is similar to the output of a SHOW PROCESSLIST statement, but using Performance Schema for this is generally preferred because it is non-blocking.
10. Use the MySQL sys schema procedure ps\_setup\_reset\_to\_default to restore the default performance\_schema configuration:

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> CALL sys.ps_setup_reset_to_default(TRUE);  
...  
| Resetting: setup_actors  
...  
| Resetting: setup_instruments  
...  
| Resetting: setup_consumers  
...  
| Resetting: setup_objects  
...  
Query OK, 0 rows affected (#.## sec)
```

11. Exit all mysql client sessions and close all Linux terminal windows.

## Practice 4-4: Using MySQL Workbench for Performance Schema Configuration, Monitoring, and Reporting

### Overview

In this practice, you take a tour of MySQL Workbench's Performance Schema configuration options, and its performance monitoring and reporting capabilities.

### Duration

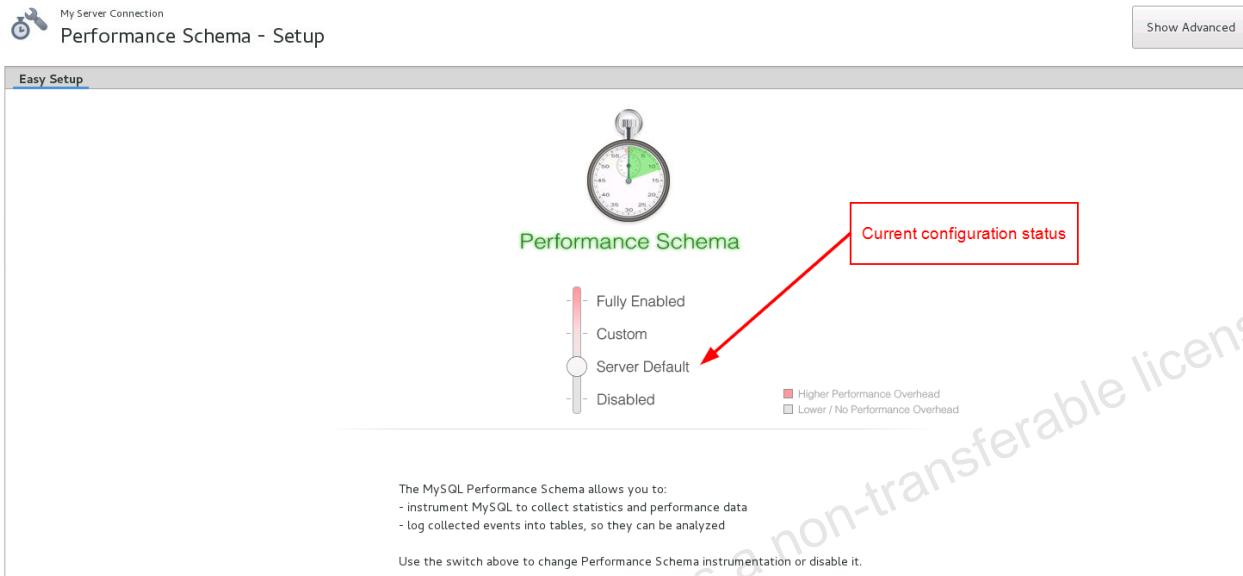
This practice should take approximately 20 minutes to complete.

### Tasks

#### Connect to the MySQL Server

1. Select Applications > Programming > MySQL Workbench on the Linux desktop. The MySQL Workbench welcome screen appears.
2. Create a new connection to the MySQL server:
  - a. Click the plus (+) symbol next to MySQL Connections.
  - b. In the Setup New Connection dialog box, enter the following details:
    - Connection Name: My Server Connection
    - Hostname: Keep the default hostname (127.0.0.1).
    - Port: Keep the default port (3306).
    - Username: `root`
    - Default Schema: `sakila`
  - c. Click Test Connection.
  - d. Enter the password `oracle` in the Connect to MySQL Server dialog box and click OK.
  - e. In the “Successfully made the MySQL connection confirmation” dialog box, click OK.
  - f. In the Setup New Connection dialog box, click OK.
3. Connect to the MySQL server:
  - a. Click the My Server Connection link on the MySQL Workbench home screen.
  - b. If you are prompted for a password:
    - 1) In the Connect to MySQL Server dialog box, enter the password `oracle` and select the “Save password in keychain” option.
    - 2) Click OK.
    - 3) When prompted, create and confirm the new keychain password of `oracle`.
  - c. The My Server Connection page opens, with the Navigation pane on the left and a Query Editor window on the right. The `sakila` database displays in bold under the Schemas section of the Navigation pane. This is because you specified `sakila` as the default schema when you created the connection.
    - **Note:** You might need to maximize the MySQL Workbench window to see the Schemas section of the Navigation pane.

- In the Navigation pane, under the Performance heading, click Performance Schema Setup. The Administration - Performance Schema Setup tab opens, displaying the Easy Setup page. Note from the switch at the center of the page that the Performance Schema is enabled and is currently using the default server configuration.



- Click the Show Advanced button at the top-right corner of the tab. New tabs appear within the Administration - Performance Schema Setup tab.



- Click the Introduction tab and read the information provided.
- Click the following tabs in turn and note both the current configuration options and those that you can enable. Do not make any changes to the default configuration at this stage.
  - Instruments
  - Consumers
  - Actors & Objects
  - Threads
  - Options
- Click the Hide Advanced button. The tabs for advanced configuration options disappear.
- Close the Administration - Performance Schema Setup tab.

## Evaluate the MySQL Workbench Performance Monitoring and Reporting Capabilities

10. Open a Linux terminal window. Enter the following command to execute the `/root/scripts/slap-test.sh` script. This script executes frequent queries against the `employees` database, to emulate a load on the MySQL server.

```
# sh /root/scripts/slap-test.sh
```

  - The script executes for several minutes. Continue with the remaining steps in this practice while the script is running.
11. In the Navigation pane of MySQL Workbench, click the Dashboard link in the Performance section. Observe the network activity, MySQL status, and InnoDB status while the script executes.
12. Click the Performance Reports link in the Navigation pane and look at the list of reports that are available.
  - a. Select Hot Spots for I/O > Top File I/O Activity Report. Note the file I/O statistics for the `employees` schema.
  - b. Select High Cost SQL Statements > Statement Analysis report. Note the statements that executed most frequently and scanned the most rows.
    - **Note:** You might need to scroll to the right to see the columns that contain these statistics.
  - c. Select the Database Schema Statistics > Unused Indexes report. Observe which employee database table indexes were not used.
  - d. Select Wait Event Times (Expert) > Waits by User by Time. Observe the wait event statistics.
  - e. Select the InnoDB Statistics > InnoDB Buffer Stats by Schema report. Note how this report uses data from `INFORMATION_SCHEMA` to summarize buffer statistics.
13. Close MySQL Workbench.
14. In the Linux terminal window, enter `Ctrl+C` to stop the execution of the `slap-test.sh` script if it is still running.
15. Exit all `mysql` client sessions and close all Linux terminal windows.

## **Solution 4-4: Using MySQL Workbench for Performance Schema Configuration, Monitoring, and Reporting**

---

There are no solution steps for this practice.

GANG LIU (gangli@baylorhealth.edu) has a non-transferable license  
to use this Student Guide.

## **Practices for Lesson 5: General Server Tuning**

## Practices for Lesson 5: Overview

---

### Overview

In these practices, you will:

- Estimate the amount of memory MySQL uses
- Manage the number of clients connected to the MySQL server
- Investigate the effects of thread caching

**Note:** The output of the various Linux commands and SQL statements shown in the solution steps for these practices might be different on your system.

### Assumptions

- The MySQL server is running.
- You have stopped the MySQL Enterprise Monitor Service Manager service, as instructed in step 35 of the practice titled “Using MySQL Enterprise Monitor” in the lesson titled “Performance Tuning Tools.”
- SysBench is installed.
- You created the `sb` database in the practice titled “Using Benchmark Tools” in the lesson titled “Performance Tuning Tools.”
- The following scripts are in the `/root/scripts` directory and are executable:
  - `perft-sysbench-clean.sh`
  - `perft-sysbench-prepare.sh`
  - `perft-sysbench-run.sh`
  - `thread_cache_hit_rate.sql`
- LibreOffice Calc or equivalent spreadsheet software is installed and the `Lesson5Practice.ods` spreadsheet exists in the `/root/labs` directory.

## Practice 5-1: Estimating Total Thread Memory Usage

### Overview

In this practice, you estimate the total thread memory usage for the MySQL server.

### Duration

This practice should take you approximately 10 minutes to complete.

### Tasks

1. Use the mysqladmin command-line client in a Linux terminal window to inspect the values of the following system variables:
  - max\_connections
  - sort\_buffer\_size
  - read\_rnd\_buffer\_size
  - join\_buffer\_size
  - binlog\_cache\_size
  - thread\_stack
  - tmp\_table\_size
  - max\_heap\_table\_size
2. Enter values for each of these variables in the “5-1” worksheet of the Lesson05Practice.ods spreadsheet in the /root/labs directory.

The spreadsheet calculates the total thread memory usage by using the following formula:

```
max_connections *  
  (sort_buffer_size +  
   read_rnd_buffer_size +  
   join_buffer_size +  
   binlog_cache_size +  
   thread_stack +  
   [the smaller of tmp_table_size and max_heap_table_size])
```

3. What is the total thread memory usage calculated by the spreadsheet?
4. What is the more realistic thread memory usage calculated by the spreadsheet?
5. Leave the Linux terminal window open for the next practice.

## Solution 5-1: Estimating Total Thread Memory Usage

---

### Steps

1. Use the mysqladmin command-line client in a Linux terminal window to inspect the values of the following system variables:

- a. The max\_connections system variable

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# mysqladmin -uroot -p variables | grep max_conn
Enter password: oracle
| max_connect_errors | 100 |
| max_connections | 151 |
```

- b. The sort\_buffer\_size system variable

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# mysqladmin -uroot -p variables | grep sort_buffer_size
Enter password: oracle
| innodb_sort_buffer_size | 1048576 |
| myisam_sort_buffer_size | 8388608 |
| sort_buffer_size | 262144 |
```

- c. The read\_rnd\_buffer\_size system variable

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# mysqladmin -uroot -p variables | grep read_rnd
Enter password: oracle
| read_rnd_buffer_size | 262144 |
```

- d. The join\_buffer\_size system variable

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# mysqladmin -uroot -p variables | grep join_buffer
Enter password: oracle
| join_buffer_size | 262144 |
```

- e. The binlog\_cache\_size system variable

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# mysqladmin -uroot -p variables | grep binlog_cache
Enter password: oracle
| binlog_cache_size | 32768 |
| max_binlog_cache_size | 18446744073709547520 |
```

- f. The thread\_stack system variable

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# mysqladmin -uroot -p variables | grep thread_st
Enter password: oracle
| thread_stack | 262144 |
```

- g. The `tmp_table_size` system variable

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# mysqladmin -uroot -p variables | grep tmp_table
Enter password: oracle
| max_tmp_tables | 32 |
| tmp_table_size | 16777216 |
```

- h. The `max_heap_table_size` system variable

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# mysqladmin -uroot -p variables | grep max_heap
Enter password: oracle
| max_heap_table_size | 16777216 |
```

2. Enter values for each of these variables in the “5-1” worksheet of the Lesson05Practice.ods spreadsheet in the /root/labs directory.

The spreadsheet calculates the total thread memory usage by using the following formula:

```
max_connections *
  (sort_buffer_size +
  read_rnd_buffer_size +
  join_buffer_size +
  binlog_cache_size +
  thread_stack +
  [the smaller of tmp_table_size and max_heap_table_size])
```

3. What is the total thread memory usage calculated by the spreadsheet?

The answer is:

$151 * (262144 + 262144 + 262144 + 32768 + 262144 + 16777216) =$
<b>2696642560 bytes (2.47 GB)</b>

4. What is the more realistic thread memory usage calculated by the spreadsheet?

The spreadsheet divides the total thread memory usage calculated in step 2 by four to determine a more realistic thread memory usage.

The answer is:

$2696642560 / 4 = 674160640 bytes (633.49 MB)$
--

**Note:** This formula will not be useful for all systems. You might need to adjust it to calculate a realistic thread memory usage value for your server workload.

5. Leave the Linux terminal window open for the next practice.

## Practice 5-2: Managing the Number of Client Connections

### Overview

In this practice, you review the response from the MySQL server when you try to open more connections than the setting of the `max_connections` system variable allows. To accomplish this objective, perform the following:

- View the current `max_connections` system variable setting.
- Execute a test against the MySQL server, which simulates more connections than the `max_connections` system variable setting allows.
- Increase the value of `max_connections` and repeat the test.

### Duration

This practice should take you approximately 30 minutes to complete.

### Tasks

1. In a Linux terminal window, log in as the `root` user and use the `mysqladmin` command to display the maximum number of connections that the MySQL server allows. What is the value of the `max_connections` system variable?
2. Examine the contents of the `/root/scripts/perft-sysbench-prepare.sh` script. This script prepares the SysBench test environment. It accepts a single argument for the number of rows to create in the `sbtest` table of the `sb` database.
3. Examine the contents of the `/root/scripts/perft-sysbench-run.sh` script. This script executes the SysBench test and accepts three arguments:
  - The number of rows in the test table (`$numrows`)
  - The number of client connections to simulate (`$numthreads`)
  - A limit for the total number of requests (`$maxrequests`. This is optional. The default value is 10000.)
4. Examine the contents of the `/root/scripts/perft-sysbench-clean.sh` script. This script drops the `sbtest` table that the SysBench prepare step creates, from the `sb` database.
5. Prepare the test environment by executing the `perft-sysbench-prepare.sh` script. Specify a test table containing 100,000 rows.
6. Execute the `perft-sysbench-run.sh` script. Specify the following arguments:
  - A test table size of 100,000 rows
  - 200 client connections
  - A limit of 1,000 requestsDoes the script produce any errors?
7. Use the `mysqladmin extended-status` command to review the maximum number of connections that MySQL has had open simultaneously since the last server restart.

8. Use the `mysql` client non-interactively to increase the number of connections that the MySQL server can handle by raising the value of `max_connections` to 250. Use the `mysqladmin` client program to verify that the change is made.
9. Execute the `perf-t-sysbench-run.sh` script again, with the same arguments as before. Does the script produce any errors?
10. Use the `mysqladmin` client to review the maximum number of connections that MySQL has had open simultaneously since the last server restart.
11. Restart the MySQL server.
12. Execute the `perf-t-sysbench-run.sh` script again, with the same arguments as before. Does the script produce any errors?
13. Use the `mysqladmin` client to review the maximum number of connections that MySQL has had open simultaneously since the last server restart.
14. Add a configuration entry to the `/etc/my.cnf` file to make the increase to the `max_connections` setting permanent.
15. Restart the MySQL server.
16. Execute the `perf-t-sysbench-run.sh` script again, with the same arguments as before. Does the script produce any errors?
17. Use the `mysqladmin` client to review the maximum number of connections that MySQL has had open simultaneously since the last server restart.
18. Execute the `perf-t-sysbench-clean.sh` script to delete the test data.
19. Remove the `max_connections` entry from the `/etc/my.cnf` file.
20. Restart the MySQL server.
21. Leave the Linux terminal window open for the next practice.

## Solution 5-2: Managing the Number of Client Connections

---

### Solution Steps

1. In a Linux terminal window, log in as the `root` user and use the `mysqladmin` command to display the maximum number of connections that the MySQL server allows.

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# mysqladmin -uroot -p variables | grep max_connections
Enter password: oracle
| max_connections | 151 |

```

– What is the value of the `max_connections` system variable? **Answer: 151**

2. Examine the contents of the `/root/scripts/perft-sysbench-prepare.sh` script. This script prepares the SysBench test environment. It accepts a single argument for the number of rows to create in the `sbtest` table of the `sb` database.

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# cat /root/scripts/perft-sysbench-prepare.sh
#!/bin/bash

# check for command-line arguments
re='^[0-9]+$'
if ! [[ $1 =~ $re ]] ; then
    echo "ERROR: Expecting number of table rows!" >&2; exit 1
fi

# prepare sysbench test table with passed in number of rows
sysbench --db-driver=mysql --test=oltp --oltp-table-size=$1 --
mysql-user=root --mysql-password=oracle --mysql-db=sb --mysql-
table-engine=innodb prepare
```

3. Examine the contents of the `/root/scripts/perft-sysbench-run.sh` script. This script executes the SysBench test and accepts three arguments:

- The number of rows in the test table (`$numrows`)
- The number of client connections to simulate (`$numthreads`)
- A limit for the total number of requests (`$maxrequests`. This is optional. The default value is 10000.)

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# cat /root/scripts/perft-sysbench-run.sh
#!/bin/bash

# check for command-line arguments
numrows=$1
numthreads=$2
# if no parameter passed for max-requests, default to 10,000
```

```

maxreqs=${3-10000}

re='^ [0-9]+$'
if ! [[ $numrows =~ $re ]] ; then
    echo "ERROR: Expecting number of rows in the test
table!" >&2; exit 1
fi
if ! [[ $numthreads =~ $re ]] ; then
    echo "ERROR: Expecting number of connections to
simulate!" >&2; exit 1
fi

# execute sysbench test with arguments specified
sysbench --db-driver=mysql --test=oltp --oltp-test-mode=simple
--oltp-reconnect-mode=query --oltp-table-size=$numrows
--mysql-user=root --mysql-password=oracle --mysql-db=sb
--max-requests=$maxreqs --num-threads=$numthreads run

```

4. Examine the contents of the /root/scripts/perft-sysbench-clean.sh script. This script drops the sbtest table that the SysBench prepare step creates, from the sb database.

Enter the following command at the Linux terminal prompt and receive the results shown:

```

# cat /root/scripts/perft-sysbench-clean.sh
#!/bin/bash

sysbench --db-driver=mysql --test=oltp --mysql-user=root --
mysql-password=oracle --mysql-db=sb cleanup

```

5. Prepare the test environment by executing the perft-sysbench-prepare.sh script. Specify a test table containing 100,000 rows.

Enter the following commands at the Linux terminal prompt and receive the results shown:

```

# cd /root/scripts
# sh perft-sysbench-prepare.sh 100000
sysbench 0.4.12: multi-threaded system evaluation benchmark

Creating table 'sbtest'...
Creating 100000 records in table 'sbtest'...
#

```

6. Execute the perft-sysbench-run.sh script. Specify the following arguments:

- A test table size of 100,000 rows
- 200 client connections
- A limit of 1,000 requests

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# sh perft-sysbench-run.sh 100000 200 1000
sysbench 0.4.12: multi-threaded system evaluation benchmark

FATAL: unable to connect to MySQL server, aborting...
FATAL: error 1040: Too many connections
FATAL: failed to connect to database server!
FATAL: thread#152: failed to connect to database server,
aborting...
#
```

- Does the script produce any errors? **Answer:** The script produced a “Too many connections” error, preventing further connections to the MySQL server.
7. Use the mysqladmin extended-status command to review the maximum number of connections that MySQL has had open simultaneously since the last server restart.

**Note:** mysqladmin extended-status displays the server status variables and their values.

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# mysqladmin -uroot -p extended-status | grep Max_used
Enter password: oracle
| Max_used_connections | 152 |
| Max_used_connections_time | <date and time> |
```

- The server has actually allowed one more client to connect than the value of max\_connections (151). This extra connection is reserved for accounts with the SUPER privilege, so that an administrator can connect to the server and troubleshoot, even if the maximum number of unprivileged clients are connected.
8. Use the mysql client non-interactively to increase the number of connections that the MySQL server can handle by raising the value of max\_connections to 250. Use the mysqladmin client program to verify that the change is made.

Enter the following commands at the Linux terminal prompt and receive the results shown:

```
# mysql -uroot -p -e"SET GLOBAL max_connections=250"
Enter password: oracle
# mysqladmin -u root -p variables | grep max_connections
Enter password: oracle
| max_connections | 250 |
```

9. Execute the perft-sysbench-run.sh script again, with the same arguments as before.

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# sh perft-sysbench-run.sh 100000 200 1000
sysbench 0.4.12: multi-threaded system evaluation benchmark

Running the test with following options:
Number of threads: 200

Doing OLTP test.
```

```

Running simple OLTP test
Using Special distribution (12 iterations, 1 pct of values are
returned in 75 pct cases)
Using "BEGIN" for starting transactions
Using auto_inc on the id column
Maximum number of requests for OLTP test is limited to 1000
Threads started!
Done.

OLTP test statistics:
  queries performed:
    read:                      1000
    write:                     0
    other:                     1000
    total:                     2000
  transactions:              1000  (6415.91 per sec.)
  deadlocks:                 0     (0.00 per sec.)
  read/write requests:       1000  (6415.91 per sec.)
  other operations:          1000  (6415.91 per sec.)

Test execution summary:
  total time:                0.1559s
  total number of events:    1000
  total time taken by event execution: 27.6211
  per-request statistics:
    min:                      10.41ms
    avg:                      27.62ms
    max:                      97.35ms
    approx. 95 percentile:    84.17ms

Threads fairness:
  events (avg/stddev):      5.0000/1.85
  execution time (avg/stddev): 0.1381/0.01

```

- Does the script produce any errors? **Answer:** The script executed without any errors.
10. Use the `mysqladmin` client to review the maximum number of connections that MySQL has had open simultaneously since the last server restart.

Enter the following command at the Linux terminal prompt and receive the results shown:

```

# mysqladmin -uroot -p extended-status | grep 'Max_used'
Enter password: oracle
| Max_used_connections | 226 |
| Max_used_connections_time | <date and time> |

```

- The value of `Max_used_connections` might be different on your server. The important thing is that the MySQL server was able to open enough connections to complete the test.

11. Restart the MySQL server.

Enter the following commands at a Linux terminal prompt and receive the results shown:

```
# service mysql restart
Shutting down MySQL.. SUCCESS!
Starting MySQL. SUCCESS!
```

12. Execute the `perf-sysbench-run.sh` script again, with the same arguments as before.

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# sh perf-sysbench-run.sh 100000 200 1000
sysbench 0.4.12: multi-threaded system evaluation benchmark

FATAL: unable to connect to MySQL server, aborting...
FATAL: error 1040: Too many connections
FATAL: failed to connect to database server!
FATAL: thread#152: failed to connect to database server,
aborting...
```

- Does the script produce any errors? **Answer:** The script produced a “Too many connections” error, preventing further connections to the MySQL server. When the MySQL server restarts, the `max_connections` setting returns to its default value.

13. Use the `mysqladmin` client to review the maximum number of connections that MySQL has had open simultaneously since the last server restart.

Enter the following command at the Linux terminal prompt:

```
# mysqladmin -uroot -p extended-status | grep 'Max_used'
Enter password: oracle
| Max_used_connections | 152 |
| Max_used_connections_time | <date and time> |
```

14. Add a configuration entry to the `/etc/my.cnf` file to make the increase to the `max_connections` setting permanent.

- a. Edit the MySQL server options file, `/etc/my.cnf`, by using a text editor such as `nano`:

```
# nano /etc/my.cnf
```

- b. Add the following line under the `[mysqld]` section of `my.cnf`:

```
[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
user=mysql
# Disabling symbolic-links is recommended to prevent assorted
# security risks
symbolic-links=0
max_connections=225
...
```

- c. Save the `my.cnf` file and close the text editor. (You can save the file in `nano` by pressing `Ctrl + O` and exit `nano` by pressing `Ctrl + X`.)
15. Restart the MySQL server.

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# service mysql restart
Shutting down MySQL.. SUCCESS!
Starting MySQL. SUCCESS!
```

16. Execute the `perf-sysbench-run.sh` script again, with the same arguments as before.

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# sh perf-sysbench-run.sh 100000 200 1000
sysbench 0.4.12: multi-threaded system evaluation benchmark

Running the test with following options:
Number of threads: 200

Doing OLTP test.
Running simple OLTP test
Using Special distribution (12 iterations, 1 pct of values are
returned in 75 pct cases)
Using "BEGIN" for starting transactions
Using auto_inc on the id column
Maximum number of requests for OLTP test is limited to 1000
Threads started!
Done.

OLTP test statistics:
    queries performed:
        read:                      1001
        write:                     0
        other:                     1001
        total:                     2002
    transactions:                1001  (7104.57 per sec.)
    deadlocks:                  0      (0.00 per sec.)
    read/write requests:        1001  (7104.57 per sec.)
    other operations:           1001  (7104.57 per sec.)

Test execution summary:
    total time:                 0.1409s
    total number of events:     1001
    total time taken by event execution: 24.4241
    per-request statistics:
        min:                      10.62ms
        avg:                      24.40ms
        max:                      63.42ms
```

```

approx. 95 percentile: 55.10ms

Threads fairness:
  events (avg/stddev): 5.0050/1.26
  execution time (avg/stddev): 0.1221/0.01

```

- Does the script produce any errors? **Answer:** No

17. Use the `mysqladmin` client to review the maximum number of connections that MySQL has had open simultaneously since the last server restart.

Enter the following command at the Linux terminal prompt and receive the results shown:

```

# mysqladmin -uroot -p extended-status | grep 'Max_used'
Enter password: oracle
| Max_used_connections | 218 |
| Max_used_connections_time | <date and time> |

```

- The value of `Max_used_connections` might be different on your machine.

18. Execute the `perf-sysbench-clean.sh` script to delete the test data.

Enter the following command at the Linux terminal prompt and receive the results shown:

```

# sh perf-sysbench-clean.sh
sysbench 0.4.12: multi-threaded system evaluation benchmark

Dropping table 'sbtest'...
Done.
#

```

19. Remove the `max_connections` entry from the `/etc/my.cnf` file.

Using a text editor such as `nano`, edit the `/etc/my.cnf` file to read as follows:

```

[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
user=mysql
# Disabling symbolic-links is recommended to prevent assorted
# security risks
symbolic-links=0

[mysqld_safe]
log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid

[client]
socket=/var/lib/mysql/mysql.sock

```

20. Restart the MySQL server.

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# service mysql restart
Shutting down MySQL.. SUCCESS!
Starting MySQL. SUCCESS!
```

21. Leave the Linux terminal window open for the next practice.

## Practice 5-3: Investigating the Effects of Multiple Simultaneous Connections

### Overview

In this practice, you evaluate the effect of numerous connections on the memory used by the MySQL server. To accomplish this objective, you perform the following:

- Set the `max_connections` system variable to handle 1,024 connections.
- Execute queries against the MySQL server by using an increasing number of connections.
- Evaluate the amount of virtual (swapped and physical) and physical (non-swapped) memory used by each execution.

**Note:** This practice uses two Linux terminal windows. The instructions refer to the terminal windows as `t1` and `t2`. To make it easier to identify each terminal, change its title by using the Terminal > Set Title menu option in each terminal window.

### Assumptions

You have a terminal window open and are logged in as the Linux `root` user. This practice refers to this terminal window as `t1`.

### Duration

This practice should take you approximately 15 minutes to complete.

### Tasks

1. In terminal window `t1`, use the `mysql` client non-interactively to increase the maximum number of connections that the MySQL server allows by setting the value of `max_connections` to 2,048.
2. In `t1`, execute the `/root/scripts/perft-sysbench-prepare.sh` script to create a test table with 5,000,000 rows.

**Note:** This step takes several minutes to complete. Allow it to finish before proceeding.
3. In `t1`, execute the `/root/scripts/perft-sysbench-run.sh` script with the following arguments:
  - Size of the test table: 5,000,000 rows
  - Number of connections to simulate: 8
  - A limit of 100,000 requests

**Note:** Do not wait for the script to complete. Proceed immediately to the next step.
4. Open a new Linux terminal window (`t2`). In `t2`, use the `top` command to analyze the MySQL server (`mysqld`) memory usage.
  - What is the average amount of virtual memory (`VIRT`: fifth column, shown in kilobytes) that your system is using?

- What is the average amount of physical memory (RES: sixth column, shown in kilobytes) that your system is using?
5. In t1, allow the `perf-t-sysbench-run.sh` script to complete and make a note of the total time for its execution.
6. Repeat steps 3 to 5, changing the number of connections parameter for each execution of `perf-t-sysbench-run.sh` to the following values:
- 256 connections:  

```
# sh perf-t-sysbench-run.sh 5000000 256 100000
```
  - 512 connections:  

```
# sh perf-t-sysbench-run.sh 5000000 512 100000
```
- Record the average amount of virtual and physical memory during each execution of the script. What happens to the MySQL memory usage as the number of connections increases?
7. Leave the t1 and t2 terminal windows open for the next practice.

## Solution 5-3: Investigating the Effects of Multiple Simultaneous Connections

### Solution Steps

1. In terminal window t1, use the `mysql` client non-interactively to increase the maximum number of connections that the MySQL server allows by setting the value of `max_connections` to 2,048.

Enter the following commands at the t1 Linux terminal prompt and receive the results shown:

```
# mysql -uroot -p -e"SET GLOBAL max_connections=2048"  
Enter password: oracle  
# mysqladmin -uroot -p variables | grep max_connections  
Enter password: oracle  
| max_connections | 2048 |
```

2. In t1, execute the `/root/scripts/perft-sysbench-prepare.sh` script to create a test table with 5,000,000 rows.

**Note:** This step takes several minutes to complete. Allow it to finish before proceeding.

Enter the following commands at the t1 Linux terminal prompt and receive the results shown:

```
# cd /root/scripts  
# sh perft-sysbench-prepare.sh 5000000  
sysbench 0.4.12: multi-threaded system evaluation benchmark  
  
Creating table 'sbtest'...  
Creating 5000000 records in table 'sbtest'...  
#
```

3. In t1, execute the `perft-sysbench-run.sh` script with the following arguments:

- Size of the test table: 5,000,000 rows
- Number of connections to simulate: 8
- A limit of 100,000 requests

**Note:** Do not wait for the script to complete. Proceed immediately to the next step.

Enter the following command at the t1 Linux terminal prompt and receive the results shown:

```
# sh perft-sysbench-run.sh 5000000 8 100000  
sysbench 0.4.12: multi-threaded system evaluation benchmark  
  
Running the test with following options:  
Number of threads: 8  
  
Doing OLTP test.
```

```

Running simple OLTP test
Using Special distribution (12 iterations, 1 pct of values are
returned in 75 pct cases)
Using "BEGIN" for starting transactions
Using auto_inc on the id column
Maximum number of requests for OLTP test is limited to 100000
Threads started!

```

4. Open a new Linux terminal window (*t2*). In *t2*, use the `top` command to analyze the MySQL server (`mysqld`) memory usage.

Enter the following command at the *t2* Linux terminal prompt and receive the results shown:

```

# top -n 8 | grep mysqld
24584 mysql 20 0 4771896 459084 20060 S 33.3 2.8 0:35.17 mysqld
24584 mysql 20 0 4771896 459084 20060 S 31.9 2.8 0:36.13 mysqld
24584 mysql 20 0 4771896 459084 20060 S 30.6 2.8 0:37.05 mysqld
24584 mysql 20 0 4771896 459084 20060 S 31.3 2.8 0:37.99 mysqld
24584 mysql 20 0 4771896 459084 20060 S 31.2 2.8 0:38.93 mysqld
24584 mysql 20 0 4771896 459084 20060 S 30.9 2.8 0:39.86 mysqld
24584 mysql 20 0 4771896 459084 20060 S 32.0 2.8 0:40.82 mysqld
24584 mysql 20 0 4771896 459084 20060 S 31.2 2.8 0:41.76 mysqld

```

- What is the average amount of virtual memory (VIRT: fifth column, shown in kilobytes) that your system is using? **Answer:** 4772 MB in the sample output
- What is the average amount of physical memory (RES: sixth column, shown in kilobytes) that your system is using? **Answer:** 459 MB in the sample output

5. In *t1*, allow the `perf-t-sysbench-run.sh` script to complete and make a note of the total time for its execution.

When the `perf-t-sysbench-run.sh` script completes, it displays the following summary information:

```

...
Test execution summary:
total time: 130.7927s
total number of events: 100009
total time taken by event execution: 1046.2253
per-request statistics:
    min: 10.33ms
    avg: 10.46ms
    max: 11.38ms
    approx. 95 percentile: 10.59ms

Threads fairness:
events (avg/stddev): 12501.1250/3.22
execution time (avg/stddev): 130.7782/0.00

```

6. Repeat steps 3 to 5, changing the number of connections parameter for each execution of `perf-t-sysbench-run.sh` to the following values:

- 256 connections:

```
# sh perf-t-sysbench-run.sh 5000000 256 100000
```

- 512 connections:

```
# sh perf-t-sysbench-run.sh 5000000 512 100000
```

- a. Record the average amount of virtual and physical memory during each execution of the script.

Sample statistics:

Number of Connections	Average Virtual Memory	Average Physical Memory	Approximate Time Elapsed
8	4772 MB	459 MB	131 secs
256	4937 MB	597 MB	11 secs
512	5227 MB	797 MB	13 secs

- b. What happens to the MySQL memory usage as the number of connections increases?

**Answer:** The amount of memory used increases with more connections. The default thread-handling model in MySQL server executes statements by using one thread per client connection. As more clients connect to the server and execute statements, the overall performance degrades and the time elapsed levels out. The MySQL thread pool plugin provides an alternative thread-handling model. This improves server performance by efficiently managing statement execution threads for large numbers of client connections.

7. Leave the `t1` and `t2` terminal windows open for the next practice.

## Practice 5-4: Investigating the Effects of Thread Caching

### Overview

In this practice, you execute several random queries against the MySQL server by using multiple connections to determine the best setting for the `thread_cache_size` system variable. To accomplish this objective, you perform the following tasks:

- Simulate an environment where 128 connections result in approximately 900 transactions that run against the MySQL server every second.
- Modify the `thread_cache_size` system variable, test, and then calculate the thread cache hit rate.
  - The closer the thread cache hit rate is to 100%, the fewer new threads the MySQL server must create to handle connections. This results in less overhead on the server.

**Note:** This practice uses two Linux terminal windows. The instructions refer to the terminal windows as `t1` and `t2`. To make it easier to identify each terminal, change its title by using the Terminal > Set Title menu option in each terminal window.

### Assumptions

- You have completed the practice titled “Investigating the Effects of Multiple Simultaneous Connections” in this lesson.
- You have two terminal windows (`t1` and `t2`) open from the previous practice.
- You have stopped the MySQL Enterprise Monitor Service Manager as instructed in step 35 of the practice titled “Using MySQL Enterprise Monitor” in the lesson titled “Performance Tuning Tools.”
- The `thread_cache_hit_rate.sql` script is in the `/root/scripts` directory.

### Duration

This practice should take you approximately 20 minutes to complete.

### Tasks

1. In the Linux terminal window `t1`, restart the MySQL server.
2. In the Linux terminal window `t2`, log in to the `mysql` client as the MySQL `root` user.
3. Use the `mysql` client in `t2` to set the global `thread_cache_size` system variable to 0 and the `max_connections` system variable to 300.
4. In `t1`, examine the `/root/scripts/thread_cache_hit_rate.sql` script. This script creates a stored procedure called `display_thread_cache_hit_rate`. The stored procedure queries the `performance_schema.global_status` table to retrieve the values of the `Threads_created` and `Connections` status variables, and uses them to calculate the thread cache hit rate.
5. Use the `mysql` client in `t2` to execute the `/root/scripts/thread_cache_hit_rate.sql` script. This script creates the `display_thread_cache_hit_rate` stored procedure in the `perf` database.

6. In the t1 terminal window, execute the `/root/scripts/perft-sysbench-run.sh` script with the following arguments:
  - The test table contains 5,000,000 rows.
  - The number of connections is 128.
  - The limit on the number of requests is 10,000.

**Note:** You prepared the SysBench environment with a table containing 5,000,000 rows in the practice titled “Evaluating the Effect of Multiple Simultaneous Connections” in this lesson, so you do not need to run the `perft-sysbench-prepare.sh` script first.
7. Use the `mysql` client in t2 to execute the `display_thread_cache_hit_rate` stored procedure in the `perft` database.
8. Repeat steps 6 and 7 twice more. Note the values of Threads Created, Connections, and Thread Cache Hit Rate (%) for each execution. Is the thread cache hit rate getting better or worse as commands execute against the MySQL server?
9. Use the `mysql` client in t2 to set the global `thread_cache_size` system variable to 8.
10. In the t1 terminal window, execute the `perft-sysbench-run.sh` script again by using the same arguments.
11. Use the `mysql` client in t2 to call the `display_thread_cache_hit_rate` stored procedure and measure the effectiveness of the resized thread cache.
12. Repeat steps 10 and 11 twice more. Note the values of Threads Created, Connections, and Thread Cache Hit Rate (%) for each iteration. Is the thread cache hit rate getting better or worse as commands execute against the MySQL server?
13. Use the `mysql` client in t2 to set the global `thread_cache_size` system variable to 16.
14. In the t1 terminal window, execute the `perft-sysbench-run.sh` script again by using the same arguments.
15. Use the `mysql` client in t2 to call the `display_thread_cache_hit_rate` stored procedure and measure the effectiveness of the resized thread cache.
16. Repeat steps 14 and 15 twice more. Note the values of Threads Created, Connections, and Thread Cache Hit Rate (%) for each iteration. Is the thread cache hit rate getting better or worse as commands execute against the MySQL server?
17. Exit the `mysql` client in t2, and close the t1 and t2 Linux terminal windows.

## Solution 5-4: Investigating the Effects of Thread Caching

### Solution Steps

1. In the Linux terminal window t1, restart the MySQL server.

Enter the following command at the t1 Linux terminal prompt and receive the results shown:

```
# service mysql restart
Shutting down MySQL.. SUCCESS!
Starting MySQL. SUCCESS!
```

2. In the Linux terminal window t2, log in to the mysql client as the MySQL root user.

Enter the following command at the t2 Linux terminal prompt and receive the results shown:

```
# mysql -uroot -p
Enter password: oracle
Welcome to the MySQL monitor. Commands end with ; or \g.
...
mysql>
```

3. Use the mysql client in t2 to set the global thread\_cache\_size system variable to 0 and the max\_connections system variable to 300.

Enter the following statements at the mysql prompt in t2 and receive the results shown:

```
mysql> SET GLOBAL thread_cache_size=0;
Query OK, 0 rows affected (#.## sec)

mysql> SHOW GLOBAL VARIABLES LIKE 'thread\_\cache\_%';
+-----+-----+
| Variable_name      | Value   |
+-----+-----+
| thread_cache_size | 0       |
+-----+-----+
1 row in set (#.## sec)

mysql> SET GLOBAL max_connections=300;
Query OK, 0 rows affected (#.## sec)

mysql> SHOW GLOBAL VARIABLES LIKE 'max\_\connections';
+-----+-----+
| Variable_name      | Value   |
+-----+-----+
| max_connections   | 300    |
+-----+-----+
1 row in set (#.## sec)
```

4. In t1, examine the `/root/scripts/thread_cache_hit_rate.sql` script. This script creates a stored procedure called `display_thread_cache_hit_rate`. The stored procedure queries the `performance_schema.global_status` table to retrieve the values of the `Threads_created` and `Connections` status variables, and uses them to calculate the thread cache hit rate.

Enter the following command at the t1 Linux terminal prompt and receive the results shown:

```
# cat /root/scripts/thread_cache_hit_rate.sql
...
SELECT VARIABLE_VALUE INTO created FROM
    performance_schema.global_status
    WHERE VARIABLE_NAME = 'Threads_created';
SELECT VARIABLE_VALUE INTO conns FROM
    performance_schema.global_status
    WHERE VARIABLE_NAME = 'Connections';
SELECT ROUND(100 - (created / conns) * 100,1) INTO hitrate;
...
```

5. Use the `mysql` client in t2 to execute the `/root/scripts/thread_cache_hit_rate.sql` script. This script creates the `display_thread_cache_hit_rate` stored procedure in the `perf` database.

Enter the following statement at the `mysql` prompt in t2 and receive the results shown:

```
mysql> SOURCE /root/scripts/thread_cache_hit_rate.sql;
Query OK, 1 row affected, 1 warning (#.## sec)

Database changed
Query OK, 0 rows affected (#.## sec)

Query OK, 0 rows affected (#.## sec)
```

6. In the t1 terminal window, execute the `/root/scripts/perf-sysbench-run.sh` script with the following arguments:

- The test table contains 5,000,000 rows.
- The number of connections is 128.
- The limit on the number of requests is 10,000.

**Note:** You prepared the SysBench environment with a table containing 5,000,000 rows in the practice titled “Evaluating the Effect of Multiple Simultaneous Connections” in this lesson, so you do not need to run the `perf-sysbench-prepare.sh` script first.

Enter the following commands at the t1 Linux terminal prompt and receive the results shown:

```
# cd /root/scripts
# sh perf-sysbench-run.sh 5000000 128 10000
sysbench 0.4.12: multi-threaded system evaluation benchmark
```

Running the test with following options:  
 Number of threads: 128  
 ...  
 Threads fairness:  
 events (avg/stddev): 78.1875/2.76  
 execution time (avg/stddev): 1.7104/0.00

7. Use the mysql client in t2 to execute the display\_thread\_cache\_hit\_rate stored procedure in the perft database.

Enter the following statements at the mysql prompt in t2 and receive the results shown:

```
mysql> USE perft
Database changed

mysql> CALL display_thread_cache_hit_rate;
+-----+-----+
| Threads Created | Connections | Thread Cache Hit Rate (%) |
+-----+-----+
| 10138          | 10141      | 0                      |
+-----+-----+
1 row in set (#.# sec)

Query OK, 0 rows affected (#.# sec)
```

8. Repeat steps 6 and 7 twice more. Note the values of Threads Created, Connections, and Thread Cache Hit Rate (%) for each execution.
  - Is the thread cache hit rate getting better or worse as commands execute against the MySQL server? **Answer:** The number of threads and connections increase, but the thread cache hit rate is zero, because the thread cache is disabled.
9. Use the mysql client in t2 to set the global thread\_cache\_size system variable to 8.

Enter the following statements at the mysql prompt in t2 and receive the results shown:

```
mysql> SET GLOBAL thread_cache_size=8;
Query OK, 0 rows affected (#.# sec)

mysql> SHOW GLOBAL VARIABLES LIKE 'thread\cache\_%';
+-----+-----+
| Variable_name   | Value  |
+-----+-----+
| thread_cache_size | 8      |
+-----+-----+
1 row in set (#.# sec)
```

10. In the t1 terminal window, execute the perft-sysbench-run.sh script again by using the same arguments.

Enter the following command at the t1 Linux terminal prompt and receive the results shown:

```
# sh perft-sysbench-run.sh 5000000 128 10000
sysbench 0.4.12: multi-threaded system evaluation benchmark

Running the test with following options:
Number of threads: 128 ...
```

11. Use the mysql client in t2 to call the display\_thread\_cache\_hit\_rate stored procedure and measure the effectiveness of the resized thread cache.

Enter the following statement at the mysql prompt in t2 and receive the results shown:

```
mysql> CALL display_thread_cache_hit_rate;
+-----+-----+-----+
| Threads Created | Connections | Thread Cache Hit Rate (%) |
+-----+-----+-----+
| 32639           | 40562     | 19.5                  |
+-----+-----+-----+
1 row in set (#.# sec)

Query OK, 0 rows affected (#.# sec)
```

12. Repeat steps 10 and 11 twice more. Note the values of Threads Created, Connections, and Thread Cache Hit Rate (%) for each iteration.

Compare your results with the following sample outputs:

```
mysql> CALL display_thread_cache_hit_rate;
+-----+-----+-----+
| Threads Created | Connections | Thread Cache Hit Rate (%) |
+-----+-----+-----+
| 34730           | 50698     | 31.5                  |
+-----+-----+-----+
1 row in set (#.# sec)

Query OK, 0 rows affected (#.# sec)

mysql> CALL display_thread_cache_hit_rate;
+-----+-----+-----+
| Threads Created | Connections | Thread Cache Hit Rate (%) |
+-----+-----+-----+
| 36737           | 60829     | 39.6                  |
+-----+-----+-----+
1 row in set (#.# sec)

Query OK, 0 rows affected (#.# sec)
```

Is the thread cache hit rate getting better or worse as commands execute against the MySQL server? **Answer:** In the first few executions, the thread cache hit rate improves. If you continue to execute the test, the thread cache hit rate will remain constant if the number of threads connected is the same.

13. Use the mysql client in t2 to set the global `thread_cache_size` system variable to 16.

Enter the following statements at the mysql prompt in t2 and receive the results shown:

```
mysql> SET GLOBAL thread_cache_size=16;
Query OK, 0 rows affected (#.## sec)

mysql> SHOW GLOBAL VARIABLES LIKE 'thread\_\cache\_%';
+-----+-----+
| Variable_name      | Value   |
+-----+-----+
| thread_cache_size | 16      |
+-----+-----+
1 row in set (#.## sec)
```

14. In the t1 terminal window, execute the `perf-t-sysbench-run.sh` script again by using the same arguments.

Enter the following command at the t1 Linux terminal prompt and receive the results shown:

```
# sh perf-t-sysbench-run.sh 5000000 128 10000
sysbench 0.4.12: multi-threaded system evaluation benchmark

Running the test with following options:
Number of threads: 128
...
```

15. Use the mysql client in t2 to call the `display_thread_cache_hit_rate` stored procedure and measure the effectiveness of the resized thread cache.

Enter the following statement at the mysql prompt in t2 and receive the results shown:

```
mysql> CALL display_thread_cache_hit_rate;
+-----+-----+-----+
| Threads Created | Connections | Thread Cache Hit Rate (%) |
+-----+-----+-----+
| 37831          | 70968     | 46.7                  |
+-----+-----+-----+
1 row in set (#.## sec)

Query OK, 0 rows affected (#.## sec)
```

16. Repeat steps 14 and 15 twice more. Note the values of Threads Created, Connections, and Thread Cache Hit Rate (%) for each iteration.

Is the thread cache hit rate getting better or worse as commands execute against the MySQL server? **Answer:** In the first few executions, the thread cache hit rate improves. If you continue to execute the test, the thread cache hit rate will remain constant if the number of threads connected is the same.

17. Exit the mysql client in t<sub>2</sub>, and close the t<sub>1</sub> and t<sub>2</sub> Linux terminal windows:

Enter the following command at the t<sub>1</sub> Linux terminal prompt:

```
# exit
```

Enter the following commands at the t<sub>2</sub> Linux terminal prompt:

```
mysql> EXIT  
Bye  
# exit
```

## **Practices for Lesson 6: Tuning Tables, Files, and Logs**

## Practices for Lesson 6: Overview

---

### Overview

In these practices, you test your knowledge of tuning the MySQL server settings in relation to tables, files, and logs.

**Note:** The output of the various Linux commands and SQL statements shown in the solution steps for these practices might be different on your system.

## Practice 6-1: Sizing the Table Cache

### Overview

In this practice, you execute the queries logged by your production system against the MySQL server to determine the best setting for the `table_open_cache` system variable. If the table cache is too small, it will result in numerous disk I/O events, which adversely affects performance. If the table cache is too large, it wastes memory.

To accomplish this objective, you use a script with the following characteristics:

- It simulates 20 connections to the MySQL server.
- Each connection executes 500 queries, five times.
- Each query needs to open 500 tables in the `many_tables` database that you will create during the practice.

You will execute the script, and then examine the server status variables to determine the table cache hit rate. You will use this information to calculate the optimal size for `table_open_cache`, modify its value, and re-execute the test by using the new setting.

### Assumptions

- The MySQL server is installed and running.
- The following files are in the `/root/scripts` directory:
  - `create_many_tables.sql`: Creates the `many_tables` database
  - `query_many_tables.sql`: Queries each of the 500 tables in the `many_tables` database
- The MySQL Enterprise Monitor service is not running. (You stopped it in the practices for the lesson titled “Performance Tuning Tools.”)

### Duration

This practice should take you approximately 20 minutes to complete.

### Tasks

1. Open a terminal window as the Linux `root` user. This is terminal window `t1`.
2. Create the `many_tables` database by executing the `/root/scripts/create_many_tables.sql` file in `t1`.  
**Note:** The script takes a few minutes to complete.
3. Open another Linux terminal window and log in to the `mysql` client. This is terminal window `t2`.
4. In `t2`, query the `INFORMATION_SCHEMA.tables` table to display a count of the number of tables in the `many_tables` database.
5. In `t1`, use a Linux tool such as `cat` or `less`, or a text editor such as `nano`, to display the contents of the `query_many_tables.sql` file in the `/root/scripts` directory.
6. In `t1`, restart the MySQL server to reset many of the server status variables.

7. In t2, query the Performance Schema's `global_variables` table to display the current values of the `table_definition_cache`, `table_open_cache`, and `table_open_cache_instances` server variables.
8. In t2, query the Performance Schema's `global_variables` table to display the current value of the `open_files_limit` server variable.
9. In t2, query the Performance Schema's `global_status` table to display the current values of the `Open_tables` and `Opened_tables` status variables.
10. In t1, use `mysqlslap` to simulate 20 concurrent connections, each executing the queries defined in the `query_many_tables.sql` file five times against the `many_tables` database.
11. In t2, query the Performance Schema's `global_status` table to display the current values of the `Open_tables` and `Opened_tables` status variables. What do you notice about the values of these variables?
12. Calculate the table cache hit rate percentage by using the following formula:

$$\text{Hit rate} = (\text{table\_open\_cache} * 100) / \text{Opened\_tables}$$

Is the `table_open_cache` system variable set optimally for this workload?

13. Assuming that this is a typical workload for the server, calculate the optimal size of `table_open_cache` by using the following formula:

$$\text{table\_open\_cache} = \text{number of tables} * \text{number of threads}$$

**Note:** This is not a typical workload for most servers, because it is unlikely that all threads are accessing all tables simultaneously. On a production server, a good general rule would be to use 50% of the value determined by this formula.

14. Edit the `/etc/my.cnf` file to modify the `table_open_cache` value to that which you determined in the preceding step.
- Note:** The `table_open_cache` setting can be set dynamically if required, without restarting the server. You will set its value in an options file to persist it across server restarts.
15. In t1, restart the MySQL server to reset many of the server status variables.
16. In t2, query the Performance Schema's `global_variables` table to display the current values of the `table_definition_cache`, `table_open_cache`, and `table_open_cache_instances` server variables. What do you notice about the value of `table_definition_cache`?
17. In t2, query the Performance Schema's `global_variables` table to display the current value of the `open_files_limit` server variable. What do you notice about the value of the `open_files_limit` server variable?
18. In t1, reissue the `mysqlslap` test from step 10.
19. In t2, query the Performance Schema's `global_status` table to display the current values of the `Open_tables` and `Opened_tables` status variables.

20. Calculate the new table cache hit rate percentage by using the same formula as before. Is this `table_open_cache` setting optimal for this server's workload?
21. In `t1`, remove the `table_open_cache` setting from `/etc/my.cnf` and restart the server.
22. In `t2`, terminate the connection to the MySQL server and close the terminal window.
23. Leave `t1` open for the next practice.

## Solution 6-1: Sizing the Table Cache

---

### Solution Steps

**Note:** These solutions show sample outputs. Your outputs might differ from those shown.

1. Open a terminal window as the Linux `root` user. This is terminal window `t1`.
2. Create the `many_tables` database by executing the `/root/scripts/create_many_tables.sql` file in `t1`.

Enter the following commands at the `t1` Linux terminal prompt and receive the results shown:

```
# cd /root/scripts
# mysql -uroot -p < create_many_tables.sql
Enter password: oracle
#
```

**Note:** The script takes a few minutes to execute.

3. Open another Linux terminal window and log in to the `mysql` client. This is terminal window `t2`.

Enter the following command at the `t2` Linux terminal prompt and receive the results shown:

```
# mysql -uroot -p
Enter password: oracle
Welcome to the MySQL monitor. Commands end with ; or \g.
...
mysql>
```

4. In `t2`, query the `INFORMATION_SCHEMA.tables` table to display a count of the number of tables in the `many_tables` database.

Enter the following command at the `t2` Linux terminal prompt and receive the results shown:

```
mysql> SELECT COUNT(*) FROM INFORMATION_SCHEMA.TABLES
      -> WHERE TABLE_SCHEMA='many_tables';
+-----+
| COUNT(*) |
+-----+
|      500 |
+-----+
1 row in set (#.# sec)
```

5. In `t1`, use a Linux tool such as `cat` or `less`, or a text editor such as `nano`, to display the contents of the `query_many_tables.sql` file in the `/root/scripts` directory.

Enter the following command at the `t1` Linux terminal prompt and receive the results shown:

```
# cat query_many_tables.sql
...
SELECT * FROM many_tables.tbl_495;
SELECT * FROM many_tables.tbl_496;
SELECT * FROM many_tables.tbl_497;
SELECT * FROM many_tables.tbl_498;
SELECT * FROM many_tables.tbl_499;
SELECT * FROM many_tables.tbl_500;
```

- The `query_many_tables.sql` file contains one `SELECT` statement for each of the tables in the `many_tables` database.
6. In `t1`, restart the MySQL server to reset many of the server status variables.

Enter the following command at the `t1` Linux terminal prompt and receive the results shown:

```
# service mysql restart
Shutting down MySQL.... SUCCESS!
Starting MySQL. SUCCESS!
```

7. In `t2`, query the Performance Schema's `global_variables` table to display the current values of the `table_definition_cache`, `table_open_cache`, and `table_open_cache_instances` server variables.

Enter the following statement at the `mysql` prompt in `t2` and receive the results shown:

```
mysql> SELECT * FROM performance_schema.global_variables
      -> WHERE variable_name LIKE '%table%cache%';
+-----+-----+
| VARIABLE_NAME          | VARIABLE_VALUE |
+-----+-----+
| table_definition_cache | 1400           |
| table_open_cache        | 2000           |
| table_open_cache_instances | 16            |
+-----+-----+
3 rows in set (#.## sec)
```

8. In `t2`, query the Performance Schema's `global_variables` table to display the current value of the `open_files_limit` server variable.

Enter the following statements at the `mysql` prompt in `t2` and receive the results shown:

```
mysql> SELECT * FROM performance_schema.global_variables
      -> WHERE variable_name LIKE 'open_files_limit';
+-----+-----+
| VARIABLE_NAME          | VARIABLE_VALUE |
+-----+-----+
| open_files_limit       | 5000           |
+-----+-----+
```

```
1 row in set (#.## sec)
```

9. In t<sub>2</sub>, query the Performance Schema's global\_status table to display the current values of the Open\_tables and Opened\_tables status variables.

Enter the following statement at the mysql prompt in t<sub>2</sub> and receive the results shown:

```
mysql> SELECT * FROM performance_schema.global_status
      -> WHERE variable_name LIKE 'Open%table%';
+-----+-----+
| VARIABLE_NAME          | VARIABLE_VALUE |
+-----+-----+
| Open_table_definitions | 757           |
| Open_tables           | 185          |
| Opened_table_definitions | 757           |
| Opened_tables          | 192          |
+-----+-----+
4 rows in set (#.## sec)
```

10. In t<sub>1</sub>, use mysqlslap to simulate 20 concurrent connections, each executing the queries defined in the query\_many\_tables.sql file five times against the many\_tables database.

Enter the following command at the t<sub>1</sub> Linux terminal prompt and receive the results shown:

```
# mysqlslap -uroot -p \
> --create-schema=many_tables \
> --query=query_many_tables.sql \
> --concurrency=20 \
> --iterations=5
Enter password: oracle
Benchmark
      Average number of seconds to run all queries: 0.149 seconds
      Minimum number of seconds to run all queries: 0.141 seconds
      Maximum number of seconds to run all queries: 0.177 seconds
      Number of clients running queries: 20
      Average number of queries per client: 500
```

11. In t<sub>2</sub>, query the Performance Schema's global\_status table to display the current values of the Open\_tables and Opened\_tables status variables. What do you notice about the values of these variables?

Enter the following statement at the mysql prompt in t<sub>2</sub> and receive the results shown:

```
mysql> SELECT * FROM performance_schema.global_status
      -> WHERE variable_name LIKE 'Open%table%';
+-----+-----+
| VARIABLE_NAME          | VARIABLE_VALUE |
+-----+-----+
```

```

| Open_table_definitions | 757           |
| Open_tables          | 2000          |
| Opened_table_definitions | 757           |
| Opened_tables         | 44374        |
+-----+-----+
4 rows in set (#.## sec)

```

- What do you notice about the values of these variables? **Answer:** The `Open_tables` status variable shows that the table cache served 2,000 requests for a table, as per the default `table_open_cache` setting that you displayed in step 7. The value of `Opened_tables` has increased dramatically, demonstrating that most requests required a thread to open a table. This is inefficient and impacts performance, because of the file I/O involved in opening tables.

12. Calculate the table cache hit rate percentage by using the following formula:

$$\text{Hit rate} = (\text{table\_open\_cache} * 100) / \text{Opened\_tables}$$

Is the `table_open_cache` system variable set optimally for this workload?

**Answer:** The value of `Opened_tables` in the sample output shown in step 11 results in the following table cache hit rate percentage:

$$\begin{aligned}\text{Table cache hit rate} &= (\text{table\_open\_cache} * 100) / \text{Opened\_tables} \\ &= 2000 * 100 / 44374 \\ &= 4.5\%.\end{aligned}$$

This is not optimal, because the hit rate should normally be above 50%.

13. Assuming that this is a typical workload for the server, calculate the optimal size of `table_open_cache` by using the following formula:

$$\text{table\_open\_cache} = \text{number of tables} * \text{number of threads}$$

**Note:** This is not a typical workload for most servers, because it is unlikely that all threads are accessing all tables simultaneously. On a production server, a good general rule would be to use 50% of the value determined by this formula.

- **Answer:** The optimal size of `table_open_cache` is  $500 * 20 = 10,000$ .

14. Edit the `/etc/my.cnf` file to modify the `table_open_cache` value to that which you determined in the preceding step.

**Note:** The `table_open_cache` setting can be set dynamically if required, without restarting the server. You will set its value in an options file to persist it across server restarts.

Using a text editor such as `nano`, modify the `/etc/my.cnf` file to read as follows:

```
[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
user=mysql
# Disabling symbolic-links is recommended to prevent assorted
security risks
```

```

symbolic-links=0
max_connections=225


```

15. In t1, restart the MySQL server to reset many of the server status variables.

Enter the following command at the t1 Linux terminal prompt and receive the results shown:

```

# service mysql restart
Shutting down MySQL.... SUCCESS!
Starting MySQL. SUCCESS!

```

16. In t2, query the Performance Schema's `global_variables` table to display the current values of the `table_definition_cache`, `table_open_cache`, and `table_open_cache_instances` server variables. What do you notice about the value of `table_definition_cache`?

Enter the following statement at the `mysql` prompt in t2 and receive the results shown:

```

mysql> SELECT * from performance_schema.global_variables
      -> WHERE variable_name LIKE 'table%cache%';
ERROR 2006 (HY000): MySQL server has gone away
No connection. Trying to reconnect...
Connection id:      3
Current database:  performance_schema

+-----+-----+
| VARIABLE_NAME          | VARIABLE_VALUE |
+-----+-----+
| table_definition_cache | 2000           |
| table_open_cache        | 10000          |
| table_open_cache_instances | 16            |
+-----+-----+
3 rows in set (#.## sec)

```

- What do you notice about the value of `table_definition_cache`?

**Answer:** The value of the `table_definition_cache` system variable has been autosized based on the new value of `table_open_cache`. It has been autosized by using the following formula:

$$\text{table\_definition\_cache} = (400 + (\text{table\_open\_cache} / 2))$$

17. In t2, query the Performance Schema's `global_variables` table to display the current value of the `open_files_limit` server variable. What do you notice about the value of the `open_files_limit` server variable?

Enter the following statement at the `mysql` prompt in t2 and receive the results shown:

```
mysql> SELECT * FROM performance_schema.global_variables
      -> WHERE variable_name LIKE 'open_files_limit';
+-----+-----+
| VARIABLE_NAME | VARIABLE_VALUE |
+-----+-----+
| open_files_limit | 20235 |
+-----+-----+
1 row in set (0.00 sec)
```

- What do you notice about the value of the `open_files_limit` server variable?

**Answer:** The value of the `open_files_limit` server variable has also been autosized based on the new value of `table_open_cache`.

18. In t1, re-issue the `mysqlslap` test from step 10.

Enter the following command at the t1 Linux terminal prompt and receive results similar to those shown:

```
# mysqlslap -uroot -p \
> --create-schema=many_tables \
> --query=query_many_tables.sql \
> --concurrency=20 \
> --iterations=5
Enter password: oracle
Benchmark
Average number of seconds to run all queries: 0.118 seconds
Minimum number of seconds to run all queries: 0.080 seconds
Maximum number of seconds to run all queries: 0.266 seconds
Number of clients running queries: 20
Average number of queries per client: 500
```

19. In t2, query the Performance Schema's `global_status` table to display the current values of the `Open_tables` and `Opened_tables` status variables.

Enter the following statement at the `mysql` prompt in t2 and receive the results shown:

```
mysql> SELECT * FROM performance_schema.global_status
      -> WHERE variable_name LIKE 'Open%table%';
+-----+-----+
| VARIABLE_NAME | VARIABLE_VALUE |
+-----+-----+
| Open_table_definitions | 757 |
| Open_tables | 8686 |
| Opened_table_definitions | 757 |
```

```
| Opened_tables | 10953 |
+-----+-----+
4 rows in set (#.## sec)
```

20. Calculate the new table cache hit rate percentage by using the same formula as before. Is this `table_open_cache` setting optimal for this server's workload?

- **Answer:** The value of `Opened_tables` in the sample output shown in step 19 results in the following table cache hit rate percentage, which is ideal for the workload (anything over 50% is preferable).

```
Table cache hit rate = (table_open_cache*100)/Opened_tables.
                      = 10000*100/10953
                      = 92 %
```

21. In t1, remove the `table_open_cache` setting from `/etc/my.cnf` and restart the server.

In t1, use a text editor such as `nano` to modify the contents of `/etc/my.cnf` so that it appears as follows:

```
[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
user=mysql
# Disabling symbolic-links is recommended to prevent assorted
# security risks
symbolic-links=0
max_connections=225

[mysqld_safe]
log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid

[client]
socket=/var/lib/mysql/mysql.sock
```

Enter the following command at the t1 Linux terminal prompt and receive the results shown:

```
# service mysql restart
Shutting down MySQL.... SUCCESS!
Starting MySQL. SUCCESS!
```

22. In t2, terminate the connection to the MySQL server and close the terminal window.

Enter the following statement at the `mysql` prompt in t2 and receive the results shown:

```
mysql> EXIT
Bye
# exit
```

23. Leave t1 open for the next practice.

## Practice 6-2: Tuning the Open Files Limit

### Overview

In this practice, you set the MySQL `open_files_limit` system variable based on the OS limit for open file descriptors. To accomplish this objective, you perform the following:

- Review the available memory on your system.
- Determine the number of concurrent open file descriptors that the OS allows.
- Update the MySQL `open_files_limit` system variable accordingly.

### Assumptions

- The MySQL server is installed and running.
- You are logged in to a terminal window as the Linux `root` user.

### Duration

This practice should take approximately 10 minutes to complete.

### Tasks

1. Enter the following command as the Linux `root` user to display the maximum number of concurrent open file descriptors that the OS allows:

```
# sysctl fs.file-max
```

**Note:** This is the “hard” limit. The maximum number of files that the MySQL server can open concurrently never exceeds this amount.

Enter the value of `fs.file-max` into the “Hard Limit” row in the “6-2” worksheet of the `/root/labs/Lesson06Practice.ods` spreadsheet.

2. Shut down the MySQL server.
3. Edit the `/etc/my.cnf` file to set the `open_files_limit` system variable to 4096.  
**Note:** The `open_files_limit` variable is not a dynamic variable and therefore, requires a server restart to take effect.  
Save the `/etc/my.cnf` file and exit the text editor.
4. Start the MySQL server.
5. Determine the effective `open_files_limit` that the MySQL server is using (the “soft” limit) by considering the following variable settings:

- `max_connections`
- `table_open_cache`
- The `open_files_limit` specified at startup

Enter these values into appropriate locations in the “6-2” worksheet of the `/root/labs/Lesson06Practice.ods` spreadsheet. The spreadsheet calculates the effective `open_files_limit` based on these values. What do you notice about the

effective `open_files_limit` that is calculated by the spreadsheet, when compared to the value that you provided in step 3?

6. Inspect the value of the MySQL global `open_files_limit` system variable. Verify that its value is the same as the effective value calculated by the spreadsheet in step 5.
7. Edit the `/etc/my.cnf` file to set the `open_files_limit` system variable to 1000000. Save the `/etc/my.cnf` file and exit the text editor.
8. Restart the MySQL server and inspect the value of the MySQL global `open_files_limit` system variable.
9. Find the process ID for the `mysqld` process and use it to inspect the operating system limits for the MySQL server by examining the contents of the `/proc/[mysqld pid]/limits` file.
10. Edit the `/etc/my.cnf` file to remove the entry for `open_files_limit`. Save the `/etc/my.cnf` file and exit the text editor.
11. Restart the MySQL server
12. Inspect the value of the `open_files_limit` system variable.
13. Leave the Linux terminal window open and remain logged in as the `root` user. This is terminal `t1` for the next practice.

## Solution 6-2: Tuning the Open Files Limit

### Solution Steps

1. Enter the following command as the Linux `root` user to display the maximum number of concurrent open file descriptors that the OS allows:

```
# sysctl fs.file-max
```

**Note:** This is the “hard” limit. The maximum number of files that the MySQL server can open concurrently never exceeds this amount.

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# sysctl fs.file-max  
fs.file-max = 1617472
```

Enter the value of `fs.file-max` into the “Hard Limit” row in the “6-2” worksheet of the `/root/labs/Lesson06Practice.ods` spreadsheet.

- 1,617,472 files in the example output shown

2. Shut down the MySQL server.

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# service mysql stop  
Shutting down MySQL...SUCCESS!
```

3. Edit the `/etc/my.cnf` file to set the `open_files_limit` system variable to 4096.

Use a text editor such as `nano` to make the following changes to the `/etc/my.cnf` file:

```
[mysqld]  
datadir=/var/lib/mysql  
socket=/var/lib/mysql/mysql.sock  
user=mysql  
# Disabling symbolic-links is recommended to prevent assorted  
# security risks  
symbolic-links=0  
open_files_limit=4096  
  
[mysqld_safe]  
log-error=/var/log/mysqld.log  
pid-file=/var/run/mysqld/mysqld.pid  
  
[client]  
socket=/var/lib/mysql/mysql.sock
```

**Note:** The `open_files_limit` variable is not a dynamic variable and therefore, requires a server restart to take effect.

Save the `/etc/my.cnf` file and exit the text editor.

4. Start the MySQL server.

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# service mysql start
Starting MySQL...SUCCESS!
```

5. Determine the effective `open_files_limit` that the MySQL server is using (the “soft” limit) by considering the following variable settings:

- `max_connections`
- `table_open_cache`
- The `open_files_limit` specified at startup

Enter the following commands at the Linux terminal prompt and receive the results shown:

```
# mysqladmin -uroot -p variables | grep max_connections
Enter password: oracle
| max_connections | 151 |
```

```
# mysqladmin -uroot -p variables | grep table_open_cache
Enter password: oracle
| table_open_cache | 2000 |
| table_open_cache_instances | 16 |
```

Enter these values into appropriate locations in the “6-2” worksheet of the `/root/labs/Lesson06Practice.ods` spreadsheet. The spreadsheet calculates the effective `open_files_limit` based on these values.

- What do you notice about the effective `open_files_limit` calculated by the spreadsheet, when compared to the value that you provided in step 3? **Answer:** The effective `open_files_limit` calculated by the spreadsheet is 4161. This is different from the value that you provided in step 3. MySQL calculates the effective `open_files_limit` by taking into account the values of the `max_connections` and `table_open_cache` system variables. It applies the largest result from three different calculations:
  - $(10 + \text{max\_connections} + (\text{table\_open\_cache} * 2))$
  - `max_connections * 5`
  - The value of `open_files_limit` at startup (or 5,000 if no value is supplied)

6. Inspect the value of the MySQL global `open_files_limit` system variable. Verify that its value is the same as the effective value calculated by the spreadsheet in step 5.

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# mysqladmin -uroot -p variables | grep open_files_limit
Enter password: oracle
| open_files_limit | 4161 |
```

- This confirms that the actual `open_files_limit` in this instance is derived from  $(10 + \text{max\_connections} + (\text{table\_open\_cache} * 2))$ .

7. Edit the `/etc/my.cnf` file to set the `open_files_limit` system variable to 1000000. Use a text editor such as nano to make the following changes to the `/etc/my.cnf` file:

```
[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
user=mysql
# Disabling symbolic-links is recommended to prevent assorted
security risks
symbolic-links=0
open_files_limit=1000000

[mysqld_safe]
log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid

[client]
socket=/var/lib/mysql/mysql.sock
```

Save the `/etc/my.cnf` file and exit the text editor.

8. Restart the MySQL server and inspect the value of the MySQL global `open_files_limit` system variable.
- Restart the MySQL server. Enter the following command at the Linux terminal prompt and receive the results shown:

```
# service mysql restart
Shutting down MySQL.. SUCCESS!
Starting MySQL. SUCCESS!
```

- Inspect the value of the `open_files_limit` system variable. Enter the following command at the Linux terminal prompt and receive the results shown:

```
# mysqladmin -uroot -p variables | grep open_files_limit
Enter password: oracle
| open_files_limit | 1000000 |
```

9. Find the process ID for the `mysqld` process and use it to inspect the operating system limits for the MySQL server by examining the contents of the `/proc/[mysqld pid]/limits` file.

Enter the following command at the Linux terminal prompt and receive the results shown:

# cat /proc/`pidof mysqld`/limits			
Limit	Soft Limit	Hard Limit	Units
Max cpu time	unlimited	unlimited	seconds
Max file size	unlimited	unlimited	bytes
Max data size	unlimited	unlimited	bytes
Max stack size	8388608	unlimited	bytes
Max core file size	unlimited	unlimited	bytes
Max resident set	unlimited	unlimited	bytes

Max processes	1024	62549	processes
<b>Max open files</b>	<b>1000000</b>	<b>1000000</b>	files
Max locked memory	65536	65536	bytes
Max address space	unlimited	unlimited	bytes
Max file locks	unlimited	unlimited	locks
Max pending signals	62549	62549	signals
Max msgqueue size	819200	819200	bytes
Max nice priority	0	0	
Max realtime priority	0	0	
Max realtime timeout	unlimited	unlimited	us

- The pidof command injects the current pid of the mysqld process into the file path before executing cat.
- The mysqld server process now uses the new maximum that you set, which must never be greater than fs.file-max.

#### 10. Edit the /etc/my.cnf file to remove the entry for open\_files\_limit.

Use a text editor such as nano to modify the /etc/my.cnf file so that it has the following contents:

```
[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
user=mysql
# Disabling symbolic-links is recommended to prevent assorted
# security risks
symbolic-links=0

[mysqld_safe]
log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid

[client]
socket=/var/lib/mysql/mysql.sock
```

Save the /etc/my.cnf file and exit the text editor.

#### 11. Restart the MySQL server

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# service mysql restart
Shutting down MySQL.. SUCCESS!
Starting MySQL. SUCCESS!
```

#### 12. Inspect the value of the open\_files\_limit system variable.

Enter the following statement at the mysql prompt and receive the results shown:

```
# mysqladmin -uroot -p variables | grep open_files_limit
Enter password: oracle
| open_files_limit | 5000 |
```

- MySQL has applied the default of 5,000, which is the largest value calculated by the three formulae listed in the solution for step 5.
13. Leave the Linux terminal window open and remain logged in as the `root` user. This is terminal `t1` for the next practice.

## Practice 6-3: Sizing the Binary Log Cache

### Overview

In this practice, you test the binary log cache size by executing a large number of `INSERTS` within a single transaction and determining whether the cache is large enough to capture the binary log entries in memory.

**Note:** This practice uses two Linux terminal windows. The instructions refer to the terminal windows as `t1` and `t2`. To make it easier to identify each terminal, change its title by using the Terminal > Set Title menu option in each terminal window.

### Assumptions

- The `binlog-cache-test-create.sql` and `binlog-cache-test-run.sql` scripts are in the `/root/scripts` directory.
- The `employees` database is installed.
- You are logged in to a terminal window as the Linux `root` user from the previous practice. This is terminal window `t1`.

### Duration

This practice should take approximately 30 minutes to complete.

### Tasks

1. Open a new Linux terminal window as the Linux `root` user. This is `t2`. In `t2`, execute the `/root/scripts/binlog-cache-test-create.sql` script.
2. In `t2`, verify that the `binlog-cache-test-create.sql` script created the `binlogcachetest_table` table in the `perft` database and that the table contains approximately 850,000 rows.
3. In `t1`, logged in as the Linux `root` user, stop the MySQL server.
4. In `t1`, edit the `/etc/my.cnf` file to perform the following operations:
  - Specify a server ID of 999 (`server_id`, required for binary logging).
  - Enable binary logging (`log_bin`).
  - Set the binary log format to `ROW` (`binlog_format=ROW`).Save the `/etc/my.cnf` file and exit the text editor.
5. In `t1`, start the MySQL server.
6. In `t1`, inspect the size of the binary log cache by displaying the value of the `binlog_cache` system variable. How large is the per-thread binary log cache?
7. In `t2`, execute a `mysqlslap` test with the following parameters:
  - The number of concurrent connections (`--concurrency`): 5
  - The number of iterations (`--iterations`): 40
  - The schema to query against (`--create-schema`): `perft`

- The SQL file containing the queries to execute (--query):  
binlog-cache-test-run.sql

**Note:** Do not wait for the script to complete; instead, proceed immediately to the next step.

- While the mysqlslap test executes in t2, switch to the root terminal window in t1 and issue the following command:

```
# lsof +r 1 -p `pidof mysqld` \
> | grep tmp > /root/Desktop/lsof.txt
...
```

**Note:** Ignore any warnings generated by the command.

- The pidof command retrieves the process ID for the mysqld process.
  - The lsof command lists all the files opened by the mysqld process in the /tmp directory, repeating every second. It logs the output to a file called lsof.txt on the Linux root user's desktop.
- When the mysqlslap test has finished executing in t2, press CTRL + C to terminate the lsof command in t1.
  - Examine the values of the Binlog\_cache\_use and Binlog\_cache\_disk\_use status variables by opening a mysql session in t2 and executing a query against the Performance Schema's global\_status table. Based on the values of these status variables, what can you say about the effectiveness of the binary log cache?
  - Exit the mysql session in t2.
  - In t1, examine the lsof.txt file on the Linux root user's desktop. What is the largest temporary file that the MySQL server has created (seventh column)? What does this tell you about the ideal size for the binary log cache?
  - In the t1 Linux terminal window, stop the MySQL server.
  - In t1, edit the contents of the /etc/my.cnf file to specify a new value for the binlog\_cache\_size system variable based on your findings in step 11.
- Note:** binlog\_cache\_size sets the size only for the transaction cache; the size of the statement cache is determined by the binlog\_stmt\_cache\_size system variable.
- Start the MySQL server and verify the changes to the binlog\_cache\_size system variable in t1. What do you notice about the value of the binlog\_cache\_size variable?
  - Test the new binary log cache settings by executing the mysqlslap test in t2, by using the same parameters that you used in step 7. Wait for the test to complete before proceeding with the next step.
  - Examine the values of the Binlog\_cache\_disk\_use and Binlog\_cache\_use status variables by opening a mysql session in t2 and executing a query against the Performance Schema's global\_status table. Is the binary log cache effective with the new settings?
  - In t1, edit the /etc/my.cnf file to remove the custom server variable settings that you used for this practice.

Save the /etc/my.cnf file and exit the text editor.

19. Restart the MySQL server.
20. Close all running mysql sessions and terminal windows.

## Solution 6-3: Sizing the Binary Log Cache

### Solution Steps

1. Open a new Linux terminal window as the Linux `root` user. This is `t2`. In `t2`, execute the `/root/scripts/binlog-cache-test-create.sql` script.

Enter the following commands at the `t2` Linux terminal prompt and receive the results shown:

```
# cd /root/scripts
# mysql -uroot -p < binlog-cache-test-create.sql
Enter password: oracle
#
```

2. In `t2`, verify that the `binlog-cache-test-create.sql` script created the `binlogcachetest_table` table in the `perft` database, and that the table contains approximately 850,000 rows.

Enter the following command at the `t2` Linux terminal prompt and receive the results shown:

```
# mysql -uroot -p \
> -e"SELECT COUNT(*) FROM perft.binlogcachetest_table;"
Enter password: oracle
+-----+
| COUNT(*) |
+-----+
|     855173 |
+-----+
```

3. In `t1`, logged in as the Linux `root` user, stop the MySQL server.

Enter the following command at the `t1` Linux terminal prompt and receive the results shown:

```
# service mysql stop
Shutting down MySQL.. SUCCESS!
```

4. In `t1`, edit the `/etc/my.cnf` file to perform the following operations:

- Specify a server ID of 999 (`server_id`, required for binary logging).
- Enable binary logging (`log_bin`).
- Set the binary log format to `ROW` (`binlog_format=ROW`).

Use a text editor, such as `nano`, to change the contents of the `/etc/my.cnf` file as follows:

```
[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
user=mysql
```

```

# Disabling symbolic-links is recommended to prevent assorted
security risks
symbolic-links=0
server_id=999
log_bin
binlog_format=ROW

[mysqld_safe]
log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid

[client]
socket=/var/lib/mysql/mysql.sock

```

Save the /etc/my.cnf file and exit the text editor.

5. In t1, start the MySQL server.

Enter the following command at the t1 Linux terminal prompt and receive the results shown:

```

# service mysql start
Starting MySQL... SUCCESS!

```

6. In t1, inspect the size of the binary log cache by displaying the value of the binlog\_cache system variable. How large is the per-thread binary log cache?

Enter the following command at the t1 Linux terminal prompt and receive the results shown:

```

# mysqladmin -uroot -p variables | grep binlog_cache
Enter password: oracle
+-----+-----+
| binlog_cache_size | 32768 |
| max_binlog_cache_size | 18446744073709547520 |

```

- How large is the per-thread binary log cache? **Answer:** The per-thread binary log cache is approximately 32K.

7. In t2, execute a mysqlslap test with the following parameters:

- The number of concurrent connections (--concurrency): 5
- The number of iterations (--iterations): 40
- The schema to query against (--create-schema): perf
- The SQL file containing the queries to execute (--query): binlog-cache-test-run.sql

**Note:** Do not wait for the script to complete; instead, proceed immediately to the next step.

Enter the following command at the t2 Linux terminal prompt and receive the results shown:

```

# mysqlslap -uroot -p \
> --concurrency=5 \

```

```
> --iterations=10 \
> --create-schema=perf \
> --query=binlog-cache-test-run.sql
Enter password: oracle
```

- This script causes repeated `UPDATES` of a column in every record of `binlogcachetest_table`, and takes several minutes to complete. Each `UPDATE` occurs within a transaction, so the MySQL server attempts to write all the changes to the binary log cache before flushing them to the binary log files.
8. While the `mysqlslap` test executes in `t2`, switch to the `root` terminal window in `t1` and issue the following command:

```
# lsof +r 1 -p `pidof mysqld` \
> | grep tmp > /root/Desktop/lsof.txt
...
```

**Note:** Ignore any warnings generated by the command.

- The `pidof` command retrieves the process ID for the `mysqld` process.
  - The `lsof` command lists all the files opened by the `mysqld` process in the `/tmp` directory, repeating every second. It logs the output to a file called `lsof.txt` on the Linux `root` user's desktop.
9. When the `mysqlslap` test has finished executing in `t2`, press `CTRL + C` to terminate the `lsof` command in `t1`.

When the `mysqlslap` test completes, it displays benchmark results similar to the following:

```
Benchmark
Average number of seconds to run all queries: 25.139 seconds
Minimum number of seconds to run all queries: 23.890 seconds
Maximum number of seconds to run all queries: 25.817 seconds
Number of clients running queries: 5
Average number of queries per client: 2
```

10. Examine the values of the `Binlog_cache_use` and `Binlog_cache_disk_use` status variables by opening a `mysql` session in `t2` and executing a query against the Performance Schema's `global_status` table. Based on the values of these status variables, what can you say about the effectiveness of the binary log cache?

Enter the following command at the `t2` Linux terminal prompt and receive the results shown:

```
# mysql -uroot -p
Enter password: oracle
Welcome to the MySQL monitor.  Commands end with ; or \g.
...
mysql>
```

Enter the following statement at the `mysql` prompt in `t2` and receive the results shown:

```
mysql> SELECT * FROM performance_schema.global_status
      -> WHERE variable_name
      -> IN('Binlog_cache_disk_use', 'Binlog_cache_use');
+-----+-----+
| VARIABLE_NAME          | VARIABLE_VALUE |
+-----+-----+
| Binlog_cache_disk_use | 50                |
| Binlog_cache_use      | 50                |
+-----+-----+
2 rows in set (#.## sec)
```

- What can you say about the effectiveness of the binary log cache? **Answer:** The binary log cache is ineffective, because it is not large enough to contain the log entries. The MySQL server creates temporary files on disk to store the log entries, which negatively affects performance.

11. Exit the `mysql` session in `t2`.

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> EXIT
Bye
#
```

12. In `t1`, examine the `lsof.txt` file on the Linux root user's desktop. What is the largest temporary file that the MySQL server has created (seventh column)? What does this tell you about the ideal size for the binary log cache?

Enter the following command at the `t1` Linux terminal prompt and receive the results shown:

```
# cat /root/Desktop/lsof.txt
...
/tmp/ibscR794 (deleted)
mysqld 15244 mysql 12u REG 8,5 0
3145922 /tmp/ibvygSMA (deleted)
mysqld 15244 mysql 39u REG 8,5 5986211
3145924 /tmp/MYYxkJJU (deleted)
mysqld 15244 mysql 40u REG 8,5 25755990
3145930 /tmp/MLsqzu8r (deleted)
mysqld 15244 mysql 41u REG 8,5 25755990
3145931 /tmp/MLdUD
...
```

- What does this tell you about the ideal size for the binary log cache? **Answer:** In this example, the largest temporary file that the `mysqld` process has created is 25755990 bytes (approximately 24.5 MB.) This data did not fit in the in-memory cache defined by the `binlog_cache_size` variable. The ideal cache size would be big enough to store this data in memory, with extra space available if required.

13. In the t1 Linux terminal window, stop the MySQL server.

Enter the following command at the t1 Linux terminal prompt and receive the results shown:

```
# service mysql stop
Shutting down MySQL.... SUCCESS!
```

14. In t1, edit the contents of the /etc/my.cnf file to specify a new value for the binlog\_cache\_size system variable based on your findings in step 11.

Use a text editor, such as nano, to change the contents of the /etc/my.cnf file as follows:

```
[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
user=mysql
# Disabling symbolic-links is recommended to prevent assorted
# security risks
symbolic-links=0
server_id=999
log_bin
binlog_format=ROW
binlog_cache_size=30000000

[mysqld_safe]
log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid

[client]
socket=/var/lib/mysql/mysql.sock
```

**Note:** binlog\_cache\_size sets the size only for the transaction cache; the size of the statement cache is determined by the binlog\_stmt\_cache\_size system variable.

15. Start the MySQL server and verify the changes to the binlog\_cache\_size system variable in t1. What do you notice about the value of the binlog\_cache\_size variable?

Enter the following commands at the t1 Linux terminal prompt and receive the results shown:

```
# service mysql start
Starting MySQL... SUCCESS!
# mysqladmin -uroot -p variables | grep binlog_cache_size
Enter password: oracle
| binlog_cache_size           | 29999104           |
| max_binlog_cache_size       | 18446744073709547520 |
```

- What do you notice about the value of the `binlog_cache_size` variable?  
**Answer:** The MySQL server autosizes the `binlog_cache_size` variable based on the value that you provided.

16. Test the new binary log cache settings by executing the `mysqlslap` test in `t2`, by using the same parameters that you used in step 7. Wait for the test to complete before proceeding with the next step.

Enter the following command at the `t2` Linux terminal prompt:

```
# mysqlslap -uroot -p \
> --concurrency=5 \
> --iterations=10 \
> --create-schema=perf \
> --query=binlog-cache-test-run.sql
Enter password: oracle
```

17. Examine the value of the `Binlog_cache_disk_use` and `Binlog_cache_use` status variables by opening a `mysql` session in `t2` and executing a query against the Performance Schema's `global_status` table. Is the binary log cache effective with the new settings?

Enter the following command at the `t2` Linux terminal prompt and receive the results shown:

```
# mysql -uroot -p
Enter password: oracle
Welcome to the MySQL monitor. Commands end with ; or \g.
...
mysql>
```

Enter the following statement at the `mysql` prompt in `t2` and receive the results shown:

```
mysql> SELECT * FROM performance_schema.global_status
-> WHERE variable_name
-> IN('Binlog_cache_disk_use', 'Binlog_cache_use');
+-----+-----+
| VARIABLE_NAME | VARIABLE_VALUE |
+-----+-----+
| Binlog_cache_disk_use | 0 |
| Binlog_cache_use | 50 |
+-----+-----+
2 rows in set (0.00 sec)
```

- Is the binary log cache effective with the new settings? **Answer:** The binary log cache is effective, because it is large enough to contain all the log file entries in memory, without creating temporary files on disk.

18. In t1, edit the /etc/my.cnf file to remove the custom server variable settings that you used for this practice.

Use a text editor, such as nano, to modify the /etc/my.cnf file so that it has the following contents:

```
[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
user=mysql
# Disabling symbolic-links is recommended to prevent assorted
# security risks
symbolic-links=0

[mysqld_safe]
log-error=/var/log/mysqld.log
pid-file=/var/run/mysqld/mysqld.pid

[client]
socket=/var/lib/mysql/mysql.sock
```

Save the /etc/my.cnf file and exit the text editor.

19. Restart the MySQL server.

Enter the following command at the t1 Linux terminal prompt and receive the results shown:

```
# service mysql restart
Shutting down MySQL.. SUCCESS!
Starting MySQL. SUCCESS!
```

20. Close all running mysql sessions and terminal windows.

## Practice 6-4: Identifying I/O Hotspots with MySQL Workbench

### Overview

In this practice, you investigate file I/O hotspots by using MySQL Workbench Performance Reports.

### Assumptions

- You have completed the “Using MySQL Workbench for Performance Schema Configuration, Monitoring, and Reporting” practice in the lesson titled “Performance Schema.”
- You have completed the other lesson practices in this Activity Guide.

### Duration

This practice should take you approximately five minutes to complete.

### Tasks

1. Open MySQL Workbench from the Applications > Programming > MySQL Workbench menu option.
2. Click the “My Server Connection” link on the MySQL Workbench home page to connect to the MySQL server. If prompted for a password for the `root` user, enter `oracle`. MySQL Workbench connects to the MySQL server and displays a list of options and available schemas, along with the “Query 1” tab for editing SQL.
3. In the “Performance” section in the left-hand pane, click Performance Reports. The list of available reports appears.
4. In the list of reports, click the “Top File I/O Activity Report” under “Hot Spots for I/O.” MySQL Workbench queries the Performance Schema to create a report that lists the files that generate the most I/O events. The report lists the total file I/O in bytes, as well as separate entries for the total and average number of read and writes, together with the associated byte value. You can order the entries by clicking the column headers. Observe the entry for `binlogcachetest_table.ibd`, which resulted from the “Sizing the Binary Log File Cache” activity in this lesson.

### Top File I/O Activity Report

Show the Files doing the most IOs in bytes

File	Total IOs (#)	Read Requests (#)	Total Read IO	Avg Read IO	Write Requests (#)	Total Write I	Avg Write	Write %
/var/lib/mysql/eddr73p1-bin.000006	128851604	15730	128851604	8191	0	0	0	0.00
/var/lib/mysql/ibdata1	23117824	1278	23052288	18038	3	65536	21845	0.28
/var/lib/mysql/ibtmp1	13844480	0	0	0	89	13844480	155556	100.00
/var/lib/mysql/perft/binlogcachetest_table.ibd	13631488	832	13631488	16384	0	0	0	0.00
/var/lib/mysql/mysql/proc.MYD	1251204	472	1251204	2651	0	0	0	0.00
/var/lib/mysql/mysql/innodb_index_stats.ibd	98304	6	98304	16384	0	0	0	0.00
/var/lib/mysql/mysql/innodb_table_stats.ibd	81920	5	81920	16384	0	0	0	0.00
/opt/mysql-advanced-5.7.17-linux-glibc2.5-x86..	75895	3	75895	25298	0	0	0	0.00
/var/lib/mysql/ib_logfile1	67072	2	66048	33024	2	1024	512	1.53
/var/lib/mysql/mysql/engine_cost.ibd	65536	4	65536	16384	0	0	0	0.00
/var/lib/mysql/mysql/plugin.ibd	65536	4	65536	16384	0	0	0	0.00
/var/lib/mysql/mysql/server_cost.ibd	65536	4	65536	16384	0	0	0	0.00
/var/lib/mysql/mysql/servers.ibd	65536	4	65536	16384	0	0	0	0.00
/var/lib/mysql/mysql/slave_master_info.ibd	65536	4	65536	16384	0	0	0	0.00
/var/lib/mysql/mysql/slave_relay_log_info.ibd	65536	4	65536	16384	0	0	0	0.00
/var/lib/mysql/mysql/slave_worker_info.ibd	65536	4	65536	16384	0	0	0	0.00
/var/lib/mysql/mysql/time_zone.ibd	65536	4	65536	16384	0	0	0	0.00
/var/lib/mysql/mysql/time_zone_leap_second.ibd	65536	4	65536	16384	0	0	0	0.00

Export...

Copy Selected

Copy Query

Refresh

- Click the “Top File I/O by Time” report. This report lists the highest file I/O by time and latency. Locate the entry for `binlogcachetest_table.ibd` in the report.

View the remaining reports in the “Hot Spots for I/O” section:

- Top I/O by Event Category
- Top I/O in Time by Event Categories
- Top I/O Time by User/Thread

- When you have finished reviewing the reports, close MySQL Workbench.

## Solution 6-4: Identifying I/O Hotspots with MySQL Workbench

---

There are no solution steps for this practice.

## **Practices for Lesson 7: Tuning InnoDB**

## Practices for Lesson 7

---

### Overview

These practices test your knowledge of tuning the InnoDB storage engine for best performance.

**Note:** The output of the various Linux commands and SQL statements shown in the solution steps for these practices might be different on your system.

## Practice 7-1: Investigating the Effects of Log Files on Transactions

---

### Overview

In this practice, you evaluate the effect of the frequency of log file writes on performance. To accomplish this objective, you perform the following:

- Evaluate a MySQL stored procedure called `innodb_inserts`.
- Use the `perf1` database to execute multiple instances of the `innodb_inserts` MySQL stored procedure.
- Change the setting of the `innodb_flush_log_at_trx_commit` system variable and repeat the test.

### Assumptions

- The MySQL server is installed and running.
- The `perf1` database is installed and populated.

### Duration

This practice should take you approximately 20 minutes to complete.

### Tasks

1. Open a Linux terminal window as the `root` user.
2. Start a `mysql` session and review the SQL code for the `innodb_inserts` MySQL stored procedure in the `perf1` database. What does this stored procedure do?
3. What is the current value of the `innodb_flush_log_at_trx_commit` system variable?
4. Execute the `innodb_inserts` stored procedure with the following parameters:
  - Number of records to insert: 1000
  - Number of `INSERTS` before `COMMIT`: 1

This execution of the `innodb_inserts` stored procedure creates a table and inserts 500 records into it, committing after every insert. Note the time taken to execute the stored procedure.

5. Repeat step 4 by using the following parameters for the `innodb_inserts` stored procedure. Enter the results of each execution in the spreadsheet.

Number of <code>INSERT</code> statements	Commit interval (number of transactions)
1000	5
1000	250
1000	500
1000	1000

6. Answer the following questions:
  - How does the execution time vary for different frequencies of `COMMIT`?

- Why?
7. Set the value of the `innodb_flush_log_at_trx_commit` system variable to 0.
  8. Execute the `innodb_inserts` stored procedure with the following parameters:
    - Number of records to insert: 1000
    - Number of INSERTS before COMMIT: 1Record your results in the spreadsheet.

9. Repeat step 8 by using the following parameters for the `innodb_inserts` stored procedure. Enter the results of each execution in the spreadsheet.

<b>Number of INSERT statements</b>	<b>Commit interval (number of transactions)</b>
1000	5
1000	250
1000	500
1000	1000

10. Answer the following questions:
  - How does setting the `innodb_flush_log_at_trx_commit` system variable to 0 affect the performance of the stored procedure?
  - Why?
  - What risk is associated with setting `innodb_flush_log_at_trx_commit` to 0?
11. Reset the global system variable `innodb_flush_log_at_trx_commit` to the default value of 1.
12. Keep the terminal window open and remain logged in to the `mysql` command-line client for the next practice.

## Solution 7-1: Investigating the Effects of Log Files on Transactions

### Solution Steps

**Note:** Enter the results of the following tasks into the “8-1” worksheet of the ~/labs/Lesson07Practice.ods spreadsheet.

1. Open a Linux terminal window as the `root` user.
2. Start a `mysql` session and review the SQL code for the `innodb_inserts` MySQL stored procedure in the `perf` database.

Enter the following command at the Linux terminal prompt:

```
# mysql -uroot -p
Enter password: oracle
Welcome to the MySQL monitor. Commands end with ; or \g.
mysql>
```

Enter the following statements at the `mysql` prompt and receive the results shown:

```
mysql> USE perf
...
Database changed
mysql> SHOW CREATE PROCEDURE innodb_inserts\G
***** 1. row *****
Procedure: innodb_inserts
sql_mode:
ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION
Create Procedure: CREATE DEFINER=`root`@`localhost`
PROCEDURE `innodb_inserts` (IN rows INT, IN rows_per_commit INT)
BEGIN
    DECLARE counter INT default 0;
    DROP TABLE IF EXISTS innodb_test;
    CREATE TABLE innodb_test (id INT AUTO_INCREMENT PRIMARY KEY,
val INT) ENGINE=innodb;
    SET autocommit = 0;
    WHILE (counter < rows) DO
        INSERT INTO innodb_test (val) values (floor(rand() * 1000000));
        IF (counter % rows_per_commit = 0) THEN
            COMMIT;
        END IF;
        SET counter = counter + 1;
    END WHILE;
END
character_set_client: utf8
```

```
collation_connection: utf8_general_ci
Database Collation: latin1_swedish_ci
1 row in set (#.## sec)
```

- What does this stored procedure do? **Answer:** The innodb\_inserts stored procedure creates the innodb\_test table and inserts a specified number of records into it, committing after a specified number of INSERTS.
3. What is the current value of the innodb\_flush\_log\_at\_trx\_commit system variable?

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> SHOW VARIABLES LIKE 'innodb_flush_log%';
+-----+-----+
| Variable_name          | Value |
+-----+-----+
| innodb_flush_log_at_timeout | 1      |
| innodb_flush_log_at trx_commit | 1      |
+-----+-----+
2 rows in set (#.## sec)
```

- **Answer:** The value of the innodb\_flush\_log\_at\_trx\_commit system variable is 1.
4. Execute the innodb\_inserts stored procedure with the following parameters:

- Number of records to insert: 1000
- Number of INSERTS before COMMIT: 1

This execution of the innodb\_inserts stored procedure creates a table and inserts 500 records into it, committing after every insert. Note the time taken to execute the stored procedure.

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> CALL innodb_inserts(1000,1);
Query OK, 0 rows affected (3.48 sec)
```

- In the preceding sample output, the stored procedure took 48.82 seconds to execute.
5. Repeat step 4 by using the following parameters for the innodb\_inserts stored procedure. Enter the results of each execution in the spreadsheet.

Number of INSERT statements	Commit interval (number of transactions)
1000	5
1000	250
1000	500
1000	1000

Enter the following statements at the mysql prompt and receive the results shown:

```
mysql> CALL innodb_inserts(1000,5);
Query OK, 1 row affected (0.81 sec)
```

```
mysql> CALL innodb_inserts(1000,250);
Query OK, 1 row affected (0.09 sec)

mysql> CALL innodb_inserts(1000,500);
Query OK, 1 row affected (0.07 sec)

mysql> CALL innodb_inserts(1000,1000);
Query OK, 1 row affected (0.06 sec)
```

6. Answer the following questions:

- How does the execution time vary for different frequencies of COMMIT? **Answer:** There is an initial reduction in execution time when INSERTS are committed more frequently. At larger intervals, there is no significant variation.
- Why? **Answer:** Although the transactions are committed at different intervals and therefore the log file flushing occurs at different times, when COMMITs are infrequent, the log files that require flushing are bigger. Therefore, there is a point at which there is no significant performance improvement to be gained by delaying the COMMIT.

7. Set the value of the `innodb_flush_log_at_trx_commit` system variable to 0.

Enter the following statements at the mysql prompt and receive the results shown:

```
mysql> SET GLOBAL innodb_flush_log_at_trx_commit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'innodb_flush_log%';
+-----+-----+
| Variable_name          | Value   |
+-----+-----+
| innodb_flush_log_at_timeout | 1       |
| innodb_flush_log_at_trx_commit | 0       |
+-----+-----+
2 rows in set (0.00 sec)
```

8. Execute the `innodb_inserts` stored procedure with the following parameters:

- Number of records to insert: 1000
- Number of INSERTS before COMMIT: 1

Record your results in the spreadsheet.

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> CALL innodb_inserts(1000,1);
Query OK, 0 rows affected (0.05 sec)
```

- In the preceding output, the stored procedure took 24.24 seconds to execute.

9. Repeat step 8 by using the following parameters for the `innodb_inserts` stored procedure. Enter the results of each execution in the spreadsheet.

Number of INSERT statements	Commit interval (number of transactions)
1000	5
1000	250
1000	500
1000	1000

Enter the following statements at the `mysql` prompt and receive the results shown:

```
mysql> CALL innodb_inserts(1000,5);
Query OK, 1 row affected (0.05 sec)

mysql> CALL innodb_inserts(1000,250);
Query OK, 1 row affected (0.05 sec)

mysql> CALL innodb_inserts(1000,500);
Query OK, 1 row affected (0.05 sec)

mysql> CALL innodb_inserts(1000,1000);
Query OK, 1 row affected (0.05 sec)
```

10. Answer the following questions:

- How does setting the `innodb_flush_log_at_trx_commit` system variable to 0 affect the performance of the stored procedure? **Answer:** Setting `innodb_flush_log_at_trx_commit` to 0 significantly improves the performance of the stored procedure.
- Why? **Answer:** When `innodb_flush_log_at_trx_commit` is set to 0, MySQL writes the log buffer to the log file and subsequently flushes the log file to disk once per second, and not when the transaction commits.
- What risk is associated with setting `innodb_flush_log_at_trx_commit` to 0? **Answer:** A mysqld process crash might erase up to one second of transactions. Setting `innodb_flush_log_at_trx_commit` to 1 is required for full ACID compliance.

11. Reset the global system variable `innodb_flush_log_at_trx_commit` to the default value of 1.

Enter the following statements at the `mysql` prompt, and receive the results shown:

```
mysql> SET GLOBAL innodb_flush_log_at_trx_commit=1;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'innodb_flush_log%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
```

```
+-----+-----+
| innodb_flush_log_at_timeout | 1      |
| innodb_flush_log_at_trx_commit | 1      |
+-----+-----+
2 rows in set (#.# sec)
```

12. Keep the terminal window open and remain logged in to the mysql command-line client for the next practice.

## Practice 7-2: Using SHOW ENGINE INNODB STATUS

### Overview

In this practice, you evaluate the output of the `SHOW ENGINE INNODB STATUS` command. To accomplish this objective, you perform the following steps:

- Use `mysqlslap` to execute multiple iterations of the `innodb_query.sql` script from multiple concurrent connections.
- Execute the `SHOW ENGINE INNODB STATUS` command and answer questions relating to its output.

**Note:** This practice uses two Linux terminal windows. The instructions refer to the terminal windows as `t1` and `t2`. To make it easier to identify each terminal, change its title by using the Terminal > Set Title menu option in each terminal window.

### Assumptions

- The `perft` database is installed and populated.
- The `innodb_query.sql` file is located in the `/root/scripts` directory.
- The terminal window containing the `mysql` session is open from the previous practice. This is referred to as terminal window `t1` in this practice.

### Duration

This practice should take you approximately 20 minutes to complete.

### Tasks

1. Open a new terminal window (`t2`) as the Linux `root` user.
2. In terminal window `t2`, restart the MySQL server.
3. In terminal window `t2`, examine the `innodb_query.sql` script in `/root/scripts`. What does the `innodb_query.sql` script do?
4. At the `mysql` session prompt from the previous practice in terminal window `t1`, execute a query that displays the number of rows in the `perft.city_huge` table.
5. Execute another query in the `mysql` session in `t1` that displays the number of rows in the `city_huge` table that would be affected by executing the query in the `innodb_query.sql` file. How many rows does the `innodb_query.sql` script update in the `city_huge` table?
6. In terminal window `t2`, execute a `mysqlslap` command with the following parameters to create a load against the MySQL server and the `city_huge` table:
  - Query to execute: `/root/scripts/innodb_query.sql`
  - Number of iterations: 10
  - Number of concurrent connections: 10
  - Verbose output

7. Wait approximately 20–30 seconds, and then enter the `SHOW ENGINE INNODB STATUS` command at the `mysql` prompt in terminal window t1.
8. Read the following descriptions of the various sections in the output of `SHOW ENGINE INNODB STATUS`, and answer each question in the “8-2” worksheet of the `~/labs/Lesson07Practice.ods` spreadsheet.

**Note:** The answers to the remaining questions depend on the values observed in the output of the `SHOW ENGINE INNODB STATUS` report, which might be significantly different on your machine.

## Semaphores

The `SEMAPHORES` section of the output helps you to determine how well MySQL handles context switching.

9. The `reservation count` and `signal count` values display how often InnoDB uses the internal sync array. The ratio between these two values represents how frequently the InnoDB storage engine uses the OS wait functions.
  - What is the `reservation count` value?
  - What is the `signal count` value?

## Transactions

The `TRANSACTIONS` section of the output provides useful information about lock contention and can help you to diagnose transaction deadlocks.

10. `Trx id counter` is a number that increments for each transaction and identifies each transaction. `Trx id counter` displays the most recent transaction ID. What is the ID of the most recent transaction?
11. The `Purge done for trx's n:0` entry is the ID of the transaction that was last purged. The `undo n:0` entry is the ID of the transaction that is currently processing. If the value is zero, InnoDB is not currently executing any transactions.
  - What is the `Purge done for trx's n:0` value?
  - What is the `undo n:0` value?
12. `History list length` is the number of unpurged transactions in the undo space. This value increases when transactions commit, and decreases when purges occur. The isolation mode that the transactions use can have a significant effect on the length of the history list. What is the `History list length` value?
13. The `LIST OF TRANSACTIONS FOR EACH SESSION` section displays all or some of the transactions, depending on how many transactions there are. This area provides detailed information about the status of the transactions.
  - What is the status of the transaction at the bottom of the list?
  - What is the `OS thread id` for this transaction?
  - Which MySQL thread is executing this transaction?
  - What is the `query id` for this transaction?
  - Which user executed this transaction?

## File I/O

The FILE I/O section of the report displays the state of the file input/output helper threads. These helper threads are responsible for insert buffer merges (insert buffer thread), asynchronous log flushes (log thread), read-ahead (read thread), and flushing of dirty buffers (write thread).

14. How many transaction log file flushes are pending [Pending flushes (fsync) log]?
15. How many buffer pool dirty page writes are pending (buffer pool)?
16. This section displays the total number of I/O operations, along with the computed averages for those operations. You can use these values to evaluate trends in system usage.
  - What is the OS file reads value?
  - What is the OS file writes value?
  - What is the OS fsyncs value?
  - What is the average number of reads per second during the reporting period?
  - What is the average number of bytes read per second during the reporting period?
  - What is the average number of writes per second during the reporting period?
  - What is the average number of fsync operations during the reporting period?

## Insert Buffer and Adaptive Hash Index

The INSERT BUFFER AND ADAPTIVE HASH INDEX section of the report displays the status of the insert buffer (Ibuf), including the segment size and free list, together with any records residing in the insert buffer. It also displays how many records were inserted into the insert buffer, how many records were merged, and how many merge operations occurred.

17. For the merged operations, how many INSERTS occurred during the reporting period?
18. For the merged operations, how many records were marked for deletion during the reporting period?
19. Calculate the insert buffer efficiency for the merged operations by determining the ratio of records marked for deletion to inserts and expressing that ratio as a percentage. What is the insert buffer efficiency for this reporting period?
20. The last section displays the hash table size, the number of used cells, and the number of buffers used by the adaptive hash index.
  - On an average, how many hash searches per second occurred during the reporting period?
  - On an average, how many non-hash searches per second occurred during the reporting period?
21. Calculate the ratio of hash index lookups to non-hash index lookups. What is ratio of hash index lookups to non-hash index lookups during the reporting period?

## Log

The LOG section of the report displays the current log sequence number, which corresponds to the number of bytes InnoDB has written to the log files during the lifetime of the tablespace. It also tells you which logs InnoDB has flushed to disk, and when the last checkpoint occurred.

22. What is the Log sequence number value?
23. What is the Log flushed up to value?
24. What is the percentage difference between these two values?
25. How many pending log flushes are there?
26. How many pending checkpoint (chkp) writes are there?
27. How many log I/O's have occurred during the reporting period?
28. What is the average number of I/O's per second?

### **Buffer Pool and Memory**

The BUFFER POOL AND MEMORY section of the report displays information that helps you to determine if the InnoDB buffer pool is sized correctly.

29. What is the total memory allocation for InnoDB?
30. What is the Buffer pool size value in MB?
31. How many free buffers are there?
32. What is the Buffer pool hit rate?

### **Row Operations**

The ROW OPERATIONS section of the report displays summary information about the number of rows that the system has read from and written to the database, along with other statistics. You can monitor these metrics to show InnoDB load over time.

33. How many rows have been inserted since system startup?
34. How many rows have been updated since system startup?
35. How many rows have been deleted since system startup?
36. How many rows have been read since system startup?

### **Completion**

37. In the t2 terminal window, terminate any running processes by pressing CTRL + C.
38. Close the t2 terminal window.
39. Keep the mysql client session in t1 open for the next practice.

## Solution 7-2: Using SHOW ENGINE INNODB STATUS

---

### Solution Steps

1. Open a new terminal window ( $t_2$ ) as the Linux `root` user.
2. In terminal window  $t_2$ , restart the MySQL server.

Enter the following command at the  $t_2$  Linux terminal prompt and receive the results shown:

```
# service mysql restart
Shutting down MySQL... SUCCESS!
Starting MySQL... SUCCESS!
```

3. In terminal window  $t_2$ , examine the `innodb_query.sql` script in `/root/scripts`.

Use a Linux command such as `cat`, or a text editor like `nano`, to examine the contents of the `innodb_query.sql` file as shown:

```
# cat /root/scripts/innodb_query.sql
START TRANSACTION;
USE perf;
UPDATE city_huge SET population = population * 1.01 WHERE name
LIKE 'Mon%';
COMMIT;
```

What does the `innodb_query.sql` script do? **Answer:** The `innodb_query.sql` script starts a transaction, increases the value in the `population` column by 1% for all city names starting with "Mon," and then commits the transaction.

4. At the `mysql` session prompt from the previous practice in terminal window  $t_1$ , execute a query that displays the number of rows in the `perf.city_huge` table.

Enter the following statements at the `mysql` prompt and receive the results shown:

```
mysql> USE perf
mysql> SELECT COUNT(*) FROM city_huge;
+-----+
| COUNT(*) |
+-----+
| 2039500 |
+-----+
1 row in set (#.### sec)
```

5. Execute another query in the `mysql` session in  $t_1$  that displays the number of rows in the `city_huge` table that would be affected by executing the query in the `innodb_query.sql` file.

```
mysql> SELECT COUNT(*) FROM city_huge WHERE name like 'Mon%';
+-----+
| COUNT(*) |
+-----+
```

```
|      7500 |
+-----+
1 row in set (#.# sec)
```

How many rows does the `innodb_query.sql` script update in the `city_huge` table?

**Answer:** 7500 rows

- In terminal window `t2`, execute a `mysqlslap` command with the following parameters to create a load against the MySQL server and the `city_huge` table:

- Query to execute: `/root/scripts/innodb_query.sql`
- Schema: `perf`
- Number of iterations: 10
- Number of concurrent connections: 10
- Verbose output

Enter the following command at the `t2` Linux terminal prompt:

```
# mysqlslap -uroot -p \
> --query="/root/scripts/innodb_query.sql" \
> --create-schema=perf \
> --iterations=10 \
> --concurrency=10 \
> -vv
Enter password: oracle
```

- Wait approximately 20–30 seconds, and then enter the `SHOW ENGINE INNODB STATUS` command at the `mysql` prompt in terminal window `t1`.

Enter the following command at the `t1` Linux terminal prompt and receive the results shown:

```
mysql> SHOW ENGINE INNODB STATUS\G
***** 1. row *****
Type: InnoDB
Name:
Status:
=====
2017-08-08 10:26:52 0x7f8c48286700 INNODB MONITOR OUTPUT
=====
Per second averages calculated from the last 59 seconds
-----
BACKGROUND THREAD
-----
srv_master_thread loops: 19 srv_active, 0 srv_shutdown, 698
srv_idle
srv_master_thread log flush and writes: 717
... 
```

8. Read the following descriptions of the various sections in the output of `SHOW ENGINE INNODB STATUS`, and answer each question in the “8-2” worksheet of the `~/labs/Lesson07Practice.ods` spreadsheet.

**Note:** The answers to the remaining questions depend on the values observed in the output of the `SHOW ENGINE INNODB STATUS` report, which might be significantly different on your machine.

### Semaphores

The `SEMAPHORES` section of the output helps you to determine how well MySQL handles context switching.

9. The reservation count and signal count values display how often InnoDB uses the internal sync array. The ratio between these two values represents how frequently the InnoDB storage engine uses the OS wait functions.

```
...
-----
SEMAPHORES
-----
OS WAIT ARRAY INFO: reservation count 13386
--Thread 140240481691392 has waited at buf0buf.cc line 3506 for
0.00 seconds the semaphore:
Mutex at 0x43618d8, Mutex BUF_POOL created buf0buf.cc:1730, lock
var 1

OS WAIT ARRAY INFO: signal count 17254
RW-shared spins 0, rounds 13813, OS waits 2240
RW-excl spins 0, rounds 741938, OS waits 9036
RW-sx spins 7451, rounds 127223, OS waits 1545
Spin rounds per wait: 13813.00 RW-shared, 741938.00 RW-excl,
17.07 RW-sx
...
```

- What is the reservation count value? **Answer:** 13386
- What is the signal count value? **Answer:** 17254

**Note:** This section can sometimes help you to identify hotspots for your workload. Even though the files listed in this section relate to the server source code, their names can provide clues as to what is going on. For example, the `buf0buf.cc` file that is shown in the output here indicates that there is some buffer pool contention.

### Transactions

The `TRANSACTIONS` section of the output provides useful information about lock contention and can help you to diagnose transaction deadlocks.

10. `Trx id counter` is a number that increments for each transaction and identifies each transaction. `Trx id counter` displays the most recent transaction ID.

```
...
-----
TRANSACTIONS
-----
```

```
Trx id counter 9783593
```

- What is the ID of the most recent transaction? **Answer:** 9783593
11. The Purge done for trx's n:o entry is the ID of the transaction that was last purged. The undo n:o entry is the ID of the transaction that is currently processing. If the value is zero, InnoDB is not currently executing any transactions.

```
...
Purge done for trx's n:o < 9783593 undo n:o < 0 state: running but idle
...
```

- What is the Purge done for trx's n:o value? **Answer:** < 9783593, in the preceding output. This means that InnoDB is up to date with purge operations. What is the undo n:o value? **Answer:** < 0: running but idle
- Note:** InnoDB purges old values of transactions only when it receives an event notification that the transactions are complete. Uncommitted transactions that are old or stale can block the purge process and use system resources. By comparing the Trx id counter and Purge done for trx's values, you can identify any uncommitted transactions that are consuming system resources.
12. History list length is the number of unpurged transactions in the undo space. This value increases when transactions commit, and decreases when purges occur. The isolation mode that the transactions use can have a significant effect on the length of the history list.

```
...
History list length 14
...
```

- What is the History list length value? **Answer:** 14
13. The LIST OF TRANSACTIONS FOR EACH SESSION section displays all or some of the transactions, depending on how many transactions there are. This area provides detailed information about the status of transactions.

```
LIST OF TRANSACTIONS FOR EACH SESSION:
```

```
...
```

```
-----
```

```
---TRANSACTION 9783584, ACTIVE 4 sec fetching rows
```

```
mysql tables in use 1, locked 1
```

```
1727 lock struct(s), heap size 188624, 4576 row lock(s), undo log entries 2280
```

```
MySQL thread id 19, OS thread handle 140240481691392, query id 77 localhost root updating
```

```

UPDATE city_huge SET population = population * 1.01 WHERE name
LIKE 'Mon%'

---TRANSACTION 9783582, ACTIVE 4 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 1136, 1 row lock(s)
MySQL thread id 18, OS thread handle 140240480892672, query id
76 localhost root updating
UPDATE city_huge SET population = population * 1.01 WHERE name
LIKE 'Mon%'

----- TRX HAS BEEN WAITING 4 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 61 page no 24940 n bits 168 index Name of
table `perf`.`city_huge` trx id 9783582 lock_mode X waiting
Record lock, heap no 2 PHYSICAL RECORD: n_fields 2; compact
format; info bits 0
 0: len 30; hex
4d6f6e61636f2d56696c6c6520202020202020202020202020202020;
asc Monaco-Ville ; (total 35 bytes);
 1: len 4; hex 80000a87; asc      ;;
-----
```

- What is the status of the transaction at the bottom of the list? **Answer:** fetching rows (your latest transaction status might be different)
  - What is the OS thread id for this transaction? **Answer:** 140240481691392
  - Which MySQL thread is executing this transaction? **Answer:** 19
  - What is the query id for this transaction? **Answer:** 77
  - Which user executed this transaction? **Answer:** root@localhost
- Note:** Use the OS thread id, MySQL thread id, and query id values to identify the source of a query for administration and troubleshooting purposes.

## File I/O

The FILE I/O section of the report displays the state of the file input/output helper threads. These helper threads are responsible for insert buffer merges (insert buffer thread), asynchronous log flushes (log thread), read-ahead (read thread), and flushing of dirty buffers (write thread).

**Note:** The answers to the questions in this section are based on the following sample output:

```

...
-----
FILE I/O
-----
I/O thread 0 state: waiting for completed aio requests (insert
buffer thread)
I/O thread 1 state: waiting for completed aio requests (log
thread)
```

```

I/O thread 2 state: waiting for completed aio requests (read
thread)
I/O thread 3 state: waiting for completed aio requests (read
thread)
I/O thread 4 state: waiting for completed aio requests (read
thread)
I/O thread 5 state: waiting for completed aio requests (read
thread)
I/O thread 6 state: waiting for completed aio requests (write
thread)
I/O thread 7 state: waiting for completed aio requests (write
thread)
I/O thread 8 state: waiting for completed aio requests (write
thread)
I/O thread 9 state: complete io for buf page (write thread)
Pending normal aio reads: [0, 0, 0, 0] , aio writes: [0, 0, 0,
0] ,
ibuf aio reads:, log i/o's:, sync i/o's:
Pending flushes (fsync) log: 0; buffer pool: 1
16400 OS file reads, 3381 OS file writes, 152 OS fsyncs
76.64 reads/s, 16384 avg bytes/read, 56.39 writes/s, 2.46
fsyncs/s
...

```

14. How many transaction log file flushes are pending [Pending flushes (fsync) log]?  
**Answer:** 0
15. How many buffer pool dirty page writes are pending (buffer pool)? **Answer:** 1
16. This section displays the total number of I/O operations, along with the computed averages for those operations. You can use these values to evaluate trends in system usage.
  - What is the OS file reads value? **Answer:** 16400
  - What is the OS file writes value? **Answer:** 3381
  - What is the OS fsyncs value? **Answer:** 152
  - What is the average number of reads per second during the reporting period?  
**Answer:** 76.64
  - What is the average number of bytes read per second during the reporting period?  
**Answer:** 16384
  - What is the average number of writes per second during the reporting period?  
**Answer:** 56.39
  - What is the average number of fsync operations during the reporting period?  
**Answer:** 2.46

### **Insert Buffer and Adaptive Hash Index**

The INSERT BUFFER AND ADAPTIVE HASH INDEX section of the report displays the status of the insert buffer (Ibuf), including the segment size and free list, together with any records

that reside in the insert buffer. It also displays how many records were inserted into the insert buffer, how many records were merged, and how many merge operations occurred.

**Note:** The answers to the questions in this section are based on the following sample output:

```

...
-----
INSERT BUFFER AND ADAPTIVE HASH INDEX
-----
Ibuf: size 1, free list len 56, seg size 58, 2782 merges
merged operations:
  insert 147519, delete mark 314199, delete 1659
discarded operations:
  insert 0, delete mark 0, delete 0
Hash table size 276707, node heap has 1399 buffer(s)
13418.58 hash searches/s, 18601.04 non-hash searches/s
...

```

- For the merged operations, how many **INSERTs** occurred during the reporting period?

**Answer:** 147519

- For the merged operations, how many records were marked for deletion during the reporting period? **Answer:** 314199

**Note:** If a page is already in the buffer pool, it is updated in place. When a page is loaded into the buffer pool, any buffered changes to the page are merged, so that users never see the unmerged changes. The insert buffer bitmap keeps track of the available space within each page so that it can prevent overflows when inserting data. Records marked for deletion are always buffered, because the flag is updated in place.

- Calculate the insert buffer efficiency for merged operations by determining the ratio of records marked for deletion to inserts and expressing that ratio as a percentage.

- What is the insert buffer efficiency for this reporting period?

**Answer:**  $147519 / 314199 * 100 = 46.95\%$

- The last section displays the hash table size, the number of used cells, and the number of buffers used by the adaptive hash index.

- On an average, how many hash searches per second occurred during the reporting period? **Answer:** 13418.58
- On an average, how many non-hash searches per second occurred during the reporting period? **Answer:** 18601.04

- Calculate the ratio of hash index lookups to non-hash index lookups.

- What is the ratio of hash index lookups to non-hash index lookups during the reporting period?
- Answer:**  $(18601.04 / 13418.58) = 1.4$ . Usually, you would want to see a better ratio than this, because hash index lookups are faster than non-hash index lookups.

## Log

The LOG section of the report displays the current log sequence number, which corresponds to the number of bytes that InnoDB has written to the log files during the lifetime of the tablespace. It also tells you which logs InnoDB has flushed to disk, and when the last checkpoint occurred.

**Note:** The answers to the questions in this section are based on the following sample output:

```
...
-----
LOG
-----
Log sequence number 4847801726
Log flushed up to 4847676053
Pages flushed up to 4824824138
Last checkpoint at 4824796186
0 pending log flushes, 0 pending chkp writes
58 log i/o's done, 0.81 log i/o's/second
...
```

22. What is the Log sequence number value? **Answer:** 4847801726
23. What is the Log flushed up to value? **Answer:** 4847676053
24. What is the percentage difference between these two values?  
**Answer:**  $((4847801726 - 4847676053) / 4847801726) * 100 = 0.003\%$   
**Note:** A difference that is greater than 30 percent might indicate that the value of the `innodb_log_buffer_size` system variable is too small.
25. How many pending log flushes are there? **Answer:** 0
26. How many pending checkpoint (chkp) writes are there? **Answer:** 0
27. How many log i/o's have occurred during the reporting period? **Answer:** 58
28. What is the average number of log i/o's per second? **Answer:** 0.81  
**Note:** The log write performance is primarily based on the `innodb_flush_logs_at trx_commit` system variable setting, which controls when the log buffer is written to and when the logs are flushed. Changing the value of `innodb_flush_logs_at trx_commit` to zero can improve log I/O performance, but you risk losing up to one second's worth of transactions in the event of a system crash.

## Buffer Pool and Memory

The BUFFER POOL AND MEMORY section of the report displays information that helps you to determine if the InnoDB buffer pool is sized correctly.

**Note:** The answers to the questions in this section are based on the following sample output:

```
...
-----
BUFFER POOL AND MEMORY
-----
```

```

Total large memory allocated 137428992
Dictionary memory allocated 1087975
Buffer pool size 8191
Free buffers 1071
Database pages 5660
Old database pages 2069
Modified db pages 5508
Pending reads 0
Pending writes: LRU 0, flush list 8, single page 0
Pages made young 70260, not young 304998
1190.83 youngs/s, 3861.36 non-youngs/s
Pages read 16369, created 734, written 3248
76.64 reads/s, 11.85 creates/s, 54.42 writes/s
Buffer pool hit rate 998 / 1000, young-making rate 32 / 1000 not
105 / 1000
Pages read ahead 0.00/s, evicted without access 0.00/s, Random
read ahead 0.00/s
LRU len: 5660, unzip_LRU len: 0
I/O sum[7241]:cur[6], unzip sum[0]:cur[0]

```

29. What is the total memory allocation for InnoDB? **Answer:** 137428992
30. What is the Buffer pool size value in MB? **Answer:** 128 MB ( $8191 * 16384$ )  

**Note:** The buffer pool size in the report corresponds to the number of pages. Each page is 16,384 bytes by default, and can be set to 4K, 8K, or 16K by specifying an appropriate value for the `innodb_page_size` variable. A `SHOW ENGINE INNODB STATUS` buffer pool size of 8,191 corresponds to an `innodb_buffer_pool_size` system variable value of 134,201,344 bytes (128 MB).
31. How many free buffers are there? **Answer:** 1071  

**Note:** If the proportion of free buffers is consistently high, you might have over-allocated `innodb_buffer_pool_size` for the size of your database. Consider reducing the value of this system variable to reclaim unused space.
32. What is the Buffer pool hit rate? **Answer:**  $998/1000 * 100 = 99.8\%$   

**Note:** The higher the hit rate, the greater the efficiency of the buffer pool. An efficient buffer pool completes most page reads from memory. An inefficient buffer pool frequently reads pages from disk. You should aim to achieve a 95 percent or greater efficiency for most systems.

## Row Operations

The ROW OPERATIONS section of the report displays summary information about the number of rows that the system has read from and written to the database, along with other statistics. You can monitor these metrics to show InnoDB load over time.

**Note:** The answers to the questions in this section are based on the following sample output:

```
...
-----
ROW OPERATIONS
-----
0 queries inside InnoDB, 0 queries in queue
0 read views open inside InnoDB
Process ID=4892, Main thread ID=140240513189632, state: sleeping
Number of rows inserted 0, updated 107281, deleted 0, read
114805
0.00 inserts/s, 1818.29 updates/s, 0.00 deletes/s, 1818.55
reads/s
-----
END OF INNODB MONITOR OUTPUT
=====

1 row in set (#.## sec)
```

33. How many rows have been inserted since system startup? **Answer:** 0
34. How many rows have been updated since system startup? **Answer:** 107281
35. How many rows have been deleted since system startup? **Answer:** 0
36. How many rows have been read since system startup? **Answer:** 114805

**Note:** The number of rows affected by an operation is not necessarily a useful measure of performance, due to the potential for large differences in the size of the rows (for example, accessing a 2-byte row is much cheaper than accessing a 2 MB BLOB). However, the number of rows is a better indicator of performance than the number of queries, which can have a much greater disparity.

## Completion

37. In the t2 terminal window, terminate any running processes by pressing CTRL + C.
38. Close the t2 terminal window.
39. Keep the mysql client session in t1 open for the next practice.

## Practice 7-3: Monitoring InnoDB Metrics in the Information Schema

---

### Overview

In this practice, you work with the `INFORMATION_SCHEMA.INNODB_METRICS` table to learn how to monitor specific InnoDB metrics.

### Assumptions

- The MySQL server is running.
- The `perf` database is installed and contains the `innodb_inserts` table.
- The `mysql` client session terminal window is open from the previous practice.

### Duration

This practice should take you approximately 15 minutes to complete.

### Tasks

1. In the `mysql` session from the previous practice, change the default database to `INFORMATION_SCHEMA`.
2. Issue a `SHOW CREATE TABLE` statement to display the structure of the `INNODB_METRICS` table. Note the information that this table stores about each of the InnoDB metrics.
  - Which field tells you which metric a row in the `INNODB_METRICS` table refers to?
  - Which field tells you the current counter value of the metric?
3. Query the `INNODB_METRICS` table to answer the following questions:
  - How many different metrics does the `INNODB_METRICS` table support?
  - How many different subsystems do these metrics belong to?
  - How many metrics are currently reporting their status?
  - Is the `dml_inserts` metric currently enabled?
4. Execute the following statement at the `mysql` command-line prompt to reset the counter for `dml_inserts`:
 

```
SET GLOBAL innodb_monitor_reset=dml_inserts;
```
5. Insert three records into the `innodb_test` table of the `perf` database.
6. Query the `INNODB_METRICS` table to retrieve information about the `dml_inserts` metric. What is the current value of the `dml_inserts` counter?
7. The `dml_inserts` metric belongs to the DML module. Execute the following statement at the `mysql` command-line prompt to reset the counters for all the metrics in the DML module:
 

```
SET GLOBAL innodb_monitor_reset = module_dml;
```
8. Re-issue the query from step 6. What do you notice about the `dml_inserts` counter values?
9. Disable the `dml_inserts` monitor.

10. Set a global variable to reset all counters for the `dml_inserts` monitor.
11. Re-enable the `dml_inserts` monitor.
12. Re-issue the query from step 6.
  - What do you notice about the `dml_inserts` counter values?
  - What are the values of the `TYPE` and `COMMENTS` columns, and what do they tell you about what the `dml_inserts` metric represents?
13. Query the `INNODB_METRICS` table to retrieve information about the `buffer_pool_pages_total` metric.
  - What is the `COUNT` value of the `buffer_pool_pages_total` metric?
  - What is the `TYPE` of the `buffer_pool_pages_total` metric?
14. Keep the `mysql` session open for the next practice.

## Solution 7-3: Monitoring InnoDB Metrics in the Information Schema

### Solution Steps

1. In the mysql session from the previous practice, change the default database to INFORMATION\_SCHEMA.

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> USE INFORMATION_SCHEMA
Reading table information for completion of table and column
names
...
Database changed.
```

2. Issue a SHOW CREATE TABLE statement to display the structure of the INNODB\_METRICS table. Note the information that this table stores about each of the InnoDB metrics.

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> SHOW CREATE TABLE INNODB_METRICS\G
***** 1. row *****
Table: INNODB_METRICS
Create Table: CREATE TEMPORARY TABLE `INNODB_METRICS` (
  `NAME` varchar(193) NOT NULL DEFAULT '',
  `SUBSYSTEM` varchar(193) NOT NULL DEFAULT '',
  `COUNT` bigint(21) NOT NULL DEFAULT '0',
  `MAX_COUNT` bigint(21) DEFAULT NULL,
  `MIN_COUNT` bigint(21) DEFAULT NULL,
  `AVG_COUNT` double DEFAULT NULL,
  `COUNT_RESET` bigint(21) NOT NULL DEFAULT '0',
  `MAX_COUNT_RESET` bigint(21) DEFAULT NULL,
  `MIN_COUNT_RESET` bigint(21) DEFAULT NULL,
  `AVG_COUNT_RESET` double DEFAULT NULL,
  `TIME_ENABLED` datetime DEFAULT NULL,
  `TIME_DISABLED` datetime DEFAULT NULL,
  `TIME_ELAPSED` bigint(21) DEFAULT NULL,
  `TIME_RESET` datetime DEFAULT NULL,
  `STATUS` varchar(193) NOT NULL DEFAULT '',
  `TYPE` varchar(193) NOT NULL DEFAULT '',
  `COMMENT` varchar(193) NOT NULL DEFAULT ''
) ENGINE=MEMORY DEFAULT CHARSET=utf8
1 row in set (#.## sec)
```

- Which field tells you which metric a row in the INNODB\_METRICS table refers to? **Answer:** The NAME field

- Which field tells you the current counter value of the metric? **Answer:** The COUNT field

3. Query the INNODB\_METRICS table to answer the following questions:

- How many different metrics does the INNODB\_METRICS table support?
  - Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> SELECT DISTINCT NAME, SUBSYSTEM
-> FROM INNODB_METRICS
-> ORDER BY NAME;
+-----+-----+
| NAME           | SUBSYSTEM |
+-----+-----+
| adaptive_hash_pages_added | adaptive_hash_index |
| adaptive_hash_pages_removed | adaptive_hash_index |
| adaptive_hash_rows_added | adaptive_hash_index |
| adaptive_hash_rows_deleted_no_hash_entry | adaptive_hash_index |
| adaptive_hash_rows_removed | adaptive_hash_index |
| adaptive_hash_rows_updated | adaptive_hash_index |
| adaptive_hash_searches | adaptive_hash_index |
| adaptive_hash_searches_btree | adaptive_hash_index |
| buffer_data_reads | buffer |
| buffer_data_written | buffer |
| ... |
| trx_rseg_history_len | transaction |
| trx_rw_commits | transaction |
| trx_undo_slots_cached | transaction |
| trx_undo_slots_used | transaction |
+-----+-----+
235 rows in set (#.# sec)
```

- Answer: 214

- How many different subsystems do these metrics belong to?
  - Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> SELECT DISTINCT SUBSYSTEM FROM INNODB_METRICS;
+-----+
| SUBSYSTEM |
+-----+
| metadata   |
| lock       |
| server     |
| buffer     |
| buffer_page_io |
| os          |
| transaction |
| purge      |
| recovery   |
| compression |
| index      |
+-----+
```

```
| adaptive_hash_index |
| file_system          |
| change_buffer         |
| dml                  |
| ddl                  |
| icp                  |
+-----+
17 rows in set (#.## sec)
```

- **Answer:** 17

- How many metrics are currently reporting their status?
  - Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> SELECT NAME FROM INNODB_METRICS WHERE STATUS='enabled';
+-----+
| NAME           |
+-----+
| lock_deadlocks |
| lock_timeouts  |
| lock_row_lock_current_waits |
| lock_row_lock_time   |
| lock_row_lock_time_max |
...
| buffer_pool_size |
| buffer_pool_reads |
| buffer_pool_read_requests |
| buffer_pool_write_requests |
...
| dml_deletes    |
| dml_updates    |
+-----+
65 rows in set (#.## sec)
```

- **Answer:** 65

- Is the `dml_inserts` metric currently enabled?
  - Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> SELECT NAME, STATUS FROM INNODB_METRICS
      -> WHERE NAME='dml_inserts';
+-----+-----+
| NAME      | STATUS   |
+-----+-----+
| dml_inserts | enabled |
+-----+-----+
1 row in set (#.## sec)
```

- **Answer:** Yes

4. Execute the following statement at the mysql command-line prompt to reset the counter for dml\_inserts:

```
SET GLOBAL innodb_monitor_reset=dml_inserts;
```

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> SET GLOBAL innodb_monitor_reset=dml_inserts;
Query OK, 0 rows affected (#.## sec)
```

5. Insert three records into the innodb\_test table of the perf database.

Enter the following statements at the mysql prompt and receive the results shown:

```
mysql> INSERT INTO perf.innodb_test (val) VALUES (101);
Query OK, 1 row affected (#.## sec)
```

```
mysql> INSERT INTO perf.innodb_test (val) VALUES (102);
Query OK, 1 row affected (#.## sec)
```

```
mysql> INSERT INTO perf.innodb_test (val) VALUES (103);
Query OK, 1 row affected (#.## sec)
```

6. Query the INNODB\_METRICS table to retrieve information about the dml\_inserts metric.

- What is the current value of the dml\_inserts counter?

Enter the following statements at the mysql prompt and receive the results shown:

```
mysql> SELECT * FROM INNODB_METRICS
-> WHERE NAME = 'dml_inserts'\G
***** 1. row *****
      NAME: dml_inserts
      SUBSYSTEM: dml
      COUNT: 3
      MAX_COUNT: 3
      MIN_COUNT: NULL
      AVG_COUNT: 0.05084745762711865
      COUNT_RESET: 3
      MAX_COUNT_RESET: 3
      MIN_COUNT_RESET: NULL
      AVG_COUNT_RESET: NULL
      TIME_ENABLED: <date and time>
      TIME_DISABLED: NULL
      TIME_ELAPSED: 59
      TIME_RESET: NULL
      STATUS: enabled
      TYPE: status_counter
      COMMENT: Number of rows inserted
1 row in set (#.## sec)
```

- **Answer:** Three. This corresponds to the three records inserted in the previous step.
7. The `dml_inserts` metric belongs to the DML module. Execute the following statement at the `mysql` command-line prompt to reset the counters for all the metrics in the DML module:

```
SET GLOBAL innodb_monitor_reset = module_dml;
```

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> SET GLOBAL innodb_monitor_reset = module_dml;
Query OK, 0 rows affected (#.## sec)
```

8. Re-issue the query from step 6.

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> SELECT * FROM INNODB_METRICS
-> WHERE NAME = 'dml_inserts'\G
***** 1. row *****
      NAME: dml_inserts
      SUBSYSTEM: dml
      COUNT: 3
      MAX_COUNT: 3
      MIN_COUNT: NULL
      AVG_COUNT: 0.01694915254237288
      COUNT_RESET: 0
      MAX_COUNT_RESET: 0
      MIN_COUNT_RESET: NULL
      AVG_COUNT_RESET: 0
      TIME_ENABLED: <date and time>
      TIME_DISABLED: NULL
      TIME_ELAPSED: 177
      TIME_RESET: <date and time>
      STATUS: enabled
      TYPE: status_counter
      COMMENT: Number of rows inserted
1 row in set (#.## sec)
```

- What do you notice about the `dml_inserts` counter values? **Answer:** Only the `COUNT_RESET`, `MAX_COUNT_RESET`, and `AVG_COUNT_RESET` column values are reset. The `COUNT`, `MAX_COUNT`, and `AVG_COUNT` fields are unchanged. The `TIME_RESET` column contains the time at which the values were reset.
9. Disable the `dml_inserts` monitor.

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> SET GLOBAL innodb_monitor_disable = dml_inserts;
Query OK, 0 rows affected (#.## sec)
```

10. Set a global variable to reset all the counters for the `dml_inserts` monitor.

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> SET GLOBAL innodb_monitor_reset_all = dml_inserts;
Query OK, 0 rows affected (#.## sec)
```

11. Re-enable the `dml_inserts` monitor.

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> SET GLOBAL innodb_monitor_enable = dml_inserts;
Query OK, 0 rows affected (#.## sec)
```

12. Re-issue the query from step 6.

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> SELECT * FROM INNODB_METRICS WHERE NAME = 'dml_inserts' \G
***** 1. row *****
      NAME: dml_inserts
    SUBSYSTEM: dml
        COUNT: 0
    MAX_COUNT: 0
    MIN_COUNT: NULL
    AVG_COUNT: 0
    COUNT_RESET: 0
    MAX_COUNT_RESET: 0
    MIN_COUNT_RESET: NULL
    AVG_COUNT_RESET: NULL
    TIME_ENABLED: <date and time>
    TIME_DISABLED: NULL
    TIME_ELAPSED: 23
    TIME_RESET: NULL
        STATUS: enabled
        TYPE: status_counter
    COMMENT: Number of rows inserted
1 row in set (0.00 sec)
```

- What do you notice about the `dml_inserts` counter values? **Answer:** All the counter values were reset. You must disable the monitor before you can reset the values of the `COUNT` and `MAX_COUNT` fields.
- What are the values of the `TYPE` and `COMMENTS` columns, and what do they tell you about what the `dml_inserts` metric represents? **Answer:** The `TYPE` column displays “status\_counter.” The `COMMENT` column displays “Number of rows inserted.” This means that the `dml_inserts` metric is a simple tally of the number of rows inserted into any InnoDB table.

13. Query the INNODB\_METRICS table to retrieve information about the buffer\_pool\_pages\_total metric.

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> SELECT * FROM INNODB_METRICS
-> WHERE NAME = 'buffer_pool_pages_total' \G
***** 1. row *****
      NAME: buffer_pool_pages_total
    SUBSYSTEM: buffer
      COUNT: 8191
    MAX_COUNT: 8191
    MIN_COUNT: 8191
    AVG_COUNT: NULL
  COUNT_RESET: 8191
MAX_COUNT_RESET: 8191
MIN_COUNT_RESET: 8191
AVG_COUNT_RESET: NULL
    TIME_ENABLED: <date and time>
TIME_DISABLED: NULL
    TIME_ELAPSED: 477
    TIME_RESET: NULL
      STATUS: enabled
      TYPE: value
    COMMENT: Total buffer pool size in pages
(innodb_buffer_pool_pages_total)
1 row in set (#.## sec)
```

- What is the COUNT value of the buffer\_pool\_pages\_total metric? **Answer:** 8191
- What is the TYPE of the buffer\_pool\_pages\_total metric? **Answer:** value. This means that this metric is not just a simple counter like dml\_inserts, but is instead an indicator of resource usage. In this instance, it refers to the total number of pages in the buffer pool, as described in the COMMENT field.

14. Keep the mysql session open for the next practice.

## Practice 7-4: Evaluating InnoDB Buffer Pool Size

### Overview

In this practice, you change the size of the InnoDB buffer pool and evaluate the effect on scripts that place a load on the MySQL server. To accomplish this objective, you perform the following:

- Determine the current InnoDB server settings.
- Use the `mysqlslap` command-line tool to create a load against the MySQL server.
- Modify the size of the InnoDB buffer pool and notice any changes in performance.
- While the `mysqlslap` test is running, execute the `SHOW ENGINE INNODB STATUS` statement to determine the buffer pool hit rate.

**Note:** This practice uses two Linux terminal windows. The instructions refer to the terminal windows as `t1` and `t2`. To make it easier to identify each terminal, change its title by using the Terminal > Set Title menu option in each terminal window.

### Assumptions

- The `perf` database is installed and contains the `city_huge` table.
- The `innodb_buffer.sql` script file is located in the `/root/scripts` directory.
- You have a `mysql` session open in a Linux terminal window from the preceding practice. This practice refers to this terminal window as `t1`.

### Duration

This practice should take you approximately 20 minutes to complete.

### Tasks

1. In the `mysql` terminal window (`t1`) session from the previous practice, list all the system variables that begin with “`innodb_buffer_pool`.”
  - What is the default size of the InnoDB buffer pool?
  - How is the size of the buffer pool related to `innodb_buffer_pool_chunk_size` and `innodb_buffer_pool_instances`?
2. Open a new Linux terminal window as the `root` Linux user. This is terminal window `t2`.
3. In terminal window `t2`, execute the following `mysqlslap` command to create a load against the MySQL server by using the `innodb_buffer.sql` script:

```
# mysqlslap -uroot -p \
> --create-schema=perf \
> -q /root/scripts/innodb_buffer.sql \
> -i 5 \
> -c 1
```

**Note:** This command takes a while to complete. While the command executes, proceed with the next step.

4. At the mysql prompt in terminal window t1, repeatedly execute the SHOW ENGINE INNODB STATUS statement until the command that you issued in terminal window t2 completes. What do you notice about the “Buffer pool hit rate” entry of the “BUFFER POOL AND MEMORY” section of the report?
  5. When the test in terminal window t2 completes, note the average execution time.
  6. In t2, open /etc/my.cnf in a text editor and add a server option to change the value of the innodb\_buffer\_pool\_chunk\_size system variable to 1 MB (1048576). This will allow you to reduce the size of the InnoDB buffer pool in the subsequent steps.
  7. In terminal window t2, restart the MySQL server to apply the changes to the /etc/my.cnf options file.
- Note:** Although you can resize the buffer pool online, changing the value of innodb\_buffer\_pool\_chunk\_size requires a server restart.
8. At the mysql prompt in terminal window t1, enter a command such as STATUS to force the mysql session to reconnect to the server.
  9. At the mysql prompt in terminal window t1, set the innodb\_buffer\_pool\_size global variable to the minimum possible size of 5242880 (5 MB).
  10. In terminal window t2, execute the following mysqlslap command to create a load against the MySQL server by using the innodb\_buffer.sql script:

```
# mysqlslap -root -p --create-schema=perf \
> -q /root/scripts/innodb_buffer.sql \
> -i 5 \
> -c 1
```

- Note:** This command takes a while to complete. While the command executes, proceed with the next step.
11. At the mysql prompt in terminal window t1, repeatedly execute the SHOW ENGINE INNODB STATUS statement until the command that you issued in terminal window t2 completes. What do you notice about the “Buffer pool hit rate” entry of the “BUFFER POOL AND MEMORY” section of the report?
  12. When the test in terminal window t2 completes, note the average execution time. What do you notice about the average execution time?
  13. Exit the mysql session in terminal window t1 and close the terminal window.
  14. In t2, open /etc/my.cnf in a text editor and remove the configuration entry that sets innodb\_buffer\_pool\_chunk\_size to 1 MB.
  15. In terminal window t2, restart the MySQL server.
  16. Close terminal window t2.

## Solution 7-4: Evaluating InnoDB Buffer Pool Size

---

### Solution Steps

1. In the mysql terminal window (`t1`) session from the previous practice, list all the system variables that begin with “`innodb_buffer_pool`.”

Enter the following statement at the mysql prompt in `t1` and receive the results shown:

```
mysql> SHOW VARIABLES LIKE 'innodb_buffer_pool%';
+-----+-----+
| Variable_name          | Value   |
+-----+-----+
| innodb_buffer_pool_chunk_size | 134217728 |
| innodb_buffer_pool_dump_at_shutdown | ON      |
| innodb_buffer_pool_dump_now       | OFF     |
| innodb_buffer_pool_dump_pct      | 25      |
| innodb_buffer_pool_filename     | ib_buffer_pool |
| innodb_buffer_pool_instances    | 1       |
| innodb_buffer_pool_load_abort   | OFF     |
| innodb_buffer_pool_load_at_startup | ON      |
| innodb_buffer_pool_load_now     | OFF     |
| innodb_buffer_pool_size         | 134217728 |
+-----+-----+
10 rows in set (0.00 sec)
```

- What is the default size of the InnoDB buffer pool? **Answer:** 134217728 (128 MB)
  - How is the size of the buffer pool related to `innodb_buffer_pool_chunk_size` and `innodb_buffer_pool_instances`? **Answer:** `innodb_buffer_pool_size` is always equal to or a multiple of `innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances`. If you specify a value for `innodb_buffer_pool_size` that is not equal to or a multiple of that figure, the buffer pool resizes to a value that is equal to or a multiple of that figure.
2. Open a new Linux terminal window as the `root` Linux user. This is terminal window `t2`.
  3. In terminal window `t2`, execute the following `mysqlslap` command to create a load against the MySQL server by using the `innodb_buffer.sql` script:

```
# mysqlslap -uroot -p \
> --create-schema=perf \
> -q /root/scripts/innodb_buffer.sql \
> -i 5 \
> -c 1
```

**Note:** This command takes a while to complete. While the command executes, proceed with the next step.

Enter the following command at the t2 Linux terminal prompt and receive the results shown:

```
# mysqlslap -uroot -p \
> --create-schema=perf \
> -q /root/scripts/innodb_buffer.sql \
> -i 5 \
> -c 1
Enter password: oracle
```

- At the mysql prompt in terminal window t1, repeatedly execute the SHOW ENGINE INNODB STATUS statement until the command that you issued in terminal window t2 completes. What do you notice about the “Buffer pool hit rate” entry of the “BUFFER POOL AND MEMORY” section of the report?

Enter the following statement at the mysql prompt and locate the highlighted entry in the output:

```
mysql> SHOW ENGINE INNODB STATUS\G
...
-----
BUFFER POOL AND MEMORY
-----
Total large memory allocated 138412032
Dictionary memory allocated 1092830
Buffer pool size     8064
Free buffers        1024
Database pages      7018
Old database pages  2570
Modified db pages   1157
Pending reads       0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young    607, not young 74564
53.95 youngs/s, 1.00 non-youngs/s
Pages read 2658, created 8430, written 2708
1.00 reads/s, 511.49 creates/s, 263.74 writes/s
Buffer pool hit rate 1000 / 1000, young-making rate 0 / 1000 not
0 / 1000
Pages read ahead 0.00/s, evicted without access 0.00/s, Random
read ahead 0.00/s
LRU len: 7018, unzip_LRU len: 0
I/O sum[1387]:cur[146], unzip sum[0]:cur[0]
```

- Answer:** The buffer pool hit rate is consistently 1000/1000, or 100%, suggesting that the contents of the database that you are querying against fit entirely in the InnoDB buffer pool.

- When the test in terminal window t2 completes, note the average execution time.

In the following sample output, the average execution time is approximately 8.56 seconds:

Benchmark

```
Average number of seconds to run all queries: 8.552 seconds
Minimum number of seconds to run all queries: 7.843 seconds
Maximum number of seconds to run all queries: 8.944 seconds
Number of clients running queries: 1
Average number of queries per client: 5
```

6. In t2, open /etc/my.cnf in a text editor and add a server option to change the value of the innodb\_buffer\_pool\_chunk\_size system variable to 1 MB (1048576). This will allow you to reduce the size of the InnoDB buffer pool in the subsequent steps.

Make the following change to /etc/my.cnf:

```
[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
user=mysql
# Disabling symbolic-links is recommended to prevent assorted
# security risks
symbolic-links=0
server_id=999
log_bin
binlog_format=ROW
binlog_cache_size=30000000
innodb_buffer_pool_chunk_size=1M
```

7. In terminal window t2, restart the MySQL server to apply the changes to the /etc/my.cnf options file.

**Note:** Although you can resize the buffer pool online, changing the value of innodb\_buffer\_pool\_chunk\_size requires a server restart.

Enter the following command at the t2 Linux terminal prompt and receive the results shown:

```
# service mysql restart
Shutting down MySQL.... SUCCESS!
Starting MySQL... SUCCESS!
```

8. At the mysql prompt in terminal window t1, enter a command such as STATUS to force the mysql session to reconnect to the server.

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> STATUS
ERROR 2006 (HY000): MySQL server has gone away
No connection. Trying to reconnect...
Connection id:      3
Current database: *** NONE ***
...
```

9. At the mysql prompt in terminal window t1, set the innodb\_buffer\_pool\_size global variable to the minimum possible size of 5242880 (5 MB).

Enter the following statement at the mysql prompt in t1 and receive the results shown:

```
mysql> SET GLOBAL innodb_buffer_pool_size=5242880;
Query OK, 0 rows affected (#.# sec)
```

**Note:** You can resize the buffer pool online, without having to restart the server. When applying the change online, you must specify the number of bytes and not megabytes as you did when changing the value of innodb\_buffer\_pool\_chunk\_size in the options file.

10. In terminal window t2, execute the following mysqlslap command to create a load against the MySQL server by using the innodb\_buffer.sql script:

```
# mysqlslap -uroot -p \
> --create-schema=perf \
> -q /root/scripts/innodb_buffer.sql \
> -i 5 \
> -c 1
```

**Note:** This command takes a while to complete. While the command executes, proceed with the next step.

Enter the following command at the t2 Linux terminal prompt and receive the results shown:

```
# mysqlslap -uroot -p \
> --create-schema=perf \
> -q /root/scripts/innodb_buffer.sql \
> -i 5 \
> -c 1
Enter password: oracle
```

11. At the mysql prompt in terminal window t1, repeatedly execute the SHOW ENGINE INNODB STATUS statement until the command that you issued in terminal window t2 completes. What do you notice about the “Buffer pool hit rate” entry of the “BUFFER POOL AND MEMORY” section of the report?

Enter the following statement at the mysql prompt and locate the highlighted entry in the output:

```
mysql> SHOW ENGINE INNODB STATUS\G
...
-----
BUFFER POOL AND MEMORY
-----
Total large memory allocated 5406720
Dictionary memory allocated 1040438
Buffer pool size    315
Free buffers        1
```

```

Database pages      287
Old database pages 0
Modified db pages  181
Pending reads      93
Pending writes: LRU 0, flush list 0, single page 3
Pages made young 0, not young 0
0.00 youngs/s, 0.00 non-youngs/s
Pages read 539149, created 5398, written 230641
23.49 reads/s, 0.00 creates/s, 72.46 writes/s
Buffer pool hit rate 863 / 1000, young-making rate 0 / 1000 not
0 / 1000
Pages read ahead 0.00/s, evicted without access 0.50/s, Random
read ahead 0.00/s
LRU len: 287, unzip_LRU len: 0
I/O sum[9959]:cur[14], unzip sum[0]:cur[0]
...

```

- **Answer:** The buffer pool hit rate fluctuates as the query executes, but never reaches 100% (1000/1000). If the buffer pool size is optimal, the hit rate should be consistently on or near 100%.
12. When the test in terminal window t2 completes, note the average execution time. What do you notice about the average execution time?

In the following sample output, the average execution time is approximately 60 seconds:

```

Benchmark
Average number of seconds to run all queries: 60.015 seconds
Minimum number of seconds to run all queries: 54.646 seconds
Maximum number of seconds to run all queries: 62.047 seconds
Number of clients running queries: 1
Average number of queries per client: 5

```

- **Answer:** The average execution time is considerably longer (60 seconds in the example shown) due to the overhead of file I/O that is required when the database does not fit entirely within the InnoDB buffer pool. It is always a good idea to make the InnoDB buffer pool as large as possible to reduce the amount of file I/O. A good guideline to use when sizing the InnoDB buffer pool is 80% of the available system RAM.

13. Exit the mysql session in terminal window t1 and close the terminal window.

Enter the following statement at the mysql prompt and receive the results shown:

```

mysql> EXIT
Bye
#

```

Enter the following command at the Linux terminal prompt to close terminal window t1:

```
# exit
```

14. In t2, open /etc/my.cnf in a text editor and remove the configuration entry that sets innodb\_buffer\_pool\_chunk\_size to 1 MB.

Delete the entry marked in bold italics as shown:

```
[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
user=mysql
# Disabling symbolic-links is recommended to prevent assorted
security risks
symbolic-links=0
server_id=999
log_bin
binlog_format=ROW
binlog_cache_size=30000000
innodb_buffer_pool_chunk_size=1M
...
```

15. In terminal window t2, restart the MySQL server.

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# service mysql restart
Shutting down MySQL... SUCCESS!
Starting MySQL.. SUCCESS!
```

16. Close terminal window t2.

Enter the following command at the Linux terminal prompt to close terminal window t1:

```
# exit
```

## **Practices for Lesson 8: Optimizing Your Schema**

## Practices for Lesson 8

---

### Overview

These practices test your knowledge of schema design and its impact on performance.

**Note:** The output of the various Linux commands and SQL statements shown in the solution steps for these practices might be different on your system.

## Practice 8-1: Comparing the Effects of Table Normalization on Query Performance

### Overview

In this practice, you evaluate the performance implications of normalized and denormalized data for different types of queries. To achieve this, you perform the following steps:

- Use the `mysqlslap` utility to execute queries that retrieve data from normalized and denormalized tables that do not require the joining of normalized tables.
- Use the `mysqlslap` utility to execute queries that retrieve data from normalized and denormalized tables that require the joining of normalized tables.

### Assumptions

- The `world` and `world_NonNorm` databases are installed.
- Python 2.7 is installed.
- The following Python scripts exist in the `/root/scripts` directory:
  - `random-pop-norm.py`
  - `random-pop-denorm.py`
  - `random-lang-norm.py`
  - `random-lang-denorm.py`

### Duration

This practice should take you approximately 20 minutes to complete.

### Tasks

1. Open a terminal window and use the `mysql` client to review the design of the `world_all` table in the `world_NonNorm` database.  
**Note:** The `world_all` table is a combination of the `city`, `country`, and `countrylanguage` tables from the `world` database in a denormalized form.
2. Use the `/root/scripts/random-pop-denorm.py` Python script to create a SQL script file called `popquery_denorm.sql` that contains 100 `SELECT` statements against the `world_all` table in the `world_NonNorm` database. Each statement retrieves records based on a random population size.
3. Verify that the `popquery_denorm.sql` file exists in `/root/scripts` and contains the queries described in the preceding step.
4. Use the `mysqlslap` command-line utility to execute a test with the following parameters:
  - Query file to execute: `/root/scripts/popquery_denorm.sql`
  - Schema: `world_NonNorm`
  - Iterations: 5
  - Concurrency: 20 threads
  - Verbose output (`-vv` option)

What is the average amount of time each thread takes to complete?

5. Use the `/root/scripts/random-pop-norm.py` Python script to create a SQL script file called `popquery_norm.sql` that contains 100 `SELECT` statements against the `country` table in the `world` database. Each statement retrieves records based on a random population size.
6. Verify that the `popquery_norm.sql` file exists in `/root/scripts` and contains the queries described in the preceding step.
7. Use the `mysqlslap` command-line utility to execute a test with the following parameters:
  - Schema to query against: `world`
  - Query file to execute: `/root/scripts/popquery_norm.sql`
  - Iterations: 5
  - Concurrency: 20 threads
  - Verbose output (`-vv` option)

What is the average amount of time each thread takes to complete?

8. Use the `/root/scripts/random-lang-denorm.py` Python script to create a SQL script file called `langquery_denorm.sql` that contains 100 `SELECT` statements against the `world_all` table in the `world_NonNorm` database. Each statement retrieves records based on a random percentage value that corresponds to a proportion of people in a country who speak a certain language.
9. Verify that the `langquery_denorm.sql` file exists in the `/root/scripts` directory and contains the queries described in the preceding step.
10. Use the `mysqlslap` command-line utility to execute a test with the following parameters:
  - Schema to query against: `world_NonNorm`
  - Query file to execute: `/root/scripts/langquery_denorm.sql`
  - Iterations: 5
  - Concurrency: 20 threads
  - Verbose output (`-vv` option)

What is the average amount of time each thread takes to complete?

11. Use the `/root/scripts/random-lang-norm.py` Python script to create a SQL script file called `langquery_norm.sql` that contains 100 `SELECT` statements against the `city`, `country`, and `countrylanguage` tables in the `world` database. Each statement retrieves records based on a random percentage value that corresponds to a proportion of people in a country who speak a certain language, but relies on table joins to retrieve that information.
12. Verify that the `langquery_norm.sql` file exists in `/root/scripts` and contains the queries described in the preceding step.

13. Use the `mysqlslap` command-line utility to execute a test with the following parameters:

- Schema to query against: `world`
- Query file to execute: `/root/scripts/langquery_norm.sql`
- Iterations: 5
- Concurrency: 20 threads
- Verbose output (`-vv` option)

What is the average amount of time each thread takes to complete?

14. Review your test results. What do they tell you about the effects of normalizing tables on query performance?

15. Leave the Linux terminal window open and remain logged in as the `root` user. The next practice refers to this terminal window as `t1`.

## Solution 8-1: Comparing the Effects of Table Normalization on Query Performance

---

### Solution Steps

1. Open a terminal window and use the mysql client to review the design of the world\_all table in the world\_NonNorm database.

**Note:** The world\_all table is a combination of the city, country, and countrylanguage tables from the world database in a denormalized form.

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# mysql -uroot -p \
> -e"SHOW CREATE TABLE world_NonNorm.world_all\G"
Enter password: oracle
***** 1. row *****

Table: world_all
Create Table: CREATE TABLE `world_all` (
  `ID` int(11) NOT NULL AUTO_INCREMENT,
  `CountryCode` char(3) NOT NULL DEFAULT '',
  `City_Name` char(35) NOT NULL DEFAULT '',
  `Country_Name` char(52) NOT NULL DEFAULT '',
  `Continent` enum('Asia','Europe','North
America','Africa','Oceania','Antarctica','South America') NOT
NULL DEFAULT 'Asia',
  `Region` char(26) NOT NULL DEFAULT '',
  `SurfaceArea` float(10,2) NOT NULL DEFAULT '0.00',
  `IndepYear` smallint(6) DEFAULT NULL,
  `Country_Population` int(11) NOT NULL DEFAULT '0',
  `LifeExpectancy` float(3,1) DEFAULT NULL,
  `GNP` float(10,2) DEFAULT NULL,
  `GNPOld` float(10,2) DEFAULT NULL,
  `LocalName` char(45) NOT NULL DEFAULT '',
  `GovernmentForm` char(45) NOT NULL DEFAULT '',
  `HeadOfState` char(60) DEFAULT NULL,
  `Capital` int(11) DEFAULT NULL,
  `Country_District` char(20) NOT NULL DEFAULT '',
  `City_Population` int(11) NOT NULL DEFAULT '0',
  `Language` char(30) NOT NULL DEFAULT '',
  `IsOfficial` enum('T','F') NOT NULL DEFAULT 'F',
  `Percentage` float(4,1) NOT NULL DEFAULT '0.0',
  PRIMARY KEY (`ID`)
) ENGINE=InnoDB AUTO_INCREMENT=30671 DEFAULT CHARSET=latin1
```

2. Use the `/root/scripts/random-pop-denorm.py` Python script to create a SQL script file called `popquery_denorm.sql` that contains 100 `SELECT` statements against the `world_all` table in the `world_NonNorm` database. Each statement retrieves records based on a random population size.

Execute the `random-pop-denorm.py` script as follows:

```
# cd /root/scripts
# python random-pop-denorm.py
Done.
#
```

3. Verify that the `popquery_denorm.sql` file exists in `/root/scripts` and contains the queries described in the preceding step.

Use a Linux command-line tool such as `cat` or `less`, or a text editor such as `nano`, to verify the contents of the `popquery_denorm.sql` file:

```
# cat popquery_denorm.sql
...
SELECT DISTINCT Country_Name, GovernmentForm FROM
world_NonNorm.world_all WHERE Country_Population < 4751527;
SELECT DISTINCT Country_Name, GovernmentForm FROM
world_NonNorm.world_all WHERE Country_Population < 14646027;
SELECT DISTINCT Country_Name, GovernmentForm FROM
world_NonNorm.world_all WHERE Country_Population < 78992956;
#
```

**Note:** The `world_all` table in the `world_NonNorm` database is denormalized and contains duplicate data. The `SELECT DISTINCT` option eliminates duplicate data from the query results. The script randomizes the population value for comparison.

4. Use the `mysqlslap` command-line utility to execute a test with the following parameters:

- Query file to execute: `/root/scripts/popquery_denorm.sql`
- Schema: `world_NonNorm`
- Iterations: 5
- Concurrency: 20 threads
- Verbose output (`-vv` option)

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# mysqlslap -uroot -p \
> --query="/root/scripts/popquery_denorm.sql" \
> --create-schema=world_NonNorm \
> --iterations=1 \
> --concurrency=5 \
> -vv
Parsing engines to use.
Enter password: oracle
```

```
Benchmark
```

```
Average number of seconds to run all queries: 5.050 seconds
Minimum number of seconds to run all queries: 5.033 seconds
Maximum number of seconds to run all queries: 5.075 seconds
Number of clients running queries: 20
Average number of queries per client: 100
```

- What is the average amount of time each thread takes to complete? **Answer:**  
Approximately 5 seconds in the preceding sample output
5. Use the `/root/scripts/random-pop-norm.py` Python script to create a SQL script file called `popquery_norm.sql` that contains 100 `SELECT` statements against the `country` table in the `world` database. Each statement retrieves records based on a random population size.

Execute the `random_pop_norm.py` script as follows:

```
# cd /root/scripts
# python random-pop-norm.py
Done.
#
```

6. Verify that the `popquery_norm.sql` file exists in `/root/scripts` and contains the queries described in the preceding step.

Use a Linux command-line tool such as `cat` or `less`, or a text editor such as `nano`, to verify the contents of the `popquery-norm.sql` file:

```
# cat popquery_norm.sql
...
SELECT DISTINCT Name, GovernmentForm FROM world.country WHERE
population < 30191139;
SELECT DISTINCT Name, GovernmentForm FROM world.country WHERE
population < 16104915;
SELECT DISTINCT Name, GovernmentForm FROM world.country WHERE
population < 46702472;
SELECT DISTINCT Name, GovernmentForm FROM world.country WHERE
population < 144280934;
SELECT DISTINCT Name, GovernmentForm FROM world.country WHERE
population < 111855131;
SELECT DISTINCT Name, GovernmentForm FROM world.country WHERE
population < 24885873;
SELECT DISTINCT Name, GovernmentForm FROM world.country WHERE
population < 85241207;
```

**Note:** The `country` table in the `world` database is in a normalized form and includes all the data that is required by the `SELECT` statement. It does not contain duplicate data.

7. Use the `mysqlslap` command-line utility to execute a test with the following parameters:
- Schema to query against: `world`
  - Query file to execute: `/root/scripts/popquery_norm.sql`

- Iterations: 5
- Concurrency: 20 threads
- Verbose output (-vv option)

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# mysqlslap -uroot -p \
> --create-schema=world \
> --query="/root/scripts/popquery_norm.sql" \
> --iterations=1 \
> --concurrency=5 \
> -vv

Parsing engines to use.

Benchmark

Average number of seconds to run all queries: 0.099 seconds

Minimum number of seconds to run all queries: 0.098 seconds
Maximum number of seconds to run all queries: 0.103 seconds
Number of clients running queries: 20
Average number of queries per client: 100
```

- What is the average amount of time each thread takes to complete? **Answer:**  
Approximately 0.1 seconds in the preceding sample output

8. Use the /root/scripts/random-lang-denorm.py Python script to create a SQL script file called langquery\_denorm.sql that contains 100 SELECT statements against the world\_all table in the world\_NonNorm database. Each statement retrieves records based on a random percentage value that corresponds to a proportion of people in a country who speak a certain language.

Execute the random\_lang\_denorm.py script as follows:

```
# cd /root/scripts
# python random-lang-denorm.py
Done.
#
```

9. Verify that the langquery\_denorm.sql file exists in the /root/scripts directory and contains the queries described in the preceding step.

Use a Linux command-line tool such as cat or less, or a text editor such as nano, to verify the contents of the langquery\_denorm.sql file:

```
# cat langquery_denorm.sql
...
SELECT City_Name, Country_Name FROM world_NonNorm.world_all
WHERE Percentage < 49;
SELECT City_Name, Country_Name FROM world_NonNorm.world_all
WHERE Percentage < 41;
SELECT City_Name, Country_Name FROM world_NonNorm.world_all
WHERE Percentage < 7;
```

```

SELECT City_Name, Country_Name FROM world_NonNorm.world_all
WHERE Percentage < 83;
SELECT City_Name, Country_Name FROM world_NonNorm.world_all
WHERE Percentage < 32;
SELECT City_Name, Country_Name FROM world_NonNorm.world_all
WHERE Percentage < 72;
SELECT City_Name, Country_Name FROM world_NonNorm.world_all
WHERE Percentage < 69;
SELECT City_Name, Country_Name FROM world_NonNorm.world_all
WHERE Percentage < 48;

```

**Note:** The `world_all` table in the `world_NonNorm` database is denormalized and contains duplicate data. The script randomizes the percentage value for comparison.

10. Use the `mysqlslap` command-line utility to execute a test with the following parameters:

- Schema to query against: `world_NonNorm`
- Query file to execute: `/root/scripts/langquery_denorm.sql`
- Iterations: 5
- Concurrency: 20 threads
- Verbose output (`-vv` option)

Enter the following command at the Linux terminal prompt and receive the results shown:

```

# mysqlslap -uroot -p \
> --create-schema=world_NonNorm \
> --query="/root/scripts/langquery_denorm.sql" \
> --iterations=1 \
> --concurrency=5 \
> -vv
Parsing engines to use.
Enter password: oracle
Benchmark
      Average number of seconds to run all queries: 6.249 seconds
      Minimum number of seconds to run all queries: 6.228 seconds
      Maximum number of seconds to run all queries: 6.306 seconds
      Number of clients running queries: 20
      Average number of queries per client: 100

```

- What is the average amount of time each thread takes to complete? **Answer:** Approximately 6.25 seconds

11. Use the `/root/scripts/random-lang-norm.py` Python script to create a SQL script file called `langquery_norm.sql` that contains 100 `SELECT` statements against the `city`, `country`, and `countrylanguage` tables in the `world` database. Each statement retrieves records based on a random percentage value that corresponds to a proportion of people in a country who speak a certain language, but relies on table joins to retrieve that information.

Execute the `random-lang-norm.py` script as follows:

```
# cd /root/scripts
# python random-lang-norm.py
Done.
#
```

12. Verify that the `langquery_norm.sql` file exists in `/root/scripts` and contains the queries described in the preceding step.

Use a Linux command-line tool such as `cat` or `less`, or a text editor such as `nano`, to verify the contents of the `langquery_norm.sql` file:

```
# cat langquery_norm.sql
...
SELECT city.Name, country.Name FROM world.city, world.country,
world.countrylanguage WHERE city.countryCode = country.Code AND
country.Code = countrylanguage.CountryCode AND Percentage < 20;
SELECT city.Name, country.Name FROM world.city, world.country,
world.countrylanguage WHERE city.countryCode = country.Code AND
country.Code = countrylanguage.CountryCode AND Percentage < 59;
SELECT city.Name, country.Name FROM world.city, world.country,
world.countrylanguage WHERE city.countryCode = country.Code AND
country.Code = countrylanguage.CountryCode AND Percentage < 33;
SELECT city.Name, country.Name FROM world.city, world.country,
world.countrylanguage WHERE city.countryCode = country.Code AND
country.Code = countrylanguage.CountryCode AND Percentage < 100;
SELECT city.Name, country.Name FROM world.city, world.country,
world.countrylanguage WHERE city.countryCode = country.Code AND
country.Code = countrylanguage.CountryCode AND Percentage < 22;
```

**Note:** The output requires data from the `city` and `country` tables, and the `WHERE` clause requires data from the `countrylanguage` table. MySQL joins these three tables together to produce the results.

13. Use the `mysqlslap` command-line utility to execute a test with the following parameters:

- Schema to query against: `world`
- Query file to execute: `/root/scripts/langquery_norm.sql`
- Iterations: 5
- Concurrency: 20 threads
- Verbose output (`-vv` option)

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# mysqlslap -uroot -p \
> --create-schema=world
> --query="/root/scripts/langquery_norm.sql" \
> --iterations=1 \
> --concurrency=5 \
> -vv
```

```
Parsing engines to use.  
Enter password: oracle  
Benchmark  
Average number of seconds to run all queries: 11.973 seconds  
Minimum number of seconds to run all queries: 11.945 seconds  
Maximum number of seconds to run all queries: 12.001 seconds  
Number of clients running queries: 20  
Average number of queries per client: 100
```

- What is the average amount of time each thread takes to complete? **Answer:** Approximately 12 seconds
14. Review your test results. What do they tell you about the effects of normalizing tables on query performance?
- Answer:** In many situations, a normalized table produces the best results, provides greater flexibility, and eliminates duplication of data. However, you might find that for some queries, especially where multiple table joins are required, denormalizing the tables improves performance.
15. Leave the Linux terminal window open and remain logged in as the `root` user. The next practice refers to this terminal window as `t1`.

## Practice 8-2: Choosing the Correct Data Type

### Overview

In this practice, you examine the structure and contents of a table and determine the most effective data types for the character string and date columns.

### Assumptions

- The `schema_test_db.sql` script is available in the `/root/scripts` directory.
- You have the Linux terminal window open and are logged in as the Linux `root` user from the previous practice. This is terminal window `t1` in this practice.

**Note:** This practice uses two Linux terminal windows. The instructions refer to the terminal windows as `t1` and `t2`. To make it easier to identify each terminal, change its title by using the Terminal > Set Title menu option in each terminal window.

### Duration

This practice should take you approximately 15 minutes to complete.

### Tasks

1. In the terminal window from the previous practice (`t1`), start an interactive `mysql` session.
2. Open another Linux terminal window prompt (`t2`) as the Linux `root` user.
3. In terminal window `t1`, create the `schema_test` database. Populate it by sourcing the `/root/scripts/schema_test_db.sql` script.
4. In `t1`, examine the structure of the `emp_resumes` table in the `schema_test` database. What is the data type of the `resume` column?
5. In `t1`, issue a SQL statement to report the minimum, maximum, and average length of the data in the `resume` column. Based on the results, is the data type of the `resume` column a good choice for the data it contains?
6. In `t1`, query the `INFORMATION_SCHEMA` database to report the length of the data in the `emp_resumes` table. Make a note of the result.
7. In terminal window `t2`, issue the `ls -alt` Linux command to display the size of the `emp_resumes` tablespace file on disk. Make a note of the result.
8. In terminal window `t2`, test query performance against the `emp_resumes` table by using the `mysqlslap` command-line utility with the following parameters:
  - Concurrency: 50
  - Iterations: 100
  - Schema: `schema_test`
  - Query: `SELECT * FROM emp_resumes ORDER BY last_name DESC;`
  - Verbose output (`-vv` option)Record the average number of seconds it takes to execute all queries.

9. Execute an `ALTER TABLE` statement at the `mysql` prompt in terminal window `t1` to change the data type of the `resume` column from `CHAR(255)` to `VARCHAR(255)`.
10. Repeat steps 6–8. Explain the results.
11. Issue a `DESCRIBE` statement on the `emp_resumes` table at the `mysql` prompt in terminal window `t1`. Note the data type of the `birth_date` and `hire_date` columns.
12. In `t1`, execute the following query on the `emp_resumes` table. Compare the actual data stored in the `birth_date` and `hire_date` columns with the data types of these columns from the previous step. Are these data types ideal for the data that is being stored?

```
SELECT last_name, birth_date, hire_date FROM emp_resumes LIMIT 10;
```
13. In terminal window `t2`, issue the `ls -alt` Linux command to display the size of the `emp_resumes` tablespace file on disk. Make a note of the result.
14. In `t1`, change the data types of the `birth_date` and `hire_date` columns in the `emp_resumes` table to `DATE`.
15. In `t1`, execute `ANALYZE TABLE` on the `emp_resumes` table. The `ANALYZE TABLE` statement performs a key distribution analysis and stores the distribution for the named table or tables. This helps to reduce storage space and improve I/O efficiency when accessing the table.
16. In terminal window `t2`, issue the `ls -alt` Linux command to display the size of the `emp_resumes` tablespace file on disk. Make a note of the result.
17. Keep both terminal windows (`t1` and `t2`) open for the next practice.

## Solution 8-2: Choosing the Correct Data Type

---

### Solution Steps

1. In the terminal window from the previous practice (t1), start an interactive mysql session.

Enter the following command at the t1 Linux terminal prompt and receive the results shown:

```
# mysql -uroot -p
Enter password: oracle
Welcome to the MySQL monitor. Commands end with ; or \g.
...
mysql>
```

2. Open another Linux terminal window prompt (t2) as the Linux root user.

3. In terminal window t1, create the schema\_test database. Populate it by sourcing the /root/scripts/schema\_test\_db.sql script.

Enter the following statements at the mysql prompt in t1 and receive the results shown:

```
mysql> CREATE DATABASE schema_test;
Query OK, 1 row affected (#.## sec)

mysql> USE schema_test
Database changed
mysql> SOURCE /root/scripts/schema_test_db.sql;
Query OK, 0 rows affected (#.## sec)
...
Query OK, 7877 rows affected (#.## sec)
Records: 7877 Duplicates: 0 Warnings: 0
Query OK, 2123 rows affected (#.## sec)
Records: 2123 Duplicates: 0 Warnings: 0
...
Query OK, 0 rows affected (#.## sec)

mysql>
```

4. In t1, examine the structure of the emp\_resumes table in the schema\_test database. What is the data type of the resume column?

Enter the following statement at the mysql prompt in t1 and receive the results shown:

```
mysql> DESC emp_resumes;
+-----+-----+-----+-----+-----+
| Field | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| emp_no | int(11)       | NO   | PRI | NULL    |       |
| birth_date | datetime     | YES  |      | NULL    |       |
```

```

| first_name | varchar(14) | NO   |       | NULL    |       |
| last_name  | varchar(16)  | NO   |       | NULL    |       |
| gender     | enum('M','F') | NO   |       | NULL    |       |
| hire_date  | datetime     | YES  |       | NULL    |       |
| resume     | char(255)    | YES  |       | NULL    |       |
+-----+-----+-----+-----+
7 rows in set (#.## sec)

```

- What is the data type of the `resume` column? **Answer:** CHAR (255)

5. In t1, issue a SQL statement to report the minimum, maximum, and average length of the data in the `resume` column. Based on the results, is the data type of the `resume` column a good choice for the data it contains?

Enter the following statement at the `mysql` prompt in t1 and receive the results shown:

```

mysql> SELECT MIN(LENGTH(resume)) AS MIN,
      -> MAX(LENGTH(resume)) AS MAX,
      -> AVG(LENGTH(resume)) AS AVG
      -> FROM emp_resumes;
+-----+-----+-----+
| MIN  | MAX  | AVG   |
+-----+-----+-----+
| 5    | 255  | 53.2058 |
+-----+-----+
1 row in set (#.## sec)

```

- Based on the results, is the data type of the `resume` column a good choice for the data it contains? **Answer:** No. There is too much variation in the length of the data for a fixed length data type, which results in wasted disk space.

6. In t1, query the `INFORMATION_SCHEMA` database to report the length of the data in the `emp_resumes` table. Make a note of the result.

Enter the following statement at the `mysql` prompt in t1 and receive the results shown:

```

mysql> SELECT DATA_LENGTH FROM INFORMATION_SCHEMA.TABLES
      -> WHERE TABLE_NAME='emp_resumes';
+-----+
| DATA_LENGTH |
+-----+
| 3686400   |
+-----+
1 row in set (#.## sec)

```

- **Answer:** The data length is 3,686,400 bytes.

7. In terminal window t2, issue the `ls -alt` Linux command to display the size of the `emp_resumes` tablespace file on disk. Make a note of the result.

Enter the following command at the t2 Linux terminal prompt and receive the results shown:

```
# ls -alt /var/lib/mysql/schema_test
total 11292
-rw-rw---- 1 mysql mysql 11534336 Aug 14 04:25 emp_resumes.ibd
drwx----- 2 mysql mysql 4096 Aug 14 04:25 .
-rw-rw---- 1 mysql mysql 8802 Aug 14 04:25 emp_resumes.frm
drwxr-xr-x 13 mysql mysql 4096 Aug 14 04:24 ..
-rw-rw---- 1 mysql mysql 65 Aug 14 04:24 db.opt
```

- **Answer:** The size of the emp\_resumes tablespace file on disk is 11,534,336 bytes.

- In terminal window t2, test query performance against the emp\_resumes table by using the mysqlslap command-line utility with the following parameters:

- Concurrency: 50
- Iterations: 100
- Schema: schema\_test
- Query: SELECT \* FROM emp\_resumes ORDER BY last\_name DESC;
- Verbose output (-vv option)

Record the average number of seconds it takes to execute all queries.

Enter the following command at the t2 Linux terminal prompt and receive the results shown:

```
# mysqlslap -uroot -p \
> --concurrency=50 \
> --iterations=100 \
> --create-schema=schema_test \
> --query="SELECT * FROM emp_resumes ORDER BY last_name DESC;" \
> -vv

Parsing engines to use.
Enter password: oracle
Generating primary key list
Generating primary key list
...
Benchmark

      Average number of seconds to run all queries: 0.178 seconds
      Minimum number of seconds to run all queries: 0.174 seconds
      Maximum number of seconds to run all queries: 0.309 seconds
      Number of clients running queries: 50
      Average number of queries per client: 1
```

- **Answer:** The average query execution time in this example is 0.178 seconds.

- Execute an ALTER TABLE statement at the mysql prompt in terminal window t1 to change the data type of the resume column from CHAR(255) to VARCHAR(255).

Enter the following statement at the mysql prompt in t1 and receive the results shown:

```
mysql> ALTER TABLE emp_resumes
-> MODIFY resume VARCHAR(255) DEFAULT NULL;
Query OK, 10000 rows affected (#.# sec)
```

Records: 10000	Duplicates: 0	Warnings: 0
----------------	---------------	-------------

10. Repeat steps 6–8. Explain the results,

- a. Enter the following statement at the mysql prompt in t1:

```
mysql> SELECT DATA_LENGTH FROM INFORMATION_SCHEMA.TABLES
      -> WHERE TABLE_NAME='emp_resumes';
+-----+
| DATA_LENGTH |
+-----+
|      1589248 |
+-----+
1 row in set (#.# sec)
```

– **Answer:** The data length is 1,589,258 bytes.

- b. Enter the following command at the t2 Linux terminal prompt:

```
# ls -alt /var/lib/mysql/schema_test
total 9240
-rw-rw---- 1 mysql mysql 9437184 Aug 14 06:36 emp_resumes.ibd
drwx----- 2 mysql mysql    4096 Aug 14 06:36 .
-rw-rw---- 1 mysql mysql    8802 Aug 14 06:36 emp_resumes.frm
drwxr-xr-x 13 mysql mysql   4096 Aug 14 04:24 ..
-rw-rw---- 1 mysql mysql     65 Aug 14 04:24 db.opt
```

– **Answer:** The size of the tablespace file on disk is 9,437,184 bytes.

- c. Enter the following command at the t2 Linux terminal prompt:

```
# mysqlslap -uroot -p \
> --concurrency=50 \
> --iterations=100 \
> --create-schema=schema_test \
> --query="SELECT * FROM emp_resumes ORDER BY last_name DESC;" \
> -vv

Parsing engines to use.
Enter password: oracle
Generating primary key list
Generating primary key list
...
Benchmark
Average number of seconds to run all queries: 0.161 seconds
Minimum number of seconds to run all queries: 0.156 seconds
Maximum number of seconds to run all queries: 0.294 seconds
Number of clients running queries: 50
Average number of queries per client: 1
```

– **Answer:** The average query execution time in this example is 0.161 seconds.

- d. Explain the results. **Answer:** By changing the data type of the resume column from a fixed width to variable width character length, you reduce the size of the table and

improve query performance slightly. This performance improvement might be significant on large, busy tables.

- Issue a DESCRIBE statement on the emp\_resumes table at the mysql prompt in terminal window t1. Note the data type of the birth\_date and hire\_date columns.

Enter the following statement at the mysql prompt in t1:

```
mysql> DESCRIBE emp_resumes;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| emp_no | int(11) | NO | PRI | NULL | |
| birth_date | datetime | YES | | NULL | |
| first_name | varchar(14) | NO | | NULL | |
| last_name | varchar(16) | NO | | NULL | |
| gender | enum('M', 'F') | NO | | NULL | |
| hire_date | datetime | YES | | NULL | |
| resume | varchar(255) | YES | | NULL | |
+-----+-----+-----+-----+-----+
7 rows in set (#.# sec)
```

- In t1, execute the following query on the emp\_resumes table. Compare the actual data stored in the birth\_date and hire\_date columns with the data types of these columns from the previous step. Are these data types ideal for the data that is being stored?

```
SELECT last_name, birth_date, hire_date FROM emp_resumes LIMIT 10;
```

- Enter the following statement at the mysql command-line prompt and receive the results shown:

```
mysql> SELECT last_name, birth_date, hire_date
-> FROM emp_resumes LIMIT 10;
+-----+-----+-----+
| last_name | birth_date | hire_date |
+-----+-----+-----+
| Facello | 1953-09-02 00:00:00 | 1986-06-26 00:00:00 |
| Simmel | 1964-06-02 00:00:00 | 1985-11-21 00:00:00 |
| Bamford | 1959-12-03 00:00:00 | 1986-08-28 00:00:00 |
| Koblick | 1954-05-01 00:00:00 | 1986-12-01 00:00:00 |
| Maliniak | 1955-01-21 00:00:00 | 1989-09-12 00:00:00 |
| Preusig | 1953-04-20 00:00:00 | 1989-06-02 00:00:00 |
| Zielinski | 1957-05-23 00:00:00 | 1989-02-10 00:00:00 |
| Kalloufi | 1958-02-19 00:00:00 | 1994-09-15 00:00:00 |
| Peac | 1952-04-19 00:00:00 | 1985-02-18 00:00:00 |
| Piveteau | 1963-06-01 00:00:00 | 1989-08-24 00:00:00 |
+-----+-----+-----+
10 rows in set (#.# sec)
```

- b. Compare the actual data stored in the `birth_date` and `hire_date` columns with the data types of these columns from the previous step. Are these data types ideal for the data that is being stored? **Answer:** No. The `last_name` and `birth_date` columns are both of type `DATETIME`. However, the time portion is unused. The `DATETIME` field requires three bytes for the date portion and up to three extra bytes for the time information, which is wasted.
13. In terminal window `t2`, issue the `ls -alt` Linux command to display the size of the `emp_resumes` tablespace file on disk. Make a note of the result.

Enter the following command at the `t2` Linux terminal prompt and receive the results shown:

```
# ls -alt /var/lib/mysql/schema_test
total 9244
-rw-r---- 1 mysql mysql 9437184 Nov 15 15:47 emp_resumes.ibd
drwxr-x--- 2 mysql mysql     4096 Nov 15 15:46 .
-rw-r---- 1 mysql mysql     8802 Nov 15 15:46 emp_resumes.frm
drwxr-x--- 13 mysql mysql    4096 Nov 15 15:24 ..
-rw-r---- 1 mysql mysql      65 Nov 15 15:24 db.opt
```

- The size of the `emp_resumes` tablespace file is 9,437,184 bytes.

14. In `t1`, change the data types of the `birth_date` and `hire_date` columns in the `emp_resumes` table to `DATE`.

Enter the following statements at the `mysql` prompt in `t1` and receive the results shown:

```
mysql> ALTER TABLE emp_resumes
-> MODIFY birth_date DATE DEFAULT NULL;
Query OK, 10000 rows affected (#.## sec)
Records: 10000  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE emp_resumes
-> MODIFY hire_date DATE DEFAULT NULL;
Query OK, 10000 rows affected (#.## sec)
Records: 10000  Duplicates: 0  Warnings: 0
```

15. In `t1`, execute `ANALYZE TABLE` on the `emp_resumes` table. The `ANALYZE TABLE` statement performs a key distribution analysis and stores the distribution for the named table or tables. This helps to reduce storage space and improve I/O efficiency when accessing the table.

Enter the following statement at the `mysql` prompt in `t1` and receive the results shown:

```
mysql> OPTIMIZE TABLE emp_resumes\G
***** 1. row *****
Table: schema_test.emp_resumes
Op: analyze
Msg_type: status
Msg_text: OK
1 row in set (#.## sec)
```

- **Note:** InnoDB tables do not support OPTIMIZE TABLE. Executing OPTIMIZE TABLE on an InnoDB table re-creates it and performs ANALYZE TABLE instead.
16. In terminal window t2, issue the ls -alt Linux command to display the size of the emp\_resumes tablespace file on disk. Make a note of the result.

Enter the following command at the t2 Linux terminal prompt and receive the results shown:

```
# ls -alt /var/lib/mysql/schema_test
total 7196
-rw-r----- 1 mysql mysql 7340032 Nov 15 15:49 emp_resumes.ibd
drwxr-x--- 2 mysql mysql     4096 Nov 15 15:49 .
-rw-r----- 1 mysql mysql     8802 Nov 15 15:49 emp_resumes.frm
drwxr-x--- 13 mysql mysql    4096 Nov 15 15:24 ..
-rw-r----- 1 mysql mysql      65 Nov 15 15:24 db.opt
```

- The size of the emp\_resumes tablespace file is now 7,340,032 bytes.
17. Keep both terminal windows (t1 and t2) open for the next practice.

## Practice 8-3: Compressing Tables

---

### Overview

In this practice, you compare the size on disk and query performance of an InnoDB table before and after compression.

### Assumptions

- You have a Linux terminal window and a `mysql` command-line session window open from the last practice.
- The `create_comp_tables.sql` script is in the `/root/scripts` directory.

**Note:** This practice uses the two Linux terminal windows from the previous practice. The instructions refer to the terminal windows as `t1` and `t2`. Terminal window `t1` hosts a `mysql` session, and terminal window `t2` is logged in as the Linux `root` user.

### Duration

This practice should take you approximately 20 minutes to complete.

### Tasks

1. In the `mysql` session in terminal window `t1`, execute the `create_compression_table.sql` script in the `/root/scripts` directory. This script creates and populates the `blobs_table` table in the `schema_test` database.
2. Execute a query that displays the first few rows of the `blobs_table` table in the `schema_test` database.
3. Evaluate the `blobs_table` table as a candidate for compression by executing the following query:

```
SELECT AVG(LENGTH(data)) AS Uncompressed,
       AVG(LENGTH(COMPRESS(data))) AS Compressed,
       AVG(LENGTH(data)) / AVG(LENGTH(COMPRESS(data))) AS Ratio
  FROM blobs_table\G
```

Enter the following statement at the `mysql` prompt in `t1` and receive the results shown:

```
mysql> SELECT AVG(LENGTH(data)) AS Uncompressed,
      -> AVG(LENGTH(COMPRESS(data))) AS Compressed,
      -> AVG(LENGTH(data)) / AVG(LENGTH(COMPRESS(data))) AS Ratio
      -> FROM blobs_table\G
***** 1. row *****
Uncompressed: 2563.1180
Compressed: 1274.1760
Ratio: 2.01158867
1 row in set (#.## sec)
```

Is the table a suitable candidate for compression?

4. In the `mysql` session in terminal window `t1`, examine the current settings of the following global system variables and enable them if necessary:
  - `innodb_file_per_table`: Must be enabled for InnoDB table compression
  - `innodb_file_format`: Is set to use the Barracuda file format
  - `innodb_cmp_per_index_enabled`: Must be ON
5. Query the `INNODB_CMP_RESET` table in the `INFORMATION_SCHEMA` database to reset the compression metrics. Verify that the `INNODB_CMP` table counters (the `compress_*` and `uncompress_*` columns) are set to zero.
6. Create three new tables called `blobs_table_cmp_4`, `blobs_table_cmp_8`, and `blobs_table_cmp_16` with the same columns as `blobs_table`. Populate them with rows from the original `blobs_table`.
7. Execute `ALTER TABLE` statements on each of the new tables to apply compression with the following values for `KEY_BLOCK_SIZE`:
  - `blobs_table_cmp_4`: `KEY_BLOCK_SIZE=4`
  - `blobs_table_cmp_8`: `KEY_BLOCK_SIZE=8` (the default value)
  - `blobs_table_cmp_16`: `KEY_BLOCK_SIZE=16`
8. In terminal window `t2`, as the Linux root user, enter a command to view the sizes of the compressed tables on disk, and compare their size to the original `blobs_table` table. Make a note of the size of each table.
9. In `t1`, test the compression efficiency of each of the tables by comparing the proportion of successful compression operations (`compress_ops_ok`) to all compression operations (`compress_ops`) in the `INFORMATION_SCHEMA.INNODB_CMP` table.
10. In `t2`, test the query performance of the `blobs_table`, `blobs_table_cmp_4`, `blobs_table_cmp_8`, and `blobs_table_cmp_16` tables by using the `mysqlslap` command-line client at the `t2` Linux terminal prompt with the following test parameters:
  - Concurrency: 50
  - Iterations: 100
  - Schema: `schema_test`
  - Query: `SELECT * FROM table name;`Which table compression setting provides optimal performance, size on disk, and query efficiency?
11. Close terminal window `t1`. Leave terminal window `t2` (where you are logged in as the Linux root user) open for the next practice.

## Solution 8-3: Compressing Tables

---

### Solution Steps

1. In the mysql session in terminal window t1, execute the `create_compression_table.sql` script in the `/root/scripts` directory. This script creates and populates the `blobs_table` table in the `schema_test` database.

Enter the following statement at the mysql prompt in t1 and receive the results shown:

```
mysql> SOURCE /root/scripts/create_compression_table.sql
Database changed
Query OK, 0 rows affected, 1 warning (#.## sec)

Query OK, 0 rows affected (#.## sec)

Query OK, 0 rows affected, 1 warning (#.## sec)

Query OK, 0 rows affected (#.## sec)

Query OK, 1 row affected (#.## sec)
```

2. Execute a query that displays the first few rows of the `blobs_table` table in the `schema_test` database.

Enter the following statements at the mysql prompt in t1 and receive the results shown:

```
mysql> USE schema_test
...
Database changed
mysql> SELECT * FROM blobs_table LIMIT 3\G
***** 1. row *****
   id: 1
  data: ♦♦♦♦e♦♦♦♦g#####<<♦♦♦♦999944444
...
```

- The `blobs_table` table contains two columns: an auto-incrementing integer ID and a `data` column that contains binary data in blob format.

3. Evaluate the `blobs_table` table as a candidate for compression by executing the following query:

```
SELECT AVG(LENGTH(data)) AS Uncompressed,
       AVG(LENGTH(COMPRESS(data))) AS Compressed,
       AVG(LENGTH(data)) / AVG(LENGTH(COMPRESS(data))) AS Ratio
  FROM blobs_table\G
```

Enter the following statement at the mysql prompt in t1 and receive the results shown:

```
mysql> SELECT AVG(LENGTH(data)) AS Uncompressed,
-> AVG(LENGTH(COMPRESS(data))) AS Compressed,
```

```

-> AVG(LENGTH(data))/AVG(LENGTH(COMPRESS(data))) AS Ratio
-> FROM blobs_table\G
***** 1. row *****
Uncompressed: 2563.1180
Compressed: 1274.1760
Ratio: 2.01158867
1 row in set (#.## sec)

```

**Note:** The `blobs_table` table is generated randomly, so your results might be different from those shown.

Is the table a suitable candidate for compression? **Answer:** Yes. In the example shown, the compressed version of the table takes up approximately half the space of the uncompressed version.

- In the `mysql` session in terminal window `t1`, examine the current settings of the following global system variables, and enable them if necessary:

- `innodb_file_per_table`: Must be enabled for InnoDB table compression
- `innodb_file_format`: Is set to use the Barracuda file format
- `innodb_cmp_per_index_enabled`: Must be ON

Enter the following statements at the `mysql` prompt in `t1` and receive the results shown:

```

mysql> SHOW GLOBAL VARIABLES LIKE 'innodb_file%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| innodb_file_format | Barracuda |
| innodb_file_format_check | ON |
| innodb_file_format_max | Barracuda |
| innodb_file_per_table | ON |
+-----+-----+
4 rows in set (#.## sec)


```

```

mysql> SHOW GLOBAL VARIABLES LIKE 'innodb_cmp%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| innodb_cmp_per_index_enabled | OFF |
+-----+-----+
1 row in set (#.## sec)

```

- The default settings are:
  - `innodb_file_format=Barracuda`
  - `innodb_file_per_table=ON`
  - `innodb_cmp_per_index_enabled=OFF`

Enable `innodb_cmp_per_index`:

```
mysql> SET GLOBAL innodb_cmp_per_index_enabled=ON;
Query OK, 0 rows affected (#.## sec)
```

- The `innodb_cmp_per_index_enabled` setting enables per-index compression statistics in the `INFORMATION_SCHEMA.INNODB_CMP_PER_INDEX` table.

5. Query the `INNODB_CMP_RESET` table in the `INFORMATION_SCHEMA` database to reset the compression metrics. Verify that the `INNODB_CMP` table counters (the `compress_*` and `uncompress_*` columns) are set to zero.

Enter the following statements at the `mysql` prompt in t1 and receive the results shown:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_CMP_PER_INDEX;
Empty set (#.## sec)

mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_CMP\G
***** 1. row *****
    page_size: 1024
  compress_ops: 0
compress_ops_ok: 0
  compress_time: 0
  uncompress_ops: 0
uncompress_time: 0
...
***** 2. row *****
    page_size: 2048
...
***** 3. row *****
    page_size: 4096
...
***** 4. row *****
    page_size: 8192
...
***** 5. row *****
    page_size: 16384
...
5 rows in set (#.## sec)
```

6. Create three new tables called `blobs_table_cmp_4`, `blobs_table_cmp_8`, and `blobs_table_cmp_16` with the same columns as `blobs_table`. Populate them with rows from the original `blobs_table`.

Enter the following statements at the `mysql` prompt in t1 and receive the results shown:

```
mysql> CREATE TABLE blobs_table_cmp_4 LIKE blobs_table;
Query OK, 0 rows affected (#.## sec)
```

```
mysql> CREATE TABLE blobs_table_cmp_8 LIKE blobs_table;
```

```

Query OK, 0 rows affected (#.## sec)

mysql> CREATE TABLE blobs_table_cmp_16 LIKE blobs_table;
Query OK, 0 rows affected (#.## sec)

mysql> INSERT INTO blobs_table_cmp_4
-> SELECT * FROM blobs_table;
Query OK, 1000 rows affected (#.## sec)
Records: 1000  Duplicates: 0  Warnings: 0

mysql> INSERT INTO blobs_table_cmp_8
-> SELECT * FROM blobs_table;
Query OK, 1000 rows affected (#.## sec)
Records: 1000  Duplicates: 0  Warnings: 0

mysql> INSERT INTO blobs_table_cmp_16
-> SELECT * FROM blobs_table;
Query OK, 1000 rows affected (#.## sec)
Records: 1000  Duplicates: 0  Warnings: 0

```

7. Execute ALTER TABLE statements on each of the new tables to apply compression with the following values for KEY\_BLOCK\_SIZE:

- blobs\_table\_cmp\_4: KEY\_BLOCK\_SIZE=4
- blobs\_table\_cmp\_8: KEY\_BLOCK\_SIZE=8 (the default value)
- blobs\_table\_cmp\_16: KEY\_BLOCK\_SIZE=16

Enter the following statements at the mysql prompt in t1 and receive the results shown:

```

mysql> ALTER TABLE blobs_table_cmp_4
-> ROW_FORMAT=COMPRESSED KEY_BLOCK_SIZE=4;
Query OK, 0 rows affected (#.## sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE blobs_table_cmp_8
-> ROW_FORMAT=COMPRESSED KEY_BLOCK_SIZE=8;
Query OK, 0 rows affected (#.## sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE blobs_table_cmp_16
-> ROW_FORMAT=COMPRESSED KEY_BLOCK_SIZE=16;
Query OK, 0 rows affected (#.## sec)
Records: 0  Duplicates: 0  Warnings: 0

```

8. In terminal window t2, as the Linux root user, enter a command to view the sizes of the compressed tables on disk, and compare their size to the original blobs\_table table. Make a note of the size of each table.

Enter the following commands at the t2 Linux terminal prompt and receive the results shown:

```
# ls -als /var/lib/mysql/schema_test
total 35932
-rw-r---- 1 mysql mysql 11534336 Nov 15 15:56 blobs_table.ibd
-rw-r---- 1 mysql mysql 9437184 Nov 15 16:05 blobs_table_cmp_16.ibd
-rw-r---- 1 mysql mysql 7340032 Nov 15 15:49 emp_resumes.ibd
-rw-r---- 1 mysql mysql 5242880 Nov 15 16:05 blobs_table_cmp_8.ibd
-rw-r---- 1 mysql mysql 3145728 Nov 15 16:05 blobs_table_cmp_4.ibd
-rw-r---- 1 mysql mysql 8802 Nov 15 15:49 emp_resumes.frm
-rw-r---- 1 mysql mysql 8586 Nov 15 16:05 blobs_table_cmp_16.frm
-rw-r---- 1 mysql mysql 8586 Nov 15 16:05 blobs_table_cmp_8.frm
-rw-r---- 1 mysql mysql 8586 Nov 15 16:05 blobs_table_cmp_4.frm
-rw-r---- 1 mysql mysql 8586 Nov 15 15:56 blobs_table.frm
drwxr-x-- 13 mysql mysql 4096 Nov 15 15:24 ..
drwxr-x-- 2 mysql mysql 4096 Nov 15 16:05 .
-rw-r---- 1 mysql mysql 65 Nov 15 15:24 db.opt
```

- The -S argument lists files in descending order of size.

9. In t1, test the compression efficiency of each of the tables by comparing the proportion of successful compression operations (compress\_ops\_ok) to all compression operations (compress\_ops) in the INFORMATION\_SCHEMA.INNODB\_CMP table.

Enter the following statement at the mysql prompt in t1 and receive the results shown:

```
mysql> SELECT page_size,
-> compress_ops_ok/compress_ops AS compress_ratio
-> FROM INFORMATION_SCHEMA.INNODB_CMP;
+-----+-----+
| page_size | compress_ratio |
+-----+-----+
| 1024 | NULL |
| 2048 | NULL |
| 4096 | 1.0000 |
| 8192 | 1.0000 |
| 16384 | 1.0000 |
+-----+-----+
5 rows in set, 2 warnings (0.00 sec)
```

- Each table's compression operations are 100% efficient in the preceding sample output. Your results should demonstrate 100% or nearly 100% efficiency.

10. In t2, test the query performance of the blobs\_table, blobs\_table\_cmp\_4, blobs\_table\_cmp\_8, and blobs\_table\_cmp\_16 tables by using the mysqlslap command-line client at the t2 Linux terminal prompt with the following test parameters:

- Concurrency: 50
- Iterations: 100
- Schema: schema\_test
- Query: `SELECT * FROM table name;`

Enter the following command at the t2 Linux terminal prompt and receive the results shown:

```
# mysqlslap -uroot -p \
> --concurrency=50 \
> --iterations=100 \
> --create-schema=schema_test \
> --query="SELECT * FROM blobs_table;" \
Enter password: oracle
Benchmark
      Average number of seconds to run all queries: 0.041 seconds
      Minimum number of seconds to run all queries: 0.038 seconds
      Maximum number of seconds to run all queries: 0.162 seconds
      Number of clients running queries: 50
      Average number of queries per client: 1
# mysqlslap -uroot -p \
> --concurrency=50 \
> --iterations=100 \
> --create-schema=schema_test \
> --query="SELECT * FROM blobs_table_cmp_4;" \
Enter password: oracle
Benchmark
      Average number of seconds to run all queries: 0.051 seconds
      Minimum number of seconds to run all queries: 0.048 seconds
      Maximum number of seconds to run all queries: 0.192 seconds
      Number of clients running queries: 50
      Average number of queries per client: 1
# mysqlslap -uroot -p \
> --concurrency=50 \
> --iterations=100 \
> --create-schema=schema_test \
> --query="SELECT * FROM blobs_table_cmp_8;" \
Enter password: oracle
Benchmark
      Average number of seconds to run all queries: 0.040 seconds
      Minimum number of seconds to run all queries: 0.037 seconds
      Maximum number of seconds to run all queries: 0.112 seconds
```

```
Number of clients running queries: 50
Average number of queries per client: 1
# mysqlslap -uroot -p \
> --concurrency=50 \
> --iterations=100 \
> --create-schema=schema_test \
> --query="SELECT * FROM blobs_table_cmp_16;""
Benchmark
Average number of seconds to run all queries: 0.049 seconds
Minimum number of seconds to run all queries: 0.047 seconds
Maximum number of seconds to run all queries: 0.135 seconds
Number of clients running queries: 50
Average number of queries per client: 1
```

Which table compression setting provides optimal performance, size on disk, and query efficiency? **Answer:** KEY\_BLOCK\_SIZE=8

11. Close terminal window t1. Leave terminal window t2 (where you are logged in as the Linux root user) open for the next practice.  
Enter the following command at the t1 Linux terminal prompt and receive the results shown:

```
mysql> EXIT
Bye
# exit
```

## Practice 8-4: Partitioning

### Overview

In this practice, you evaluate the effect of partitioning tables on performance. To accomplish this objective, you perform the following tasks:

- Partition an existing table by using range partitioning.
- Use `mysqlslap` to execute queries against both the original and partitioned tables to compare performance.

### Assumptions

- You have a terminal window open from the previous practice and are logged in as the Linux `root` user.
- The `create_city_no_partition.sql` and `create_city_partition.sql` scripts are located in the `/root/scripts` directory.

### Duration

This practice should take you approximately 15 minutes to complete.

### Tasks

1. In the Linux terminal prompt that is open from the previous practice (where you are logged in as the Linux `root` user), restart the MySQL server to restore the default configuration settings.
2. Start an interactive `mysql` session.
3. Change the current database to `schema_test`.
4. Execute `SOURCE` statements on the following script files in the `/root/scripts` directory to create the `city_partition` and `city_no_partition` tables in the `schema_test` database:
  - `create_city_no_partition.sql`
  - `create_city_partition.sql`
5. Execute `SHOW TABLE STATUS` on the `city_no_partition` table. What value does the `Create_options` column in the output contain?
6. Execute `SHOW TABLE STATUS` on the `city_partition` table. What value does the `Create_options` column in the output contain?
7. Compare the `CREATE TABLE` statements used to create the `city_no_partition` and `city_partition` tables. How is the `city_partition` table partitioned? What other differences are there between the tables?
8. Query the `PARTITIONS` table in the `INFORMATION_SCHEMA` database to retrieve the number of rows and the data length for each partition in the `city_partition` table. Record the number of rows for each partition in the “8-4” worksheet of the

Lesson08Practice.ods spreadsheet. The spreadsheet calculates the number of rows in each partition as a percentage of all rows.

9. Execute an EXPLAIN PARTITIONS SELECT statement with each of the following query parameters to estimate the number of rows that MySQL must examine and the partitions involved:

- Query 1: The names of all cities in the `city_partition` table with populations between 5,000 and 10,000
- Query 2: The names of all cities in the `city_partition` table with populations between 7,500 and 1,500,000

Enter the number of rows and the partitions involved into the spreadsheet.

10. Log out of the `mysql` session and become the Linux root user.
11. Compare the query performance of the `city_no_partition` and `city_partition` tables by executing a `mysqlslap` test against each table, with the following parameters:

- `--concurrency: 5`
- `--iterations: 10`
- `--create-schema: schema_test`
- `--query: "SELECT Name, Population FROM table name WHERE Population BETWEEN 5000 and 10000;"`

Note the average time taken to execute all the queries.

12. Compare the query performance of the `city_no_partition` and `city_partition` tables by executing a `mysqlslap` test against each of the tables, with the following parameters:

- `--concurrency: 5`
- `--iterations: 10`
- `--create-schema: schema_test`
- `--query: "SELECT Name, Population FROM table name WHERE Population BETWEEN 7500 AND 1500000"`

Note the average time taken to execute all the queries.

13. Based on your results in steps 11 and 12, what is the effect of partitioning on query performance?
14. Drop the `schema_test` database.
15. Close all open terminal windows.

## Solution 8-4: Partitioning

---

### Solution Steps

1. In the Linux terminal prompt that is open from the previous practice (where you are logged in as the Linux `root` user), restart the MySQL server to restore the default configuration settings.

Enter the following commands at the Linux terminal prompt:

```
# service mysql restart
Shutting down MySQL.... SUCCESS!
Starting MySQL.. SUCCESS!
```

2. Start an interactive `mysql` session.

Enter the following command at the Linux terminal prompt:

```
# mysql -uroot -p
Enter password: oracle
Welcome to the MySQL monitor. Commands end with ; or \g.
...
mysql>
```

3. Change the current database to `schema_test`.

Enter the following statement at the `mysql` prompt:

```
mysql> USE schema_test
Reading table information for completion of table and column
names
You can turn off this feature to get a quicker startup with -A
Database changed
```

4. Execute `SOURCE` statements on the following script files in the `/root/scripts` directory to create the `city_partition` and `city_no_partition` tables in the `schema_test` database:

- `create_city_no_partition.sql`
- `create_city_partition.sql`

Enter the following statements at the `mysql` prompt and receive the results shown:

```
mysql> SOURCE /root/scripts/create_city_no_partition.sql
Query OK, 0 rows affected (0.00 sec)
...
Query OK, 22409 rows affected (0.51 sec)
Records: 22409  Duplicates: 0  Warnings: 0

Query OK, 22289 rows affected (0.55 sec)
Records: 22289  Duplicates: 0  Warnings: 0
...
```

```
mysql> SOURCE /root/scripts/create_city_partition.sql
Query OK, 0 rows affected (0.00 sec)
...
Query OK, 22462 rows affected (0.54 sec)
Records: 22462 Duplicates: 0 Warnings: 0

Query OK, 22289 rows affected (0.47 sec)
Records: 22289 Duplicates: 0 Warnings: 0
...
```

5. Execute `SHOW TABLE STATUS` on the `city_no_partition` table. What value does the `Create_options` column in the output contain?

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> SHOW TABLE STATUS LIKE 'city_no_p%'\G
***** 1. row *****
      Name: city_no_partition
      Engine: InnoDB
      Version: 10
      Row_format: Dynamic
          Rows: 202986
      Avg_row_length: 95
      Data_length: 19415040
      Max_data_length: 0
      Index_length: 4734976
      Data_free: 4194304
      Auto_increment: 204751
      Create_time: 2017-08-21 10:02:09
      Update_time: 2017-08-21 10:02:16
      Check_time: NULL
      Collation: latin1_swedish_ci
      Checksum: NULL
      Create_options:
          Comment:
1 row in set (#.## sec)
```

- **Answer:** The `Create_options` column in the `SHOW TABLE STATUS` output for the `city_no_partitions` table is empty.

6. Execute `SHOW TABLE STATUS` on the `city_partition` table. What value does the `Create_options` column in the output contain?

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> SHOW TABLE STATUS LIKE 'city_p%'\G
***** 1. row *****
```

```

        Name: city_partition
        Engine: InnoDB
        Version: 10
        Row_format: Dynamic
          Rows: 203344
        Avg_row_length: 95
        Data_length: 19447808
      Max_data_length: 0
        Index_length: 7913472
        Data_free: 12582912
      Auto_increment: 204751
        Create_time: 2017-08-21 10:02:25
        Update_time: 2017-08-21 10:02:38
        Check_time: NULL
        Collation: latin1_swedish_ci
        Checksum: NULL
Create_options: partitioned
        Comment:
1 row in set (#.## sec)

```

- **Answer:** The `Create_options` column in the `SHOW TABLE STATUS` output states that the `city_partition` table is partitioned.
7. Compare the `CREATE TABLE` statements used to create the `city_no_partition` and `city_partition` tables. How is the `city_partition` table partitioned? What other differences are there between the tables?

Enter the following statements at the `mysql` prompt and receive the results shown:

```

mysql> SHOW CREATE TABLE city_no_partition\G
***** 1. row *****
      Table: city_no_partition
Create Table: CREATE TABLE `city_no_partition` (
  `ID` int(11) NOT NULL AUTO_INCREMENT,
  `Name` char(35) NOT NULL DEFAULT '',
  `CountryCode` char(3) NOT NULL DEFAULT '',
  `District` char(20) NOT NULL DEFAULT '',
  `Population` int(11) NOT NULL DEFAULT '0',
  PRIMARY KEY (`ID`),
  KEY `CountryCode` (`CountryCode`)
) ENGINE=InnoDB AUTO_INCREMENT=204751 DEFAULT CHARSET=latin1
1 row in set (#.## sec)

mysql> SHOW CREATE TABLE city_partition\G
***** 1. row *****
      Table: city_partition

```

```

Create Table: CREATE TABLE `city_partition` (
    `ID` int(11) NOT NULL AUTO_INCREMENT,
    `Name` char(35) NOT NULL DEFAULT '',
    `CountryCode` char(3) NOT NULL DEFAULT '',
    `District` char(20) NOT NULL DEFAULT '',
    `Population` int(11) NOT NULL DEFAULT '0',
    PRIMARY KEY (`ID`, `Population`),
    KEY `CountryCode` (`CountryCode`)
) ENGINE=InnoDB AUTO_INCREMENT=204751 DEFAULT CHARSET=latin1
/*!50100 PARTITION BY RANGE (Population)
PARTITION p1 VALUES LESS THAN (100000) ENGINE = InnoDB,
PARTITION p2 VALUES LESS THAN (200000) ENGINE = InnoDB,
PARTITION p3 VALUES LESS THAN MAXVALUE ENGINE = InnoDB) */
```

1 row in set (#.## sec)

- **Answer:** The city\_partition table uses range partitioning on the value of the Population column. To support partitioning on the Population column, the Population column must be part of the primary key.
8. Query the PARTITIONS table in the INFORMATION\_SCHEMA database to retrieve the number of rows and the data length for each partition in the city\_partition table. Record the number of rows for each partition in the “8-4” worksheet of the Lesson08Practice.ods spreadsheet. The spreadsheet calculates the number of rows in each partition as a percentage of all rows.

Enter the following statement at the mysql prompt and receive the results shown:

```

mysql> SELECT PARTITION_NAME, TABLE_ROWS, DATA_LENGTH
-> FROM INFORMATION_SCHEMA.PARTITIONS
-> WHERE TABLE_NAME='city_partition';
+-----+-----+-----+
| PARTITION_NAME | TABLE_ROWS | DATA_LENGTH |
+-----+-----+-----+
| p1            |      25302 |     2637824 |
| p2            |      94870 |     8929280 |
| p3            |      82519 |     7880704 |
+-----+-----+-----+
3 rows in set (#.## sec)
```

9. Execute an EXPLAIN PARTITIONS SELECT statement with each of the following query parameters to estimate the number of rows that MySQL must examine and the partitions involved:
- Query 1: The names of all cities in the city\_partition table with populations between 5,000 and 10,000
  - Query 2: The names of all cities in the city\_partition table with populations between 7,500 and 1,500,000

Enter the number of rows and the partitions involved into the spreadsheet.

For query 1, enter the following statement at the mysql prompt and receive the results shown:

```
mysql> EXPLAIN PARTITIONS SELECT Name FROM city_partition
-> WHERE Population BETWEEN 5000 AND 10000\G
***** 1. row *****
      id: 1
      select_type: SIMPLE
      table: city_partition
      partitions: p1
      type: ALL
      possible_keys: NULL
      key: NULL
      key_len: NULL
      ref: NULL
      rows: 25955
      filtered: 11.11
      Extra: Using where
1 row in set, 2 warnings (#.## sec)
```

For query 2, enter the following statement at the mysql prompt and receive the results shown:

```
mysql> EXPLAIN PARTITIONS SELECT Name FROM city_partition
-> WHERE Population BETWEEN 7500 AND 1500000\G
***** 1. row *****
      id: 1
      select_type: SIMPLE
      table: city_partition
      partitions: p1,p2,p3
      type: ALL
      possible_keys: NULL
      key: NULL
      key_len: NULL
      ref: NULL
      rows: 203344
      filtered: 11.11
      Extra: Using where
1 row in set, 2 warnings (#.## sec)
```

- Query 1 involves querying 25,955 rows from a single partition, p1.
- Query 2 involves querying 203,344 rows across three partitions: p1, p2, and p3.
- **Note:** The preceding statements result in a warning message. This is because the generic partitioning handler in the MySQL server is deprecated at version 5.7.17 and will be removed in MySQL 8.0. Subsequently, MySQL expects individual

storage engines to provide their own partitioning handler. InnoDB already has its own handler.

10. Log out of the mysql session and become the Linux root user.

```
mysql> EXIT
Bye
#
```

11. Compare the query performance of the city\_no\_partition and city\_partition tables by executing a mysqlslap test against each table, with the following parameters:

- --concurrency: 5
- --iterations: 10
- --create-schema: schema\_test
- --query: "SELECT Name, Population FROM table name WHERE Population BETWEEN 5000 and 10000;"

Note the average time taken to execute all the queries.

For the city\_no\_partition table test, enter the following command at the Linux terminal prompt and receive the results shown:

```
# mysqlslap -uroot -p \
> --concurrency=5 \
> --iterations=10 \
> --create-schema=schema_test \
> --query="SELECT Name, Population FROM city_no_partition WHERE
Population BETWEEN 5000 AND 10000;"
Enter password: oracle
Benchmark
Average number of seconds to run all queries: 0.073 seconds
Minimum number of seconds to run all queries: 0.073 seconds
Maximum number of seconds to run all queries: 0.074 seconds
Number of clients running queries: 5
Average number of queries per client: 1
```

For the city\_partition table test, enter the following command at the Linux terminal prompt and receive the results shown:

```
# mysqlslap -uroot -p \
> --concurrency=5 \
> --iterations=10 \
> --create-schema=schema_test \
> --query="SELECT Name, Population FROM city_partition WHERE
Population BETWEEN 5000 AND 10000;"
Enter password: oracle
Benchmark
Average number of seconds to run all queries: 0.012 seconds
Minimum number of seconds to run all queries: 0.012 seconds
Maximum number of seconds to run all queries: 0.013 seconds
```

```
Number of clients running queries: 5
Average number of queries per client: 1
```

- The execution time is faster in the partitioned table than in the non-partitioned table. The query examines only a single partition (p1).
12. Compare the query performance of the `city_no_partition` and `city_partition` tables by executing a `mysqlslap` test against each of the tables, with the following parameters:

- `--concurrency: 5`
- `--iterations: 10`
- `--create-schema: schema_test`
- `--query: "SELECT Name, Population FROM table name WHERE Population BETWEEN 7500 AND 1500000"`

Note the average time taken to execute all the queries.

For the `city_no_partition` table test, enter the following command at the Linux terminal prompt and receive the results shown:

```
# mysqlslap -uroot -p \
> --concurrency=5 \
> --iterations=10 \
> --create-schema=schema_test \
> --query="SELECT Name, Population FROM city_no_partition WHERE
Population BETWEEN 7500 AND 1500000;"

Enter password: oracle
Benchmark
Average number of seconds to run all queries: 0.125 seconds
Minimum number of seconds to run all queries: 0.110 seconds
Maximum number of seconds to run all queries: 0.136 seconds
Number of clients running queries: 5
Average number of queries per client: 1
```

For the `city_partition` table test, enter the following command at the Linux terminal prompt and receive the results shown:

```
# mysqlslap -uroot -p \
> --concurrency=5 \
> --iterations=10 \
> --create-schema=schema_test \
> --query="SELECT Name, Population FROM city_partition WHERE
Population BETWEEN 7500 AND 1500000;"

Enter password: oracle
Benchmark
Average number of seconds to run all queries: 0.139 seconds
Minimum number of seconds to run all queries: 0.120 seconds
Maximum number of seconds to run all queries: 0.149 seconds
Number of clients running queries: 5
```

Average number of queries per client: 1

13. Based on your results in steps 11 and 12, what is the effect of partitioning on query performance?

**Answer:** For queries that use data from a single partition, the performance is better than accessing the same data from a non-partitioned table. For queries that must access all partitions to retrieve data, the performance is often worse than accessing the same data from a non-partitioned table.

14. Drop the schema\_test database.

Enter the following command at the Linux terminal prompt:

```
# mysql -uroot -p -e"DROP DATABASE schema_test;"  
Enter password: oracle
```

15. Close all open terminal windows.

## **Practices for Lesson 9: Monitoring Queries**

## Practices for Lesson 9: Overview

---

### Overview

These practices test your ability to identify poorly performing queries by using a range of techniques and tools.

**Note:** The output of the various Linux commands and SQL statements shown in the solution steps for these practices might be different on your system.

### Assumptions

- You have completed the practices for the lesson titled “Performance Tuning Tools.”
- The `perf` database is installed and contains the `city_huge` table.
- The `employees` sample database is installed.

## Practice 9-1: Monitoring Statements

### Overview

In this practice, you execute various SQL statements and observe how those statements affect the counts stored in the statement status variables.

### Duration

This practice should take you approximately 10 minutes to complete.

### Tasks

1. Open a Linux terminal window and use the `mysql` client to connect to the MySQL server.
2. Execute the `FLUSH STATUS` statement to reset the value of the session status variables to zero.
3. Display the value of the `Questions` session status variable. Explain the value of the `Questions` session status variable.
4. List all the session status variables that begin with '`Com_`'. What information do these status variables store?
5. Display the values of the session status variables that correspond to the number of `INSERT`, `DELETE`, and `SELECT` statements executed by the server since the `FLUSH STATUS` statement that you issued in step 2.
6. List all the session status variables beginning with '`Select_`'.
7. Execute the following `INSERT` statement to add a new row to the `employees` table in the `employees` database:

```
INSERT INTO employees (emp_no, birth_date, first_name,
last_name, gender, hire_date)
VALUES(500000, '1969-07-03', 'Marcus', 'Aurellius', 'M', '1999-03-14');
```

8. Display the value of the session status variable that stores a count of the number of `INSERT` statements executed on the server for this session.
9. Delete the row that you added to the `employees` table in step 7.
10. Display the value of the session status variable that stores a count of the number of `DELETE` statements executed on the server for this session.
11. Issue a `SELECT` query that lists all employees with the string '`wic`' in their surname (the `last_name` column) in descending alphabetical order of their first name (the `first_name` column).
12. Display the value of the session status variable that stores a count of the number of `SELECT` statements executed on the server for this session.
13. List all the session status variables beginning with '`Select_`'. Explain the results.
14. Leave the Linux terminal window with the `mysql` client session open for the next practice.

## Solution 9-1: Monitoring Statements

---

### Solution Steps

1. Open a Linux terminal window and use the mysql client to connect to the MySQL server.

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# mysql -uroot -p
Enter password: oracle
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is
...
mysql>
```

2. Execute the FLUSH STATUS statement to reset the value of the session status variables to zero.

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> FLUSH STATUS;
Query OK, 0 rows affected (#.# sec)
```

3. Display the value of the Questions session status variable. Explain the value of the Questions session status variable.

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> SHOW SESSION STATUS LIKE 'Questions';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Questions     | 1      |
+-----+-----+
1 row in set (#.# sec)
```

- **Answer:** The Questions session status variable's value is 1. This is because Questions records the number of statements sent to the server by clients, including the SHOW SESSION STATUS statement itself. It does not include statements executed within stored programs like the Queries status variable does.

4. List all session status variables beginning with 'Com\_'. What information do these status variables store?

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> SHOW SESSION STATUS LIKE 'Com_%';
+-----+-----+
| Variable_name           | Value |
+-----+-----+
| Com_admin_commands      | 0     |
+-----+-----+
```

```

| Com_assign_to_keycache | 0   |
| Com_alter_db          | 0   |
| Com_alter_db_upgrade  | 0   |
| Com_alter_event       | 0   |
| ...                   |     |
| Com_update             | 0   |
| Com_update_multi      | 0   |
| Com_xa_commit          | 0   |
| Com_xa_end             | 0   |
| Com_xa_prepare         | 0   |
| Com_xa_recover         | 0   |
| Com_xa_rollback        | 0   |
| Com_xa_start           | 0   |
| Com_stmt_reprepares   | 0   |
| Compression            | OFF |
+-----+-----+
149 rows in set (#.# sec)

```

- **Answer:** The `Com_*` status variables count the number of times specific statements are executed. For example, `Com_update` counts the number of UPDATE statements executed by the server.
5. Display the values of the session status variables that correspond to the number of INSERT, DELETE, and SELECT statements executed by the server since the FLUSH STATUS statement that you issued in step 2.

Enter the following statements at the mysql prompt and receive the results shown:

```

mysql> SHOW SESSION STATUS LIKE 'Com_insert';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Com_insert    | 0    |
+-----+-----+
1 row in set (#.# sec)

mysql> SHOW SESSION STATUS LIKE 'Com_delete';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Com_delete    | 0    |
+-----+-----+
1 row in set (#.# sec)

mysql> SHOW SESSION STATUS LIKE 'Com_select';
+-----+-----+

```

```

| Variable_name | Value |
+-----+-----+
| Com_select    | 0     |
+-----+-----+
1 row in set (#.# sec)

```

6. List all the session status variables beginning with 'Select\_'.

Enter the following statement at the mysql prompt and receive the results shown:

```

mysql> SHOW SESSION STATUS LIKE 'Select_%';
+-----+-----+
| Variable_name      | Value |
+-----+-----+
| Select_full_join   | 0     |
| Select_full_range_join | 0     |
| Select_range       | 0     |
| Select_range_check | 0     |
| Select_scan        | 0     |
+-----+-----+
5 rows in set (#.# sec)

```

- The 'Select\_\*' status variables provide a more detailed breakdown of the number of SELECT statements represented by Com\_select, including how many SELECT statements involved table joins, how many retrieved rows based on a range of values, and how many performed a full table scan.

7. Execute the following INSERT statement to add a new row to the employees table in the employees database:

```

INSERT INTO employees (emp_no, birth_date, first_name,
last_name, gender, hire_date)
VALUES(500000, '1969-07-03', 'Marcus', 'Aurelius', 'M', '1999-03-14');

```

Enter the following statements at the mysql prompt and receive the results shown:

```

mysql> USE employees
Reading table information ...
Database changed
mysql> INSERT INTO employees
-> (emp_no, birth_date, first_name, last_name,
-> gender, hire_date)
-> VALUES(500000, '1969-07-03', 'Marcus', 'Aurelius',
-> 'M', '1999-03-14');
Query OK, 1 row affected (#.# sec)

```

8. Display the value of the session status variable that stores a count of the number of INSERT statements executed on the server for this session.

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> SHOW SESSION STATUS LIKE 'Com_insert';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Com_insert    | 1     |
+-----+-----+
1 row in set (#.## sec)
```

- The value of `Com_insert` was incremented by one after the server processed the `INSERT` statement that you executed in step 7.

9. Delete the row that you added to the `employees` table in step 7.

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> DELETE FROM employees WHERE last_name='Aurelius';
Query OK, 1 row affected (#.## sec)
```

10. Display the value of the session status variable that stores a count of the number of `DELETE` statements executed on the server for this session.

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> SHOW SESSION STATUS LIKE 'Com_delete';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Com_delete    | 1     |
+-----+-----+
1 row in set (#.## sec)
```

- The value of `Com_delete` was incremented by one after the server processed the `DELETE` statement that you executed in step 9.

11. Issue a `SELECT` query that lists all employees with the string '`wic`' in their surname (the `last_name` column) in descending alphabetical order of their first name (the `first_name` column).

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> SELECT * FROM employees WHERE last_name LIKE '%wic%'
      -> ORDER BY first_name DESC\G
      ...
      emp_no: 431093
      birth_date: 1952-07-09
      first_name: Aamod
      last_name: Wolniewicz
      gender: M
      hire_date: 1989-08-21
***** 1318. row *****
```

```

    emp_no: 101668
birth_date: 1961-10-11
first_name: Aamer
last_name: Warwick
gender: F
hire_date: 1987-10-01
1318 rows in set (#.## sec)

```

12. Display the value of the session status variable that stores a count of the number of SELECT statements executed on the server for this session.

Enter the following statement at the mysql prompt and receive the results shown:

```

mysql> SHOW SESSION STATUS LIKE 'Com_select';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Com_select    | 1     |
+-----+-----+
1 row in set (#.## sec)

```

- The value of Com\_select was incremented by one because of the SELECT query in the preceding step.

13. List all session status variables beginning with 'Select\_'. Explain the results.

```

mysql> SHOW SESSION STATUS LIKE 'Select_%';
+-----+-----+
| Variable_name          | Value |
+-----+-----+
| Select_full_join        | 0     |
| Select_full_range_join | 0     |
| Select_range            | 0     |
| Select_range_check      | 0     |
| Select_scan           | 1     |
+-----+-----+
5 rows in set (#.## sec)

```

- **Answer:** The value of Select\_scan is incremented by one as a result of the SELECT query that you executed in step 11. This tells you that the query had to read every row in the table to generate the results. This is inefficient and suggests that either the last\_name column does not have an index or for some reason, the optimizer cannot use it. If this query executes frequently, it might be a good candidate for optimization. In this instance, the last\_name column does not have an index, as can be seen by executing a SHOW CREATE TABLE statement:

```

mysql> SHOW CREATE TABLE employees\G
***** 1. row *****
Table: employees

```

```
Create Table: CREATE TABLE `employees` (
  `emp_no` int(11) NOT NULL,
  `birth_date` date NOT NULL,
  `first_name` varchar(14) NOT NULL,
  `last_name` varchar(16) NOT NULL,
  `gender` enum('M','F') NOT NULL,
  `hire_date` date NOT NULL,
  PRIMARY KEY (`emp_no`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (#.## sec)
```

- The only index in the employees table is the PRIMARY KEY on the emp\_no column.

14. Leave the Linux terminal window with the mysql client session open for the next practice.

## Practice 9-2: Using the Slow Query Log

### Overview

In this practice, you configure the slow query log to record queries that take longer than a specified duration to execute, and examine the contents of the slow query log by using mysqldumpslow.

### Duration

This practice should take you approximately 10 minutes to complete.

### Tasks

1. Use the `mysql` client session from the preceding practice to inspect the values of the `slow_query_log` and `slow_query_log_file` global server variables. Is the slow query log enabled on this server? Note the default location of the slow query log file.
2. If necessary, enable the slow query log.
3. Determine the current value of the `long_query_time` session variable.
4. Change the value of the `long_query_time` session variable so that MySQL records all queries that take more than half a second to execute in the slow query log.
5. Inspect the current values of the following global session variables:
  - `log_slow_admin_statements`
  - `log_queries_not_using_indexes`

What do these settings tell you about the default behavior of the slow query log?

6. Execute the following `SELECT` query against the `employees` database and note the time it takes to execute:

```
SELECT emp_no, salary FROM salaries
WHERE from_date BETWEEN '1986-01-01' AND '1986-01-07'
ORDER BY from_date, salary;
```
7. Open a new Linux terminal window and use `cat`, `less`, or a similar Linux command to display the contents of the slow query log file. You determined the location of the slow query log file in step 1. Verify that the query that you executed in the previous step is recorded in the slow query log.
8. Large slow query logs can be difficult to parse. Use the `mysqldumpslow` utility with the `-g` argument to locate the query that you executed in step 6. What do you notice about the output of `mysqldumpslow`, when compared to the contents of the slow query log file that you examined in the preceding step?
9. Restart the MySQL server to restore the default server settings.
10. Exit all `mysql` client sessions and leave a single terminal window open with the Linux `root` user logged in for the next practice.

## Solution 9-2: Using the Slow Query Log

---

### Solution Steps

1. Use the mysql client session from the preceding practice to inspect the values of the slow\_query\_log and slow\_query\_log\_file global server variables. Is the slow query log enabled on this server? Note the default location of the slow query log file.

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> SHOW VARIABLES LIKE 'slow_query%';
+-----+-----+
| Variable_name      | Value
+-----+-----+
| slow_query_log     | OFF
| slow_query_log_file | /var/lib/mysql/host_name-slow.log
+-----+-----+
2 rows in set (#.## sec)
```

- Is the slow query log enabled on this server? **Answer:** The slow query log is currently disabled. The default slow query log file is *host\_name-slow.log* in the /var/lib/mysql directory.

2. If necessary, enable the slow query log.

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> SET GLOBAL slow_query_log=ON;
Query OK, 0 rows affected (#.## sec)
```

- **Note:** slow\_query\_log is a global variable. You cannot set it on a per-session basis.

3. Determine the current value of the long\_query\_time session variable.

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> SHOW SESSION VARIABLES LIKE 'long_query_time';
+-----+-----+
| Variable_name      | Value
+-----+-----+
| long_query_time    | 10.000000
+-----+-----+
1 row in set (0.01 sec)1 row in set (#.## sec)
```

- **Answer:** The default value of long\_query\_time is 10 seconds.
- **Note:** You can specify a value for the long\_query\_time system variable for all sessions, or for individual sessions.

4. Change the value of the long\_query\_time session variable so that MySQL records all queries that take more than half a second to execute in the slow query log.

Enter the following statements at the mysql prompt and receive the results shown:

```
mysql> SET SESSION long_query_time=0.5;
Query OK, 0 rows affected (#.## sec)

mysql> SHOW SESSION VARIABLES LIKE 'long_query_time';
+-----+-----+
| Variable_name      | Value   |
+-----+-----+
| long_query_time   | 0.500000 |
+-----+-----+
1 row in set (#.## sec)
```

5. Inspect the current values of the following global session variables:

- log\_slow\_admin\_statements
- log\_queries\_not\_using\_indexes

What do these settings tell you about the default behavior of the slow query log?

Enter the following statements at the mysql prompt and receive the results shown:

```
mysql> SHOW GLOBAL VARIABLES LIKE 'log_slow_admin%';
+-----+-----+
| Variable_name          | Value  |
+-----+-----+
| log_slow_admin_statements | OFF   |
+-----+-----+
1 row in set (#.## sec)

mysql> SHOW GLOBAL VARIABLES LIKE 'log_queries_%';
+-----+-----+
| Variable_name          | Value  |
+-----+-----+
| log_queries_not_using_indexes | OFF   |
+-----+-----+
1 row in set (#.## sec)
```

What do these settings tell you about the default behavior of the slow query log? **Answer:**

- The default value of the log\_slow\_admin\_statements setting is OFF, which means that MySQL will not consider a query as “slow” if the query consists of an administrative statement. Administrative statements include ALTER TABLE, ANALYZE TABLE, CHECK TABLE, CREATE INDEX, DROP INDEX, OPTIMIZE TABLE, and REPAIR TABLE.
- The default value of the log\_queries\_not\_using\_indexes setting is OFF, which means that MySQL will not record queries that do not use indexes in the slow query log, even if they exceed the long\_query\_time threshold.

- **Note:** You cannot specify values for `log_slow_admin_statements` and `log_queries_not_using_indexes` on a per-session basis.
6. Execute the following `SELECT` query against the `employees` database and note the time that it takes to execute:

```
SELECT emp_no, salary FROM salaries
WHERE from_date BETWEEN '1986-01-01' AND '1986-01-07'
ORDER BY from_date, salary;
```

Enter the following statements at the `mysql` prompt and receive the results shown:

```
mysql> USE employees;
...
Database changed
mysql> SELECT emp_no, salary FROM salaries
-> WHERE from_date BETWEEN '1986-01-01' AND '1986-01-07'
-> ORDER BY from_date, salary;
+-----+-----+
| emp_no | salary |
+-----+-----+
| 82240 | 40000 |
| 241962 | 40000 |
...
| 97466 | 91912 |
+-----+-----+
354 rows in set (0.70 sec)
```

- In the preceding example output, the query takes 0.7 seconds to execute, which is above the threshold set in the `long_query_time` session variable. Therefore, it should appear in the slow query log.
7. Open a new Linux terminal window and use `cat`, `less`, or a similar Linux command to display the contents of the slow query log file. You determined the location of the slow query log file in step 1. Verify that the query that you executed in the previous step is recorded in the slow query log.

Enter the following command at a new Linux terminal prompt and receive the results shown:

```
# cat /var/lib/mysql/host_name-slow.log
/usr/local/mysql/bin/mysqld, Version: 5.7.17-enterprise-
commercial-advanced-log (MySQL Enterprise Server - Advanced
Edition (Commercial)). started with:
Tcp port: 3306 Unix socket: /var/lib/mysql/mysql.sock
Time Id Command Argument
# Time: date_and_time
# User@Host: root[root] @ localhost [] Id: 213
# Query_time: 0.699740 Lock_time: 0.000072 Rows_sent: 354
Rows_examined: 2844401
```

```

use employees;
SET timestamp=timestamp;
SELECT emp_no, salary FROM salaries WHERE from_date BETWEEN
'1986-01-01' AND '1986-01-07' ORDER BY from_date, salary;

```

8. Large slow query logs can be difficult to parse. Use the mysqldumpslow utility with the -g argument to locate the query that you executed in step 6. What do you notice about the output of mysqldumpslow, when compared to the contents of the slow query log file that you examined in the preceding step?

Enter the following command at the Linux terminal prompt and receive the results shown:

```

# mysqldumpslow -g 'SELECT emp_no, salary' \
> /var/lib/mysql/host_name-slow.log

Reading mysql slow query log from /var/lib/mysql/edddr73p1-
slow.log
Count: 1 Time=0.00s (0s) Lock=0.00s (0s) Rows=0.0 (0),
0users@0hosts
# Time: N-N-11T10:N:N.625240Z
# User@Host: root[root] @ localhost [] Id: N
# Query_time: N.N Lock_time: N.N Rows_sent: N Rows_examined:
N
use employees;
SET timestamp=N;
SELECT emp_no, salary FROM salaries WHERE from_date BETWEEN
'S' AND 'S' ORDER BY from_date, salary

```

**Note:** mysqldumpslow “abstracts” numerical and string values with N and ‘S’, respectively. This is so that it can group queries that are identical, except for the values that appear in the WHERE clause. You can change this behavior by setting the -a and -n options when executing mysqldumpslow:

- -a: Do not abstract numerical and string values.
- -n: Abstract only numerical values with at least the specified number of digits.

9. Restart the MySQL server to restore the default server settings.

Enter the following command at the Linux terminal prompt and receive the results shown:

```

# service mysql restart
Shutting down MySQL..... SUCCESS!
Starting MySQL... SUCCESS!

```

10. Exit all mysql client sessions and leave a single terminal window open with the Linux root user logged in for the next practice.

## Practice 9-3: Identifying Slow Queries with MySQL Enterprise Monitor Query Analyzer

### Overview

In this practice, you use MySQL Enterprise Monitor Query Analyzer to identify slow queries as possible candidates for optimization.

### Duration

This practice should take you approximately 15 minutes to complete.

### Tasks

1. Use the `mysqlmonitor.sh` helper script at the Linux terminal prompt to check if the MySQL Enterprise Monitor Service Manager is running, and start it if necessary.  
To check the current status of the MySQL Enterprise Monitor Service Manager, execute the following command:

```
# sh /opt/mysql/enterprise/monitor/mysqlmonitorctl.sh status
```

To start the MySQL Enterprise Monitor Service Manager, execute the following command:

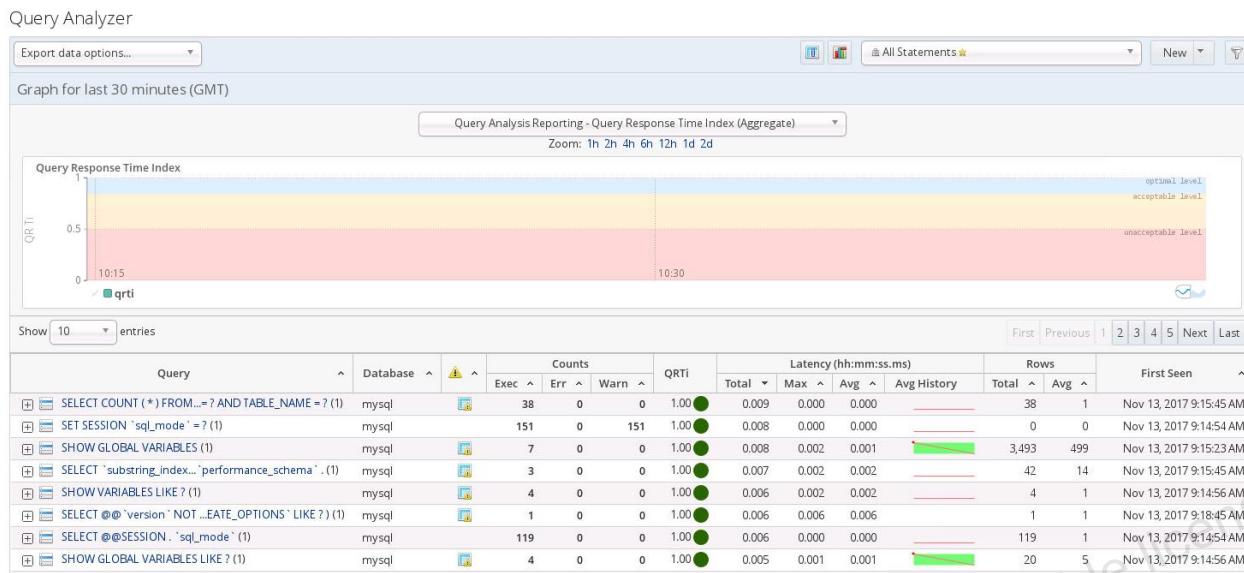
```
# sh /opt/mysql/enterprise/monitor/mysqlmonitorctl.sh start
```

Enter the following commands at the Linux terminal prompt and receive the results shown:

```
# sh /opt/mysql/enterprise/monitor/mysqlmonitorctl.sh status
MySQL Enterprise MySQL is not running
MySQL Enterprise Tomcat is not running

# sh /opt/mysql/enterprise/monitor/mysqlmonitorctl.sh start
Starting mysql service [ OK ]
<date and time> mysqld_safe Logging to
'/opt/mysql/enterprise/monitor/mysql/data/host-name.err'.
<date and time> mysqld_safe Logging to
'/opt/mysql/enterprise/monitor/mysql/data/host-name.err'.
<date and time> mysqld_safe Starting mysqld daemon with
databases from /opt/mysql/enterprise/monitor/mysql/data
Starting tomcat service [ OK ]
```

2. Open MySQL Enterprise Monitor from the Applications > Programming > MySQL Enterprise Monitor Linux desktop menu option.
3. Log in to MySQL Enterprise Monitor as the `monitormanager` user with the password `oracle`. The MySQL Enterprise Monitor Global Overview page displays.
4. Click the Queries link in the menu on the left-hand side of the Global Overview page. The Query Analyzer page appears and shows a graph of all query activity in past 30 minutes and detailed query statistics in the table below it:



**Note:** Many of the queries in the table relate to the mysql system database.

5. From the Refresh drop-down list at the top right-hand corner of the page, change the refresh interval so that the page redisplays every 30 seconds.
6. In the Linux terminal window, start an interactive mysql session.

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# mysql -uroot -p
Enter password: oracle
Welcome to the MySQL monitor. Commands end with ; or \g.
...
mysql>
```

7. Change the current database to perf by entering the following statement at the mysql prompt and receiving the results shown:

```
mysql> USE perf;
Database changed
```

8. Execute the following query against the city\_huge table in the perf database and receive the results shown:

```
mysql> SELECT * FROM city_huge
      -> WHERE Name LIKE '%cos%' AND District LIKE '%ie%';
Empty set (#.# sec)
```

9. Re-issue the query in the preceding step four times more.

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> SELECT * FROM city_huge WHERE Name LIKE '%cos%' AND
      District LIKE '%ie%';
Empty set (#.# sec)
```

```

mysql> SELECT * FROM city_huge WHERE Name LIKE '%cos%' AND
District LIKE '%ie%';
Empty set (#.## sec)

mysql> SELECT * FROM city_huge WHERE Name LIKE '%cos%' AND
District LIKE '%ie%';
Empty set (#.## sec)

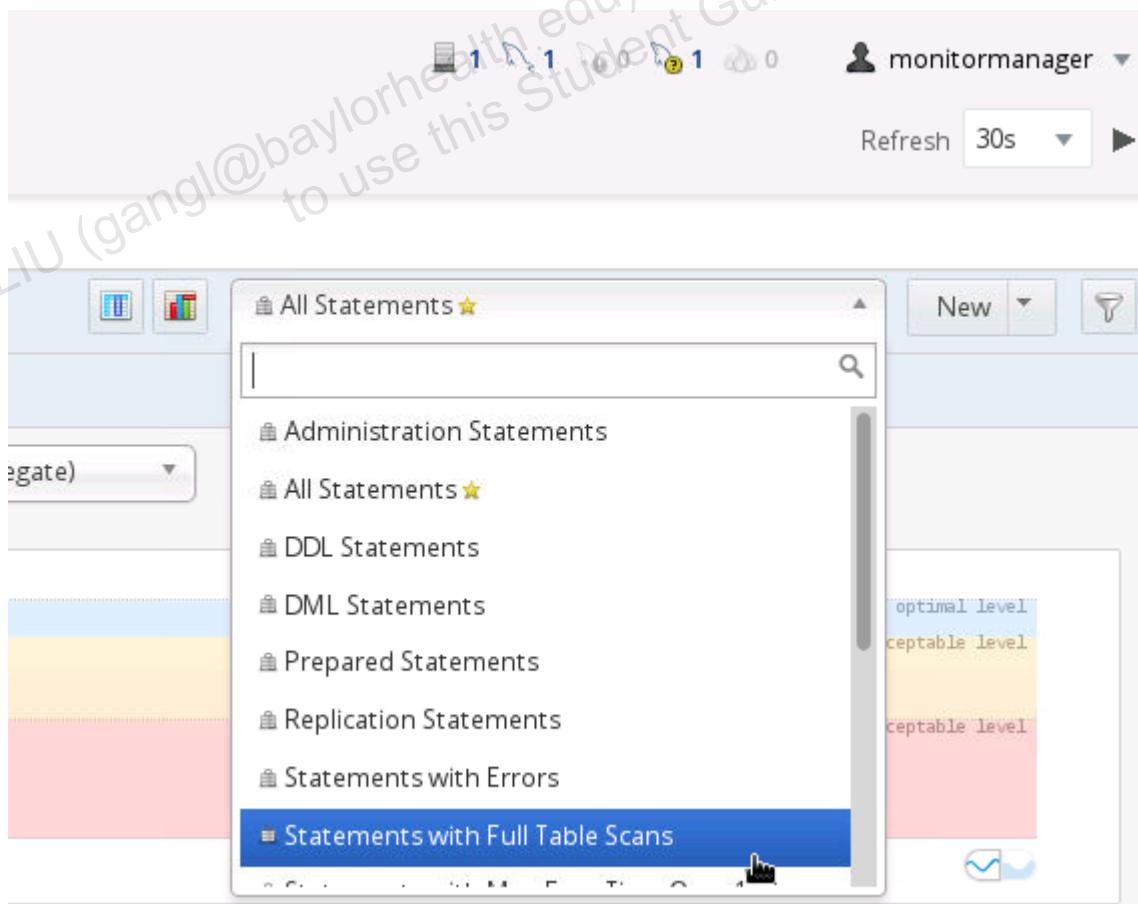
mysql> SELECT * FROM city_huge WHERE Name LIKE '%cos%' AND
District LIKE '%ie%';
Empty set (#.## sec)

```

10. In MySQL Enterprise Monitor, wait for the Query Analyzer page to refresh. When the page refreshes, click the Pause button to disable further refreshes:



11. From the drop-down list of filters that currently displays All Statements, select Statements with Full Table Scans.



12. In the table at the bottom of the Query Analyzer page that displays query statistics, click the Database column heading to display all queries by schema name in reverse alphabetical order. This puts the results from the `perf` database on the first page of the list of query statistics.

Query	Database	Counts	QRTi	Latency (hh:mm:ss.ms)				Rows	First Seen
				Exec	Err	Warn	Total	Max	Avg
SHOW SCHEMAS (1)	perf	1 0 0 1.00	0.000	0.000	0.000	0.000	0.000	15	15 8:22:46 AM
SELECT * FROM `city_hug...` AND `District` LIKE ? (1)	perf	5 0 0 0.00	7.664	5.586	1.533	0	0	0	8:24:57 AM
SHOW TABLES (1)	perf	1 0 0 1.00	0.000	0.000	0.000	0.000	0.000	5	5 8:22:46 AM
SHOW GLOBAL STATUS (2)	mysql	96 0 0 1.00	0.075	0.001	0.001	33,984	354	Mar 30, 2017 4:10:12 PM	
SHOW SCHEMAS LIKE ? (2)	mysql	8 0 0 1.00	0.002	0.015	0.000	4	0	Mar 30, 2017 4:10:12 PM	
SELECT `ROUND` (SUM ( ...ary_by_digest` LIMIT ? (2)	mysql	48 0 0 1.00	0.201	0.009	0.004	48	1	Mar 30, 2017 4:10:13 PM	
SELECT GROUP_CONCAT ( ?...` , `d` . `db` LIMIT ? (2)	mysql	13 0 0 1.00	0.003	0.000	0.000	13	1	Mar 30, 2017 4:10:13 PM	
SELECT `GROUP_CONCAT` ( ?...` , `d` . `db` `LIMIT ? (2)	mysql	13 0 0 1.00	0.005	0.001	0.000	13	1	Mar 30, 2017 4:10:13 PM	
SELECT `digest` AS `dig...` `nolIndexUsedCount` , (2)	mysql	45 0 0 1.00	0.155	0.010	0.003	11,049	246	Mar 30, 2017 4:10:13 PM	
SELECT `plugin_name` FR...ORDER BY `plugin_name` (2)	mysql	53 0 0 1.00	0.038	0.017	0.001	1,643	31	Mar 30, 2017 4:10:12 PM	

Note how the QRTi (Query Response Time Index) value for the query is colored red, indicating a suboptimal query response time. Also, note the execution count (5) and latency statistics for the query.

13. Click the link in the Query column for the `city_huge` query. A dialog box appears and displays the Canonical Query tab.

Canonical Query Example Query Explain Query Graphs

Overview of information collected and aggregated for queries of this form.

**Canonical Form**

[truncated](#) | [full](#) | [formatted](#)

---

```
SELECT
  *
FROM
  `city_huge`
```

**WHERE**

```
NAME LIKE ? AND `District` LIKE ?
```

---

**Execution Time Statistics**

Max Time	Min Time	Avg Time	Total Time	Standard Deviation
5.586	0.505	1.533	7.664	

**Row Statistics**

Max Rows	Min Rows	Avg Rows	Total Rows	Standard Deviation	Total Size	Max Size
		0	0			

**Execution Summary**

Executions	Errors	Warnings	Table Scans	Bad Index Used
5	0	0	5	0

**Time Span**  
From Sep 29, 2017 8:05:57 AM to Sep 29, 2017 8:35:57 AM.

**First Seen**  
Sep 29, 2017 8:24:57 AM

Close

Note the normalized version of the SQL text under “Canonical Form” and the number of full table scans under “Execution Summary.” A full table scan is computationally expensive. If this query executes frequently, you could improve its performance by adding an index.

14. Click the Example Query tab. To see an example (non-normalized) query, you must configure MySQL Enterprise Monitor and Performance Schema:

Canonical Query   Example Query   Explain Query   Graphs

No example query available. Make sure **Enable Example Query** is marked in Query Analysis configuration.

**Configure Query Analysis.** This operation will navigate to the Advisor Configuration page, and the relevant Advisor instances will be selected. Modify the Configuration to enable Query Analysis Example and Explain capture.

If you are using the **Performance Schema**, you need to adjust the server's configuration by running this command:

```
UPDATE performance_schema.setup_consumers SET enabled = 'YES' WHERE
```

To make this change permanent, adjust your */etc/my.cnf* file (location may vary) to include the following line:

```
performance_schema_consumer_events_statements_history_long = ON
```

Please note that the query may also have been removed due to configured table size limitations, look for the *performance\_schema\_events\_statements\_history\_size* system variable on the same file and adjust appropriately.

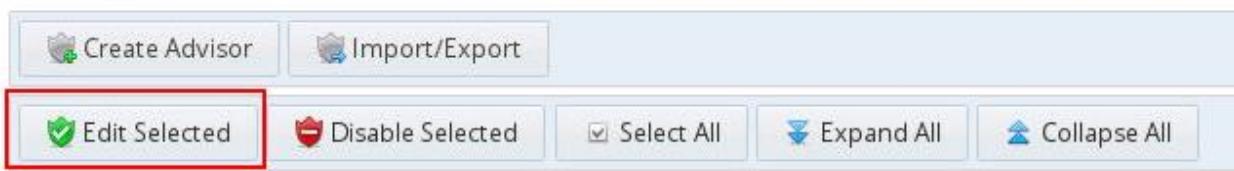
**Close**

- Click the Configure Query Analysis link on the Example Query tab. This takes you to the Manage Advisors page. Ensure that the Query Analysis Reporting advisor check box is selected:

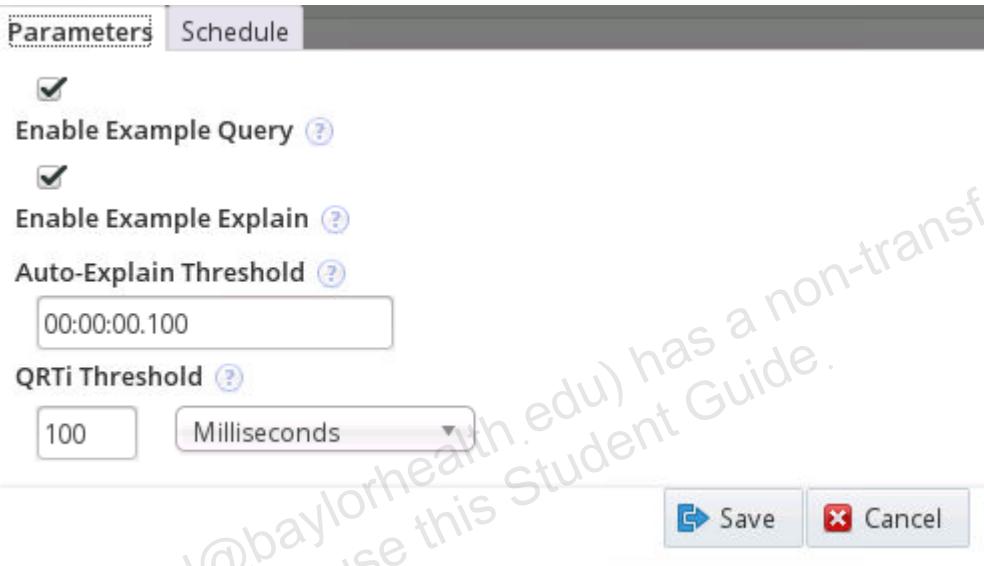
Performance	Count: 22
Query Analysis	Count: 4
<input type="checkbox"/> Item	Info Coverage Schedule Event Handling
<input type="checkbox"/> Average Statement Execution Time Advisor	(?) 100% (1/1)   0  0  0
<input checked="" type="checkbox"/> Query Analysis Reporting	(?) 100% (1/1)   0  0  0
<input type="checkbox"/> Query Pileup Advisor	(?) 100% (1/1)   0  0  0

- With the Query Analysis Reporting advisor check box selected, click the Edit Selected button at the top of the Manage Advisors page:

## Manage Advisors



17. In the dialog box that displays, move the cursor over the Question Mark icons to see detailed information about each entry. The default optimal value for the Query Response Time Index is 100 milliseconds. Query Analyzer considers any query with a QRTi value greater than 100 milliseconds to be suboptimal.



18. Ensure that the Enable Example Query and Enable Example Explain check boxes are selected, and click Save to close the dialog box.
19. At the mysql prompt in the Linux terminal window, list the contents of the Performance Schema setup\_consumers table.

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> SELECT * FROM performance_schema.setup_consumers;
+-----+-----+
| NAME           | ENABLED |
+-----+-----+
| events_stages_current | NO      |
| events_stages_history | NO      |
| events_stages_history_long | NO     |
| events_statements_current | YES    |
| events_statements_history | YES    |
| events_statements_history_long | NO    |
| events_transactions_current | NO    |
| events_transactions_history | NO    |
| events_transactions_history_long | NO    |
```

```

| events_waits_current | NO   |
| events_waits_history | NO   |
| events_waits_history_long | NO  |
| global_instrumentation | YES  |
| thread_instrumentation | YES  |
| statements_digest    | YES  |
+-----+-----+
15 rows in set (#.## sec)

```

- In order to display sample query information, Query Analyzer requires that the `events_statements_history_long` consumer is enabled. It is currently disabled.
20. Enable the `events_statements_history_long` consumer in the Performance Schema by entering the following statement at the `mysql` prompt and receive the results shown:

```

mysql> UPDATE performance_schema.setup_consumers
      -> SET ENABLED = 'YES'
      -> WHERE NAME = 'events_statements_history_long';
Query OK, 1 row affected (#.## sec)
Rows matched: 1  Changed: 1  Warnings: 0

```

21. At the `mysql` prompt, re-issue the query from step 8 and receive the results shown:

```

mysql> SELECT * FROM city_huge
      -> WHERE Name LIKE '%cos%' AND District LIKE '%ie%';
Empty set (#.## sec)

```

22. Return to the Query Analyzer page in MySQL Enterprise Monitor by clicking the Queries menu option on the left-hand side of the Manage Advisors page.
23. From the drop-down list of filters that currently displays All Statements, select Statements with Full Table Scans.
24. Click the link in the Query column for the `city_huge` query. A dialog box appears and displays the Canonical Query tab. Note that the number of executions and table scans are both incremented by one.
25. Click the Example Query tab and note that the full text of the query is now available.

Canonical Query Example Query Explain Query Graphs

The query with the longest execution time during the Time Span (usually the slowest but not always).

**Sampled Query**  
[truncated](#) | [full](#) | [formatted](#)

---

```

SELECT
  *
FROM
  perft.
  city_huge

WHERE
  Name like '%cos%' and District like '%ie%'

```

Execution Time  
600 ms

Date [REDACTED]

User root

Thread ID 583

From Host localhost

To Host

Source Location None found.

Comments None found.

Close

26. Click the Explain Query tab. This tab shows the output of executing EXPLAIN on the query. The EXPLAIN statement enables you to view the MySQL optimizer's execution plan for the query. It tells you that the query uses a full table scan and is unable to use an index. If this query executes frequently, it would benefit from having a usable index on the `city_huge` table.

Canonical Query Example Query Explain Query Graphs

Explain of a query that occurred during the Time Span (usually the slowest but not always).

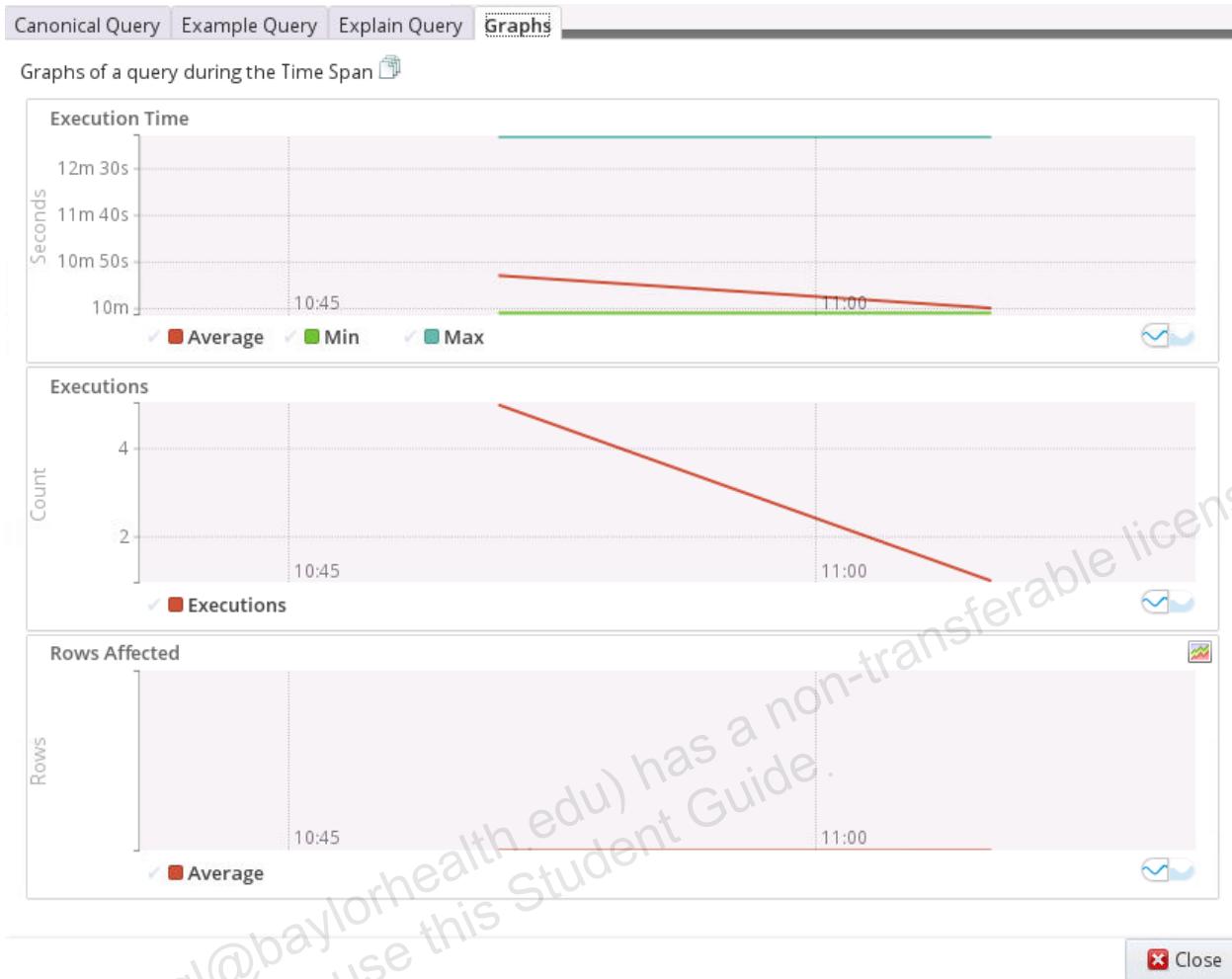
**Explain**

id	select_type	table	type	possible_keys	key	key_len	ref	rows	extra
1	SIMPLE	city_huge	ALL	null	null	0	null	2028428	Using where

Close

**Note:** The EXPLAIN statement is covered in the lesson titled “Optimizing Queries.”

27. Click the Graphs tab. This tab displays the execution metrics for the query in graphical format.



28. Close the dialog box.
29. Close the browser window.
30. Open a new Linux terminal window and stop the MySQL Enterprise Monitor Manager Service by executing the following command:

```
# sh /opt/mysql/enterprise/monitor/mysqlmonitorctl.sh stop
Stopping tomcat service . [ OK ]
Stopping mysql service .. [ OK ]
```

Close the Linux terminal window when the command completes.

31. Keep the mysql command-line client session open for the next practice.

## **Solution 9-3: Identifying Slow Queries with MySQL Enterprise Monitor Query Analyzer**

---

There is no solution for this practice. Follow the practice task instructions.

## Practice 9-4: Identifying Slow Queries with `sys` Views

---

### Overview

In this practice, you query `sys` views to identify queries that might be good candidates for optimization.

### Assumptions

- You have completed the practice titled “Identifying Slow Queries with MySQL Enterprise Monitor Query Analyzer” in this lesson’s practices.
- You have a `mysql` command-line client session open in a Linux terminal window.

### Duration

This practice should take you approximately five minutes to complete.

### Tasks

1. Query the `sys` schema `statements_with_full_table_scans` view for all queries executed against the `perf1` database and locate the query against the `city_huge` table that you performed in the preceding practice.
2. Query the `sys` schema `statements_with_runtimes_in_95th_percentile` view for all queries executed against the `perf1` database and locate the query against the `city_huge` table that you performed in the preceding practice.
3. Query the `sys` schema `statement_analysis` view for all queries executed against the `perf1` database and locate the query against the `city_huge` table that you performed in the preceding practice.
4. Terminate any `mysql` sessions and close all Linux terminal windows.

## Solution 9-4: Identifying Slow Queries with sys Views

### Solution Steps

1. Query the sys schema statements\_with\_full\_table\_scans view for all queries executed against the perf database and locate the query against the city\_huge table that you performed in the preceding practice.

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> SELECT * FROM sys.statements_with_full_table_scans
-> WHERE db='perf'\G
***** 1. row *****
      query: SELECT `IFNULL` ( SUM ( `TRX_L ...
MATION_SCHEMA` . `INNODB_TRX`
      db: perf
      exec_count: 9
      total_latency: 973.06 us
      no_index_used_count: 9
      no_good_index_used_count: 0
      no_index_used_pct: 100
      rows_sent: 9
      rows_examined: 0
      rows_sent_avg: 1
      rows_examined_avg: 0
      first_seen: <date and time>
      last_seen: <date and time>
      digest: 5dc3c8159692bd5d23f983ee54fe15bb
***** 2. row *****
      query: SELECT * FROM `city_huge` WHERE ...
      LIKE ? AND `District` LIKE ?
      db: perf
      exec_count: 6
      total_latency: 8.18 s
      no_index_used_count: 6
      no_good_index_used_count: 0
      no_index_used_pct: 100
      rows_sent: 0
      rows_examined: 12237000
      rows_sent_avg: 0
      rows_examined_avg: 2039500
      first_seen: <date and time>
      last_seen: <date and time>
      digest: 76261dd14d96f3e032a46512cf995395
...
...
```

- Queries that result in full table scans and execute frequently are usually good candidates for optimization.
2. Query the sys schema statements\_with\_runtimes\_in\_95th\_percentile view for all queries executed against the perf database and locate the query against the city\_huge table that you performed in the preceding practice.

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> SELECT * FROM statements_with_runtimes_in_95th_percentile
-> WHERE db='perf'\G
*****
query: SELECT * FROM `city_huge` WHER ... LIKE ?
AND `District` LIKE ?
db: perf
full_scan: *
exec_count: 6
err_count: 0
warn_count: 0
total_latency: 8.18 s
max_latency: 5.59 s
avg_latency: 1.36 s
rows_sent: 0
rows_sent_avg: 0
rows_examined: 12237000
rows_examined_avg: 2039500
first_seen: <date and time>
last_seen: <date and time>
digest: 76261dd14d96f3e032a46512cf995395
...
3 rows in set (#.## sec)
```

- The statements\_with\_runtimes\_in\_95th\_percentile view shows the slowest of all queries running on the system and is a good place to identify possible candidates for optimization.
3. Query the sys schema statement\_analysis view for all queries executed against the perf database and locate the query against the city\_huge table that you performed in the preceding practice.

```
mysql> SELECT * FROM sys.statement_analysis
-> WHERE db='perf'\G
...
*****
query: SELECT * FROM `city_huge` WHER ... LIKE ?
AND `District` LIKE ?
db: perf
full_scan: *
```

```
exec_count: 6
err_count: 0
warn_count: 0
total_latency: 8.18 s
max_latency: 5.59 s
avg_latency: 1.36 s
lock_latency: 523.00 us
rows_sent: 0
rows_sent_avg: 0
rows_examined: 12237000
rows_examined_avg: 2039500
rows_affected: 0
rows_affected_avg: 0
tmp_tables: 0
tmp_disk_tables: 0
rows_sorted: 0
sort_merge_passes: 0
digest: 76261dd14d96f3e032a46512cf995395
first_seen: <date and time>
last_seen: <date and time>
5 rows in set (#.## sec)
```

- The statement\_analysis view provides similar information to MySQL Enterprise Monitor Query Analyzer and is a good resource for identifying slow queries.
4. Terminate any mysql sessions and close all Linux terminal windows.

## Practice 9-5: Using MySQL Workbench Query Statistics

### Overview

In this practice, you enable Query Statistics in MySQL Workbench so that you can view performance metrics for individual queries.

### Assumptions

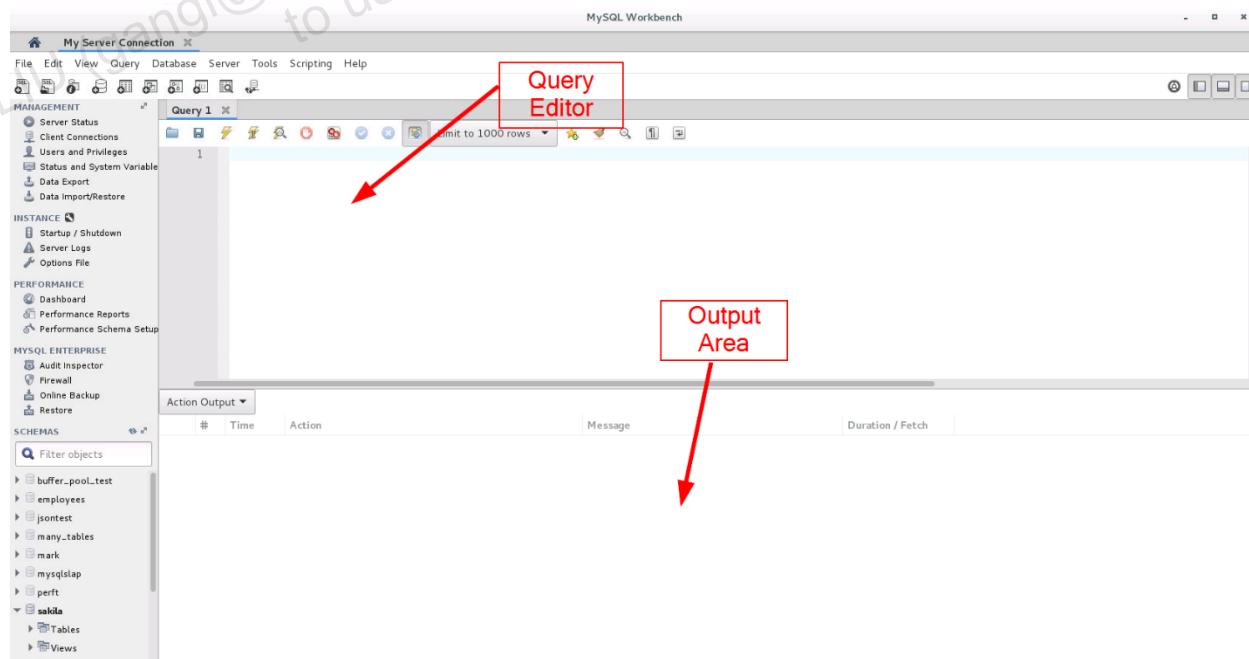
You have created a connection to the MySQL server in MySQL Workbench, as instructed in the practice titled “Using MySQL Workbench for Performance Schema Configuration, Monitoring, and Reporting” in the lesson titled “Performance Schema.”

### Duration

This practice should take you approximately 10 minutes to complete.

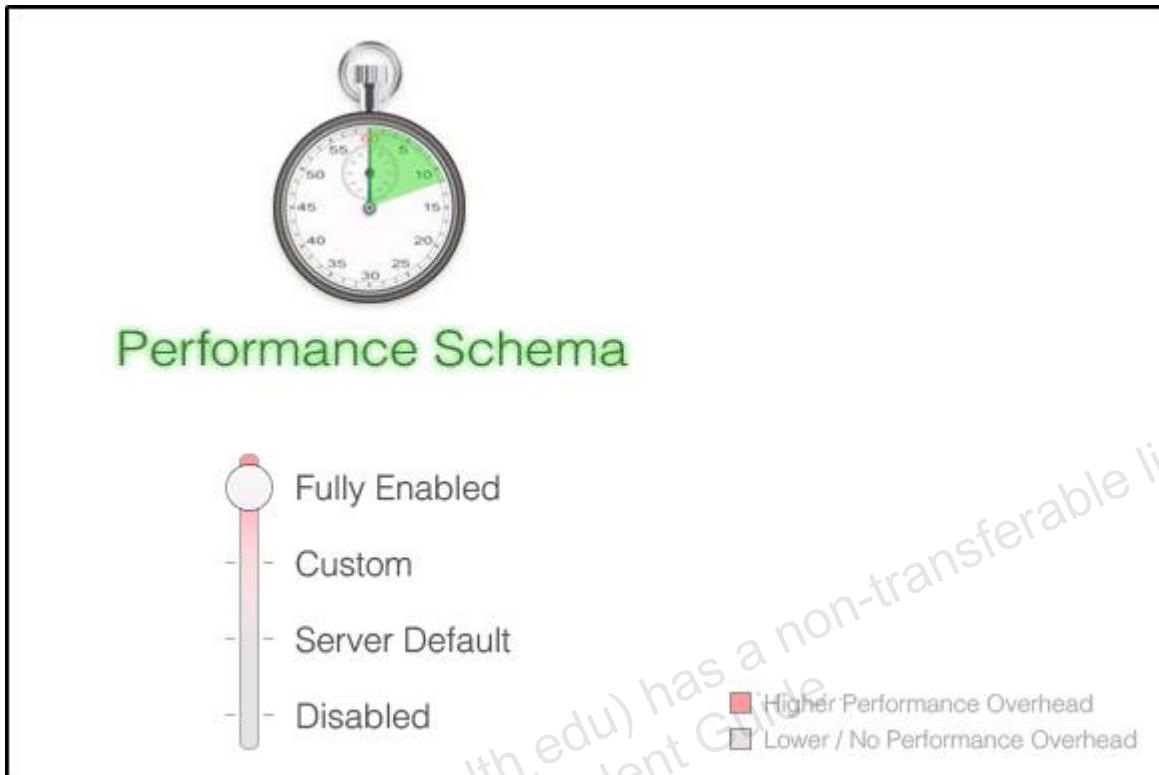
### Tasks

1. In the Linux desktop, open MySQL Workbench by selecting the Applications > Programming > MySQL Workbench menu option.
2. On the MySQL Workbench home page, click the My Server Connection link to establish a connection with the MySQL server.  
MySQL Workbench connects to the MySQL server and displays the Query 1 tab.
3. Ensure that you can see the output area at the bottom of the query tab. Expand this area if necessary. If the output area does not appear at all, make it visible by setting the View > Panels > Show Output Area menu option.

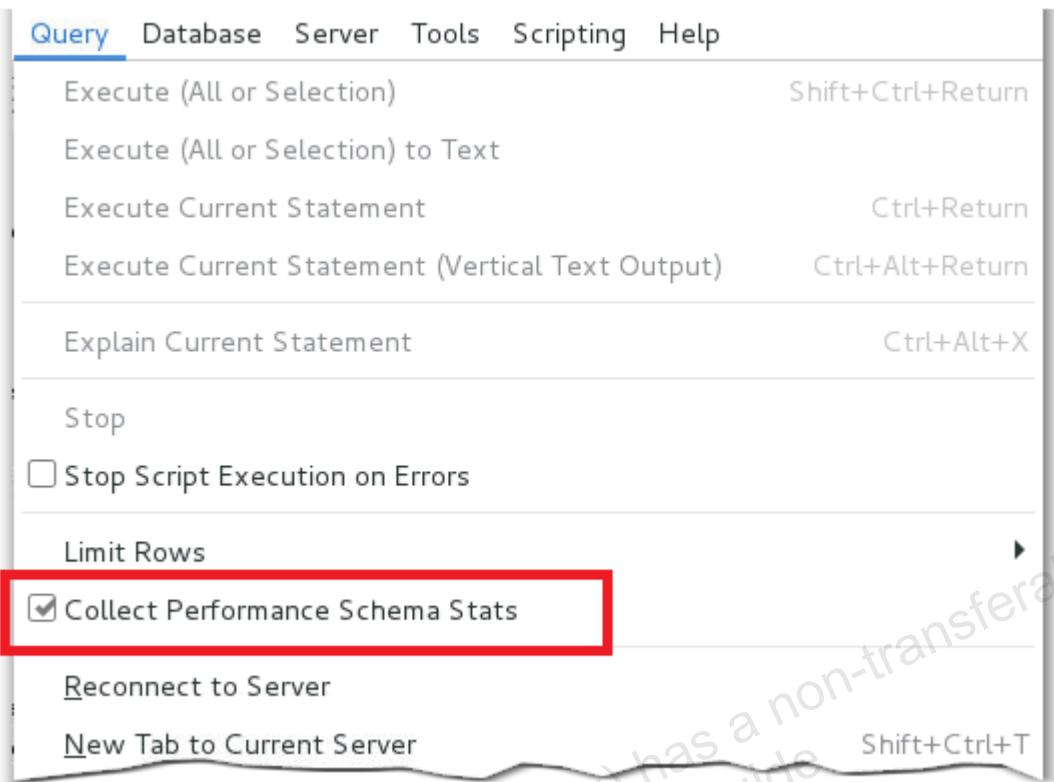


4. In the left-hand pane, under Performance, click Performance Schema Setup. The Performance Schema Easy Setup page displays.

5. Drag the slider in the middle of the page to Fully Enabled. In the warning dialog box that appears, click Enable Everything.



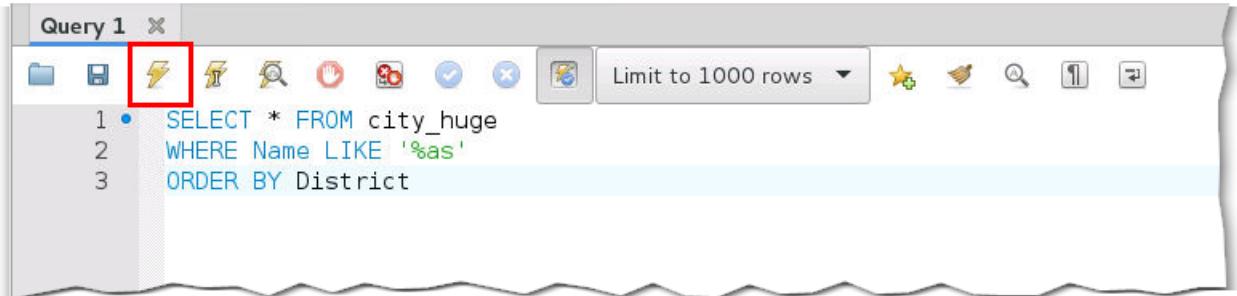
6. In the MySQL Workbench Query menu, ensure that the Collect Performance Schema Stats check box is selected:



7. Close the Performance Schema Setup tab.
8. In the left-hand pane, under Schemas, right-click the `perf1` database and select “Set as Default Schema.”
9. In the Query 1 tab editor, enter the following query:

```
SELECT * FROM city_huge
WHERE Name LIKE '%as%'
ORDER BY District
```

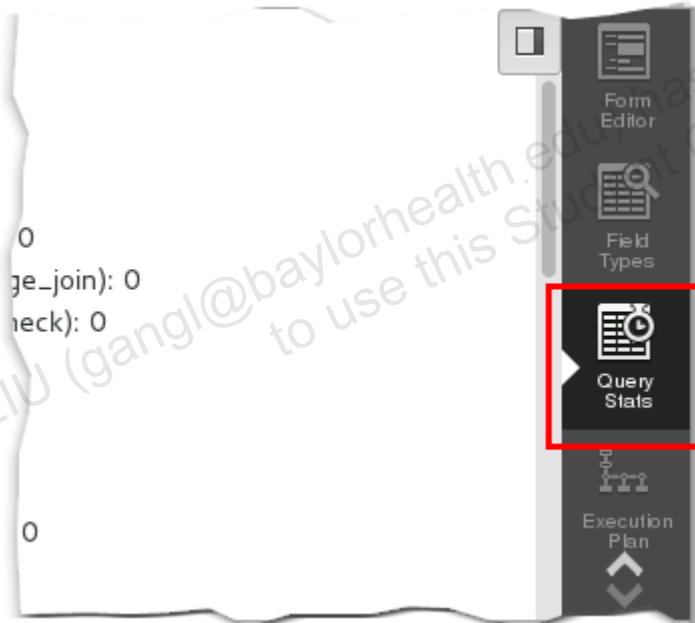
10. Execute the query by clicking the Lightning Bolt icon:



11. When the query completes, the Result Grid displays. Expand this area so that you can see it properly.

The screenshot shows the MySQL Workbench interface. On the left, the 'My Server Connection' sidebar lists various database management options like 'Users and Privileges', 'Status and System Variable', and 'Performance Schema Setup'. The main area is titled 'Query 1' and contains a SQL query: 'SELECT \* FROM city\_huge'. Below the query is a 'Result Grid' table with columns: #, ID, Name, CountryCode, District, and Population. Six rows of data are shown, all from the 'Andalusia' district. To the right of the result grid is a 'city\_huge 2' tab showing 'Action Output' with two log entries. The first entry is '1 11:55:39 SELECT \* FROM city\_huge WHERE Name LIKE "%as" ORDE...' and the second is '2 11:56:40 SELECT \* FROM city\_huge WHERE Name LIKE "%as" ORDE...'. Both entries show '1000 row(s) returned' and execution times of '0.621 sec / 0.0003...' and '0.607 sec / 0.0003...' respectively. A vertical toolbar on the right side includes icons for 'Result Grid', 'Form Editor', 'Field Types', and 'Query Stats' (which is highlighted with a red box).

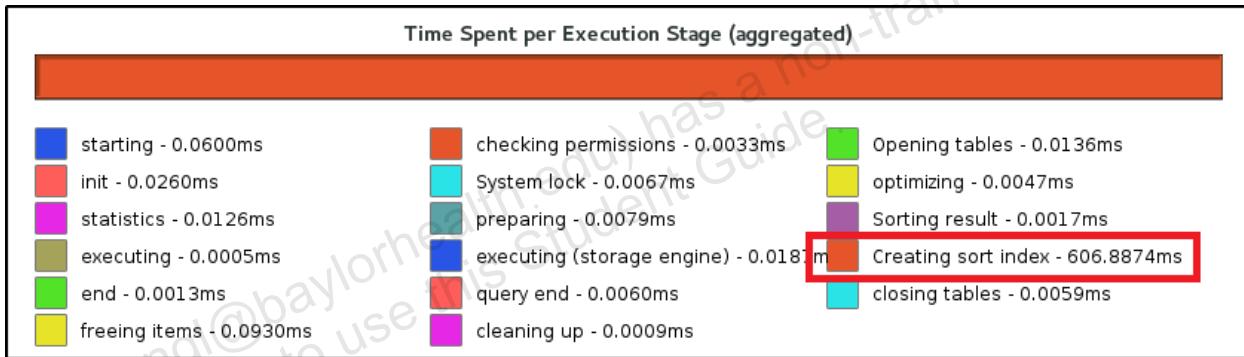
12. Change the output type from Results Grid to Query Statistics:



13. Examine the statistics for the query that you executed in step 10. Note the number of rows examined and the use of full table scans.

Query Statistics	
<b>Timing (as measured at client side):</b>	<b>Joins per Type:</b>
Execution time: 0:00:0.60685515	Full table scans (Select_scan): 1
<b>Timing (as measured by the server):</b>	Joins using table scans (Select_full_join): 0
Execution time: 0:00:0.60715094	Joins using range search (Select_full_range_join): 0
Table lock wait time: 0:00:0.00010500	Joins with range checks (Select_range_check): 0
<b>Errors:</b>	Joins using range (Select_range): 0
Had Errors: NO	
Warnings: 0	
<b>Rows Processed:</b>	<b>Sorting:</b>
Rows affected: 0	Sorted rows (Sort_rows): 1000
Rows sent to client: 1000	Sort merge passes (Sort_merge_passes): 0
<b>Rows examined:</b> 2040500	Sorts with ranges (Sort_range): 0
	Sorts with table scans (Sort_scan): 1
<b>Temporary Tables:</b>	<b>Index Usage:</b>
Temporary disk tables created: 0	No Index used
Temporary tables created: 0	
	<b>Other Info:</b>
	Event Id: 88110
	Thread Id: 3257

14. Scroll down to view a graphical display of the duration of each execution stage. Note the overhead involved in creating a temporary index for sorting the query results by District.



15. Close MySQL Workbench.

## **Solution 9-5: Using MySQL Workbench Query Statistics**

---

### **Overview**

There is no solution for this practice. View the practice task instructions.

GANG LIU (gangli@baylorhealth.edu) has a non-transferable license  
to use this Student Guide.

## **Practices for Lesson 10: Optimizing Queries**

## Practices for Lesson 10: Overview

---

### Overview

These practices test your ability to determine the optimizer's execution plan for a query and to improve the performance of slow queries.

**Note:** The output of the various Linux commands and SQL statements shown in the solution steps for these practices might be different on your system.

### Assumptions

- The world database is installed and populated.
- You have installed the JSON-DataView Mozilla Firefox extension in the “Course Environment Overview” practice for the lesson titled “Introduction.”

## Practice 10-1: Understanding the Query Execution Plan with the EXPLAIN Statement

### Overview

In this practice, you examine the MySQL optimizer's execution plan for several different SELECT queries by using the EXPLAIN command.

To accomplish this, you perform the following tasks:

- Determine the optimizer's strategy for a particular query.
- Add indexes to the tables that the query accesses and evaluate the effect on the query execution plan.

**Note:** The practice steps include instructions to remove the indexes that you add during the practice, to restore the original state of the world database.

### Duration

This practice should take you approximately 30 minutes to complete.

### Tasks

1. Open a Linux terminal window and start an interactive mysql session.
2. Make world the current database.
3. Execute the following statement to view the EXPLAIN output for a query against the city table:

```
EXPLAIN SELECT * FROM city WHERE ID=3803\G
```

  - What is the select type for this query?
  - What is the access method for this query?
  - How many rows must MySQL read to perform this query?
  - Does this access strategy result in good performance? Why?
4. Execute the following statement to view the EXPLAIN output for a query against the city table that uses an expression to determine the value of the ID field to search for:

```
EXPLAIN SELECT * FROM city WHERE ID=3800+3\G
```

Is there any difference in the execution plan for this query, when compared to the query that you looked at in step 3? Why?

5. Execute the following statement that uses a different expression to determine the value of the ID field to search for:

```
EXPLAIN SELECT * FROM city WHERE ID-3=3800\G
```

Is there any difference in the execution plan for this query, when compared to the query that you looked at in step 4? Why?

6. Execute the following statement to view the EXPLAIN output for a query that joins the country and countrylanguage tables:

```
EXPLAIN SELECT country.Name FROM country, countrylanguage
```

```
WHERE country.Code = countrylanguage.CountryCode
AND countrylanguage.Language = 'English' \G
```

- Explain the join type for the first table in the join.
  - Explain the join type for the second table in the join.
7. Execute the following statement to view the EXPLAIN output for a query against the `city` table:
- ```
EXPLAIN SELECT Name FROM City WHERE District='California' \G
```
- What is the access method for this query?
  - Does this access strategy result in good performance? Why?
8. Display the optimizer cost for the query in the preceding step by viewing the extended EXPLAIN information that is available in JSON format.
- What cost did the optimizer calculate for this query?
  - Which columns must MySQL access to resolve the query?
9. Create a covering index called `cov1` that indexes the appropriate columns in the `city` table so that the query that you examined in the previous step executes without accessing the table rows.
10. Re-issue the EXPLAIN statement from step 7. Explain the difference in the output.
11. Display the optimizer cost for the query in the preceding step. What is the effect of adding the covering index on the query cost?
12. Drop the `cov1` index from the `city` table.
13. Add an index on the `IndepYear` column of the `country` table.
14. Execute the following statement to view the EXPLAIN output for a query against the `country` table:

```
EXPLAIN SELECT * FROM Country
WHERE IndepYear = 1905 OR IndepYear IS NULL \G
```

- What is the access method for this query?
  - Does this access strategy result in good performance? Why?
  - What other optimizations does MySQL use for this query?
15. Drop the index on the `IndepYear` column of the `country` table to restore the original structure of the `country` table.
16. Execute the following statement to view the EXPLAIN output for a query against the `Country` table:

```
EXPLAIN SELECT * FROM city
WHERE ID = 50 OR District = 'Michigan' \G;
```

- What is the access method for this query?
  - Does this access strategy result in good performance? Why?
17. Drop the index on the `District` column of the `city` table.
18. Execute the following statement to view the EXPLAIN output for a query against the `country` and `city` tables that uses a subquery in the WHERE clause:

```
EXPLAIN SELECT * FROM city
WHERE CountryCode IN (
    SELECT Code FROM country WHERE Code LIKE 'USA'
) \G
```

- What is the join type for the first table in the join?
  - What is the select type for the second table in the join? Why?
  - What is the join type for the second table in the join?
19. Execute the following statement to view the EXPLAIN output for a query against the `country` and `city` tables that uses a different subquery in the WHERE clause:

```
EXPLAIN SELECT * FROM country
WHERE country.Population < (
    SELECT MAX(city.Population) FROM city
) \G
```

- What is the select type for the first table in the join?
  - What is the select type for the second table in the join?
  - Why is the select type for the second table in this join different from the one that MySQL uses for the second table in the preceding step?
20. Add an index on the `Population` column of the `city` table.

21. Execute the following statement to view the EXPLAIN output for a query against the `city` table:

```
EXPLAIN SELECT * FROM city WHERE Population > 1000000 \G
```

- What is the access method for this query?
  - Does this access strategy result in good performance? Why?
22. Remove the index on the `Population` column of the `city` table to restore the original structure of the table.
23. Execute the following statement to view the EXPLAIN output for a query against the `city` table:

```
EXPLAIN SELECT ID FROM city \G
```

- What is the access method for this query?
  - Does this access strategy result in good performance? Why?
24. Execute the following statement to view the EXPLAIN output for a query against the `city` table:

```
EXPLAIN SELECT * FROM city WHERE Population > 1000000 \G
```

- What is the access method for this query?
  - Does this access strategy result in good performance? Why?
25. Leave the terminal window with the `mysql` session open for the next practice.

## Solution 10-1: Understanding the Query Execution Plan with the EXPLAIN Statement

---

### Solution Steps

1. Open a Linux terminal window and start an interactive mysql session.

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# mysql -uroot -p
Enter password: oracle
Welcome to the MySQL monitor. Commands end with ; or \g.
...
mysql>
```

2. Make world the current database.

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> USE world
...
Database changed
mysql>
```

3. Execute the following statement to view the EXPLAIN output for a query against the city table:

```
EXPLAIN SELECT * FROM city WHERE ID=3803\G
```

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> EXPLAIN SELECT * FROM city WHERE ID=3803\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: city
    partitions: NULL
        type: const
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
        ref: const
      rows: 1
     filtered: 100.00
       Extra: NULL
1 row in set, 1 warning (#.# sec)
```

- What is the select type for this query? **Answer:** SIMPLE. This means that the SELECT does not use a UNION or subqueries.
- What is the access method for this query? **Answer:** const
- How many rows must MySQL read to perform this query? **Answer:** 1

- Does this access strategy result in good performance? Why? **Answer:** Yes. The `const` type applies when you compare all parts of a PRIMARY KEY or UNIQUE index to constant values. In this example, the `const` type tells you that the table has at the most one matching row, which MySQL reads at the start of the query. Because there is only one row, the optimizer treats this row's column values as constants and reads them only once.
4. Execute the following statement to view the EXPLAIN output for a query against the `city` table that uses an expression to determine the value of the `ID` field to search for:

```
EXPLAIN SELECT * FROM city WHERE ID=3800+3\G
```

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> EXPLAIN SELECT * FROM city WHERE ID=3803+3\G
*****
   id: 1
select_type: SIMPLE
  table: city
 partitions: NULL
      type: const
possible_keys: PRIMARY
        key: PRIMARY
    key_len: 4
      ref: const
     rows: 1
  filtered: 100.00
    Extra: NULL
1 row in set, 1 warning (#.# sec)
```

- Is there any difference in the execution plan for this query, when compared to the query that you looked at in step 3? Why? **Answer:** No. The optimizer evaluates the expression `3800+3` before executing the query.
5. Execute the following statement that uses a different expression to determine the value of the `ID` field to search for:

```
EXPLAIN SELECT * FROM city WHERE ID-3=3800\G
```

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> EXPLAIN SELECT * FROM city WHERE ID-3=3800\G
*****
   id: 1
select_type: SIMPLE
  table: city
 partitions: NULL
      type: ALL
possible_keys: NULL
        key: NULL
    key_len: NULL
```

```

    ref: NULL
rows: 4188
filtered: 100.00
Extra: Using where
1 row in set, 1 warning (#.## sec)

```

- Is there any difference in the execution plan for this query, when compared to the query that you looked at in step 4? Why? **Answer:** Yes. The optimizer cannot reduce the expression `ID-3 = 3800` to a single value before executing the query and therefore must scan the full table (`type: ALL`). Note that the number of rows is an estimate based on InnoDB statistics and does not correspond to the exact number of rows in the `city` table (4,097).
6. Execute the following statement to view the EXPLAIN output for a query that joins the `country` and `countrylanguage` tables:

```

EXPLAIN SELECT country.Name FROM country, countrylanguage
WHERE country.Code = countrylanguage.CountryCode
AND countrylanguage.Language = 'English'\G

```

Enter the following statement at the mysql prompt and receive the results shown:

```

mysql> EXPLAIN SELECT country.Name FROM country, countrylanguage
-> WHERE country.Code = countrylanguage.CountryCode
-> AND countrylanguage.Language = 'English'\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: countrylanguage
     partitions: NULL
      type: index
possible_keys: PRIMARY,CountryCode
        key: CountryCode
      key_len: 3
         ref: NULL
        rows: 984
    filtered: 10.00
      Extra: Using where; Using index
***** 2. row *****
      id: 1
  select_type: SIMPLE
        table: country
     partitions: NULL
      type: eq_ref
possible_keys: PRIMARY
        key: PRIMARY
      key_len: 3

```

```

    ref: world.countrylanguage.CountryCode
    rows: 1
    filtered: 100.00
    Extra: NULL
2 rows in set, 1 warning (#.## sec)

```

- Explain the join type for the first table in the join. **Answer:** The join type for the first table in the join is `index`. The index on the `CountryCode` column acts as a covering index for the query, as you can see from the `Using index` entry in the `Extra` column. A covering index can retrieve all the required data from the index tree without reading rows from the table.
  - Explain the join type for the second table in the join. **Answer:** The join type for the second table in the join is `eq_ref`. Other than `system` and `const`, this is the best possible join type. MySQL uses the `eq_ref` access method for comparing indexed columns by using the `=` operator. The optimizer uses this join type when the join uses all parts of a `PRIMARY KEY` or `UNIQUE NOT NULL` index.
7. Execute the following statement to view the EXPLAIN output for a query against the `city` table:
- ```
EXPLAIN SELECT Name FROM City WHERE District='California'\G
```
- Enter the following statement at the `mysql` prompt and receive the results shown:
- ```

mysql> EXPLAIN SELECT Name FROM city
      -> WHERE District = 'California'\G
***** 1. row *****
      id: 1
      select_type: SIMPLE
      table: city
      partitions: NULL
      type: ref
      possible_keys: District
      key: District
      key_len: 20
      ref: const
      rows: 68
      filtered: 100.00
      Extra: NULL
1 row in set, 1 warning (#.## sec)

```
- What is the access method for this query? **Answer:** `ref`
  - Does this access strategy result in good performance? Why? **Answer:** Maybe. The `ref` access method indicates that MySQL cannot select a single row based on the key value. Performance is good only if the key used by the optimizer matches only a few rows.
8. Display the optimizer cost for the query in the preceding step by viewing the extended EXPLAIN information that is available in JSON format.

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> EXPLAIN FORMAT=JSON SELECT Name FROM city
-> WHERE District = 'California' \G
***** 1. row *****
EXPLAIN: {
  "query_block": {
    "select_id": 1,
    "cost_info": {
      "query_cost": "81.60"
    },
    "table": {
      "table_name": "city",
      "access_type": "ref",
      "possible_keys": [
        "District"
      ],
      "key": "District",
      "used_key_parts": [
        "District"
      ],
      "key_length": "20",
      "ref": [
        "const"
      ],
      "rows_examined_per_scan": 68,
      "rows_produced_per_join": 68,
      "filtered": "100.00",
      "cost_info": {
        "read_cost": "68.00",
        "eval_cost": "13.60",
        "prefix_cost": "81.60",
        "data_read_per_join": "4K"
      },
      "used_columns": [
        "Name",
        "District"
      ]
    }
  }
}
1 row in set, 1 warning (#.## sec)
```

- What cost did the optimizer calculate for this query? **Answer:** 81.60 (the value of the `query_cost` field in the JSON output. The `cost_info` JSON object displays a more detailed breakdown of the cost.)
  - Which columns must MySQL access to resolve the query? **Answer:** The `Name` and `District` columns (the elements in the JSON `used_columns` array field)
9. Create a covering index called `cov1` that indexes the appropriate columns in the `city` table so that the query that you examined in the previous step executes without accessing the table rows.

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> CREATE INDEX cov1 ON city (District, Name);
Query OK, 0 rows affected (#.## sec)
Records: 0  Duplicates: 0  Warnings: 0
```

10. Re-issue the `EXPLAIN` statement from step 7.

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> EXPLAIN SELECT Name FROM city WHERE District =
'California'\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: city
    partitions: NULL
       type: ref
  possible_keys: District,cov1
        key: cov1
    key_len: 20
       ref: const
      rows: 68
  filtered: 100.00
     Extra: Using index
1 row in set, 1 warning (#.## sec)
```

- Explain the difference in the output. **Answer:** The query now uses the `cov1` index to resolve the query, without having to access the table rows. The output shows this by the presence of the `Using index` entry in the `Extra` column.

11. Display the optimizer cost for the query in the preceding step.

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> EXPLAIN FORMAT=JSON SELECT Name FROM city
-> WHERE District = 'California'\G
***** 1. row *****
EXPLAIN: {
  "query_block": {
    "select_id": 1,
```

```

"cost_info": {
    "query_cost": "15.08"
},
"table": {
    "table_name": "city",
    "access_type": "ref",
    "possible_keys": [
        "District",
        "cov1"
    ],
    "key": "cov1",
    "used_key_parts": [
        "District"
    ],
    "key_length": "20",
    "ref": [
        "const"
    ],
    "rows_examined_per_scan": 68,
    "rows_produced_per_join": 68,
    "filtered": "100.00",
    "using_index": true,
    "cost_info": {
        "read_cost": "1.48",
        "eval_cost": "13.60",
        "prefix_cost": "15.08",
        "data_read_per_join": "4K"
    },
    "used_columns": [
        "Name",
        "District"
    ]
}
}
}

1 row in set, 1 warning (#.## sec)

```

- What is the effect of adding a covering index on the query cost? **Answer:** The cost of the query is significantly lower (15.08 instead of 81.60).
12. Drop the cov1 index from the city table.

Enter the following statement at the mysql prompt and receive the results shown:

```

mysql> ALTER TABLE city DROP INDEX cov1;
Query OK, 0 rows affected (#.## sec)

```

```
Records: 0    Duplicates: 0    Warnings: 0
```

13. Add an index on the `IndepYear` column of the `country` table.

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> ALTER TABLE Country ADD INDEX (IndepYear);
Query OK, 0 rows affected (#.# sec)
Records: 0    Duplicates: 0    Warnings: 0
```

14. Execute the following statement to view the EXPLAIN output for a query against the `country` table:

```
EXPLAIN SELECT * FROM Country
WHERE IndepYear = 1905 OR IndepYear IS NULL\G
```

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> EXPLAIN SELECT * FROM country
-> WHERE IndepYear = 1905 OR IndepYear IS NULL\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: country
    partitions: NULL
        type: ref_or_null
possible_keys: IndepYear
          key: IndepYear
      key_len: 3
        ref: const
      rows: 48
  filtered: 100.00
      Extra: Using index condition
1 row in set, 1 warning (#.# sec)
```

- What is the access method for this query? **Answer:** `ref_or_null`
- Does this access strategy result in good performance? Why? **Answer:** Maybe. The `ref_or_null` access method is similar to `ref`, but MySQL does an extra search for rows that contain null values. Performance is good only if the key that the optimizer uses matches only a few rows.
- What other optimizations does MySQL use for this query? **Answer:** Index condition pushdown. If MySQL can evaluate parts of the `WHERE` condition by using only columns from the index, the MySQL server pushes this part of the `WHERE` condition down to the storage engine. The storage engine then evaluates the pushed index condition by using the index entry and only if this is satisfied is the row read from the table. This optimization can reduce the number of times the storage engine must access the base table and the number of times the MySQL server must access the storage engine.

15. Drop the index on the `IndepYear` column of the `country` table to restore the original structure of the `country` table.

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> DROP INDEX IndepYear ON country;
Query OK, 0 rows affected (#.## sec)
Records: 0  Duplicates: 0  Warnings: 0
```

16. Execute the following statement to view the EXPLAIN output for a query against the Country table:

```
EXPLAIN SELECT * FROM city
WHERE ID = 50 OR District = 'Michigan' \G;
```

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> EXPLAIN SELECT * FROM city
-> WHERE ID = 50 OR District = 'Michigan' \G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: city
    partitions: NULL
        type: index_merge
possible_keys: PRIMARY,District
          key: PRIMARY,District
      key_len: 4,20
        ref: NULL
      rows: 9
  filtered: 100.00
Extra: Using union(PRIMARY,District); Using where
1 row in set, 1 warning (#.## sec)
```

- What is the access method for this query? **Answer:** index\_merge
- Does this access strategy result in good performance? Why? **Answer:** Maybe. The index\_merge access strategy tells you that MySQL has found multiple indexes that it can use for the query. In this case, it uses the primary key and the District index. The index\_merge method retrieves rows with multiple range scans and merges the results. Performance is good if there are high selectivity indexes on the table. The Extra column Using union entry shows how MySQL merges index scans for the index\_merge join type.
  - **Selectivity** is a ratio between the number of values in a column and the number of values that are distinct or unique. High selectivity means that the ratio is approaching 1:1.

17. Drop the index on the District column of the city table.

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> DROP INDEX District ON city;
Query OK, 0 rows affected (#.## sec)
Records: 0  Duplicates: 0  Warnings: 0
```

18. Execute the following statement to view the EXPLAIN output for a query against the country and city tables that uses a subquery in the WHERE clause:

```
EXPLAIN SELECT * FROM city
WHERE CountryCode IN (
    SELECT Code FROM country WHERE Code LIKE 'USA'
) \G
```

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> EXPLAIN SELECT * FROM city
-> WHERE CountryCode IN (
->     SELECT Code FROM country WHERE Code LIKE 'USA'
-> ) \G
*****
      id: 1
  select_type: SIMPLE
        table: country
    partitions: NULL
        type: range
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 3
        ref: NULL
      rows: 1
  filtered: 100.00
    Extra: Using where; Using index
*****
      id: 1
  select_type: SIMPLE
        table: city
    partitions: NULL
        type: ref
possible_keys: CountryCode
          key: CountryCode
      key_len: 3
        ref: world.country.Code
      rows: 18
  filtered: 100.00
    Extra: NULL
2 rows in set, 1 warning (#.## sec)
```

- What is the join type for the first table in the join? **Answer:** range. MySQL uses an index to retrieve only the rows that are in a given range. The `key` column in the output row indicates which index is used. The `key_len` column in the output contains the longest key part that MySQL uses. The `ref` column value is `NULL` for this join type.

- What is the select type for the second table in the join? Why? **Answer:** SIMPLE. You might expect the second table select type to be DEPENDENT SUBQUERY. In earlier versions of MySQL, this was the case. In later versions of MySQL, the optimizer uses semi-join strategies to improve subquery execution. Semi-joins differ from inner joins in that they return only one matching row from the joined table. In this example, the optimizer recognizes that it needs to return only one instance of each CountryCode from the country table. The most important factor for the IN clause is that there is a match, not how many matches there are.
  - You can view the rewritten query by executing SHOW WARNINGS:

```
mysql> SHOW WARNINGS\G
*****
1. row ****
Level: Note
Code: 1003
Message: /* select#1 */ select `world`.`city`.'ID` AS
`ID`, `world`.`city`.'Name` AS
`Name`, `world`.`city`.'CountryCode` AS
`CountryCode`, `world`.`city`.'District` AS
`District`, `world`.`city`.'Population` AS `Population` from
`world`.`country` join `world`.`city` where
((`world`.`city`.'CountryCode` = `world`.`country`.'Code`) and
(`world`.`country`.'Code` like 'USA'))
1 row in set (#.## sec)
```

- What is the join type for the second table in the join? **Answer:** ref. MySQL reads all rows with matching index values from this table for each combination of rows from the previous tables. The ref column in the output shows which columns or constants MySQL compares to the index named in the key column to select rows from the table.
19. Execute the following statement to view the EXPLAIN output for a query against the country and city tables that uses a different subquery in the WHERE clause:

```
EXPLAIN SELECT * FROM country
WHERE country.Population < (
  SELECT MAX(city.Population) FROM city
)\G
```

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> EXPLAIN SELECT * FROM country
-> WHERE country.Population < (
->   SELECT MAX(city.Population) FROM city
-> )\G
*****
1. row ****
id: 1
select_type: PRIMARY
table: country
partitions: NULL
type: ALL
possible_keys: NULL
```

```

        key: NULL
key_len: NULL
      ref: NULL
     rows: 239
  filtered: 33.33
    Extra: Using where
***** 2. row *****
      id: 2
select_type: SUBQUERY
      table: city
     partitions: NULL
       type: ALL
possible_keys: NULL
      key: NULL
key_len: NULL
      ref: NULL
     rows: 4188
  filtered: 100.00
    Extra: NULL
2 rows in set, 1 warning (#.## sec)

```

- What is the select type for the first table in the join? **Answer:** PRIMARY. This corresponds to the outermost SELECT query in the statement.
  - What is the select type for the second table in the join? **Answer:** SUBQUERY
  - Why is the select type for the second table in this join different from the one that MySQL uses for the second table in the preceding step? **Answer:** The optimizer can use the semi-join strategy only when there is an IN or ANY subquery that appears at the top level of the WHERE or ON clause, and where there is no grouping or aggregation. This subquery uses the aggregation function MAX() and therefore cannot use a semi-join.
20. Add an index on the Population column of the city table.

Enter the following statement at the mysql prompt and receive the results shown:

```

mysql> ALTER TABLE city ADD INDEX (Population);
Query OK, 0 rows affected (#.## sec)
Records: 0  Duplicates: 0  Warnings: 0

```

21. Execute the following statement to view the EXPLAIN output for a query against the city table:

```
EXPLAIN SELECT * FROM city WHERE Population > 1000000\G
```

Enter the following statement at the mysql prompt and receive the results shown:

```

mysql> EXPLAIN SELECT * FROM city
      -> WHERE Population > 1000000\G
***** 1. row *****

```

```

        id: 1
select_type: SIMPLE
    table: city
  partitions: NULL
      type: range
possible_keys: Population
          key: Population
    key_len: 4
        ref: NULL
       rows: 237
  filtered: 100.00
     Extra: Using index condition
1 row in set, 1 warning (#.## sec)

```

- What is the access method for this query? **Answer:** range
  - Does this access strategy result in good performance? Why? **Answer:** Yes. With the range access strategy, MySQL retrieves rows only within a given range of values by using the index specified in the `key` column to select the rows. Performance is good because MySQL optimizes many range operations.
22. Remove the index on the `Population` column of the `city` table to restore the original structure of the table.

Enter the following statement at the `mysql` prompt and receive the results shown:

```

mysql> DROP INDEX Population ON city;
Query OK, 0 rows affected (#.## sec)
Records: 0  Duplicates: 0  Warnings: 0

```

23. Execute the following statement to view the `EXPLAIN` output for a query against the `city` table:

```
EXPLAIN SELECT ID FROM city\G
```

Enter the following statement at the `mysql` prompt and receive the results shown:

```

mysql> EXPLAIN SELECT ID FROM city\G
***** 1. row *****
        id: 1
select_type: SIMPLE
    table: city
  partitions: NULL
      type: index
possible_keys: NULL
          key: CountryCode
    key_len: 3
        ref: NULL
       rows: 4188
  filtered: 100.00

```

***Extra: Using index***

```
1 row in set, 1 warning (#.# sec)
```

- What is the access method for this query? **Answer:** index
  - Does this access strategy result in good performance? Why? **Answer:** Probably not. The index access method tells you that MySQL scans the entire index tree. Even though this is preferable to a full table scan, it might still result in poor performance.
24. Execute the following statement to view the EXPLAIN output for a query against the city table:

```
EXPLAIN SELECT * FROM city WHERE Population > 1000000\G
```

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> EXPLAIN SELECT * FROM city WHERE Population > 1000000\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: city
    partitions: NULL
      type: ALL
possible_keys: NULL
      key: NULL
     key_len: NULL
       ref: NULL
      rows: 4188
  filtered: 33.33
     Extra: Using where
1 row in set, 1 warning (#.# sec)
```

- What is the access method for this query? **Answer:** ALL
  - Does this access strategy result in good performance? Why? **Answer:** No. The ALL access method tells you that MySQL performs a full table scan, reading every single row from the table to find matches. MySQL cannot use any indexes to reduce the number of rows it must read. This type of query results in the poorest performance.
25. Leave the terminal window with the mysql session open for the next practice.

## Practice 10-2: Improving the Performance of a Query

### Overview

In this practice, you improve the performance of a specific query. To accomplish this, you perform the following tasks:

- Examine the query execution plan by using the EXPLAIN command.
- Determine what changes to make to improve the response time.
- Implement the changes.

### Duration

This practice should take you approximately 20 minutes to complete.

### Tasks

1. In the mysql session from the previous practice, execute the following statement to view the EXPLAIN output for a query against the city table in the world database:

```
EXPLAIN SELECT Name FROM city
WHERE District = 'Zhejiang'
ORDER BY Population DESC\G
```

- What is the access method for this query?
  - Which indexes can the optimizer use for this query?
  - What information does the Extra column contain?
  - Does this query perform well? Why?
  - What change must you make to the table to prevent the query from performing a full table scan?
2. Improve the performance of the query by making the change identified in the previous step.
  3. Re-issue the EXPLAIN statement from step 1.
    - What is the access method for this query?
    - What key does the optimizer use?
    - What information does the Extra column contain?
    - Does the change you implemented in the preceding step improve the performance of the query? Why?
    - What must you do to prevent this query from having to perform a filesort?
  4. Remove the necessity for a filesort operation by making the changes identified in the previous step.
  5. Re-issue the EXPLAIN statement from step 1.
    - What key does the optimizer use?
    - What information does the Extra column contain?
    - Does this change improve the performance of the query? Why?
  6. What must you do to prevent this query from requiring a primary key lookup after finding the indexed row?

7. Make the necessary change identified in the previous step and remove any indexes that the change makes redundant.
8. Re-issue the `EXPLAIN` statement from step 1.
  - What key does the optimizer use?
  - What information does the `Extra` column contain?
  - Does this change improve the performance of the query? Why?
9. Drop any indexes that you added in this practice to return the `city` table to its original state.
10. Exit the `mysql` session but leave the Linux terminal window open for the next practice.

## Solution 10-2: Improving the Performance of a Query

---

### Solution Steps

1. In the mysql session from the previous practice, execute the following statement to view the EXPLAIN output for a query against the city table in the world database:

```
EXPLAIN SELECT Name FROM city
WHERE District = 'Zhejiang'
ORDER BY Population DESC\G
```

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> EXPLAIN SELECT Name FROM city
-> WHERE District = 'Zhejiang'
-> ORDER BY Population DESC\G
***** 1. row *****
    id: 1
  select_type: SIMPLE
        table: city
    partitions: NULL
        type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 4188
     filtered: 10.00
       Extra: Using where; Using filesort
1 row in set, 1 warning (#.# sec)
```

- What is the access method for this query? **Answer:** ALL
- Which indexes can the optimizer use for this query? **Answer:** None
- What information does the Extra column contain? **Answer:**
  - Using where: The query uses a WHERE clause to filter the output rows.
  - Using filesort: MySQL must perform an extra pass through the table to work out how to retrieve the rows in sorted order. The sort is performed by iterating through all the rows according to the join type and storing the sort key and pointer to the row for all rows that match the WHERE clause. MySQL sorts the keys, and then retrieves the rows in the required order.
- Does this query perform well? Why? **Answer:** No. This query requires a full table scan (type: ALL), cannot use any keys (possible\_keys: NULL), and requires a sort on the Population column (Extra: ... Using filesort). A filesort requires MySQL to perform quick sorts on the data in the sort buffer. The amount of data it can fit into the buffer depends on the sort\_buffer\_size system variable. If the sort buffer cannot contain all the data that it must sort, MySQL feeds the data to the sort buffer in portions, and then performs a sort merge operation to combine them. If this is

the case, MySQL uses a temporary file to store the data portions during the sorting operation.

- What change must you make to the table to prevent the query from performing a full table scan? **Answer:** You must add an index to the District column of the city table.
- 2. Improve the performance of the query by making the change identified in the previous step.

Enter the following statement at the mysql prompt:

```
mysql> ALTER TABLE city ADD INDEX (District);
Query OK, 0 rows affected (#.# sec)
Records: 0  Duplicates: 0  Warnings: 0
```

- 3. Re-issue the EXPLAIN statement from step 1.

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> EXPLAIN SELECT Name FROM city
      -> WHERE District = 'Zhejiang'
      -> ORDER BY Population DESC\G
***** 1. row *****
    id: 1
  select_type: SIMPLE
        table: city
    partitions: NULL
      type: ref
possible_keys: District
      key: District
    key_len: 20
        ref: const
      rows: 16
  filtered: 100.00
    Extra: Using index condition; Using filesort
1 row in set, 1 warning (#.# sec)
```

- What is the access method for this query? **Answer:** ref
- What key does the optimizer use? **Answer:** District
- What information does the Extra column contain? **Answer:** Using index condition; Using filesort
- Does the change that you implemented in the preceding step improve the performance of the query? Why? **Answer:** Yes. Now the query does not require a full table scan. Instead, it uses the ref access method to retrieve the rows from the index on the District column. In addition, the storage engine matches rows (Extra: Using index condition.)
  - Index Condition Pushdown (ICP) is an optimization that MySQL uses when it can retrieve rows from a table by using an index. This reduces the number of times the storage engine must access the base table and reduces the number of times the MySQL server must access the storage engine. If MySQL can evaluate the WHERE

condition by using only columns from the index, it delegates this responsibility to the storage engine. The only rows that MySQL reads from the table are those that match the `WHERE` condition.

- What must you do to prevent this query from having to perform a `filesort`?  
**Answer:** Add a composite index to the `city` table that includes the `District` and `Population` columns. The index on the `District` column is no longer required in this case, because MySQL can use the index prefix.
4. Remove the necessity for a `filesort` operation by making the changes identified in the previous step.

Enter the following statements at the `mysql` prompt and receive the results shown:

```
mysql> ALTER TABLE city
      -> ADD INDEX Dist_Pop (District, Population);
Query OK, 0 rows affected (#.## sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE city
      -> DROP INDEX District;
Query OK, 0 rows affected (#.## sec)
Records: 0  Duplicates: 0  Warnings: 0
```

5. Re-issue the `EXPLAIN` statement from step 1.

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> EXPLAIN SELECT Name FROM city
      -> WHERE District = 'Zhejiang'
      -> ORDER BY Population DESC\G
*****
          1. row ****
              id: 1
              select_type: SIMPLE
                  table: city
                  partitions: NULL
                  type: ref
              possible_keys: Dist_Pop
                  key: Dist_Pop
                  key_len: 20
                  ref: const
                  rows: 16
                  filtered: 100.00
                  Extra: Using where
1 row in set, 1 warning (#.## sec)
```

- What key does the optimizer use? **Answer:** The key that you added in the preceding step. (`Dist_Pop` in the example solution)
- What information does the `Extra` column contain? **Answer:** Using where

- Does this change improve the performance of the query? Why? **Answer:** Yes. Adding a composite key on the District and Population columns removes the requirement to perform a filesort. The only disadvantage is that the MySQL server cannot push the index condition down to the storage engine.
6. What must you do to prevent this query from requiring a primary key lookup after finding the indexed row? **Answer:** Add a composite index to the city table that includes the District, Population, and Name columns. Drop the composite index on the District and Population columns, because MySQL can use the index prefix.
7. Make the necessary change identified in the previous step and remove any indexes that the change makes redundant.

Enter the following statements at the mysql prompt and receive the results shown:

```
mysql> ALTER TABLE city
      -> ADD INDEX Dist_Pop_Name (District, Population, Name);
Query OK, 0 rows affected (#.# sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE city
      -> DROP INDEX Dist_Pop;
Query OK, 0 rows affected (#.# sec)
Records: 0  Duplicates: 0  Warnings: 0
```

8. Re-issue the EXPLAIN statement from step 1.

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> EXPLAIN SELECT Name FROM city
      -> WHERE District = 'Zhejiang'
      -> ORDER BY Population DESC\G
*****
           1. row ****
    id: 1
  select_type: SIMPLE
        table: city
     partitions: NULL
       type: ref
  possible_keys: Dist_Pop_Name
        key: Dist_Pop_Name
    key_len: 20
      ref: const
     rows: 16
  filtered: 100.00
    Extra: Using where; Using index
1 row in set, 1 warning (#.# sec)
```

- What key does the optimizer use? **Answer:** Dist\_Pop\_Name
- What information does the Extra column contain? **Answer:** Using where and Using index

- Does this change improve the performance of the query? Why? **Answer:** Yes. The query now retrieves all the data that it needs from the index, without accessing the base table (Extra: ... Using index). This type of index is known as a *covering index*. Performance will be very good for reads, but there might be a negative impact on performance for table updates because MySQL must update the index too.
9. Drop any indexes that you added in this practice to return the `city` table to its original state.

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> ALTER TABLE city
      -> DROP INDEX Dist_Pop_Name;
Query OK, 0 rows affected (#.## sec)
Records: 0  Duplicates: 0  Warnings: 0
```

10. Exit the `mysql` session but leave the Linux terminal window open for the next practice.

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> EXIT
Bye
#
```

## Practice 10-3: Tracing the Optimizer

### Overview

In this practice, you enable optimizer tracing and examine its output. This gives you an insight into how the MySQL optimizer generates the execution plan for a specific query. To accomplish this, you perform the following steps:

- Enable optimizer tracing.
- Execute a query.
- Send the optimizer trace output to a dump file.
- Disable optimizer tracing.
- Open the trace output in a web application that makes it easy to navigate.
- Examine the optimizer trace output.

### Duration

This practice should take you approximately 10 minutes to complete.

### Tasks

1. Using a text editor such as `nano`, edit `/etc/my.cnf` and add the following entry at the bottom of the `[mysqld]` section:

```
[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
user=mysql
# Disabling symbolic-links is recommended to prevent assorted
security risks
symbolic-links=0
secure_file_priv=/tmp
...
```

**Note:** The `--secure-file-priv` system variable limits the effects of the data import and export operations. It is set to `NULL` by default, which prohibits these operations. The value that you provide in the `/etc/my.cnf` file enables you to export database content to the server's `/tmp` directory in a later step.

2. Restart the MySQL server by executing the following command at a Linux terminal prompt and receiving the results shown:

```
# service mysql restart
Shutting down MySQL.. SUCCESS!
Starting MySQL.. SUCCESS!
```

3. Start a `mysql` command-line client session by entering the following command and receiving the results shown:

```
# mysql -uroot -p
```

```
Enter password: oracle
Welcome to the MySQL monitor. Commands end with ; or \g.
...
mysql>
```

4. Enable the optimizer trace by issuing the following commands at the mysql prompt and receiving the results shown:

```
mysql> SET optimizer_trace='enabled=ON';
Query OK, 0 rows affected (#.## sec)

mysql> SET @@end_markers_in_json=OFF;
Query OK, 0 rows affected (#.## sec)

mysql> SET @@optimizer_trace_max_mem_size=32768;
Query OK, 0 rows affected (#.## sec)
```

**Note:** Setting @@end\_markers\_in\_json to OFF makes the output harder to read, but ensures better JSON compliance. Setting @@optimizer\_trace\_max\_mem\_size to 32768 increases the length of the trace that MySQL can store.

5. Execute the following query against the world database city, country, and countrylanguage tables at the mysql prompt and receive the results shown:

```
mysql> USE world
Reading table information ...
Database changed
mysql> SELECT city.Name, Language
      -> FROM countrylanguage, country, city
      -> WHERE city.CountryCode = country.Code
      -> AND city.ID = country.Capital
      -> AND city.Population >= 1000000
      -> AND countrylanguage.CountryCode = country.Code;
...
Harare	Ndebele
Harare	Nyanja
Harare	Shona
+-----+-----+
351 rows in set (#.## sec)
```

6. Query the OPTIMIZER\_TRACE table in the INFORMATION\_SCHEMA database, sending the contents of the TRACE column to a file in the /tmp directory called trace.json.

Enter the following statement at the mysql prompt and receive the results shown:

```
mysql> SELECT TRACE INTO DUMPFILE "/tmp/trace.json"
      -> FROM INFORMATION_SCHEMA.OPTIMIZER_TRACE;
Query OK, 1 row affected (#.## sec)
```

7. Disable the optimizer trace by issuing the following command at the mysql prompt:

```
mysql> SET optimizer_trace='enabled=OFF';
Query OK, 0 rows affected (#.## sec)
```

8. Exit the mysql session and use a Linux command such as cat or less to inspect the contents of the /tmp/trace.json file:

```
# less /tmp/trace.json
{
  "steps": [
    {
      "join_preparation": {
        "select#": 1,
        "steps": [
          {
            "expanded_query": "/* select#1 */ select
`city`.`Name` AS `Name`, `countrylanguage`.`Language` AS
`Language` from `countrylanguage` join `country` join `city`
where ((`city`.`CountryCode` = `country`.`Code`) and
(`city`.`ID` = `country`.`Capital`) and (`city`.`Population` >=
1000000) and (`countrylanguage`.`CountryCode` =
`country`.`Code`))"
          }
        ]
      }
    },
    {
      "join_optimization": {
        "select#": 1,
        "steps": [
          ...
        ]
      }
    }
  ]
}
```

- This is the raw JSON output of the optimizer trace.

9. Open the trace output in Mozilla Firefox by using the File > Open File menu option and browsing to the /tmp/trace.json file.

The JSON-DataView Mozilla Firefox extension that you installed in the practice titled “Course Environment Overview” for the lesson titled “Introduction” displays the raw JSON output of the optimizer trace with syntax highlighting. You can expand and collapse individual nodes by clicking the “-” and “+” links next to the nodes.

**Note:** If the Firefox menu is not visible, press the ALT key to display it.

10. Identify the nodes for the three main phases of the query optimization process:  
join\_preparation, join\_optimization, and join\_execution.
11. Identify the following steps within the join\_optimization node:

- **condition\_processing:** This step applies various logical transformations to the SQL statement, including equality\_propagation, constant\_propagation, and trivial\_condition\_removal.
  - **rows\_estimation:** This estimates the number of rows from each table that participates in the join and calculates the cost for a full scan of that table.
  - **considered\_execution\_plans:** Each child node of this node represents one possible execution path. The table entry represents the table that the optimizer is considering as a candidate for the first table in the join order. It then develops the plan further by evaluating possible access methods, the number of rows MySQL must read, and the calculated cost of each stage and the plan as a whole.
12. Locate the join\_execution node and its select# field.  
The select# field identifies the query execution plan from all the candidate plans in the considered\_execution\_plans node that the optimizer determines as being the best (that is, the one with the lowest cost) for this particular query.
13. Close Mozilla Firefox.  
14. Exit the Linux terminal prompt.

## Solution 10-3: Tracing the Optimizer

---

There are no solution steps for this practice. Follow the practice task instructions.

GANG LIU (gangli@baylorhealth.edu) has a non-transferable license  
to use this Student Guide.

## **Practices for Lesson 11: Optimizing Locking Operations**

## Practices for Lesson 11: Overview

---

### Overview

In these practices, you will use MySQL statements and sys and Performance Schema queries to investigate locks on database objects.

**Note:** These practices use three Linux terminal windows. The instructions refer to the terminal windows as t1, t2, and t3. If you do not already have three such terminal windows open, open them before you start the practice. To make it easier to identify each terminal, change its title by using the Terminal > Set Title menu option in each terminal window.

## Practice 11-1: Troubleshooting Blocked Queries

---

### Overview

In this practice, you investigate the causes of blocked queries by using SHOW PROCESSLIST, Information Schema, and SHOW ENGINE INNODB STATUS.

### Duration

This practice should take you approximately 30 minutes to complete.

### Tasks

1. In the Linux terminal windows t<sub>1</sub>, t<sub>2</sub>, and t<sub>3</sub>, log in to the mysql client.
2. Use the mysql client in t<sub>1</sub> to start a new transaction and execute a query that selects all the rows and all the columns in the employees.departments table with the FOR UPDATE modifier.
3. Use the mysql client in t<sub>2</sub> to insert a new department called Distribution with a dept\_no of d010.
4. In t<sub>2</sub>, increase the InnoDB lock timeout to 3600 seconds and re-execute the INSERT statement from the preceding step. Do not wait for the statement to complete; proceed with the next step.
5. In t<sub>3</sub>, execute SHOW PROCESSLIST.
6. In t<sub>3</sub>, execute SHOW ENGINE INNODB STATUS to list all running transactions and examine the output for any information about pending locks.
7. In t<sub>3</sub>, enable the InnoDB lock monitor.
8. In t<sub>3</sub>, execute SHOW ENGINE INNODB STATUS again and examine the output for the extra information provided by the InnoDB lock monitor.
9. In t<sub>3</sub>, disable the InnoDB lock monitor.
10. In t<sub>3</sub>, query the sys.innodb\_lock\_waits view to display the details of the blocked and blocking transactions. Relate the transaction IDs to the output of SHOW ENGINE INNODB STATUS in step 8. Make a note of the blocking\_pid column value that provides the process list ID of the blocking transaction.
11. In t<sub>3</sub>, query the Performance Schema's threads table with the process list ID that you determined in the preceding step to determine the thread ID of the blocking transaction.
12. In t<sub>3</sub>, query the sys.session view to display the details of the latest query executed for the thread that you identified in the preceding step.
13. In t<sub>3</sub>, kill the blocking connection. Use the syntax in the sql\_kill\_blocking\_connection column of sys.innodb\_lock\_waits to perform this operation.
14. In t<sub>2</sub>, check the status of the INSERT operation that you executed in step 4.
15. In t<sub>2</sub>, delete the new row that you added to the departments table.

16. In t3, enable the `events_transactions_current` consumer in the Performance Schema.
17. In t3, enable the `transaction` instrument in the Performance Schema.
18. Re-establish the `mysql` client connection in terminal window t1 by executing a command such as `SHOW STATUS`.
19. Repeat steps 2 and 3. Do not wait for the statement in step 3 to complete; proceed with step 20.
20. Repeat steps 10 and 11 to determine the process list ID and thread ID of the blocking transaction.
21. In t3, query the `events_transactions_current` table in the Performance Schema to retrieve the details of the blocking transaction.
22. In t3, execute the following query to join the Performance Schema `events_transactions_current` and `events_statements_history` tables and view the list of queries that the transaction has executed. Substitute `<thread_ID>` with the thread ID that you determined in step 20.

```
SELECT t.THREAD_ID, s.SQL_TEXT
  FROM performance_schema.events_transactions_current t
INNER JOIN performance_schema.events_statements_history s
    ON s.THREAD_ID = t.THREAD_ID
   AND s.NESTING_EVENT_ID = t.EVENT_ID
  WHERE t.THREAD_ID = <thread_id>
 ORDER BY s.EVENT_ID;
```

23. In t1, roll back the transaction.
24. In t2, verify that the `INSERT` operation has completed successfully.
25. In t2, delete the new row that you added to the `departments` table.
26. Leave terminal windows t1, t2, and t3 open and connected to the MySQL server for the next practice.

## Solution 11-1: Troubleshooting Blocked Queries

---

### Solution Steps

1. In the Linux terminal windows t<sub>1</sub>, t<sub>2</sub>, and t<sub>3</sub>, log in to the mysql client.

At the t<sub>1</sub> Linux terminal window prompt, enter the following command and receive the results shown:

```
# mysql -uroot -p
Enter password: oracle
Welcome to the MySQL monitor. Commands end with ; or \g.
...
mysql>
```

At the t<sub>2</sub> Linux terminal window prompt, enter the following command and receive the results shown:

```
# mysql -uroot -p
Enter password: oracle
Welcome to the MySQL monitor. Commands end with ; or \g.
...
mysql>
```

At the t<sub>3</sub> Linux terminal window prompt, enter the following command and receive the results shown:

```
# mysql -uroot -p
Enter password: oracle
Welcome to the MySQL monitor. Commands end with ; or \g.
...
mysql>
```

2. Use the mysql client in t<sub>1</sub> to start a new transaction and execute a query that selects all the rows and all the columns in the employees.departments table with the FOR UPDATE modifier.

Enter the following statements at the mysql prompt in t<sub>1</sub> and receive the results shown:

```
mysql> USE employees;
...
Database changed
mysql> START TRANSACTION;
Query OK, 0 rows affected (#.## sec)

mysql> SELECT * FROM departments FOR UPDATE;
+-----+-----+
| dept_no | dept_name      |
+-----+-----+
| d009    | Customer Service |
```

```

d005	Development
d002	Finance
d003	Human Resources
d001	Marketing
d004	Production
d006	Quality Management
d008	Research
d007	Sales
+-----+-----+
9 rows in set (#.## sec)

```

- Use the mysql client in t2 to insert a new department called Distribution with a dept\_no of d010.

At the mysql prompt in the t2 terminal window, enter the following commands and receive the results shown:

```

mysql> USE employees;
...
Database changed
mysql> INSERT INTO departments VALUES ('d010', 'Distribution');

```

- The statement does not complete because it is waiting for a lock that is currently held by the transaction in t1. After 50 seconds (the default value of the innodb\_lock\_wait\_timeout system variable), it times out with the following message:

```
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

- In t2, increase the InnoDB lock timeout to 3600 seconds and re-execute the INSERT statement from the preceding step. Do not wait for the statement to complete; proceed with the next step.

At the mysql prompt in the t2 terminal window, enter the following command and receive the results shown:

```

mysql> SET innodb_lock_wait_timeout=3600;
Query OK, 0 rows affected (#.## sec)

mysql> INSERT INTO departments VALUES ('d010', 'Distribution');

```

- The statement does not complete.

- In t3, execute SHOW PROCESSLIST.

At the mysql prompt in the t3 terminal window, enter the following command and receive the results shown:

```

mysql> SHOW PROCESSLIST\G
***** 1. row *****
Id: 3
User: root

```

```

Host: localhost
db: employees
Command: Sleep
Time: 129
State:
Info: NULL
***** 2. row *****
Id: 4
User: root
Host: localhost
db: employees
Command: Query
Time: 16
State: update
Info: insert into departments values ('d010', 'Distribution')
***** 3. row *****
Id: 5
User: root
Host: localhost
db: NULL
Command: Query
Time: 0
State: starting
Info: SHOW PROCESSLIST
3 rows in set (#.## sec)

```

- The State column of the row that represents the session that executes the INSERT operation shows that the process is in the update state. This might indicate that the statement is blocked, but might also indicate that it is executing normally.
- 6. In t3, execute SHOW ENGINE INNODB STATUS to list all running transactions and examine the output for any information about pending locks.

At the mysql prompt in the t3 terminal window, enter the following command and receive the results shown:

```

mysql> SHOW ENGINE INNODB STATUS\G
...
-----
TRANSACTIONS
-----
Trx id counter 9801231
Purge done for trx's n:o < 9801228 undo n:o < 0 state: running
but idle
History list length 1

```

```

LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 421542859791984, not started
0 lock struct(s), heap size 1136, 0 row lock(s)
---TRANSACTION 9801230, ACTIVE 127 sec inserting
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 1136, 1 row lock(s), undo
log entries 1
MySQL thread id 4, OS thread handle 140067476825856, query id 38
localhost root update
insert into departments values ('d010', 'Distribution')
----- TRX HAS BEEN WAITING 127 SEC FOR THIS LOCK TO BE
GRANTED:
RECORD LOCKS space id 55 page no 4 n bits 80 index dept_name of
table `employees`.`departments` trx id 9801230 lock mode X locks
gap before rec insert intention waiting
Record lock, heap no 3 PHYSICAL RECORD: n_fields 2; compact
format; info bits 0
0: len 7; hex 46696e616e6365; asc Finance;;
1: len 4; hex 64303032; asc d002;;
-----
---TRANSACTION 9801228, ACTIVE 240 sec
3 lock struct(s), heap size 1136, 19 row lock(s)
MySQL thread id 3, OS thread handle 140067608020736, query id 23
localhost root
...
-----
END OF INNODB MONITOR OUTPUT
=====

1 row in set (#.## sec)

```

- The INSERT statement has been waiting for a lock for several seconds. This output does not show any other transaction that is using that table, so you cannot establish which transaction is holding the blocking lock.

7. In t3, enable the InnoDB lock monitor.

Enter the following statement at the mysql prompt in t3 and receive the results shown:

```

mysql> SET GLOBAL innodb_status_output=ON;
Query OK, 0 rows affected (#.## sec)

mysql> SET GLOBAL innodb_status_output_locks=ON;
Query OK, 0 rows affected (#.## sec)

```

8. In t3, execute SHOW ENGINE INNODB STATUS again and examine the output for the extra information provided by the InnoDB lock monitor.

Enter the following statement at the mysql prompt in t3 and receive the results shown:

```
mysql> SHOW ENGINE INNODB STATUS\G
...
-----
TRANSACTIONS
-----
Trx id counter 9801231
Purge done for trx's n:o < 9801228 undo n:o < 0 state: running
but idle
History list length 1
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 421542859791984, not started
0 lock struct(s), heap size 1136, 0 row lock(s)
---TRANSACTION 9801230, ACTIVE 245 sec inserting
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 1136, 1 row lock(s), undo
log entries 1
MySQL thread id 4, OS thread handle 140067476825856, query id 38
localhost root update
insert into departments values ('d010', 'Distribution')
----- TRX HAS BEEN WAITING 245 SEC FOR THIS LOCK TO BE
GRANTED:
RECORD LOCKS space id 55 page no 4 n bits 80 index dept_name of
table `employees`.`departments` trx id 9801230 lock_mode X locks
gap before rec insert intention waiting
Record lock, heap no 3 PHYSICAL RECORD: n_fields 2; compact
format; info bits 0
 0: len 7; hex 46696e616e6365; asc Finance;;
 1: len 4; hex 64303032; asc d002;;
-----
TABLE LOCK table `employees`.`departments` trx id 9801230 lock
mode IX
RECORD LOCKS space id 55 page no 4 n bits 80 index dept_name of
table `employees`.`departments` trx id 9801230 lock_mode X locks
gap before rec insert intention waiting
Record lock, heap no 3 PHYSICAL RECORD: n_fields 2; compact
format; info bits 0
 0: len 7; hex 46696e616e6365; asc Finance;;
 1: len 4; hex 64303032; asc d002;;
-----
---TRANSACTION 9801228, ACTIVE 358 sec
3 lock struct(s), heap size 1136, 19 row lock(s)
MySQL thread id 3, OS thread handle 140067608020736, query id 23
localhost root
```

```

TABLE LOCK table `employees`.`departments` trx id 9801228 lock mode IX
RECORD LOCKS space id 55 page no 4 n bits 80 index dept_name of
table `employees`.`departments` trx id 9801228 lock mode X
Record lock, heap no 1 PHYSICAL RECORD: n_fields 1; compact
format; info bits 0
 0: len 8; hex 73757072656d756d; asc supremum;;
  Record lock, heap no 2 PHYSICAL RECORD: n_fields 2; compact
format; info bits 0
  0: len 9; hex 4d61726b6574696e67; asc Marketing;;
  1: len 4; hex 64303031; asc d001;;
  ...
-----
END OF INNODB MONITOR OUTPUT
=====
Query OK, 1 row affected (#.## sec)

```

- If you enable the InnoDB lock monitor, you see all the locks that each transaction holds, so you can find the lock for which the `INSERT` transaction is waiting. The lock monitor shows an entry for each locked record in each index. On busy systems, this means that you might see many thousands of individual record locks, which makes the lookup difficult.

9. In t3, disable the InnoDB lock monitor.

Enter the following statements at the `mysql` prompt in t3 and receive the results shown:

```

mysql> SET GLOBAL innodb_status_output=OFF;
Query OK, 0 rows affected (#.## sec)

mysql> SET GLOBAL innodb_status_output_locks=OFF;
Query OK, 0 rows affected (#.## sec)

```

10. In t3, query the `sys.innodb_lock_waits` view to display the details of the blocked and blocking transactions. Relate the transaction IDs to the output of `SHOW ENGINE INNODB STATUS` in step 8. Make a note of the `blocking_pid` column value that provides the process list ID of the blocking transaction.

Enter the following statement at the `mysql` prompt in t3 and receive the results shown:

```

mysql> SELECT * FROM sys.innodb_lock_waits\G
***** 1. row *****
      wait_started: <date and time>
      wait_age: 00:07:55
      wait_age_secs: 475
      locked_table: `employees`.`departments`

```

```

        locked_index: dept_name
        locked_type: RECORD
waiting trx_id: 9801230
waiting_trx_started: <date and time>
waiting_trx_age: 00:07:55
waiting_trx_rows_locked: 1
waiting_trx_rows_modified: 1
waiting_pid: 4
waiting_query: insert into departments values
('d010', 'Distribution')
waiting_lock_id: 9801230:55:4:3
waiting_lock_mode: X,GAP
blocking trx id: 9801228
blocking pid: 3
blocking_query: NULL
blocking_lock_id: 9801228:55:4:3
blocking_lock_mode: X
blocking_trx_started: <date and time>
blocking_trx_age: 00:09:48
blocking_trx_rows_locked: 19
blocking_trx_rows_modified: 0
sql_kill_blocking_query: KILL QUERY 3
sql_kill_blocking_connection: KILL 3
1 row in set, 3 warnings (#.## sec)

```

- The sys.innodb\_lock\_waits view shows both the blocked transaction and information about the transaction that is blocking it.
11. In t3, query the Performance Schema's threads table with the process list ID that you determined in the preceding step to determine the thread ID of the blocking transaction. Enter the following statement at the mysql prompt in t3 and receive the results shown:

```

mysql> SELECT THREAD_ID FROM performance_schema.threads
-> WHERE PROCESSLIST_ID=3;
+-----+
| THREAD_ID |
+-----+
|      28   |
+-----+
1 row in set (#.## sec)

```

- **Note:** Both PROCESSLIST\_ID and THREAD\_ID might be different on your system.
12. In t3, query the sys.session view to display the details of the latest query executed for the thread that you identified in the preceding step. Enter the following statement at the mysql prompt in t3 and receive the results shown:

```

mysql> SELECT * FROM sys.session WHERE thd_id=28\G
***** 1. row ****
    thd_id: 28
    conn_id: 3
        user: root@localhost
        db: employees
    command: Sleep
        state: NULL
        time: 788
    current_statement: NULL
    statement_latency: NULL
        progress: NULL
    lock_latency: 74.00 us
    rows_examined: 9
        rows_sent: 9
    rows_affected: 0
        tmp_tables: 0
    tmp_disk_tables: 0
        full_scan: NO
    last_statement: select * from departments for update
last_statement_latency: 158.52 us
    current_memory: 0 bytes
        last_wait: NULL
    last_wait_latency: NULL
        source: NULL
    trx_latency: NULL
    trx_state: NULL
    trx_autocommit: NULL
        pid: 20588
    program_name: mysql
1 row in set (0.03 sec)

```

- The output shows the query that the blocking thread is currently executing.
- **Note:** The output of the sys.session view is similar to that of sys.processlist and therefore of SHOW PROCESSLIST, but it does not include background threads. It contains only details of the latest query that a session executed. If the same session executes other queries after the one that caused the block, you will need to query the Performance Schema events\_statements\_history or events\_statements\_history\_long tables to retrieve this information.

13. In t3, kill the blocking connection. Use the syntax in the

`sql_kill_blocking_connection` column of `sys.innodb_lock_waits` to perform this operation.

Enter the following statement at the mysql prompt in t3 and receive the results shown:

```
mysql> KILL 3;
Query OK, 0 rows affected (#.## sec)
```

- The connection ID might be different on your system.

14. In t2, check the status of the INSERT operation that you executed in step 4.

```
mysql> INSERT INTO departments
-> VALUES ('d010', 'Distribution');
Query OK, 1 row affected (#.## sec)
```

- The query is no longer blocked and completes successfully.

15. In t2, delete the new row that you added to the departments table.

Enter the following statement at the mysql prompt in t2 and receive the results shown:

```
mysql> DELETE FROM departments WHERE dept_no = 'd010';
Query OK, 1 row affected (#.## sec)
```

16. In t3, enable the events\_transactions\_current consumer in the Performance Schema.

Enter the following statement at the mysql prompt in t3 and receive the results shown:

```
mysql> UPDATE performance_schema.setup_consumers
-> SET ENABLED = 'YES'
-> WHERE NAME = 'events_transactions_current';
Query OK, 1 row affected (#.## sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

17. In t3, enable the transaction instrument in the Performance Schema.

Enter the following statement at the mysql prompt in t3 and receive the results shown:

```
mysql> UPDATE performance_schema.setup_instruments
-> SET ENABLED = 'YES'
-> WHERE NAME = 'transaction';
Query OK, 1 row affected (#.## sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

18. Re-establish the mysql client connection in terminal window t1 by executing a command such as SHOW STATUS.

Enter the following statement at the mysql prompt in t1 and receive the results shown:

```
mysql> SHOW STATUS;
No connection. Trying to reconnect...
Connection id: 8
Current database: employees
mysql>
```

19. Repeat steps 2 and 3. Do not wait for the statement in step 3 to complete; proceed with step 20.

At the mysql prompt in the t1 terminal window, enter the following commands and receive the results shown:

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (#.## sec)

mysql> SELECT * FROM departments FOR UPDATE;
+-----+-----+
| dept_no | dept_name      |
+-----+-----+
d009	Customer Service
d005	Development
d002	Finance
d003	Human Resources
d001	Marketing
d004	Production
d006	Quality Management
d008	Research
d007	Sales
+-----+-----+
9 rows in set (#.## sec)
```

At the mysql prompt in the t2 terminal window, enter the following command and receive the results shown:

```
mysql> INSERT INTO departments
-> VALUES ('d010', 'Distribution');
```

– The statement does not complete.

20. Repeat steps 10 and 11 to determine the process list ID and thread ID of the blocking transaction.

**Note:** The process list ID and thread ID shown in the following example output might be different on your system.

Enter the following statement at the mysql prompt in t3 and receive the results shown:

```
mysql> SELECT * FROM sys.innodb_lock_waits\G
***** 1. row *****
      wait_started: <date and time>
      wait_age: 00:01:13
      wait_age_secs: 73
      locked_table: `employees`.`departments`
      locked_index: dept_name
      locked_type: RECORD
      waiting_trx_id: 9801258
      waiting trx started: <date and time>
      waiting trx age: 00:01:13
```

```

waiting trx rows locked: 1
waiting trx rows modified: 1
    waiting pid: 4
        waiting query: insert into departments values
('d010', 'Distribution')
            waiting lock id: 9801258:55:4:3
            waiting lock mode: X,GAP
            blocking trx id: 9801257
                blocking pid: 8
                    blocking query: NULL
                    blocking lock id: 9801257:55:4:3
                    blocking lock mode: X
                    blocking trx started: <date and time>
                    blocking trx age: 00:01:28
                    blocking trx rows locked: 19
blocking trx rows modified: 0
    sql kill blocking query: KILL QUERY 8
sql kill blocking connection: KILL 8
1 row in set, 3 warnings (#.## sec)

```

Enter the following statement at the mysql prompt in t3 and receive the results shown:

```

mysql> SELECT THREAD_ID FROM performance_schema.threads
-> WHERE PROCESSLIST_ID = 8;
+-----+
| THREAD_ID |
+-----+
|      33   |
+-----+
1 row in set (0.00 sec)

```

21. In t3, query the events\_transactions\_current table in the Performance Schema to retrieve details of the blocking transaction.

Enter the following statement at the mysql prompt in t3 and receive the results shown:

```

mysql> SELECT *
-> FROM performance_schema.events_transactions_current
-> WHERE THREAD_ID = 33\G
***** 1. row *****
    THREAD_ID: 33
    EVENT_ID: 20
    END_EVENT_ID: NULL
    EVENT_NAME: transaction
    STATE: ACTIVE
    TRX_ID: NULL
    GTID: AUTOMATIC

```

```

        XID_FORMAT_ID: NULL
        XID_GTRID: NULL
        XID_BQUAL: NULL
        XA_STATE: NULL
        SOURCE: transaction.cc:209
        TIMER_START: NULL
        TIMER_END: NULL
        TIMER_WAIT: NULL
        ACCESS_MODE: READ WRITE
        ISOLATION_LEVEL: REPEATABLE READ
        AUTOCOMMIT: NO
        NUMBER_OF_SAVEPOINTS: 0
NUMBER_OF_ROLLBACK_TO_SAVEPOINT: 0
        NUMBER_OF_RELEASE_SAVEPOINT: 0
        OBJECT_INSTANCE_BEGIN: NULL
        NESTING_EVENT_ID: 19
        NESTING_EVENT_TYPE: STATEMENT
1 row in set (#.## sec)

```

- The events\_transactions\_current table provides more information about the blocking transaction.

22. In t3, execute the following query to join the Performance Schema

events\_transactions\_current and events\_statements\_history tables and view the list of queries that the transaction has executed. Substitute <thread\_ID> with the thread ID that you determined in step 20.

```

SELECT t.THREAD_ID, s.SQL_TEXT
FROM performance_schema.events_transactions_current t
INNER JOIN performance_schema.events_statements_history s
ON s.THREAD_ID = t.THREAD_ID
AND s.NESTING_EVENT_ID = t.EVENT_ID
WHERE t.THREAD_ID = <thread_id>
ORDER BY s.EVENT_ID;

```

Enter the following statement at the mysql prompt in t3 and receive the results shown:

```

mysql> SELECT t.THREAD_ID, s.SQL_TEXT
-> FROM performance_schema.events_transactions_current t
-> INNER JOIN performance_schema.events_statements_history s
-> ON s.THREAD_ID = t.THREAD_ID
-> AND s.NESTING_EVENT_ID = t.EVENT_ID
-> WHERE t.THREAD_ID = 33
-> ORDER BY s.EVENT_ID;
+-----+-----+
| THREAD_ID | SQL_TEXT          |
+-----+-----+

```

```
|       33 | SELECT * FROM departments FOR UPDATE |
+-----+
1 row in set (#.## sec)
```

- This approach allows you to view the statement history for a specific transaction and is especially useful if a transaction has executed more than one statement.

23. In t1, roll back the transaction.

Enter the following statement at the mysql prompt in t1 and receive the results shown:

```
mysql> ROLLBACK;
Query OK, 0 rows affected (#.## sec)
```

24. In t2, verify that the INSERT operation completes successfully.

```
mysql> INSERT INTO departments
-> VALUES ('d010', 'Distribution');
Query OK, 1 row affected (#.## sec)
```

- The query is no longer blocked and completes successfully.

25. In t2, delete the new row that you added to the departments table.

Enter the following statement at the mysql prompt in t2 and receive the results shown:

```
mysql> DELETE FROM departments WHERE dept_no = 'd010';
Query OK, 1 row affected (#.## sec)
```

26. Leave terminal windows t1, t2, and t3 open and connected to the MySQL server for the next practice.

## Practice 11-2: Investigating Metadata Locks

### Overview

In this practice, you use the Performance Schema to view the metadata locks on a table.

### Assumptions

You have completed the practice titled “Troubleshooting Blocked Queries” in this lesson.

### Duration

This practice should take you approximately 15 minutes to complete.

### Tasks

1. In t1, enable the `global_instrumentation` consumer in the Performance Schema.
2. In t1, enable the `wait/lock/metadata/sql/mdl` instrument in the Performance Schema.
3. In t1, create a new database called `locktest`.
4. In t1, execute the following query to create the `tbl1` table in the `locktest` database:

```
CREATE TABLE tbl1 (
    ID int,
    NAME VARCHAR(255)
);
```

5. In t1, start a transaction and issue a query that selects everything in the `tbl1` table.
6. In t2, drop the `NAME` column from the `tbl1` table in the `locktest` database. Do not wait for the statement to complete; proceed to the next step.
7. In t3, execute the following query against the `metadata_locks` and `threads` tables in the Performance Schema to show which metadata locks are being held on the `tbl1` table:

```
SELECT OBJECT_NAME, LOCK_TYPE, LOCK_STATUS,
       THREAD_ID, PROCESSLIST_ID, PROCESSLIST_INFO
  FROM performance_schema.metadata_locks
 INNER JOIN performance_schema.threads
    ON THREAD_ID = OWNER_THREAD_ID
 WHERE OBJECT_SCHEMA = 'locktest'
   AND PROCESSLIST_ID <> CONNECTION_ID();
```

8. In t1, roll back the transaction.
9. In t2, check the status of the table modification statement that you issued in step 6.
10. In t2, drop the `locktest` database.
11. In t2, set the global `innodb_lock_wait_timeout` variable back to its default setting of 50.

12. Exit all open mysql command-line client sessions and terminal windows.
13. Enter the following at the t1, t2, and t3 terminal windows:

## Solution 11-2: Investigating Metadata Locks

---

### Solution Steps

1. In t1, enable the `global_instrumentation` consumer in the Performance Schema.

Enter the following statement at the `mysql` prompt in t1 and receive the results shown:

```
mysql> UPDATE performance_schema.setup_consumers
      -> SET ENABLED = 'YES'
      -> WHERE NAME = 'global_instrumentation';
Query OK, 0 rows affected (#.# sec)
Rows matched: 1  Changed: 0  Warnings: 0
```

2. In t1, enable the `wait/lock/metadata/sql/mdl` instrument in the Performance Schema.

Enter the following statement at the `mysql` prompt in t1 and receive the results shown:

```
mysql> UPDATE performance_schema.setup_instruments
      -> SET ENABLED = 'YES'
      -> WHERE NAME = 'wait/lock/metadata/sql/mdl';
Query OK, 1 row affected (#.# sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

3. In t1, create a new database called `locktest`.

Enter the following statement at the `mysql` prompt in t1 and receive the results shown:

```
mysql> CREATE DATABASE locktest;
Query OK, 1 row affected (#.# sec)
```

4. In t1, execute the following query to create the `tbl1` table in the `locktest` database:

```
CREATE TABLE tbl1 (
    ID int,
    NAME VARCHAR(255)
);
```

Enter the following statements at the `mysql` prompt in t1 and receive the results shown:

```
mysql> USE locktest;
Database changed
mysql> CREATE TABLE tbl1 (ID int, NAME VARCHAR(255));
Query OK, 0 rows affected (#.# sec)
```

5. In t1, start a transaction and issue a query that selects everything in the `tbl1` table.

Enter the following statement at the `mysql` prompt in t1 and receive the results shown:

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (#.# sec)

mysql> SELECT * FROM tbl1;
Empty set (#.# sec)
```

6. In t2, drop the NAME column from the `tbl1` table in the `locktest` database. Do not wait for the statement to complete; proceed to the next step.

Enter the following statements at the `mysql` prompt in t1 and receive the results shown:

```
mysql> USE locktest;
...
Database changed
mysql> ALTER TABLE tbl1
-> DROP NAME;
```

- The statement does not complete.

7. In t3, execute the following query against the `metadata_locks` and `threads` tables in the Performance Schema to show which metadata locks are being held on the `tbl1` table:

```
SELECT OBJECT_NAME, LOCK_TYPE, LOCK_STATUS,
       THREAD_ID, PROCESSLIST_ID, PROCESSLIST_INFO
  FROM performance_schema.metadata_locks
 INNER JOIN performance_schema.threads
    ON THREAD_ID = OWNER_THREAD_ID
 WHERE OBJECT_SCHEMA = 'locktest'
   AND PROCESSLIST_ID <> CONNECTION_ID();
```

Enter the following statement at the `mysql` prompt in t3 and receive the results shown:

```
mysql> SELECT OBJECT_NAME, LOCK_TYPE, LOCK_STATUS,
->     THREAD_ID, PROCESSLIST_ID, PROCESSLIST_INFO
->     FROM performance_schema.metadata_locks
->     INNER JOIN performance_schema.threads
->     ON THREAD_ID = OWNER_THREAD_ID
->     WHERE OBJECT_SCHEMA = 'locktest'
->     AND PROCESSLIST_ID <> CONNECTION_ID();
*****
          1. row *****
OBJECT_NAME: NULL
LOCK_TYPE: INTENTION_EXCLUSIVE
LOCK_STATUS: GRANTED
THREAD_ID: 29
PROCESSLIST_ID: 4
PROCESSLIST_INFO: ALTER TABLE tbl1 DROP NAME
*****
          2. row *****
OBJECT_NAME: tbl1
LOCK_TYPE: SHARED_UPGRADABLE
LOCK_STATUS: GRANTED
THREAD_ID: 29
PROCESSLIST_ID: 4
PROCESSLIST_INFO: ALTER TABLE tbl1 DROP NAME
*****
          3. row *****
```

```

OBJECT_NAME: tbl11
LOCK_TYPE: EXCLUSIVE
LOCK_STATUS: PENDING
THREAD_ID: 29
PROCESSLIST_ID: 4
PROCESSLIST_INFO: ALTER TABLE tbl11 DROP NAME
***** 4. row *****
OBJECT_NAME: tbl11
LOCK_TYPE: SHARED_READ
LOCK_STATUS: GRANTED
THREAD_ID: 33
PROCESSLIST_ID: 8
PROCESSLIST_INFO: NULL
4 rows in set (#.## sec)

```

- The output shows the locks on the `tbl11` table. The process with the ID of 4 in the preceding example output is waiting for an exclusive metadata lock on the `tbl11` table. MySQL has granted all the other locks on the table.
8. In `t1`, roll back the transaction.

Enter the following statement at the `mysql` prompt in `t1` and receive the results shown:

```

mysql> ROLLBACK;
Query OK, 0 rows affected (#.## sec)

```

9. In `t2`, check the status of the table modification statement that you issued in step 6.

```

mysql> ALTER TABLE tbl11
-> DROP NAME;
Query OK, 0 rows affected (#.## sec)
Records: 0  Duplicates: 0  Warnings: 0

```

- The statement completes successfully.

10. In `t2`, drop the `locktest` database.

Enter the following statement at the `mysql` prompt in `t2` and receive the results shown:

```

mysql> DROP DATABASE locktest;
Query OK, 1 row affected (#.## sec)

```

11. In `t2`, set the global `innodb_lock_wait_timeout` variable back to its default setting of 50.

Enter the following statement at the `mysql` prompt in `t2` and receive the results shown:

```

mysql> SET GLOBAL innodb_lock_wait_timeout = 50;
Query OK, 0 rows affected (#.## sec)

```

12. Exit all open `mysql` command-line client sessions and terminal windows.

13. Enter the following at the t1, t2, and t3 terminal windows:

```
mysql> EXIT  
Bye  
# exit
```

GANG LIU (gangli@baylorhealth.edu) has a non-transferable license  
to use this Student Guide.

## **Practices for Lesson 12: Tuning Replication**

## Practices for Lesson 12: Overview

---

### Overview

In these practices, you will test your ability to diagnose replication lag.

### Assumptions

The following files are present:

- /root/labs/repl.cnf
- /root/scripts/repl-insert.sql

## Practice 12-1: Diagnosing Replication Lag by Using Binary Log File Name and Position

### Overview

In this practice, you create a simple replicated system with one master replicating to one slave. You then simulate replication lag and diagnose it by comparing the binary log file names and positions between the master and slave replication servers.

### Duration

This practice should take you approximately 25 minutes to complete.

### Tasks

1. Using a text editor such as `nano`, or a Linux tool like `cat` or `less`, examine the contents of the `/root/labs/repl.cnf` file.
2. Create and initialize two new MySQL server instances by executing the following commands at a Linux terminal prompt:

```
# mysqld --no-defaults --initialize-insecure --user=mysql  
--datadir=/var/lib/mysql11  
  
# mysqld --no-defaults --initialize-insecure --user=mysql  
--datadir=/var/lib/mysql12
```
3. Start the two server instances with the configuration information in the `/root/labs/repl.cnf` file, by executing the following command at the Linux terminal prompt:

```
# mysqld_multi --defaults-file=/root/labs/repl.cnf start 1-2
```
4. Connect to the first (replication master) server on port 3311 by using the `mysql` command-line utility.
5. Use the `PROMPT` command to change the prompt from `mysql>` to `master>`. This will make it easier for you to identify the master server in the subsequent steps.
6. Create a user called `repl@127.0.0.1` on the master with a password of “oracle,” and grant the user the `REPLICATION SLAVE` privilege.
7. Issue an appropriate `FLUSH TABLES` statement on the master that locks all tables for all databases with a global read lock.
8. Issue the `SHOW MASTER STATUS` statement on the master and note the current binary log file name and position:
9. Open a new Linux terminal window and connect to the MySQL server on port 3312. This server will act as the replication slave.
10. Use the `PROMPT` command to change the prompt from `mysql>` to `slave>`.
11. On the replication slave, issue a `CHANGE MASTER TO` statement to specify the following information about the master:

- The port that the master is running on
  - The name, host, and password of the user account with the REPLICATION SLAVE privilege
  - The master log file name
  - The master log file position
12. Start the replication slave.
13. On the replication master, display the process list.
14. On the replication master, create a new database called `repltest` and make it the current database.
15. On the replication master, create a new table called `repltbl` in the `repltest` database, by executing the following statement and receiving the results shown:

```
master> CREATE TABLE repltbl (
    ->     id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    ->     name VARCHAR(30) NOT NULL,
    ->     email VARCHAR(50)
    -> );
```

Query OK, 0 rows affected (#.## sec)

16. On the replication slave, issue the `SHOW DATABASES` statement.
17. On the replication slave, make `repltest` the current database and list its tables.
18. Stop the replication slave.
19. On the replication slave, issue the following statement to introduce a 10-minute delay on the master. This simulates replication lag for the subsequent steps:

```
slave> CHANGE MASTER TO MASTER_DELAY = 600;
```

Query OK, 0 rows affected (#.## sec)

20. Start the replication slave.
21. On the replication master, insert a new row into the `repltest.repltbl` table with the following column values:
- `id`: John Smith
  - `email`: `jsmith@mycompany.com`
22. On the replication slave, issue a statement to display the contents of the `repltest.repltbl` table.
23. On the replication slave, show the slave's current status. Note the values of `Master_Log_File`, `Read_Master_Log_Pos`, `Relay_Master_Log_File`, `Exec_Master_Log_File_Pos`, and `Seconds_Behind_Master`. What does this tell you about the current replication status?
24. Leave both the Linux terminal windows open and connected to the MySQL servers for the next practice.

## Solution 12-1: Diagnosing Replication Lag by Using Binary Log File Name and Position

### Solution Steps

1. Using a text editor such as nano, or a Linux tool like cat or less, examine the contents of the /root/labs/repl.cnf file.

The contents of the repl.cnf file are as follows:

```
[mysqld1]
datadir=/var/lib/mysql1
port=3311
socket=/var/lib/mysql1/mysql.sock
server-id=1
user=mysql
log-bin=mysql1-bin
relay-log=mysql1-relay-bin
log-slave-updates=ON
log-error=mysql1
report-host=localhost
report-port=3311
relay-log-recovery=1
master-info-repository=TABLE
relay-log-info-repository=TABLE
#gtid-mode=ON
#enforce-gtid-consistency

[mysqld2]
datadir=/var/lib/mysql2
port=3312
socket=/var/lib/mysql2/mysql.sock
server-id=2
user=mysql
log-bin=mysql2-bin
relay-log=mysql2-relay-bin
log-slave-updates=ON
log-error=mysql2
report-host=localhost
report-port=3312
relay-log-recovery=1
master-info-repository=TABLE
relay-log-info-repository=TABLE
#gtid-mode=ON
```

```
#enforce-gtid-consistency
```

- The `repl.cnf` file provides the configuration settings for two servers (`mysqld1` and `mysqld2`) in a replication setup. The GTID settings (`gtid-mode` and `enforce-gtid-consistency`) are commented out.

2. Create and initialize two new MySQL server instances by executing the following commands at a Linux terminal prompt:

```
# mysqld --no-defaults --initialize-insecure --user=mysql  
--datadir=/var/lib/mysql1
```

```
# mysqld --no-defaults --initialize-insecure --user=mysql  
--datadir=/var/lib/mysql2
```

3. Start the two server instances with the configuration information in the `/root/labs/repl.cnf` file, by executing the following command at the Linux terminal prompt:

```
# mysqld_multi --defaults-file=/root/labs/repl.cnf start 1-2
```

4. Connect to the first (replication master) server on port 3311 by using the `mysql` command-line utility.

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# mysql -uroot -h127.0.0.1 -P3311 --skip-password  
Welcome to the MySQL monitor. Commands end with ; or \g.  
...  
mysql>
```

5. Use the `PROMPT` command to change the prompt from `mysql>` to `master>`. This will make it easier for you to identify the master server in the subsequent steps.

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> PROMPT master> ;  
master>
```

6. Create a user called `repl@127.0.0.1` on the master with a password of “oracle,” and grant the user the `REPLICATION SLAVE` privilege.

Enter the following statements at the `mysql` prompt and receive the results shown:

```
master> CREATE USER 'repl'@'127.0.0.1' IDENTIFIED BY 'oracle';  
Query OK, 0 rows affected (#.## sec)  
  
master> GRANT REPLICATION SLAVE ON *.* TO 'repl'@'127.0.0.1';  
Query OK, 0 rows affected (#.## sec)
```

7. Issue an appropriate `FLUSH TABLES` statement on the master that locks all tables for all databases with a global read lock.

Enter the following statement at the `master>` prompt and receive the results shown:

```
master> FLUSH TABLES WITH READ LOCK;  
Query OK, 0 rows affected (#.## sec)
```

8. Issue the `SHOW MASTER STATUS` statement on the master and note the current binary log file name and position:

Enter the following statement at the `master>` prompt and receive the results shown:

```
master> SHOW MASTER STATUS\G
***** 1. row *****
      File: mysql1-bin.000002
      Position: 611
      Binlog_Do_DB:
      Binlog_Ignore_DB:
      Executed_Gtid_Set:
1 row in set (#.## sec)
```

- In the example output, the current binary log file name is `mysql1-bin.000002` and the log file position is 611. The file name and position might be different on your system.

9. Open a new Linux terminal window and connect to the MySQL server on port 3312. This server will act as the replication slave.

Enter the following statement at the `mysql` prompt in a new Linux terminal window and receive the results shown:

```
# mysql -uroot -h127.0.0.1 -P3312
Welcome to the MySQL monitor. Commands end with ; or \g.
...
mysql>
```

10. Use the `PROMPT` command to change the prompt from `mysql>` to `slave>`.

Enter the following statement at the `mysql` prompt and receive the results shown:

```
mysql> PROMPT slave> ;
slave>
```

11. On the replication slave, issue a `CHANGE MASTER TO` statement to specify the following information about the master:

- The port the master is running on
- The name, host, and password of the user account with the `REPLICATION SLAVE` privilege
- The master log file name
- The master log file position

Enter the following statement at the `slave>` prompt and receive the results shown:

```
slave> CHANGE MASTER TO
      -> MASTER_HOST='127.0.0.1',
      -> MASTER_USER='rep1',
      -> MASTER_PASSWORD='oracle',
      -> MASTER_LOG_FILE='mysql1-bin.000002',
      -> MASTER_LOG_POS=611,
```

```
-> MASTER_PORT=3311;
Query OK, 0 rows affected, 1 warning (#.##)
```

12. Start the replication slave.

Enter the following statement at the `slave>` prompt and receive the results shown:

```
slave> START SLAVE;
Query OK, 0 rows affected (#.## sec)
```

13. On the replication master, display the process list.

Enter the following statement at the `slave>` prompt and receive the results shown:

```
master> SHOW PROCESSLIST\G
***** 1. row *****
Id: 4
User: root
Host: localhost:52768
db: NULL
Command: Query
Time: 0
State: starting
Info: SHOW PROCESSLIST
***** 2. row *****
Id: 5
User: repl
Host: localhost:52770
db: NULL
Command: Binlog Dump
Time: 84
State: Master has sent all binlog to slave; waiting for more updates
Info: NULL
2 rows in set (#.## sec)
```

- The slave server is awaiting updates from the master.

14. On the replication master, create a new database called `repltest` and make it the current database.

Enter the following statements at the `master>` prompt and receive the results shown:

```
master> CREATE DATABASE repltest;
Query OK, 1 row affected (#.## sec)

master> USE repltest;
Database changed
```

15. On the replication master, create a new table called `repltbl` in the `repltest` database, by executing the following statement and receiving the results shown:

```
master> CREATE TABLE repltbl (
    ->   id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    ->   name VARCHAR(30) NOT NULL,
    ->   email VARCHAR(50)
    -> );
```

Query OK, 0 rows affected (#.## sec)

16. On the replication slave, issue the SHOW DATABASES statement.

Enter the following statement at the slave> prompt and receive the results shown:

```
slave> SHOW DATABASES;
+-----+
| Database      |
+-----+
| information_schema |
| mysql          |
| performance_schema |
| repltest       |
| sys            |
+-----+
5 rows in set (#.## sec)
```

17. On the replication slave, make repltest the current database and list its tables.

Enter the following statements at the slave> prompt and receive the results shown:

```
slave> USE repltest;
Database changed.

slave> SHOW TABLES;
+-----+
| Tables_in_repltest |
+-----+
| repltbl        |
+-----+
1 row in set (#.## sec)
```

- The repltest database and its table also exist on the slave. Replication is working.

18. Stop the replication slave.

Enter the following statement at the slave> prompt and receive the results shown:

```
slave> STOP SLAVE;
```

Query OK, 0 rows affected (#.## sec)

19. On the replication slave, issue the following statement to introduce a 10-minute delay on the master. This simulates replication lag for the subsequent steps:

```
slave> CHANGE MASTER TO MASTER_DELAY = 600;
Query OK, 0 rows affected (#.## sec)
```

20. Start the replication slave.

Enter the following statement at the `slave>` prompt and receive the results shown:

```
slave> START SLAVE;
Query OK, 0 rows affected (#.## sec)
```

21. On the replication master, insert a new row into the `repltest.repltbl` table with the following column values:

- `id`: John Smith
- `email`: `jsmith@mycompany.com`

Enter the following statement at the `master>` prompt and receive the results shown:

```
master> INSERT INTO repltbl (name, email)
      -> VALUES ('John Smith', 'jsmith@mycompany.com');
Query OK, 1 row affected (#.## sec)
```

22. On the replication slave, issue a statement to display the contents of the `repltest.repltbl` table.

Enter the following statement at the `slave>` prompt and receive the results shown:

```
slave> SELECT * FROM repltbl;
Empty set (#.## sec)
```

- The update on the master has not replicated to the slave.

23. On the replication slave, show the slave's current status. Note the values of `Master_Log_File`, `Read_Master_Log_Pos`, `Relay_Master_Log_File`, `Exec_Master_Log_File_Pos`, and `Seconds_Behind_Master`. What does this tell you about the current replication status?

Enter the following statement at the `slave>` prompt and receive the results shown:

```
slave> SHOW SLAVE STATUS\G
***** 1. row *****
Slave_IO_State: Waiting for master to send event
Master_Host: 127.0.0.1
Master_User: repl
Master_Port: 3311
Connect_Retry: 60
Master_Log_File: mysql1-bin.000002
Read_Master_Log_Pos: 1355
Relay_Log_File: mysql2-relay-bin.000002
Relay_Log_Pos: 321
Relay_Master_Log_File: mysql1-bin.000002
Slave_IO_Running: Yes
```

```
Slave_SQL_Running: Yes
...
Exec_Master_Log_Pos: 1051
...
Seconds_Behind_Master: 152
...
SQL_Delay: 600
SQL_Remaining_Delay: 448
Slave_SQL_Running_State: Waiting until MASTER_DELAY
seconds after master executed event
...
1 row in set (#.## sec)
```

- **Answer:** By comparing the current log file name and position of the slave (`Master_Log_File` and `Exec_Master_Log_Position`) to that of the master (`Relay_Master_Log_File` and `Exec_Master_Log_Pos` in the output of `SHOW SLAVE STATUS` or `Position` in the output of `SHOW MASTER STATUS`), you can determine that the slave is lagging behind the master.
24. Leave both the Linux terminal windows open and connected to the MySQL servers for the next practice.

## Practice 12-2: Diagnosing Replication Lag by Using GTID Sets

### Overview

In this practice, you enable GTID replication and use the `GTID_SUBTRACT()` function to determine which transactions have been applied to the master but not to the slave.

### Duration

This practice should take you approximately 20 minutes to complete.

### Tasks

1. Open a new Linux terminal window and issue the following command at the terminal prompt to stop both the replication master and replication slave servers:

```
# mysqld_multi --defaults-file=/root/labs/repl.cnf \
> --user=root stop 1-2
```

2. Open the `/root/labs/repl.cnf` file in a text editor, such as `nano`, `vi`, or `emacs`, and uncomment the following lines:

```
# gtid-mode=ON
# enforce-gtid-consistency
```

3. Start the replication master and replication slave servers by issuing the following command at the Linux terminal prompt:

```
# mysqld_multi --defaults-file=/root/labs/repl.cnf \
> --user=root start 1-2
```

4. In the terminal window with the connection to the replication master, issue `STATUS` to reconnect the connection.
5. In the terminal window with the connection to the replication slave, issue `STATUS` to reconnect the connection.
6. Stop the replication slave.
7. On both the replication master and the replication slave, issue the `RESET MASTER` command.
8. On the replication slave, issue the following `CHANGE MASTER TO` statement to instruct the master to use GTIDs:

```
slave> CHANGE MASTER TO MASTER_AUTO_POSITION=1;
Query OK, 0 rows affected (#.## sec)
```

9. Start the replication slave.
10. On the replication master, insert a new row into the `reptest.repltbl` table with the following column values:
  - `id`: Helen Jones
  - `email`: `hjones@mycompany.com`

11. On the replication master, issue `SHOW MASTER STATUS` and note the last executed GTID set.
12. On the replication slave, issue `SHOW SLAVE STATUS` and note the retrieved and executed GTID sets.
13. On the replication slave, query the `replication_connection_status` table in the Performance Schema to provide an alternative view of the retrieved GTIDs.
14. On the replication slave, keep querying the value of the global `gtid_executed` variable until the received transaction set identified in the preceding two steps executes.
15. On the replication slave, issue `SHOW SLAVE STATUS`.
16. The `/root/scripts/repl-insert.sql` script inserts multiple rows into the `repltbl` table in the `repltest` database. Execute the `repl-insert.sql` script on the replication master.
17. On the replication slave, query the `replication_connection_status` table in the Performance Schema to list the retrieved GTIDs.
18. On the replication slave, display the last executed GTID.
19. Use the `GTID_SUBTRACT()` function to calculate which GTIDs are missing, by executing the following statement on the replication slave, replacing `<retrieved GTIDs>` with the GTID set identifier that you determined in step 17.

```
SELECT GTID_SUBTRACT(<retrieved GTIDs>, @@global.gtid_executed)
AS MissingGTIDs, IF(GTID_SUBTRACT(<retrieved GTIDs>,
@@global.gtid_executed) = '', 'YES', 'NO') AS CaughtUp FROM
performance_schema.replication_connection_status;
```

20. Close the `master>` and `slave>` sessions and their Linux terminal windows.
21. Delete the contents of the `/var/lib/mysql1` and `/var/lib/mysql2` data directories.
22. Restart the usual MySQL server (the one that you have used for all the other practices in this course).
23. Close all open Linux terminal windows.

## Solution 12-2: Diagnosing Replication Lag by Using GTID Sets

---

### Solution Steps

1. Open a new Linux terminal window and issue the following command at the terminal prompt to stop both the replication master and replication slave servers:

```
# mysqld_multi --defaults-file=/root/labs/repl.cnf \
> --user=root stop 1-2
```

2. Open the /root/labs/repl.cnf file in a text editor, such as nano, vi, or emacs, and uncomment the following lines:

```
# gtid-mode=ON
# enforce-gtid-consistency
```

3. Start the replication master and replication slave servers by issuing the following command at the Linux terminal prompt:

```
# mysqld_multi --defaults-file=/root/labs/repl.cnf \
> --user=root start 1-2
```

4. In the terminal window with the connection to the replication master, issue STATUS to reconnect the connection.

Enter the following statement at the master> prompt and receive the results shown:

```
master> STATUS
ERROR 2013 (HY000): Lost connection to MySQL server during query
master>
```

5. In the terminal window with the connection to the replication slave, issue STATUS to reconnect the connection.

Enter the following statement at the slave> prompt and receive the results shown:

```
slave> STATUS
ERROR 2013 (HY000): Lost connection to MySQL server during query
slave>
```

6. Stop the replication slave.

Enter the following statement at the slave> prompt and receive the results shown:

```
slave> STOP SLAVE;
Query OK, 0 rows affected (#.## sec)
```

7. On both the replication master and the replication slave, issue the RESET MASTER command.

Enter the following statement at the master> prompt and receive the results shown:

```
master> RESET MASTER;
Query OK, 0 rows affected (#.## sec)
```

Enter the following statement at the slave> prompt and receive the results shown:

```
slave> RESET MASTER;
```

```
Query OK, 0 rows affected (#.## sec)
```

8. On the replication slave, issue the following CHANGE MASTER TO statement to instruct the master to use GTIDs:

```
slave> CHANGE MASTER TO MASTER_AUTO_POSITION=1;
Query OK, 0 rows affected (#.## sec)
```

9. Start the replication slave.

Enter the following statement at the slave> prompt and receive the results shown:

```
slave> START SLAVE;
Query OK, 0 rows affected (#.## sec)
```

10. On the replication master, insert a new row into the reptest.repltbl table with the following column values:

- id: Helen Jones
- email: hjones@mycompany.com

Enter the following statements at the master> prompt and receive the results shown:

```
master> USE reptest;
Database changed
master> INSERT INTO repltbl (name, email)
-> VALUES ('Helen Jones', 'hjones@mycompany.com');
Query OK, 1 row affected (#.## sec)
```

11. On the replication master, issue SHOW MASTER STATUS and note the last executed GTID set.

Enter the following statement at the master> prompt and receive the results shown:

```
master> SHOW MASTER STATUS\G
***** 1. row *****
      File: mysql1-bin.000001
      Position: 459
      Binlog_Do_DB:
      Binlog_Ignore_DB:
Executed_Gtid_Set: 8d2cb537-b4c2-11e7-bb69-3417eb9a345a:1
1 row in set (#.## sec)
```

12. On the replication slave, issue SHOW SLAVE STATUS and note the retrieved and executed GTID sets.

Enter the following statement at the slave> prompt and receive the results shown:

```
slave> SHOW SLAVE STATUS\G
...
      SQL_Delay: 600
      SQL_Remaining_Delay: 504
      Slave_SQL_Running_State: Waiting until MASTER_DELAY
seconds after master executed event
```

```

Master_Retry_Count: 86400
Master_Bind:
Last_IO_Error_Timestamp:
Last_SQL_Error_Timestamp:
Master_SSL_Crl:
Master_SSL_Crlpath:
Retrieved_Gtid_Set: 8d2cb537-b4c2-11e7-bb69-
3417eb9a345a:1

Executed_Gtid_Set:
Auto_Position: 1
Replicate_Rewrite_DB:
Channel_Name:
Master_TLS_Version:
1 row in set (#.## sec)

```

- The last retrieved GTID set is the one that was executed on the master. However, because of the simulated replication lag, this GTID set has not executed on the slave. The GTID set identifier will be different on your machine.

13. On the replication slave, query the `replication_connection_status` table in the Performance Schema to provide an alternative view of the retrieved GTIDs.

Enter the following statement at the `slave>` prompt and receive the results shown:

```

slave> SELECT * FROM
-> performance_schema.replication_connection_status\G
***** 1. row *****
CHANNEL_NAME:
GROUP_NAME:
SOURCE_UUID: 8d2cb537-b4c2-11e7-bb69-3417eb9a345a
THREAD_ID: 31
SERVICE_STATE: ON
COUNT_RECEIVED_HEARTBEATS: 66
LAST_HEARTBEAT_TIMESTAMP: 2017-10-19 15:06:29
RECEIVED_TRANSACTION_SET: 8d2cb537-b4c2-11e7-bb69-
3417eb9a345a:1
LAST_ERROR_NUMBER: 0
LAST_ERROR_MESSAGE:
LAST_ERROR_TIMESTAMP: 0000-00-00 00:00:00
1 row in set (#.## sec)

```

14. On the replication slave, keep querying the value of the global `gtid_executed` variable until the received transaction set identified in the preceding two steps executes.

Repeatedly enter the following statement at the `slave>` prompt and proceed to the next step when you receive the results shown:

```

slave> SELECT @@global.gtid_executed;
+-----+

```

```
| @@global.gtid_executed           |
+-----+
| 8d2cb537-b4c2-11e7-bb69-3417eb9a345a:1 |
+-----+
1 row in set (#.# sec)
```

15. On the replication slave, issue SHOW SLAVE STATUS.

Enter the following statement at the slave> prompt and receive the results shown:

```
slave> SHOW SLAVE STATUS\G
***** 1. row *****
...
          SQL_Delay: 600
          SQL_Remaining_Delay: NULL
          Slave_SQL_Running_State: Slave has read all relay log;
waiting for more updates
          Master_Retry_Count: 86400
          Master_Bind:
          Last_IO_Error_Timestamp:
          Last_SQL_Error_Timestamp:
          Master_SSL_Crl:
          Master_SSL_Crlpath:
          Retrieved_Gtid_Set: 8d2cb537-b4c2-11e7-bb69-
3417eb9a345a:1
          Executed_Gtid_Set: 8d2cb537-b4c2-11e7-bb69-
3417eb9a345a:1
          Auto_Position: 1
          Replicate_Rewrite_DB:
          Channel_Name:
          Master_TLS_Version:
1 row in set (0.00 sec)
```

- The retrieved GTID set has now executed, and the slave is waiting for more updates from the master.

16. The /root/scripts/repl-insert.sql script inserts multiple rows into the repltbl table in the repltest database. Execute the repl-insert.sql script on the replication master.

Enter the following statement at the master> prompt and receive the results shown:

```
master> SOURCE /root/scripts/repl-insert.sql
Reading table information for completion of table and column
names
You can turn off this feature to get a quicker startup with -A

Database changed
Query OK, 1 row affected (0.09 sec)
```

```

Query OK, 1 row affected (0.08 sec)

Query OK, 1 row affected (0.09 sec)

Query OK, 1 row affected (0.08 sec)

Query OK, 1 row affected (0.09 sec)

Query OK, 1 row affected (0.08 sec)

Query OK, 1 row affected (0.09 sec)

Query OK, 1 row affected (0.09 sec)

Query OK, 1 row affected (0.08 sec)

Query OK, 1 row affected (0.08 sec)

```

17. On the replication slave, query the `replication_connection_status` table in the Performance Schema to list the retrieved GTIDs.

Enter the following statement at the `slave>` prompt and receive the results shown:

```

slave> SELECT * FROM
-> performance_schema.replication_connection_status\G
***** 1. row *****
CHANNEL_NAME:
GROUP_NAME:
SOURCE_UUID: 8d2cb537-b4c2-11e7-bb69-3417eb9a345a
THREAD_ID: 31
SERVICE_STATE: ON
COUNT_RECEIVED_HEARTBEATS: 114
LAST_HEARTBEAT_TIMESTAMP: 2017-10-19 15:30:33
RECEIVED_TRANSACTION_SET: 8d2cb537-b4c2-11e7-bb69-
3417eb9a345a:1-12
LAST_ERROR_NUMBER: 0
LAST_ERROR_MESSAGE:
LAST_ERROR_TIMESTAMP: 0000-00-00 00:00:00
1 row in set (#.## sec)

```

18. On the replication slave, display the last executed GTID.

Enter the following statement at the `slave>` prompt and receive the results shown:

```
slave> SELECT @@global.gtid_executed;
+-----+
| @@global.gtid_executed |
+-----+
| 8d2cb537-b4c2-11e7-bb69-3417eb9a345a:1 |
+-----+
1 row in set (#.## sec)
```

- Because of the replication lag, the slave has not caught up with the updates on the master.

19. Use the `GTID_SUBTRACT()` function to calculate which GTIDs are missing, by executing the following statement on the replication slave, replacing `<retrieved GTIDs>` with the GTID set identifier that you determined in step 17.

```
SELECT GTID_SUBTRACT(<retrieved GTIDs>, @@global.gtid_executed)
AS MissingGTIDs, IF(GTID_SUBTRACT(<retrieved GTIDs>,
@@global.gtid_executed) = '', 'YES', 'NO') AS CaughtUp FROM
performance_schema.replication_connection_status;
```

Enter the following statement at the `slave>` prompt and receive results similar to those shown:

```
slave> SELECT GTID_SUBTRACT('8d2cb537-b4c2-11e7-bb69-
3417eb9a345a:1-12', @@global.gtid_executed) AS MissingGTIDs,
IF(GTID_SUBTRACT('8d2cb537-b4c2-11e7-bb69-3417eb9a345a:1-12',
@@global.gtid_executed) = '', 'YES', 'NO') AS CaughtUp FROM
performance_schema.replication_connection_status;
+-----+-----+
| MissingGTIDs | CaughtUp |
+-----+-----+
| 8d2cb537-b4c2-11e7-bb69-3417eb9a345a:2-12 | NO      |
+-----+-----+
1 row in set (#.## sec)
```

- In the sample output shown, transactions 2–12 from the GTID set have not yet been applied on the slave.

20. Close the `master>` and `slave>` sessions and their Linux terminal windows.

In the remaining Linux terminal windows, stop both the servers by issuing the following command:

```
# mysql_multi --defaults-file=/root/labs/repl.cnf \
> --user=root stop 1-2
```

21. Delete the contents of the `/var/lib/mysql1` and `/var/lib/mysql2` data directories.

Enter the following commands at the Linux terminal prompt:

```
# rm -rf /var/lib/mysql1
# rm -rf /var/lib/mysql2
```

22. Restart the usual MySQL server (the one that you have used for all the other practices in this course).

Enter the following command at the Linux terminal prompt and receive the results shown:

```
# service mysql start  
Starting MySQL... SUCCESS!
```

23. Close all open Linux terminal windows.

## **Practices for Lesson 13: Conclusion**

## Practices for Lesson 13: Overview

---

There are no practices for this lesson.

## **Appendix A: Performance Enhancements in MySQL 5.7**

## Appendix A: Overview

---

### Overview

The following summarizes some of the key changes in MySQL 5.7 that affect performance. Note that this list is not exhaustive and does not include many optimizations that are less visible to the database administrator.

## Features Added in MySQL 5.7

### InnoDB Enhancements

*Dynamic buffer pool sizing.* The `innodb_buffer_pool_size` parameter is dynamic, allowing you to resize the buffer pool without restarting the server. The resizing operation involves moving pages to a new location in memory, which the server does in chunks. Chunk size is configurable by using the new `innodb_buffer_pool_chunk_size` configuration option. You can monitor the progress of resizing by using the new `Innodb_buffer_pool_resize_status` status variable.

*Enhancements to buffer pool dump and load operations.* A new system variable, `innodb_buffer_pool_dump_pct`, allows you to specify the percentage of most recently used pages in each buffer pool to read out and dump. When InnoDB background tasks result in I/O activity, InnoDB attempts to limit the number of buffer pool load operations per second by using the `innodb_io_capacity` setting.

*Multi-threaded page cleaning.* InnoDB supports multiple page cleaner threads for flushing dirty pages from buffer pool instances. Use the new system variable, `innodb_page_cleaners`, to specify the number of page cleaner threads. The default value of 1 maintains the previous configuration in which there is a single page cleaner thread. This enhancement builds on work completed in MySQL 5.6, which introduced a single page cleaner thread to offload buffer pool flushing work from the InnoDB master thread.

*DDL performance for InnoDB temporary tables.* The performance of DDL operations on temporary tables has been improved by optimizing the `CREATE TABLE`, `DROP TABLE`, `TRUNCATE TABLE`, and `ALTER TABLE` statements.

*Spatial indexes.* InnoDB supports indexing of spatial data types by using `SPATIAL` indexes, including the use of `ALTER TABLE ... ALGORITHM=INPLACE` for online operations (`ADD SPATIAL INDEX`).

*Native partitioning.* Previously, InnoDB relied on the `ha_partition` handler, which creates a handler object for each partition. With native partitioning, a partitioned InnoDB table uses a single, partition-aware handler object. This enhancement reduces the amount of memory required for partitioned InnoDB tables.

### Performance Schema

*Configuration improvements:*

- Memory is allocated on demand – you do not need to restrict the size of individual tables.
- You can turn statistics on/off for a specific host or user.
- You can tune the size of `SQL_DIGEST`.

`sys` schema. MySQL distributions now include the `sys` schema, which is a set of objects that help DBAs and developers interpret data collected by the Performance Schema. Use the `sys` schema objects for typical tuning and diagnosis use cases.

*Metadata locking information.* The `metadata_locks` table tells you which thread is waiting for a lock and which thread holds the lock, and works with many database objects, not just tables.

*Table lock information.* The `table_locks` table provides not only information about table locks, but also statistics related to open tables.

*Memory diagnostics.* Before 5.7, it was very difficult to diagnose where MySQL uses memory. The new `memory_*` tables in Performance Schema enable you to do this.

*Stored routine instrumentation.* Performance Schema now collects statistics from stored routines, thus enabling you to determine what happens within the stored routine and which queries the stored routine executes.

*Prepared statements.* The new `prepared_statement_instances` table tells you which thread owns a given prepared statement and how many times the prepared statement executes, and also provides optimizer statistics, similar to that available in `events_statements_*`.

*Variables instrumentation.* System and status variables are now available in Performance Schema tables, and you should query `performance_schema` instead of `INFORMATION_SCHEMA` for this information. This also affects the operation of the `SHOW VARIABLES` and `SHOW STATUS` statements. The value of the `show_compatibility_56` system variable (OFF by default) affects the output produced from and the privileges required for system and status variable statements and tables. Applications that require 5.6 behavior should set this variable to ON until they have been migrated to the new behavior for system variables and status variables.

*Replication instrumentation.* Data from `SHOW SLAVE STATUS` is now available in the `replication_*` tables. The information collected in these tables is especially useful for GTID-based replication setups. The tables support replication channels for multi-threaded slaves.

*Group Replication Performance Schema tables.* MySQL 5.7 adds several new tables to the Performance Schema to provide information about replication groups and channels.

## Optimizer Enhancements

*Index condition pushdown support for partitioned tables.* Queries on partitioned tables by using the InnoDB or MyISAM storage engine may employ the index condition pushdown optimization.

*EXPLAIN for named connections.* You can use EXPLAIN to obtain the execution plan for an explainable statement that is executing in a named connection, by executing `EXPLAIN [options] FOR CONNECTION connection_id;`.

*Optimizer hints for individual statements.* You provide hints to the optimizer within individual SQL statements, which enables finer control over statement execution plans than can be achieved by using the `optimizer_switch` system variable. Hints are also permitted in the statements that are used with EXPLAIN, which enables you to see how hints affect execution plans.

## Features Deprecated or Removed in MySQL 5.7

### Overview

This section summarizes the main items that are deprecated or removed in MySQL 5.7. Most of these are scheduled for removal in MySQL 8.0.

*Query cache.* The query cache is deprecated and will be removed in MySQL 8.0. Deprecation includes the following items:

- The `FLUSH QUERY CACHE` and `RESET QUERY CACHE` statements
- The `SQL_CACHE` and `SQL_NO_CACHE` `SELECT` modifiers
- The following system variables: `have_query_cache`, `ndb_cache_check_time`, `query_cache_limit`, `query_cache_min_res_unit`, `query_cache_size`, `query_cache_type`, and `query_cache_wlock_invalidate`
- The following status variables: `Qcache_free_blocks`, `Qcache_free_memory`, `Qcache_hits`, `Qcache_inserts`, `Qcache_lowmem_prunes`, `Qcache_not_cached`, `Qcache_queries_in_cache`, and `Qcache_total_blocks`

*Metadata locks system variables.* The `metadata_locks_cache_size` and `metadata_locks_hash_instances` system variables are deprecated. These do nothing as of MySQL 5.7.4.

*Extended EXPLAIN.* The `EXTENDED` and `PARTITIONS` keywords for the `EXPLAIN` statement are deprecated. These keywords are still recognized but are now unnecessary because their effect is always enabled.

*Information Schema PROFILING table.* The `INFORMATION_SCHEMA.PROFILING` table is deprecated. Use the Performance Schema instead.

*Information Schema INNODB\_LOCKS and INNODB\_LOCK\_WAIT tables.* These are deprecated and will be entirely replaced in MySQL 8.0 by Performance Schema tables.

*Performance Schema timing information.* The Performance Schema `setup_timers` table is deprecated and will be removed in MySQL 8.0, as will the `TICK` row in the `performance_timers` table.

*SHOW ENGINE INNODB MUTEX output has been removed.* Comparable information can be generated by creating views on Performance Schema tables.

*InnoDB table and tablespace monitors.* The InnoDB Tablespace Monitor and InnoDB Table Monitor are removed. For the Table Monitor, equivalent information can be obtained from the InnoDB INFORMATION\_SCHEMA tables.

*InnoDB monitor/lock monitor.* The specially named tables that are used to enable and disable the standard InnoDB Monitor and InnoDB Lock Monitor (`innodb_monitor` and `innodb_lock_monitor`) are removed and replaced by two dynamic system variables: `innodb_status_output` and `innodb_status_output_locks`.

GANG LIU (gangli@baylorhealth.edu) has a non-transferable license  
to use this Student Guide.

## **Appendix B: Server Tuning Checklist**

## Appendix B: Overview

---

### Overview

MySQL 5.7 ships with excellent default settings for most workloads and you will usually not need to change anything unless you have identified that there is a performance issue. However, the following InnoDB settings are largely dependent on the amount of RAM that your server has available and adjusting these accordingly might result in performance benefits.

Follow the advice in the lesson tilted “Performance Tuning Concepts” when considering a change to the default values, which can be summarized as follows:

- Benchmark your existing system.
- Make small changes to the existing values, one setting at a time.
- Benchmark again.
- Repeat as required.

### Core InnoDB Settings

#### Sizing the InnoDB Buffer Pool

Start by setting `innodb_buffer_pool_size` to 50% of the total system RAM, and increase if necessary, up to around 75%. The value of `innodb_buffer_pool_size` never needs to be larger than the size of the database.

#### Sizing the InnoDB Log Files

The `innodb_log_file_size` setting controls the size of each log file in a log group. Generally, the combined size of the log files should be large enough that the server can smooth out peaks and troughs in workload activity, which often means that there is enough redo log space to handle more than an hour of write activity. The larger the value, the less checkpoint flush activity is required in the buffer pool, which reduces the amount of disk I/O. Start with a value of 128 MB and increase as necessary up to a maximum of 2 GB. There is no benefit to setting `innodb_log_file_size` to a value that is larger than `innodb_buffer_pool_size`.

#### Frequency of InnoDB Log File Flushing

The default value of `innodb_flush_log_at_trx_commit` is 1. This setting controls the balance between strict ACID compliance for commit operations and the higher performance that is possible when MySQL executes commit-related I/O operations in batches. You can achieve better performance by changing the default value to 0 or 2, but then you can lose up to a second's worth of transactions in a crash.

#### InnoDB Flush Method

The `innodb_flush_method` setting defines the method that is used to flush data to InnoDB data and log files, which affect I/O throughput. The default value is `NULL`, which uses `fsync` on Unix-like systems and `async_unbuffered` on Windows. You can achieve a performance benefit by setting this option to `O_DIRECT`. This instructs InnoDB to use `O_DIRECT` (or

`directio()` on Solaris) to open the data files, and uses `fsync()` to flush both the data and log files. This option is available only on some GNU/Linux versions, FreeBSD, and Solaris. It is not available on Windows.

## Optional Settings

The settings listed in the preceding paragraphs provide a good basis for a new installation of MySQL 5.7. Consider the following settings only if they are applicable to your workload and environment.

### Lock Mode for Autoincrement Variables

You might want to set `innodb_autoinc_lock_mode` to 2, which is “interleaved” mode if you are using `binlog_format=ROW` (the default) or `binlog_format=MIXED`. This is the fastest and most scalable lock mode, but it is not safe when using statement-based replication or recovery scenarios that involve replaying SQL statements from the binary log.

### InnoDB Flush Capacity

Consider changing these settings only for write-intensive workloads and if you know how many IOPS (input/output operations per second) your storage devices can achieve. If you have a fast disk (such as an SSD), increasing the values of these settings can improve the speed at which InnoDB flushes dirty pages to disk. Set `innodb_io_capacity_max` to the maximum setting that your storage device supports and `innodb_io_capacity` to 50% of the maximum value.

GANG LIU (gangli@baylorhealth.edu) has a non-transferable license  
to use this Student Guide.

## **Appendix C: Using MySQL with Oracle Tools**

## Appendix C: Overview

---

### Overview

If you currently work with Oracle Databases, you will probably be using Oracle tools to connect to, administer, monitor, and develop against those databases. This appendix shows you how you can use Oracle Enterprise Manager and SQL Developer to work with MySQL databases.

# Using Oracle Enterprise Manager with the MySQL Plug-In

## Overview

The Oracle Enterprise Manager 13c Enterprise Manager System Monitoring Plug-in for MySQL enables you to monitor availability, performance, and configuration information for MySQL instances alongside your Oracle databases. This section demonstrates the functionality included in the Oracle Enterprise Manager System Monitoring Plug-in for MySQL.

## Central Management

- After you log in to Enterprise Manager Cloud Control, the Welcome page appears. Select the Targets > All Targets menu option at the top of the page:

- The All Targets option presents you with a list of the many different types of instances managed by Oracle Enterprise Manager:

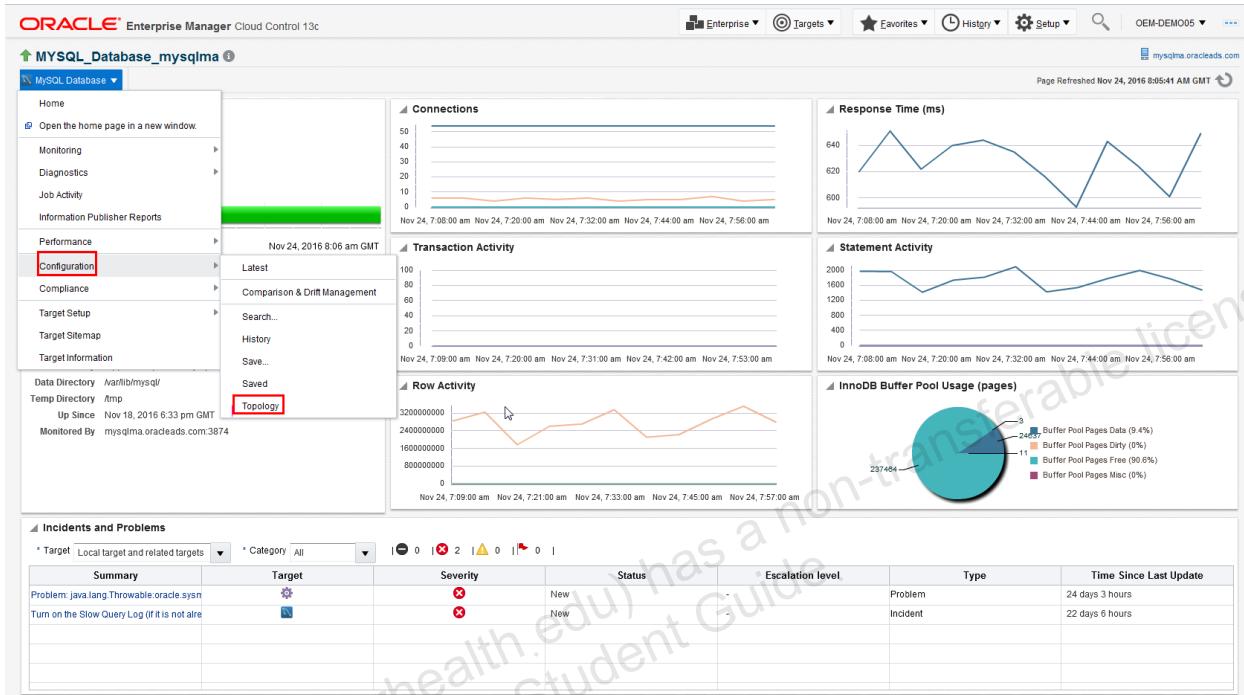
The screenshot shows the Oracle Enterprise Manager Cloud Control 13c interface. The top navigation bar includes links for Enterprise, Targets, Favorites, History, Setup, and OEM-DEMO05. The main content area is titled "All Targets". On the left, a "Refine Search" sidebar lists categories such as Target Type (Cloud, Databases, Groups, Systems and Services, Middleware, Servers, Storage and Network), Target Status (Down, Up, N/A), and Target Version (12.1.3.0, 12.1.3.0.0). A search bar at the top of the sidebar allows searching by target name. The main list of targets includes entries for various Oracle components, with several MySQL instances highlighted in green. At the bottom right, there are buttons for Save Search, Target Status, and a summary of Discovered Targets (6) and Targets Found (40).

- Click the MySQL Databases link in the Refine Search pane on the left-hand side of the page to filter the list so that it displays only MySQL instances:

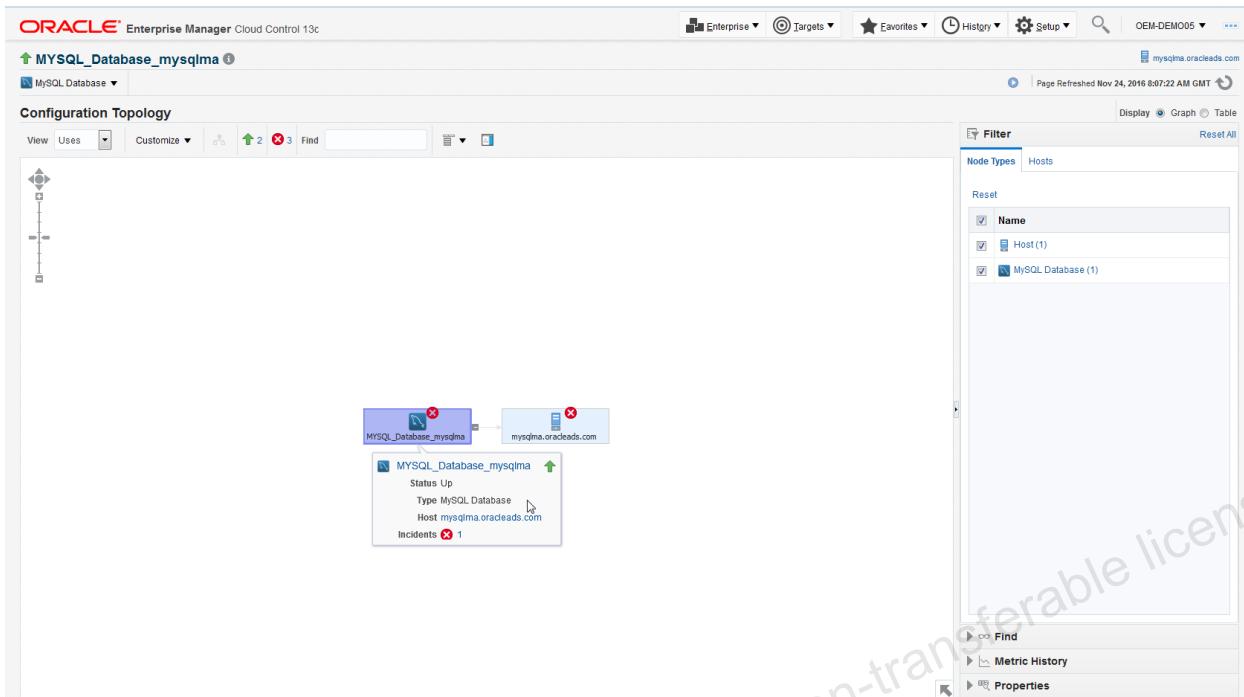
This screenshot shows the same Oracle Enterprise Manager interface after applying the MySQL Database filter. The Refine Search pane now highlights "MySQL Database (4)". The main list of targets is much shorter, displaying only the four MySQL instances that were previously filtered. The rest of the targets from the first screenshot are no longer visible.

- Click a MySQL instance in the list to show the dashboard page for the instance. This high-level dashboard shows configuration details, incidents and problems, and other metrics. On the left-hand side, you can see Availability History. This shows how long your instance has been gathering statistics as a percentage of the past week. The Configuration summary shows details of your instance such as the TCP/IP port. There are several performance charts covering the past 45 minutes. You can check here for spikes or dips in connections,

response time, transaction activity, statement activity, row activity, and buffer pool usage. The Response Time (ms) chart shows the amount of time a simple SELECT statement issued by a client takes to reach the MySQL server. The Incidents and Problems panel on this screen highlights any issues with the system. You can drill down into any of the reported issues for suggestions on how to solve the problem:



- To view the topology, select the Configuration > Topology option from the MySQL Database menu, as shown in the preceding screenshot.
- Click the red X next to the MySQL Dolphin icon and hover the cursor over the box that represents this MySQL instance. You will see a description of the instance and the number of reported incidents associated with it:



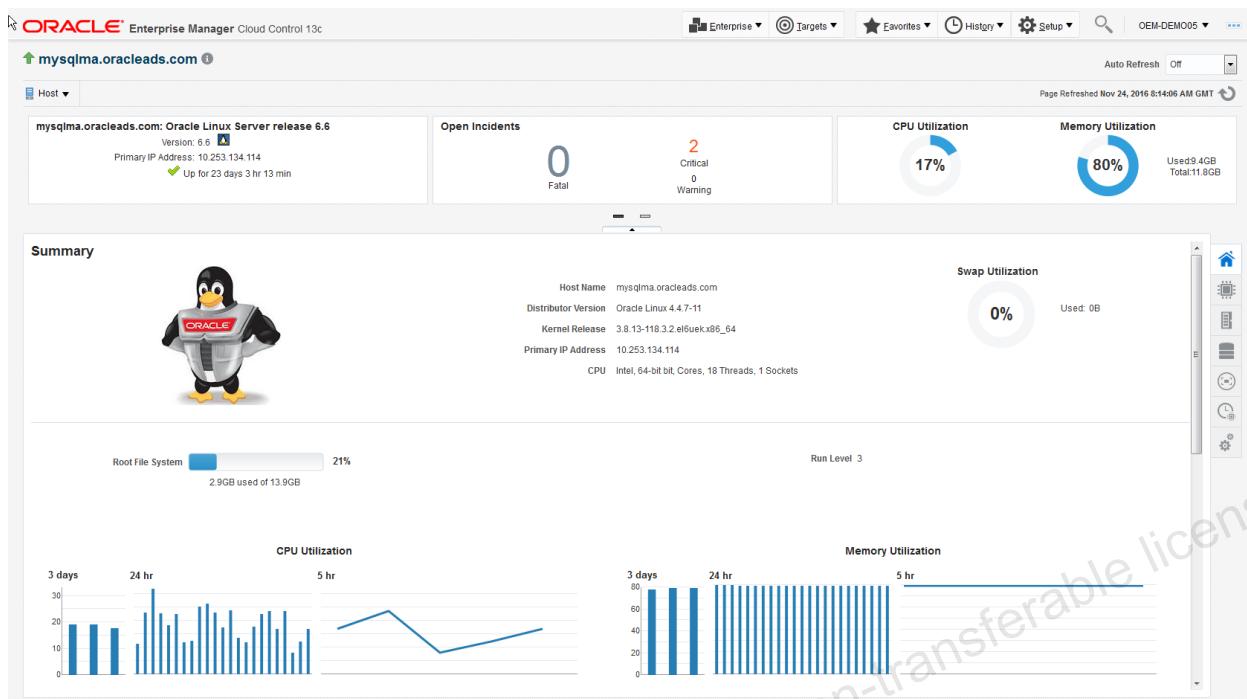
- Click the number of incidents to see the list of incidents, with additional information:

| Severity                                                                                                   | Target   | Priority | Status | Age         | Time Since Last Update | Owner | Ackn Escal | Type | Category |      |
|------------------------------------------------------------------------------------------------------------|----------|----------|--------|-------------|------------------------|-------|------------|------|----------|------|
| Turn on the Slow Query Log (if it is not already turned on) and monitor what goes into it. Statements t... | MySQL... | None     | New    | 22 days ... | 22 days 6 hours        | -     | No         | No   | Incident | Load |

- Click an incident in the list to see the details of OEM's recommendation for resolving the issue:

9. Click the Back button twice to return to the topology view.
10. Click the machine name that represents the host that the MySQL instance is running on to inspect the hardware. Note the description in the box that gives more information about memory utilization, CPU utilization, disk usage, and swap space. You can see the number of open incidents (if any) on the machine.

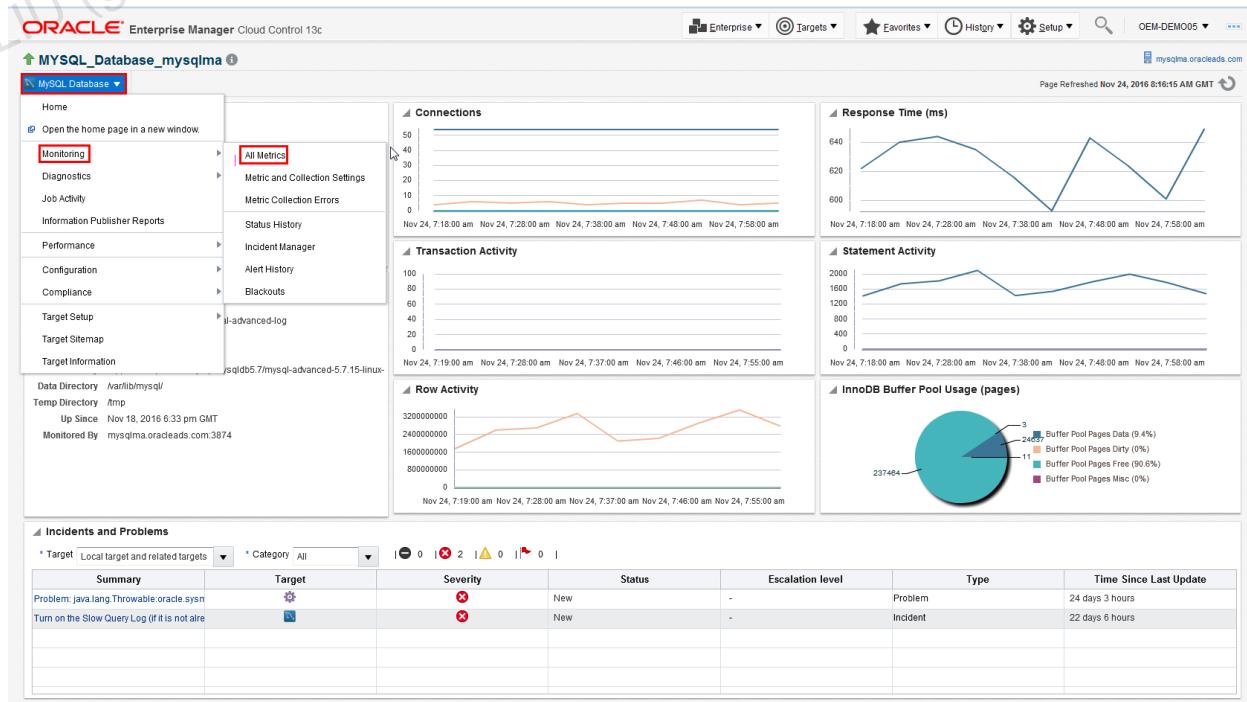
11. Click the number of incidents reported for the host to see more details:



## Real-Time Monitoring

The following demonstrates some of the metrics that OEM provides that help record real-time activities within the MySQL instance.

12. Click Back to return to the topology view, and then click the MySQL instance.
13. From the MySQL Database menu in the left-hand pane, select the Monitoring > All Metrics option:



14. The All Metrics page appears. From the list of metric categories on the left-hand side of the page, select Firewall Activity:

15. The following screenshot highlights the Access Denied (Delta) metric category. Click the Access Denied (Delta) metric to see what thresholds are set:

| Metric                    | Thresholds | Real Time Value |
|---------------------------|------------|-----------------|
| Access Denied (Delta)     | Set        |                 |
| Access Granted (Delta)    | Not Set    |                 |
| Access Suspicious (Delta) | Set        |                 |
| Cached Entries (Delta)    | Not Set    |                 |

16. In the example screenshot shown as follows, you can see that this metric raises a warning event if there are five “access denied” incidents or a critical event if there are 10 or more. No access denied events were raised since the alert was created.

The screenshot shows the Oracle Enterprise Manager interface for MySQL Database metrics. The left sidebar lists various metrics categories under 'All Metrics'. The 'Access Denied (Delta)' category is selected and expanded, showing its description: 'The number of statements rejected by MySQL Enterprise Firewall'. On the right, there are sections for 'Statistics' (with 'Last Known Value' as 'No data') and 'Thresholds' (set to 'Warning / Critical 5 / 10' with 'Comparison Operator >=' and 'Occurrences Before Alert 1'). Below these are sections for 'Metric Value History' and 'Table View'.

17. Check whether the connection limit has been reached. Start by clicking the Threads Activity metric category:

This screenshot is similar to the previous one but shows the 'Threads Activity' category selected in the left sidebar. The 'Threads Activity' category includes metrics like 'InnoDB Adaptive Hash Activity', 'InnoDB Bufferpool Activity', and 'InnoDB Connection Activity'. The 'Access Denied (Delta)' category is still visible in the list.

18. In the list of Threads Activity category metrics, click Connection Limit Usage (Rate):

**All Metrics**

Threads Activity

| Metric                               | Thresholds | Real Time Value |
|--------------------------------------|------------|-----------------|
| Cached                               | Not Set    | 0               |
| Connected                            | Not Set    | 54              |
| <b>Connection Limit Usage (Rate)</b> | Set        | 21.6            |
| Created (Delta)                      | Not Set    | 0               |
| Running                              | Not Set    | 4               |
| Slow Launch Threads (Delta)          | Not Set    | 0               |
| Thread Pooling Not Enabled           | Set        | 0               |
| Too Many Concurrent Queries Running  | Set        | 4               |

Data shown in above table is collected in real time.

19. In the following example screenshot, you can see that the connection limit threshold is set to warn you when 85 connections exist and to raise a critical event when there are 95 connections. So far, the threshold has not been reached:

**All Metrics**

**Connection Limit Usage (Rate)**

Description: Once the max conn limit for the MySQL server is reached, no other user conn's can be established and errors occur on the client.

| Statistics                                        | Thresholds                  |
|---------------------------------------------------|-----------------------------|
| Last Known Value: 21.6                            | Warning / Critical: 85 / 95 |
| Collection Timestamp: Nov 24, 2016 8:17:31 AM GMT | Comparison Operator: >      |
| Average Value: 21.6                               | Occurrences Before Alert: 1 |
| Low / High Value: 21.2 / 22                       | Corrective Actions: None    |

**Metric Value History**

24  
20  
16  
12  
8  
4  
0

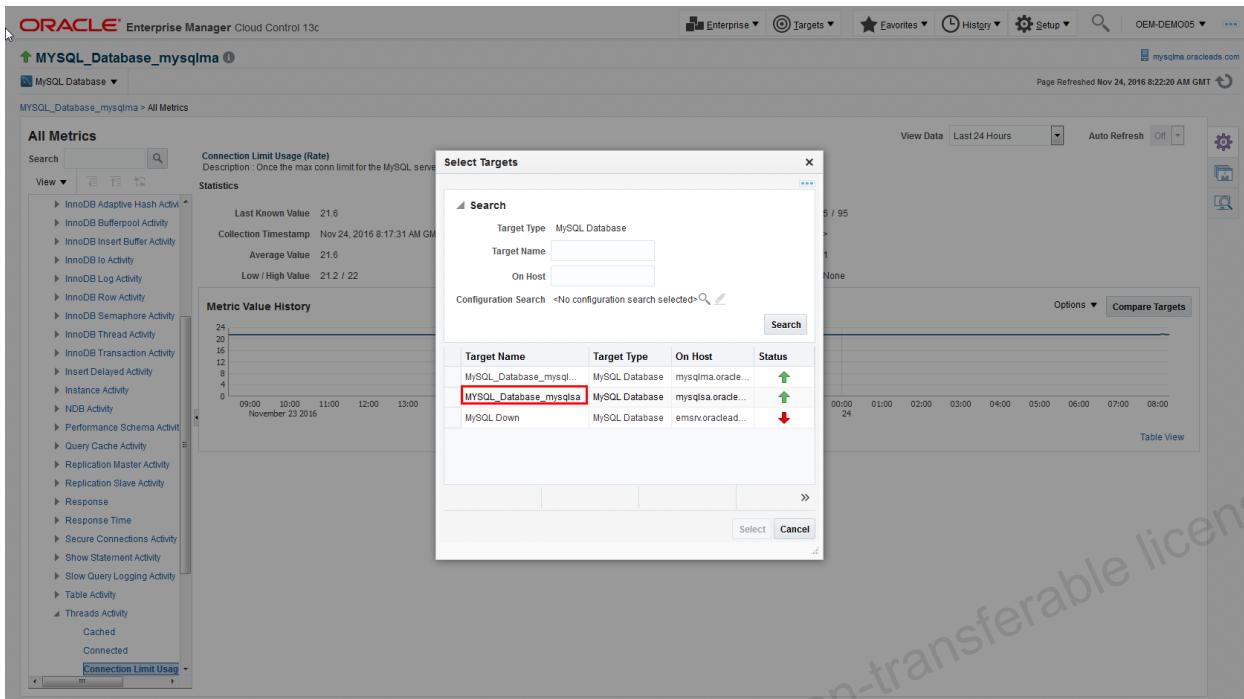
09:00 10:00 11:00 12:00 13:00 14:00 15:00 16:00 17:00 18:00 19:00 20:00 21:00 22:00 23:00 00:00 01:00 02:00 03:00 04:00 05:00 06:00 07:00 08:00

November 23 2016

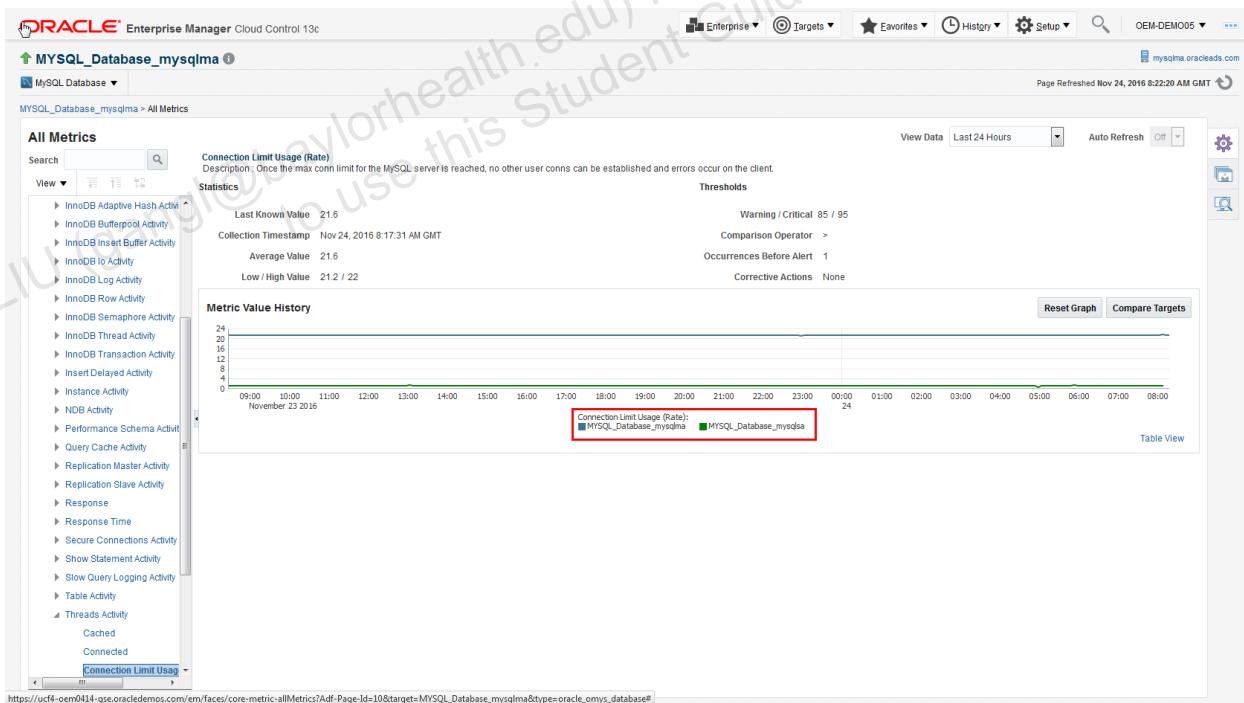
■ Connection Limit Usage (Rate)

Options ▾ Compare Targets

20. So far, you considered only a single MySQL instance. In the next steps, you will consider the status of a slave in relation to its master. Click the Compare Targets button shown in the screenshot in the preceding step.
21. In the Select Targets dialog box, select the Target Name of the slave and click the Select button:



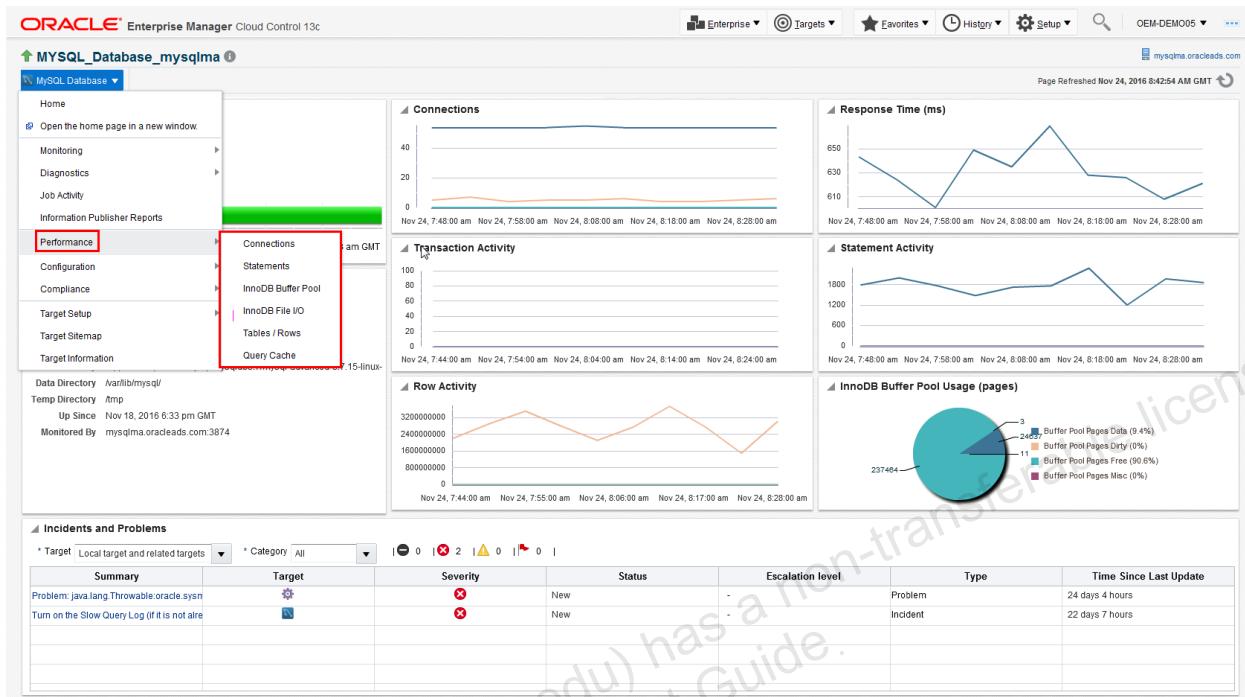
22. The graph that displays shows the status of the master in blue and the slave in green for this metric [Connection Limit Usage (Rate)]:



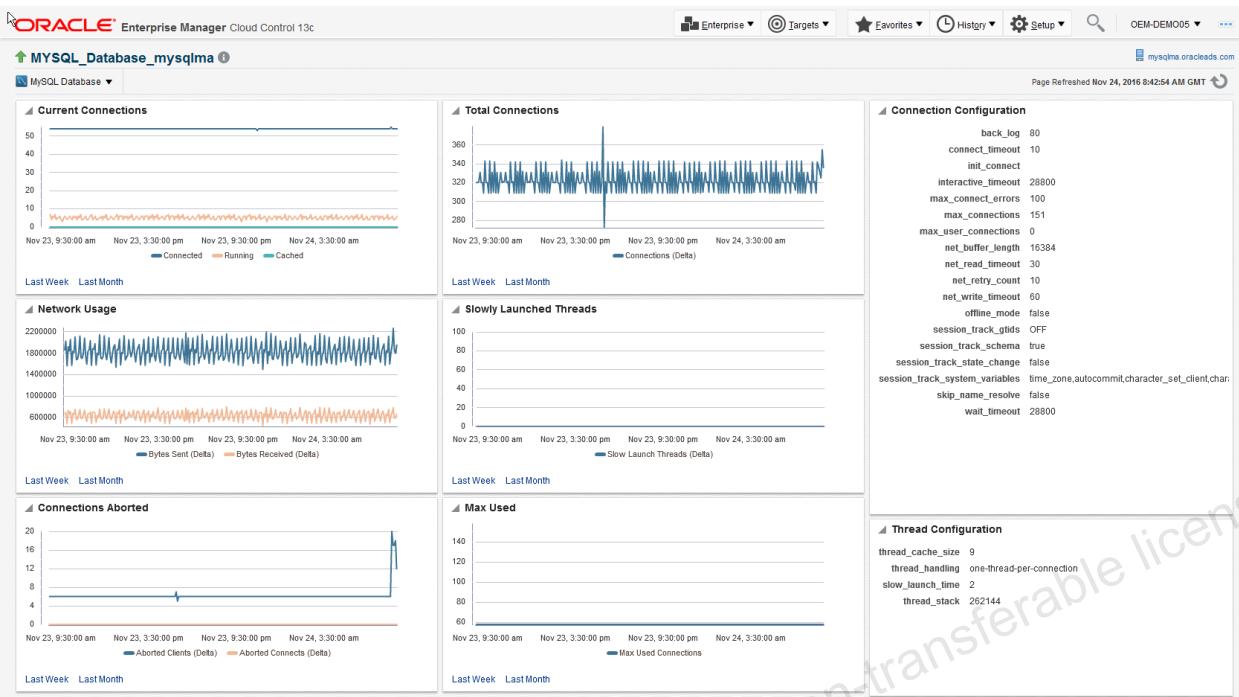
## Diagnosing Performance Issues

23. Click the Targets button at the top of the page, and select All Targets from the drop-down list that displays. You see a list of all the different instances managed by Oracle Enterprise Manager.
24. In the Refine Search pane on the left-hand side, select MySQL Databases.

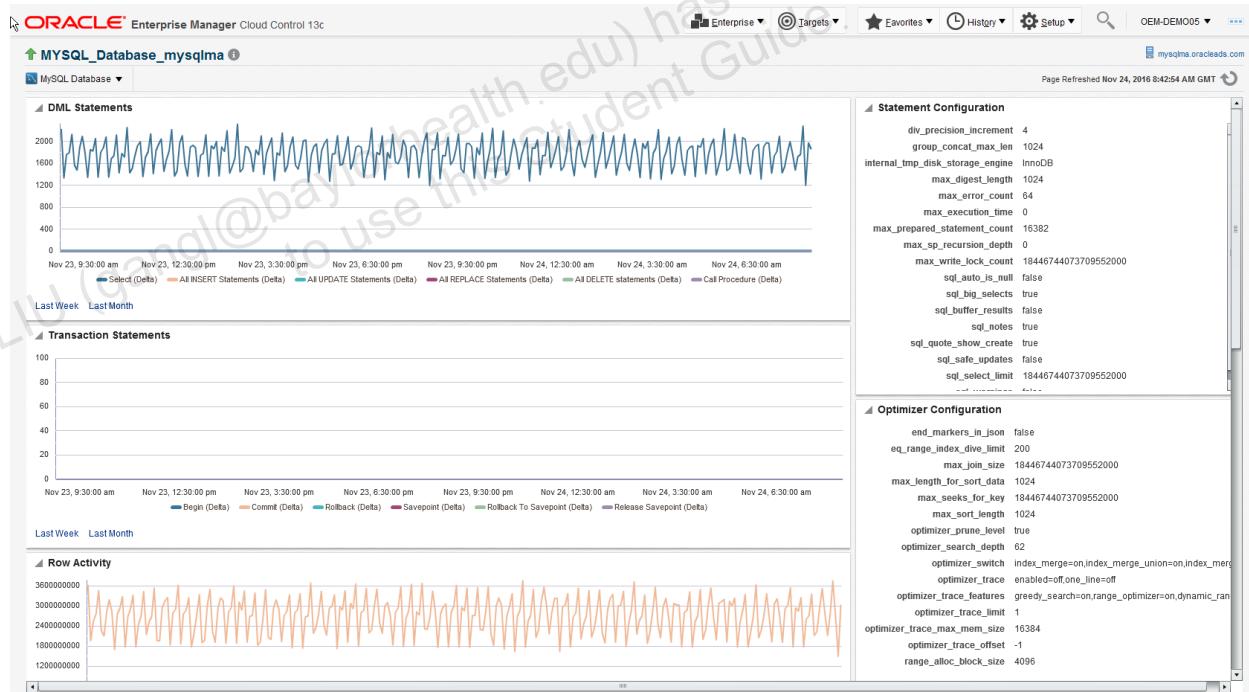
25. Click the name of a MySQL instance to view its dashboard page.
26. Select the Performance option in the MySQL Database menu. A submenu displays showing the performance metrics that are available. Select the Connections option:



When looking at each performance area, you see a chart of the current settings on the left-hand side of the screen. Each chart shows metrics for the past 24 hours by default, but you can also view metrics for the past week and past month. The configuration details for these metrics appear on the right. This makes it easy to compare the configuration of the server variables against the current metrics. You may want to change the settings depending on what you see.

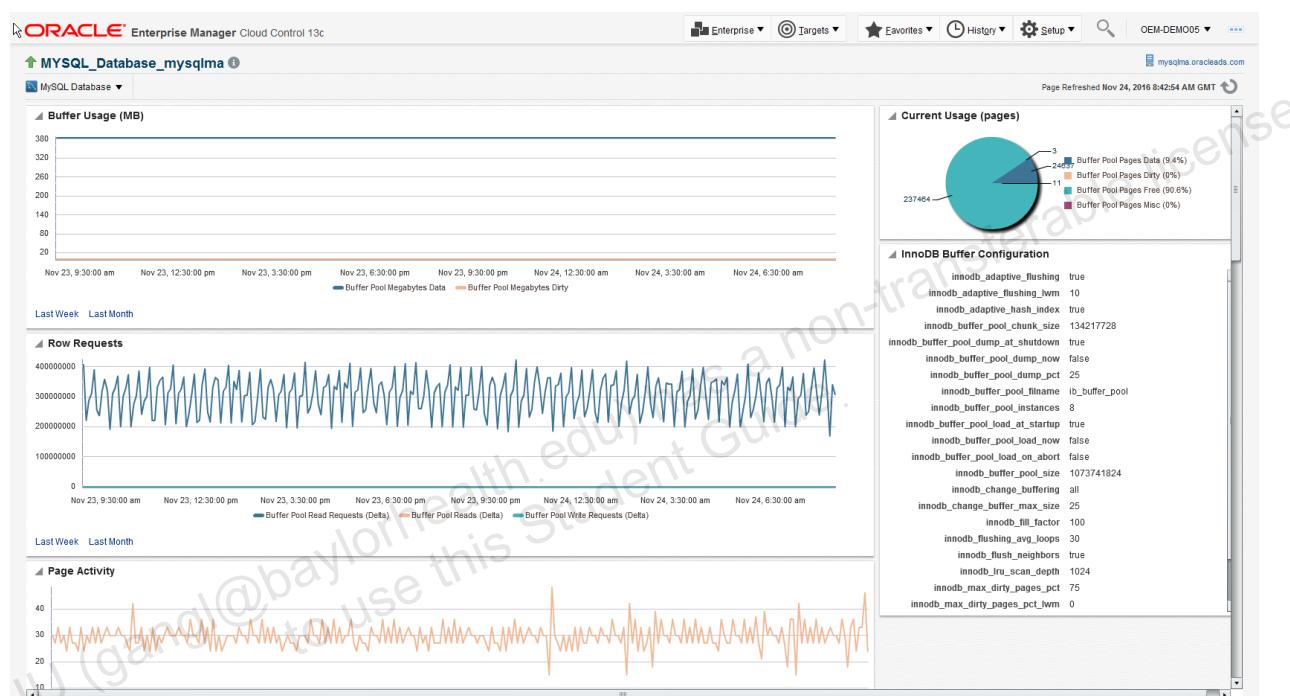


27. Select the MySQL Database > Performance > Statements menu option:



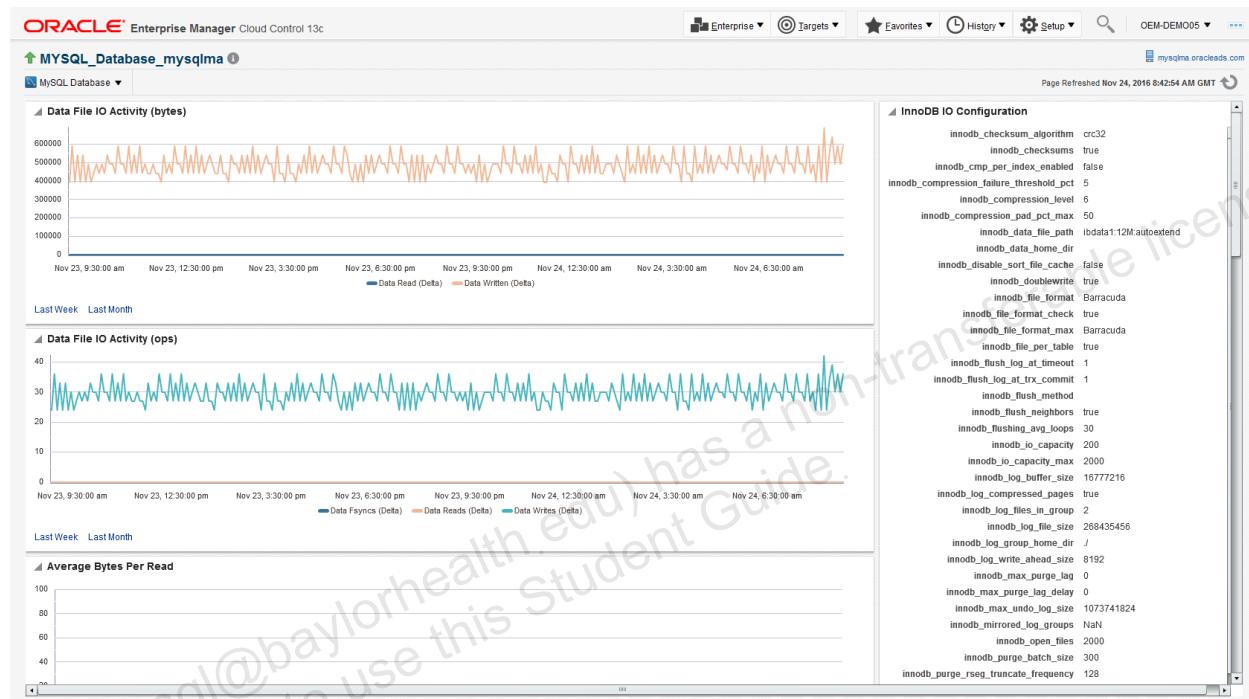
- The “DML Statements” chart includes SELECTS, INSERTS, UPDATES, and DELETES. The legend indicates how each type of statement is represented.
  - The “Transaction Statements” chart summarizes the number of transaction control statements that have executed: BEGIN, COMMIT, ROLLBACK, SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT.

- The “Index Usage Ratio” chart shows the delta in rows accessed by indexes compared to rows accessed by table scans. If you see slow queries, you can look here to check index usage.
  - The “Temporary Tables” chart displays the number of temporary tables created and how many were created on disk. Sorting often involves using temporary tables, so you can compare these metrics to those in the “Sort Activity” chart.
  - The “Statement Activity” summary on the right displays the current configuration settings for all metrics reported in the Statements section.
28. Select the MySQL Database > Performance > InnoDB Buffer Pool menu option:



- The “Data File IO Activity (bytes)” chart shows read/write activity in bytes.
- The “Data File IO Activity (ops)” chart shows the number of read/write fsync operations.
- The “Average Bytes Per Read” chart shows the number of bytes per read (associated with page size; the default is 16K).
- The “Double Write Activity” chart shows the number of double-write operations and the number of pages per write.
- “Redo Log IO Activity” shows the bytes that were written to the redo log (transaction log).
- “Redo Log Activity (ops)” shows the number of writes to the redo log.
- “Redo Log Waits” shows the number of times the buffer was too small to accommodate all the entries, resulting in them being written to disk. Ideally, this number should be zero.
- “Pending IO” shows the number of pending async I/O requests.
- “Pending Flushes” shows pending buffer and redo log flush operations.

- “Redo Log Waits,” “Pending IO,” and “Pending Flushes” should ideally be close to zero.
  - The “Open Files” metric shows how many files are open. The default is one file per table or partition for InnoDB tables.
  - The server settings associated with these InnoDB metrics are summarized in the “InnoDB Buffer Configuration” section on the right-hand side.
29. Select the MySQL Database > Performance > InnoDB File I/O menu option:



- The “Open Tables” chart on the left shows the total number of tables that have been opened.
- The “Currently Open Tables” chart shows the number of tables in the table cache. If “Opened Tables” spikes and “Currently Open Tables” dips frequently, you might need to increase the size of the table cache.
- The “Temporary Tables” chart shows the number of tables that were created and the proportion of those that were written to disk.
- The “Row Reads” chart shows the average number of index hits per second versus table scans.
- The “Row Writes” chart shows the number of requests to write a row. This might be very high during certain routine operations, such as a backup. The highest spike in the chart in the screenshot corresponds with a daily backup, which occurs between 1 AM and 2 AM GMT.
- “Sorts” shows the number of sorts performed. “Rows Sorted” shows the number of sorted rows. There is a correlation between these two charts.
- The “Sort Merge Passes” chart displays the number of merge passes performed by the sort algorithm, which should ideally be close to zero.

- The “Table Locks” chart shows the number of times a table lock request was granted and the number of times it could not be granted. Ideally, you want the value of “Table Locked Waited” to be low.
- The “Table Scan Ratio” chart shows the ratio of hits via indexes versus table scans as the number of rows accessed.
- The “Table Configuration” section on the right summarizes the relevant server variables.

# Using Oracle SQL Developer to Connect to MySQL Databases

## Overview

Oracle SQL Developer allows you to inspect MySQL database objects and execute queries against MySQL databases. However, not all functionality within SQL developer is available for MySQL connections. To work with MySQL databases within Oracle SQL Developer, you must first download and install an appropriate JDBC driver for MySQL and configure SQL Developer to use it.

## Installing the Driver

1. Download the official MySQL Connector/J JDBC driver from one of the following locations:
  - <https://support.oracle.com>: Enterprise Edition software for Oracle customers (Patches and Updates Tab, Product Search)
  - <https://edelivery.oracle.com>: For Enterprise Edition trial software (Select Product Pack: MySQL Database)
  - <https://dev.mysql.com/downloads/connector/j/>: For the Community (GPL) edition
2. Install Connector/J by extracting the .jar file from the downloaded archive to the required location on your machine—for example, using the Enterprise Edition (commercial) version of Connector/J:

```
# cd /stage/MySQL-Connectors/connector-j
# unzip ~/Downloads/mysql-connector-java-commercial-5.1.44.zip
Archive: /root/Downloads/mysql-connector-java-commercial-5.1.44.zip
  creating: mysql-connector-java-commercial-5.1.44/
  inflating: mysql-connector-java-commercial-5.1.44/CHANGES
  inflating: mysql-connector-java-commercial-5.1.44/LICENSE.mysql
  inflating: mysql-connector-java-commercial-5.1.44/README
  inflating: mysql-connector-java-commercial-5.1.44/README.txt
  inflating: mysql-connector-java-commercial-5.1.44/mysql-connector-
java-commercial-5.1.44-bin.jar
# ls /stage/MySQL-Connectors/connector-j/mysql-connector-java-
commercial-5.1.44/
CHANGES      mysql-connector-java-commercial-5.1.44-bin.jar
README.txt
LICENSE.mysql README
```

3. If this is the first time that you have used SQL Developer on this machine, you must first specify the location of the Java JDK. Launch `sqldeveloper` from a Linux terminal window and you are prompted to enter the location of the JDK:

```
# sqldeveloper

Oracle SQL Developer
Copyright (c) 1997, 2017, Oracle and/or its affiliates. All
rights reserved.
```

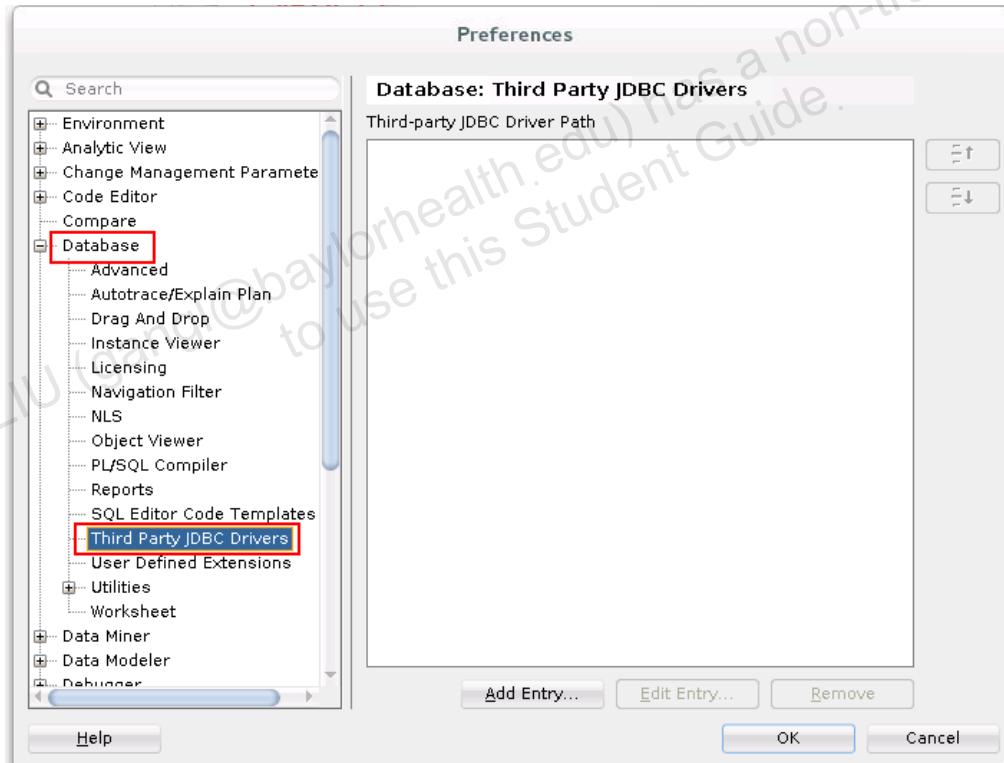
```
Type the full pathname of a JDK installation (or Ctrl-C to quit), the path will be stored in
/root/.sqldeveloper/17.3.1/product.conf
/usr/lib/jvm/java-1.8.0-openjdk
```

**Notes:**

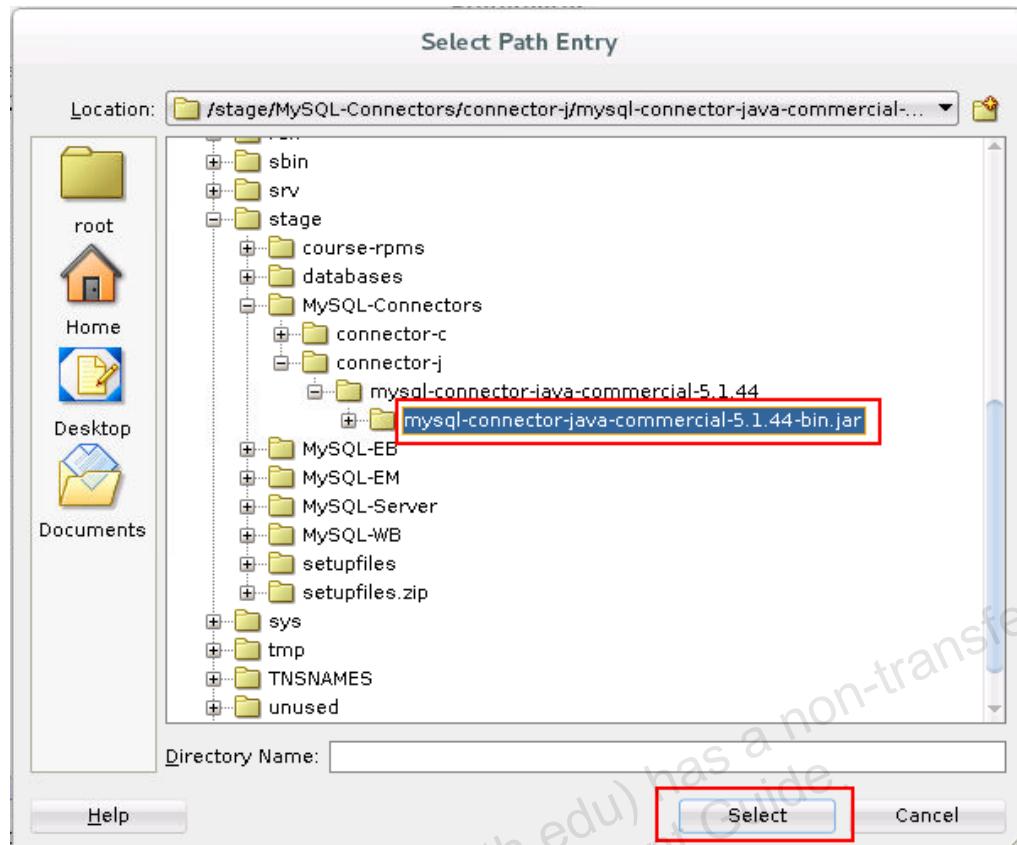
- See <http://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/index.html> for the correct JDK to use for your version of SQL Developer.
  - You can also edit the `~/.sqldeveloper/<version>/product.conf` file to specify the location of the JDK.
4. If you have completed the previous step, or have an existing installation of SQL Developer, launch it from the Linux desktop menu.

**Registering the MySQL JDBC Driver**

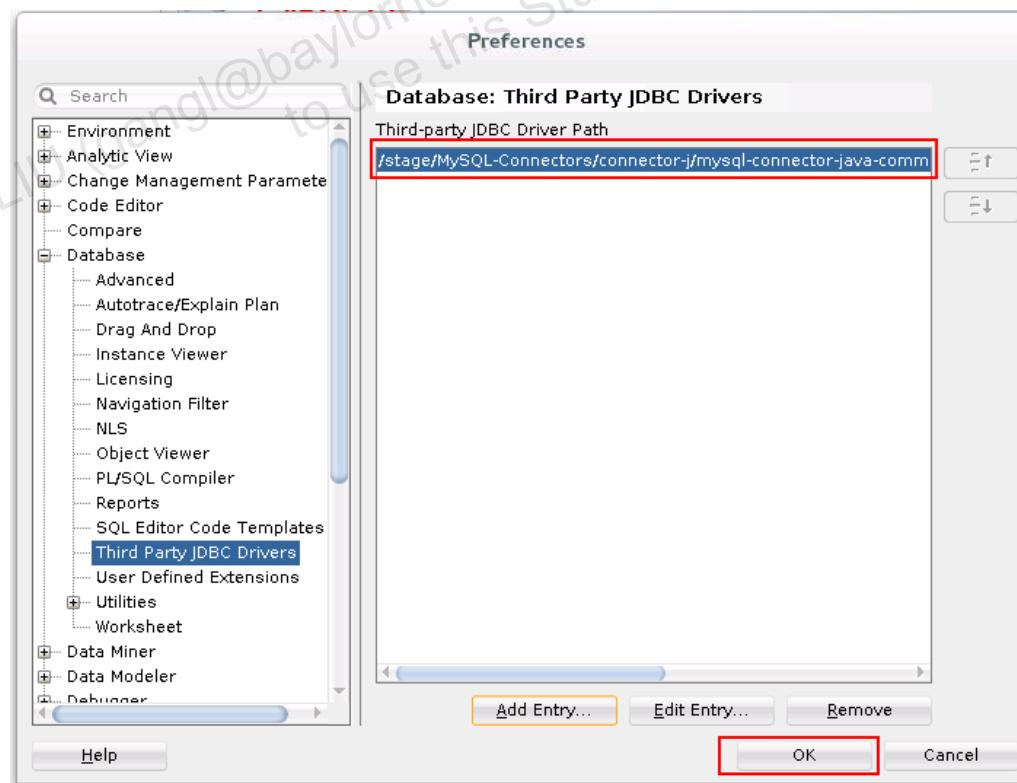
5. In SQL Developer, select the Tools > Preferences menu option and navigate to the Database > Third Party JDBC Drivers setting on the left-hand side of the Preferences pane:



6. Click the Add Entry button, browse to the Connector/J .jar file, and click Select:



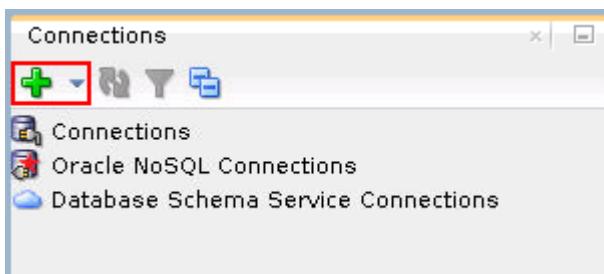
7. Verify that the MySQL Connector/J driver is listed as a third party JDBC driver and click OK:



8. Restart SQL Developer to apply the changes.

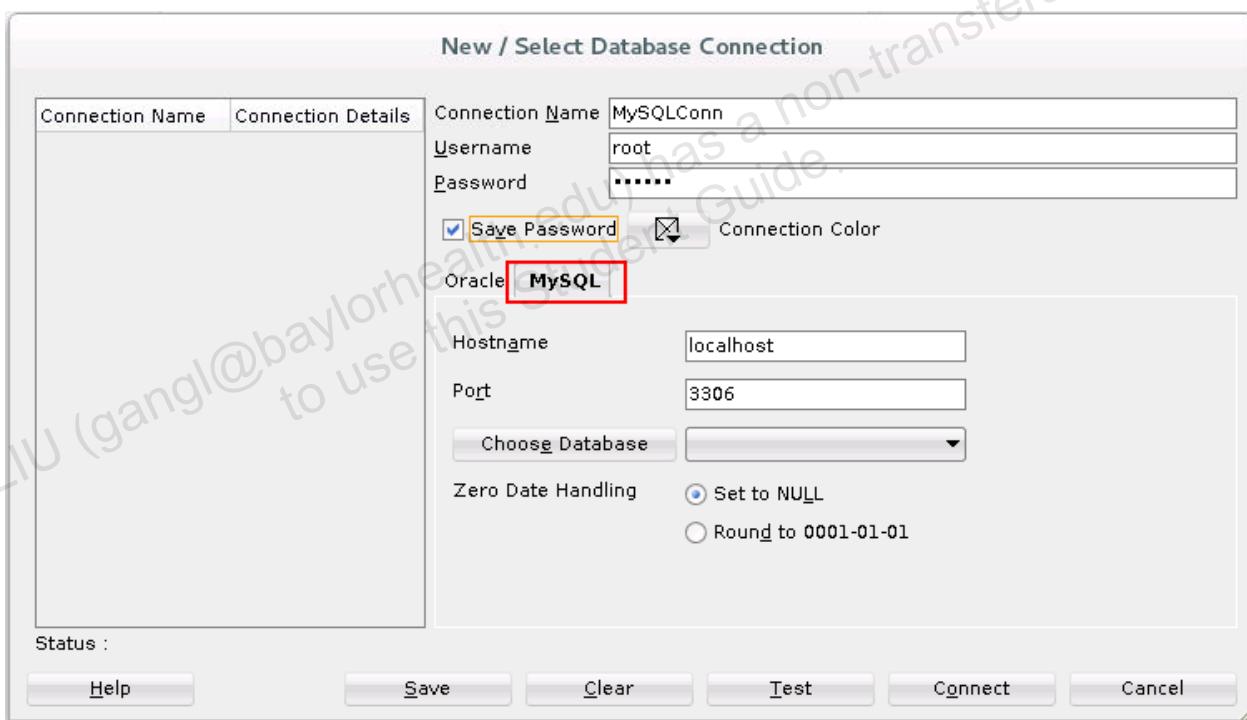
## Connecting to MySQL

9. Click the plus symbol in the Connections pane:

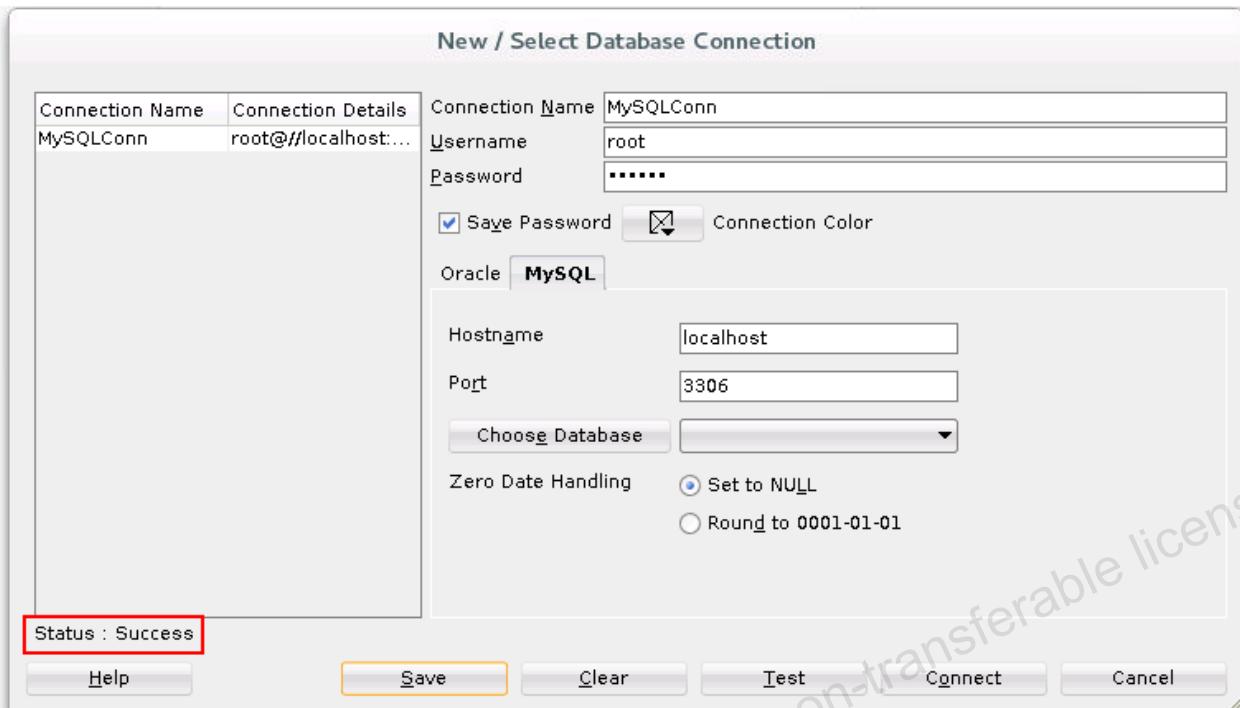


10. In the New/Select Database Connection dialog box, open the MySQL tab and enter the following details:

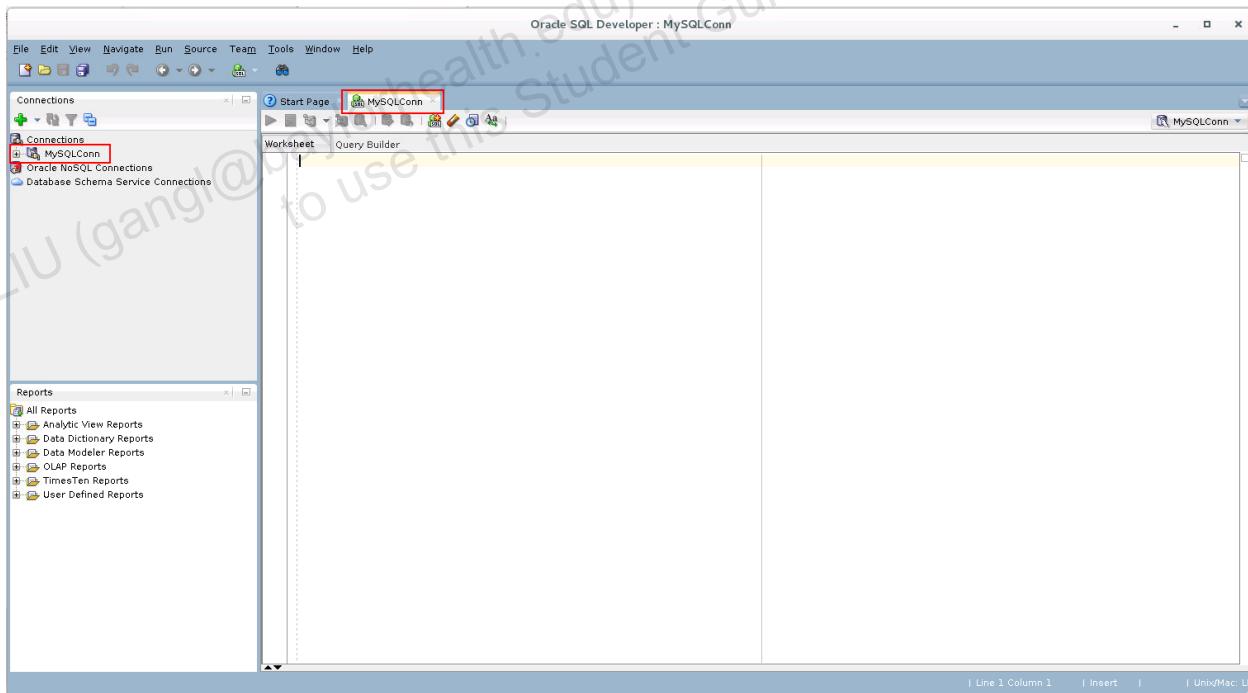
- Connection Name: A name for the connection
- Username and Password
- Hostname and Port



11. Click Save, and then click Test. If the connection succeeds by using the supplied parameters, the Status entry reads as follows:

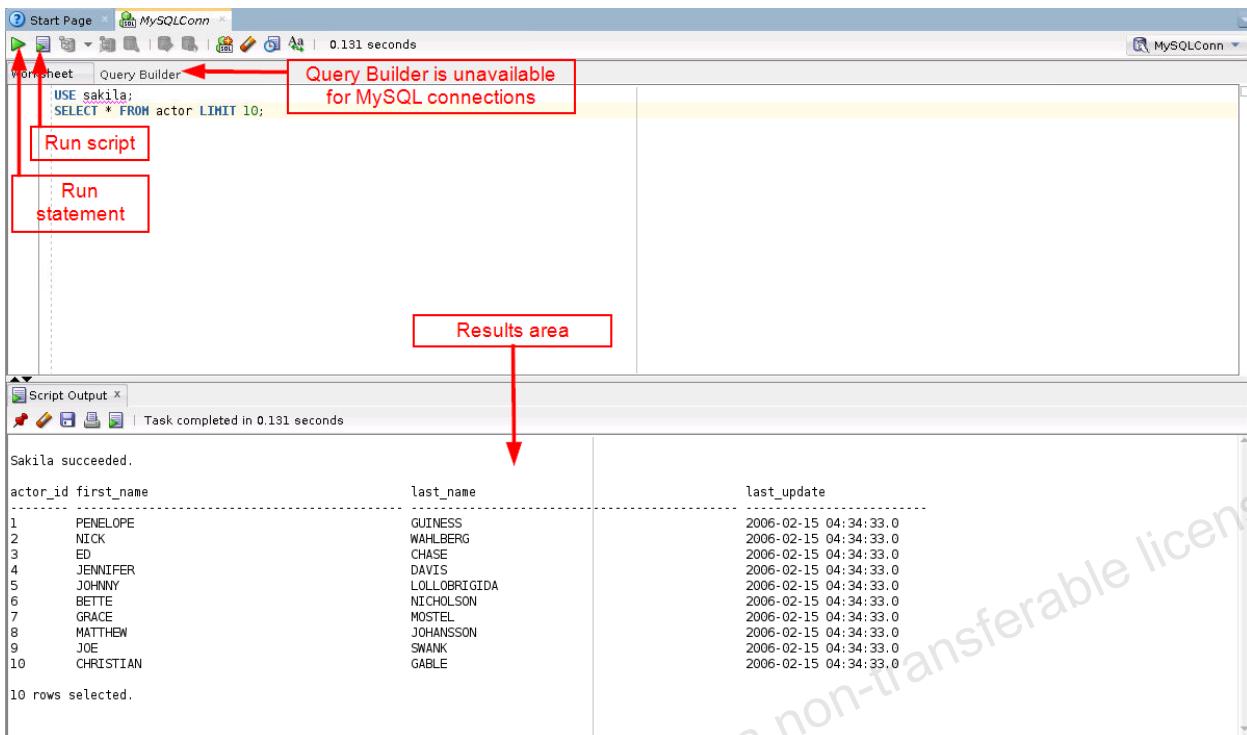


12. Click Connect and a new tab opens, identified by the name of the connection. The connection appears in the list of connections in the Connection pane:



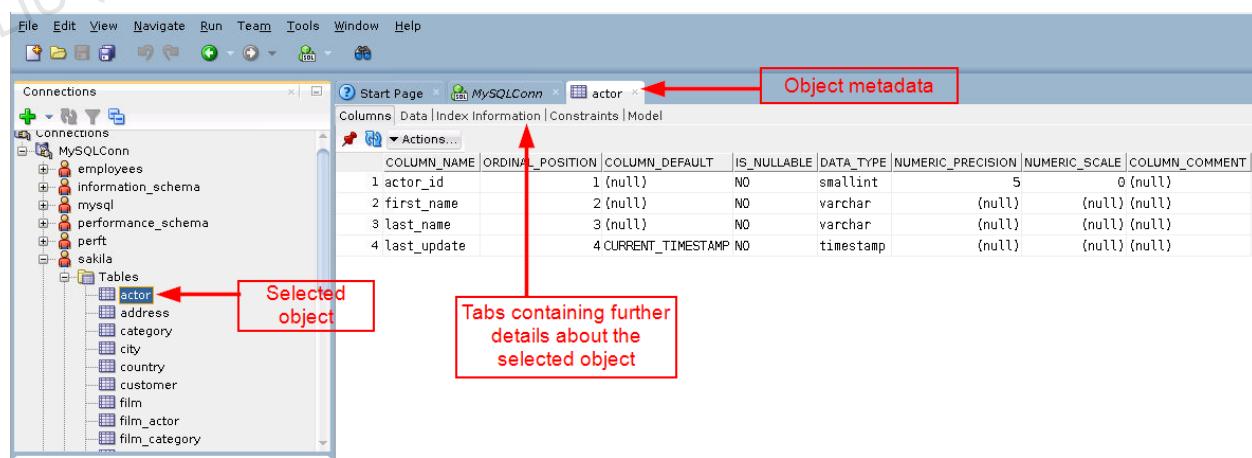
## Usage

13. Enter SQL on the Worksheet page and click the appropriate button to execute either a single SQL statement or all SQL statements on the page. The results appear in a new pane at the bottom of the screen:



**Note:** The Query Builder is unavailable for MySQL connections.

14. Expand the new connection in the Connections pane. You can view all the available schemas and can drill down to review the individual database objects that they contain. When you select an object, a new tab appears with the object metadata. The range of metadata available depends on the type of object that you select. For example, the `actors` table in the `sakila` database enables you to view information about columns, indexes, and constraints, and the Model tab provides options to export schema information to an SQL file or directly to an Oracle database for data modelling.



GANG LIU (gangli@baylorhealth.edu) has a non-transferable license  
to use this Student Guide.