

MySQL Performance Tuning

Activity Guide

D61820GC20

Edition 2.0

May 2011

D73031

ORACLE®

Copyright © 2011, 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.

Author

Jeff Gorton

Technical Contributors and Reviewers

Paul Dubois, John Russell, James Day, Kimseong Loh, Mattias Jonsson, Ole Solberg

This book was published using: *Oracle Tutor*

Table of Contents

Practices for Lesson 1: Introduction.....	1-1
Overview of Practices for Lesson 1.....	1-2
Practices for Lesson 2: Performance Tuning Basics.....	2-1
Overview of Practices for Lesson 2.....	2-2
Practices for Lesson 3: Performance Tuning Tools.....	3-1
Practices for Lesson 3: Overview.....	3-2
Practice 3-1: MySQL Monitoring Tools.....	3-3
Solutions 3-1: MySQL Monitoring Tools	3-8
Practice 3-2: Open Source Community Monitoring Tools.....	3-23
Solutions 3-2: Open Source Community Monitoring Tools	3-29
Practice 3-3: Benchmark Tools.....	3-42
Solutions 3-3: Benchmark Tools	3-45
Practices for Lesson 4: MySQL Server Tuning	4-1
Practices for Lesson 4.....	4-2
Practice 4-1: Effects of Thread Caching	4-3
Solutions 4-1: Effects of Thread Caching	4-7
Practice 4-2: Table Caching	4-15
Solutions 4-2: Table Caching.....	4-18
Practice 4-3: Setting open_files_limit	4-24
Solutions 4-3: Setting open_files_limit	4-25
Practice 4-4: Setting max_connections	4-26
Solutions 4-4: Setting max_connections	4-28
Practice 4-5: Evaluating the Effects of Numerous Connections	4-33
Solutions 4-5: Evaluating the Effects of Numerous Connections	4-37
Practice 4-6: Total Thread Memory Usage.....	4-45
Solutions 4-6: Total Thread Memory Usage	4-46
Practice 4-7: Sort Queries	4-48
Solutions 4-7: Sort Queries.....	4-54
Practices for Lesson 5: MySQL Query Cache	5-1
Practices for Lesson 5.....	5-2
Practice 5-1: Query Cache	5-3
Solutions 5-1: Query Cache.....	5-10
Practices for Lesson 6: InnoDB.....	6-1
Practices for Lesson 6.....	6-2
Practice 6-1: Committing Transactions.....	6-3
Solutions 6-1: Committing Transactions	6-7
Practice 6-2: SHOW ENGINE INNODB STATUS	6-11
Solutions 6-2: SHOW ENGINE INNODB STATUS	6-17
Practice 6-3: InnoDB Monitors.....	6-26
Solutions 6-3: InnoDB Monitors	6-30
Practice 6-4: InnoDB Settings.....	6-35
Solutions 6-4: InnoDB Settings	6-40
Practices for Lesson 7: MyISAM.....	7-1
Practices for Lesson 7	7-2
Practice 7-1: Optimizing MyISAM	7-3

Solutions 7-1: Optimizing MyISAM.....	7-6
Practice 7-2: MyISAM Table Locks	7-9
Solutions 7-2: MyISAM Table Locks	7-14
Practice 7-3: Key Cache Effectiveness	7-21
Solutions 7-3: Key Cache Effectiveness.....	7-24
Practice 7-4: Full-Text Indexing	7-28
Solutions 7-4: Full-Text Indexing.....	7-30
Practices for Lesson 8: Other MySQL Storage Engines and Issues.....	8-1
Practices for Lesson 8.....	8-2
Practice 8-1: Large Objects	8-3
Solutions 8-1: Large Objects	8-5
Practice 8-2: MEMORY Storage Engine.....	8-7
Solution 8-2: MEMORY Storage Engine.....	8-10
Practices for Lesson 9: Schema Design and Performance.....	9-1
Practices for Lesson 9.....	9-2
Practice 9-1: Schema Design	9-3
Solutions 9-1: Schema Design.....	9-5
Practice 9-2: Data Types.....	9-8
Solutions 9-2: Data Types	9-12
Practice 9-3: Indexes	9-17
Solutions 9-3: Indexes	9-21
Practice 9-4: Partitioning	9-27
Solutions 9-4: Partitioning.....	9-30
Practices for Lesson 10: MySQL Query Performance.....	10-1
Practices for Lesson 10.....	10-2
Practice 10-1: EXPLAIN Outputs	10-3
Solutions 10-1: EXPLAIN Outputs	10-7
Practice 10-2: Improve Query Executions	10-14
Solutions 10-2: Improve Query Executions.....	10-17
Practice 10-3: Locate and Correct Problematic Queries.....	10-21
Solutions 10-3: Locate and Correct Problematic Queries.....	10-25
Practices for Lesson 11: Performance Tuning Extras	11-1
Overview of Practices for Lesson 11	11-2
Practices for Lesson 12: Conclusion	12-1
Overview of Practices for Lesson 12.....	12-2

Practices for Lesson 1: Introduction

Chapter 1

Overview of Practices for Lesson 1

Practices for Lesson 1

There are no practices for this lesson.

Practices for Lesson 2: Performance Tuning Basics

Chapter 2

Overview of Practices for Lesson 2

Practices for Lesson 2

There are no practices for this lesson.

Practices for Lesson 3: Performance Tuning Tools

Chapter 3

Practices for Lesson 3: Overview

Practices Overview

In these practices, you are introduced to a large number of MySQL and open-source community performance-tuning tools.

Practice 3-1: MySQL Monitoring Tools

Overview

In this practice, you use and compare the different MySQL monitoring tools. To accomplish this objective, you do the following:

- Use the tools available for the `mysql` client.
 - This includes using various `SHOW` statements along with built-in information and performance databases.
- Use the `mysqladmin` client tool.
- Set up and review the information that can be obtained from the MySQL Enterprise Monitor.

Assumptions

- The MySQL server is installed and running.
- The `mysqladmin` client is available.
- The MySQL Enterprise Monitor and Agent have been installed.
- The `10min.sql` sample file is accessible and located in the `/stage/scripts` directory.

Duration

- This practice should take 60 minutes to complete.

Prerequisite Task

1. To prevent skewed results in the practices, execute the following commands as `root` in a system terminal window:

```
sys> rm -rf /etc/init.d/mysql-monitor-server
sys> ln -s /opt/mysql/enterprise/monitor/mysqlmonitorctl.sh \
/etc/init.d/mysql-monitor-server
sys> /etc/init.d/mysql-monitor-server stop
sys> /etc/init.d/mysql-monitor-agent stop
```

Note: These commands are discussed in a later practice.

Tasks

1. Using the `mysql` client, show all the server status variables that start with the word “Threads%”.
 - How many server status variables are displayed? _____

Note: Use the login name `root` and the password `oracle` to access the MySQL server. The sample solutions demonstrate entering the password on the command line. Entering the password on the command line is not recommended in a real-world environment. In the classroom, entering the password on the command line prevents the MySQL server from prompting you for the password and supports the execution of multiple commands at once.
2. Using the `mysql` client, show all the tables in the `INFORMATION_SCHEMA` that start with the letter “S”.
 - How many `INFORMATION_SCHEMA` tables start with the letter “S”? _____

3. Using the `mysql` client, show the columns from the `SESSION_STATUS` table in the `INFORMATION_SCHEMA` database.
 - How many columns does the `SESSION_STATUS` table of the `INFORMATION_SCHEMA` database contain?

4. Using the `mysql` client, use the `SESSION_STATUS` table in the `INFORMATION_SCHEMA` database to show all the server status variables that start with the word “Threads%”.
 - Compare the output from step 1 with this output. Are they similar or different?

5. As `root`, stop the MySQL server and then start it with the `--performance_schema` option.
6. Using the `mysql` client, show all the tables in the `performance_schema` database.
7. Using the `mysql` client, display all the records in the `threads` table of the `performance_schema` database.
8. In the `/stage/scripts/` directory, use the `mysql` client to run `10min.sql` against the MySQL server.

Note: The `10min.sql` file executes a SQL statement against the `employees` table that takes approximately 10 minutes to complete.
9. In a separate terminal window, use the `mysql` client to show all the open tables that are currently open and being used by entering the following command:

```
sys> mysql -uroot -poracle -e"SHOW OPEN TABLES WHERE In_use > 0"
```

Note: The `In_use` column identifies the number of table locks or lock requests there are for the table. If a table is open but not being used, the value is 0.

 - How many non-TEMPORARY tables are currently open and being used? _____
10. In the second terminal window (opened in step 9), use the `mysql` client to show all the MySQL threads currently running.
 - How many MySQL server processes are currently running? _____
 - What is the process ID for the query executed in step 9? _____
11. In the second terminal window (opened in step 9), use the `mysqladmin` client to display a short server status message for the MySQL server.
 - How many active threads are currently connected to the MySQL server? _____
 - How many tables are currently opened by the MySQL server? _____
12. In the second terminal window (opened in step 9), use the `mysqladmin` client to kill the thread for the query executed in step 8.

13. Close the terminal window opened in step 9.
 14. In the first terminal window, what is the error that was displayed when the execution of the query was terminated?
-

15. Open the Firefox browser and enter the following URL:

```
file:///home/oracle/dashboard_final.swf
```

Note: This opens the Enterprise Monitor demo. This demo includes sound. Check with your instructor before viewing to minimize distractions in the classroom environment.

To view the entire demo, you may need to choose Full Screen from the View menu.

16. Exit the Firefox browser.

17. As root, start the MySQL Enterprise Service Manager by entering the following command:

```
sys> /etc/init.d/mysql-monitor-server start
```

18. As root, start the MySQL Enterprise Monitor Agent by entering the following command:

```
sys> /etc/init.d/mysql-monitor-agent start
```

19. Open the Firefox internet browser and enter the following URL:

```
Localhost:18080
```

Note: Port 18080 is the default port for the MySQL Enterprise Monitor Dashboard.

20. In the browser window that appears (MySQL Enterprise Dashboard Setup), fill in the form elements with the following values:

- Under the Create Administrator heading:
 - Username: admin
 - Password: oracle
 - Confirm Password: oracle
- Under the Create Agent Credentials heading:
 - Username: agent
 - Password: oracle
 - Confirm Password: oracle
- Click the Complete Setup button to continue

Note: In a working environment, the other fields in this setup form would need to be filled in. In the classroom environment, the fields entered are satisfactory to complete the setup.

21. In the next window, choose the settings in the pop-up box for your current location and language.

- In the **Timezone** field, choose your timezone from the drop-down list.
- In the **Locale**, choose the language that you want the MySQL Enterprise Dashboard to display.
- Click the save user settings button to continue.

22. Along the top of the webpage, click the Monitor tab.

- In the left-hand box (with the title **Servers**), what is the name of the server that the MySQL Enterprise Monitor is currently monitoring?

- In the right-hand box (with the title **All Servers heat chart**), select the plus symbol to the left of the words **All Servers**. What is the name of the server that is being monitored by the heat chart?

-
- What is the color of the Agent Status for the server being monitored? _____
 - What is the color of the Server Status for the server being monitored? _____
 - In the center box (with the title All Servers Graphs), how many graphs are displayed?

- How active is your server based on the graph details?

23. Along the top of the webpage, click the Advisors tab.

- This tab provides you best practice Advisors that are designed to automatically examine a MySQL server's configuration, security, and performance levels, and identify any deviations from best practice rules that are built by the database experts at MySQL.
- Under the All Servers - Scheduled Advisors heading, select the plus symbol to the left of the words Heat Chart. Click the link for the text "CPU I/O Usage Excessive". What does this advisor tell you?

- Click the Hide button to close the pop-up box that was opened.

24. Along the top of the webpage, click the Events tab.

- This tab provides you a listing of alerts that have been captured by the MySQL Enterprise Monitor. The alerts range from *Critical* (requires immediate attention) to *Info* (issues that do not affect the operation of your server). In addition, alerts also appear on the Monitor screen in order of severity.

25. Along the top of the webpage, click the Graphs tab.

- This tab provides you with a complete listing of all graphs available for you to view. You can select any one of the graphs to view simply by selecting the plus symbol to the left of the graph to view. You can also view all the graphs by clicking the expand all button.

26. Along the top of the webpage, click the Query Analyzer tab.

- This tab lets you monitor the statements being executed on a monitored server and retrieve information about the query, number of executions, and the execution times of each query.

Note: In the classroom environment, this function has not been enabled.

27. Along the top of the webpage, click the Replication tab.

- This tab provides you a quick summary view of the state of your replication servers or, if you wish, you can drill down and determine specifics about any master or slave.

Note: In the classroom environment, there is a single instance of the MySQL server running and results in no replication groups being found by MySQL Enterprise Monitor.

28. Along the top of the webpage, click the Settings tab.

- This tab explores the configuration settings in more detail, and shows you how to manage servers, users, notification groups, Simple Network Management Protocol (SNMP) traps, log files, and the product information screen.

29. Along the top of the webpage, click the What's New? tab.

- This tab provides you with a simplified interface to view updates and news related to your MySQL Enterprise Subscription.

30. In the top right-hand corner of the webpage, click the Log Out link.

31. Exit the browser and return to the terminal window that was originally opened in step 1.

32. As root, stop the MySQL Enterprise Monitor Agent by entering the following command:

```
sys> /etc/init.d/mysql-monitor-agent stop
```

33. As root, stop the MySQL Enterprise Service Manager by entering the following command:

```
sys> /etc/init.d/mysql-monitor-server stop
```

34. Open the Firefox browser and enter the following URL:

```
file:///home/oracle/queryanalyzer_final.swf
```

Note: This opens the Query Analyzer demo. This demo includes sound. Check with your instructor before viewing to minimize distractions in the classroom environment.

To view the entire demo, you may need to choose Full Screen from the View menu.

35. Exit the Firefox browser.

Solutions 3-1: MySQL Monitoring Tools

Prerequisite Task

- To prevent skewed results in the practices, execute the following commands as `root` in a system terminal window:

```
sys> rm -rf /etc/init.d/mysql-monitor-server
sys> ln -s /opt/mysql/enterprise/monitor/mysqlmonitorctl.sh \
/etc/init.d/mysql-monitor-server
sys> /etc/init.d/mysql-monitor-server stop
sys> /etc/init.d/mysql-monitor-agent stop
```

Note: These files are discussed in a later practice.

Tasks

Note: These solutions are sample solutions and are not expected to be the exact output you see in your execution of the steps.

- Using the `mysql` client, show all the server status variables that start with the word “Threads%”.

```
sys> mysql -uroot -poracle -e"SHOW STATUS LIKE 'Threads%'"
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Threads_cached | 0
| Threads_connected | 1
| Threads_created | 1
| Threads_running | 1
+-----+-----+
```

- How many server status variables are displayed? 4

Note: Use the login name `root` and the password `oracle` to access the MySQL server. The sample solutions demonstrate entering the password on the command line. Entering the password on the command line is not recommended in a real-world environment. In the classroom, entering the password on the command line prevents the MySQL server from prompting you for the password and supports the execution of multiple commands at once.

- Using the `mysql` client, show all the tables in the `INFORMATION_SCHEMA` that start with the letter “S”.

```
sys> mysql -uroot -poracle -e"SHOW TABLES FROM \
> INFORMATION_SCHEMA LIKE 'S%'"
+-----+
| Tables_in_INFORMATION_SCHEMA (S%) |
+-----+
| SCHEMATA
| SCHEMA_PRIVILEGES
| SESSION_STATUS
| SESSION_VARIABLES
| STATISTICS
+-----+
```

- How many `INFORMATION_SCHEMA` tables start with the letter “S”? 5

3. Using the `mysql` client, show the columns from the `SESSION_STATUS` table in the `INFORMATION_SCHEMA` database.

```
sys> mysql -uroot -poracle -e"DESC \
> INFORMATION_SCHEMA.SESSION_STATUS"
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| VARIABLE_NAME | varchar(64) | NO | | NULL | |
| VARIABLE_VALUE | varchar(1024) | YES | | | |
+-----+-----+-----+-----+-----+
```

- How many columns does the `SESSION_STATUS` table of the `INFORMATION_SCHEMA` database contain?

2

4. Using the `mysql` client, use the `SESSION_STATUS` table in the `INFORMATION_SCHEMA` database to show all the server status variables that start with the word “Threads%”.

```
sys> mysql -uroot -poracle -e"SELECT * FROM \
> INFORMATION_SCHEMA.SESSION_STATUS WHERE VARIABLE_NAME \
> LIKE 'Thread%'"
+-----+-----+
| VARIABLE_NAME | VARIABLE_VALUE |
+-----+-----+
| THREADS_CACHED | 0 |
| THREADS_CONNECTED | 1 |
| THREADS_CREATED | 1 |
| THREADS_RUNNING | 1 |
+-----+-----+
```

- Compare the output from step 1 with this output. Are they similar or different?

They are the same server status variables, the only difference is the capitalization.

5. As root, stop the MySQL server and then start it with the `--performance_schema` option.

```
sys> /etc/init.d/mysql stop
sys> /etc/init.d/mysql start --performance_schema
```

6. Using the mysql client, show all the tables in the performance_schema database.

```
sys> mysql -uroot -poracle -e"SHOW TABLES FROM \
> performance_schema"
+-----+
| Tables_in_performance_schema |
+-----+
| cond_instances
| events_waits_current
| events_waits_history
| events_waits_history_long
| events_waits_summary_by_instance
| events_waits_summary_by_thread_by_event_name
| events_waits_summary_global_by_event_name
| file_instances
| file_summary_by_event_name
| file_summary_by_instance
| mutex_instances
| performance_timers
| rwlock_instances
| setup_consumers
| setup_instruments
| setup_timers
| threads
+-----+
```

7. Using the mysql client, display all the records in the threads table of the performance_schema database.

```
sys> mysql -uroot -poracle -e"SELECT * FROM \
> performance_schema.threads"
+-----+-----+-----+
| THREAD_ID | PROCESSLIST_ID | NAME
+-----+-----+-----+
| 0 | 0 | thread/sql/main
| 11 | 0 | thread/innodb/srv_master_thread
| 18 | 2 | thread/sql/one_connection
| 7 | 0 | thread/innodb/io_handler_thread
| 1 | 0 | thread/innodb/io_handler_thread
| 10 | 0 | thread/innodb/io_handler_thread
| 15 | 0 | thread/innodb/srv_lock_timeout_thread
| 3 | 0 | thread/innodb/io_handler_thread
| 9 | 0 | thread/innodb/io_handler_thread
| 12 | 0 | thread/innodb/srv_monitor_thread
| 5 | 0 | thread/innodb/io_handler_thread
| 8 | 0 | thread/innodb/io_handler_thread
| 4 | 0 | thread/innodb/io_handler_thread
| 6 | 0 | thread/innodb/io_handler_thread
| 13 | 0 | thread/innodb/srv_error_monitor_thread
| 2 | 0 | thread/innodb/io_handler_thread
| 16 | 0 | thread/sql/signal_handler
+-----+-----+-----+
```

8. In the /stage/scripts/ directory, use the mysql client to run 10min.sql against the MySQL server.

```
sys> mysql -uroot -poracle < /stage/scripts/10min.sql
```

Note: The 10min.sql file executes a SQL statement against the employees table that takes approximately 10 minutes to complete.

- In a separate terminal window, use the mysql client to show all the open tables that are currently open and being used by entering the following command:

```
sys> mysql -uroot -poracle -e"SHOW OPEN TABLES WHERE In_use > 0"
+-----+-----+-----+-----+
| Database | Table      | In_use | Name_locked |
+-----+-----+-----+-----+
| employees | dept_manager | 1 | 0 |
| employees | dept_emp    | 1 | 0 |
| employees | salaries    | 1 | 0 |
+-----+-----+-----+-----+
```

Note: The In_use column identifies the number of table locks or lock requests there are for the table. If a table is open but not being used, the value is 0.

- How many non-Temporary tables are currently open and being used? 3

- In the second terminal window (opened in step 9), use the mysql client to show all the MySQL threads currently running.

```
sys> mysql -uroot -poracle -e"SHOW PROCESSLIST\G"
***** 1.row *****
Id: 5
User: root
Host: localhost
db: NULL
Command: Query
Time: 120
State: Sending data
Info: SELECT employees.dept_emp.from_date, employees.dept_manager.from_date,
employees.salaries.fro
***** 2.row *****
Id: 8
User: root
Host: localhost
db: NULL
Command: Query
Time: 0
State: NULL
Info: SHOW PROCESSLIST
```

- How many MySQL server processes are currently running? 2
- What is the process ID for the query executed in step 9? 5

- In the second terminal window (opened in step 9), use the mysqladmin client to display a short server status message for the MySQL server.

```
sys> mysqladmin -uroot -poracle status
Uptime: 32 Threads: 2 Questions: 15 Slow queries: 0 Opens:
36 Flush tables: 1 Open tables: 29 Queries per second avg:
0.468
```

- How many active threads are currently connected to the MySQL server? 2
- How many tables are currently opened by the MySQL server? 29

- In the second terminal window (opened in step 9), use the mysqladmin client to kill the thread for the query executed in step 8.

- Note:** The thread ID was identified in step 10.

```
sys> mysqladmin -uroot -poracle kill 5
```

13. Close the terminal window opened in step 9.
14. In the first terminal window, what is the error that was displayed when the execution of the query was terminated?

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

15. Open the Firefox browser and enter the following URL:

file:///home/oracle/dashboard_final.swf

Note: This opens the Enterprise Monitor demo. This demo includes sound. Check with your instructor before viewing to minimize distractions in the classroom environment. To view the entire demo, you may need to choose Full Screen from the View menu.



16. Exit the Firefox browser.
17. As root, start the MySQL Enterprise Service Manager by entering the following command:

```
sys> /etc/init.d/mysql-monitor-server start
/opt/mysql/enterprise/monitor/mysqlmonitorctl.sh : mysql started
110228 21:36:02 mysqld_safe Logging to
'/opt/mysql/enterprise/monitor/mysql/data/mysqld.log'.
110228 21:36:02 mysqld_safe Starting mysqld daemon with databases from
/opt/mysql/enterprise/monitor/mysql/data/
Using CATALINA_BASE:  /opt/mysql/enterprise/monitor/apache-tomcat
Using CATALINA_HOME:   /opt/mysql/enterprise/monitor/apache-tomcat
Using CATALINA_TMPDIR: /opt/mysql/enterprise/monitor/apache-tomcat/temp
Using JRE_HOME:        /opt/mysql/enterprise/monitor/java
```

18. As root, start the MySQL Enterprise Monitor Agent by entering the following command:

```
sys> /etc/init.d/mysql-monitor-agent start
```

19. Open the Firefox internet browser and enter the following URL:

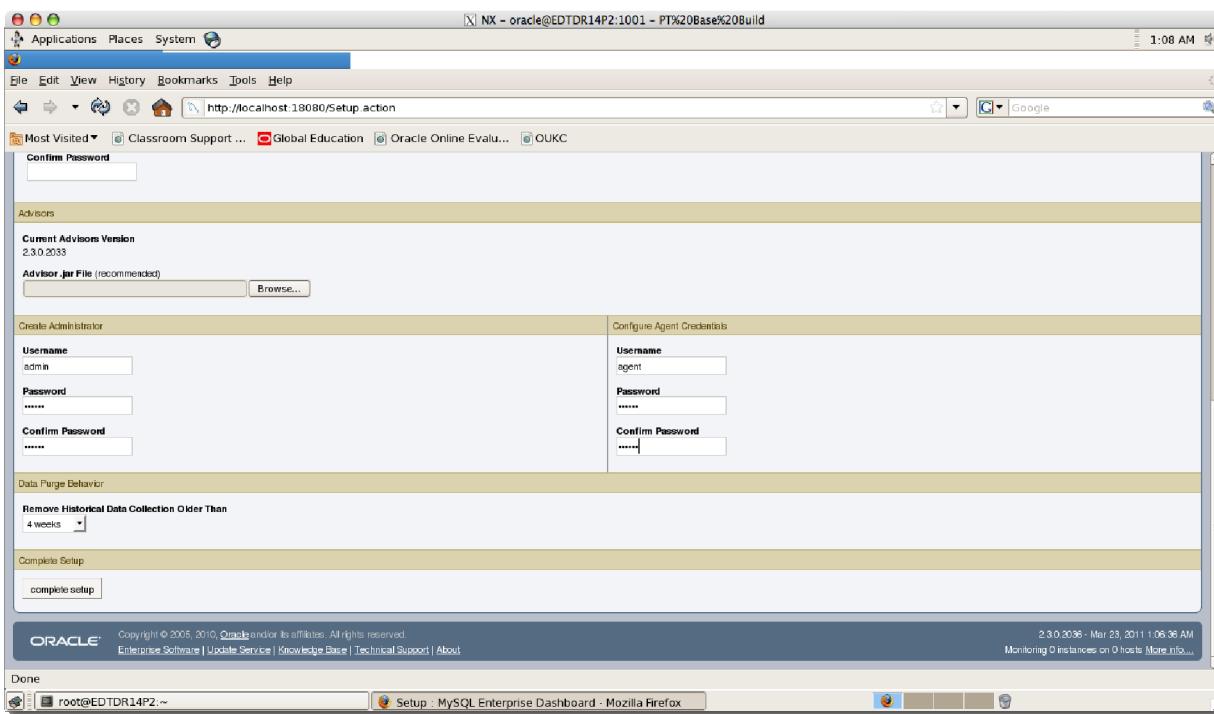
```
Localhost:18080
```

Note: Port 18080 is the default port for the MySQL Enterprise Monitor Dashboard.

20. In the browser window that appears (MySQL Enterprise Dashboard Setup), fill in the form elements with the following values:

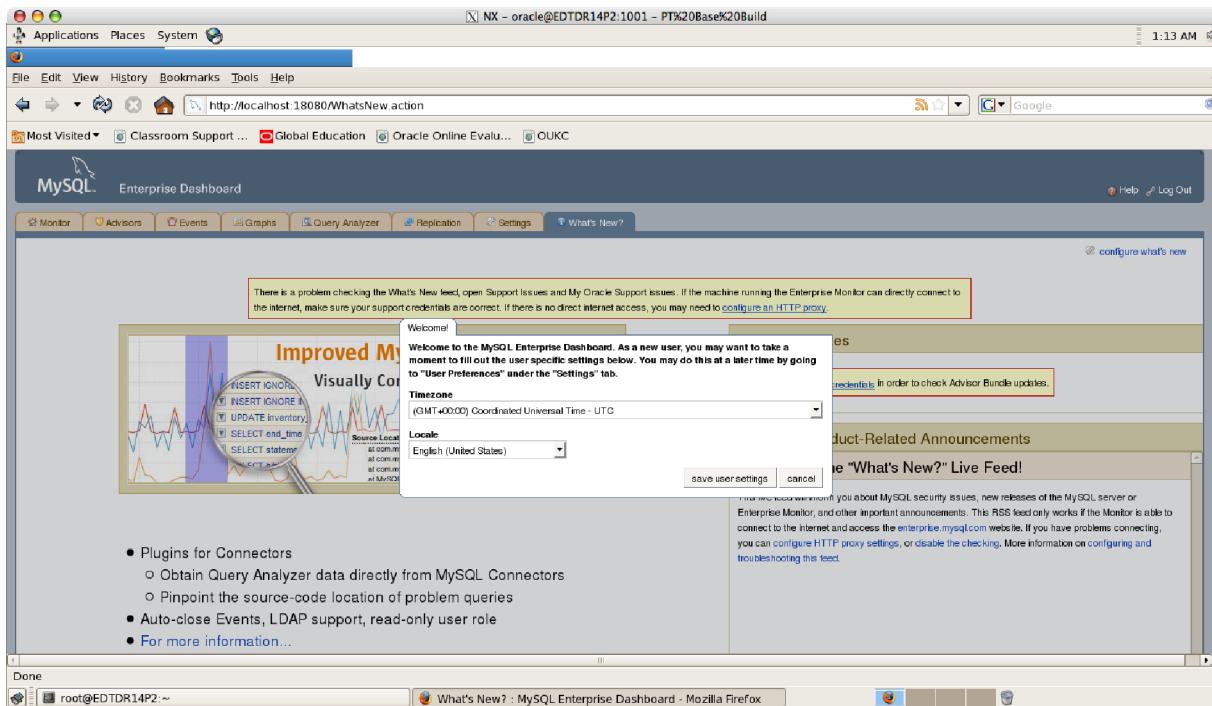
- Under the Create Administrator heading:
 - Username: admin
 - Password: oracle
 - Confirm Password: oracle
- Under the Create Agent Credentials heading:
 - Username: agent
 - Password: oracle
 - Confirm Password: oracle
- Click the Complete Setup button to continue

Note: In a working environment, the other fields in this setup form would need to be filled in. In the classroom environment, the fields entered are satisfactory to complete the setup.



21. In the next window, choose the settings in the pop-up box for your current location and language.

- In the **Timezone** field, choose your timezone from the drop-down list.
- In the **Locale**, choose the language that you want the MySQL Enterprise Dashboard to display.
- Click the save user settings button to continue.



22. Along the top of the webpage, click the Monitor tab.

- In the left-hand box (with the title **Servers**), what is the name of the server that the MySQL Enterprise Monitor is currently monitoring?

EDTDR14P2:3306

- In the right-hand box (with the title **All Servers heat chart**), select the plus symbol to the left of the words **All Servers**. What is the name of the server that is being monitored by the heat chart?

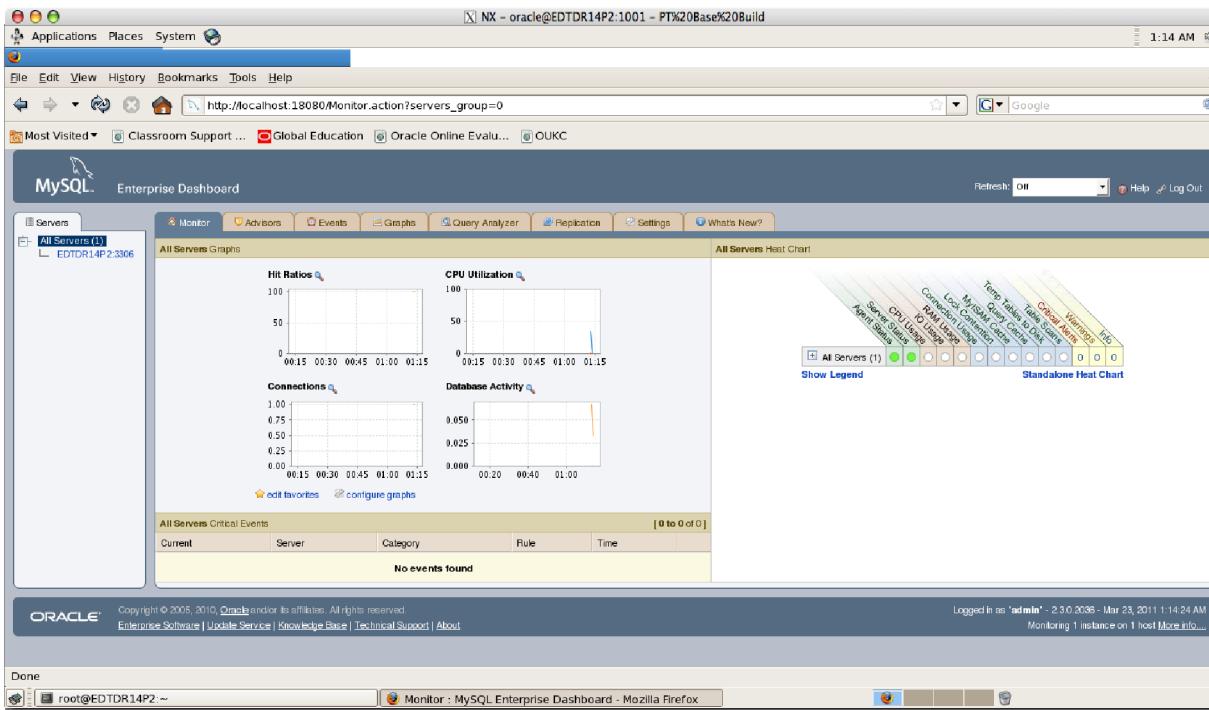
EDTDR14P2:3306

- What is the color of the Agent Status for the server being monitored? Green
- What is the color of the Server Status for the server being monitored? Green
- In the center box (with the title All Servers Graphs), how many graphs are displayed?

Four (Hit Ratios, CPU Utilization, Connections, and Database Activity)

- How active is your server based on the graph details?

The graphs are currently empty because nothing is currently executing against the MySQL server.

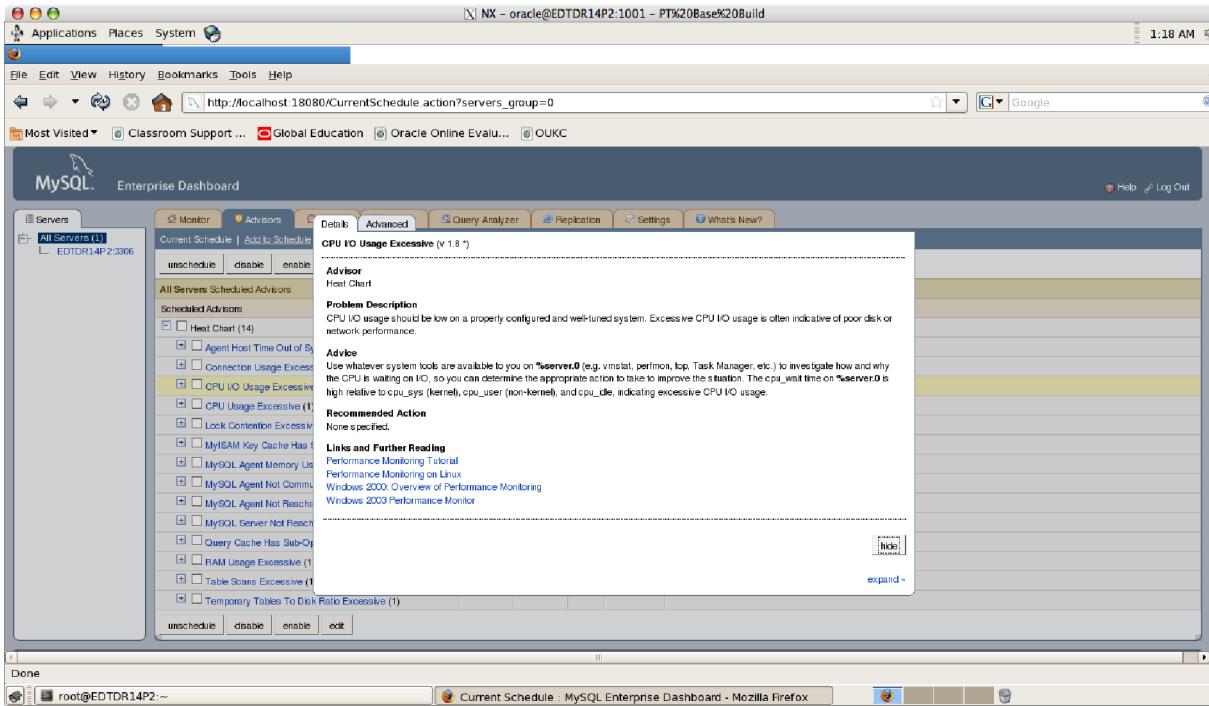


23. Along the top of the webpage, click the Advisors tab.

- This tab provides you best practice Advisors that are designed to automatically examine a MySQL server's configuration, security, and performance levels, and identify any deviations from best practice rules that are built by the database experts at MySQL.
- Under the All Servers - Scheduled Advisors heading, select the plus symbol to the left of the words Heat Chart. Click the link for the text "CPU I/O Usage Excessive". What does this advisor tell you?

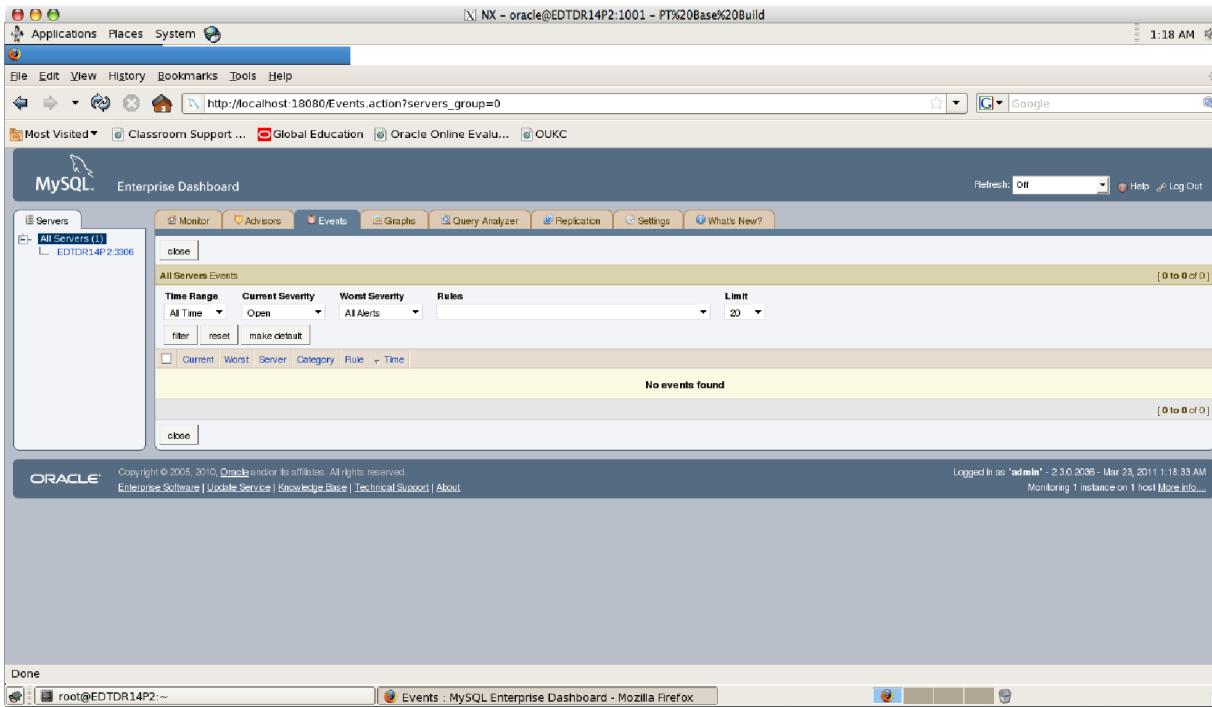
CPU I/O usage should be low on a properly configured and well-tuned system.
Excessive CPU I/O usage is often indicative of poor disk or network performance.

- Click the Hide button to close the pop-up box that was opened.



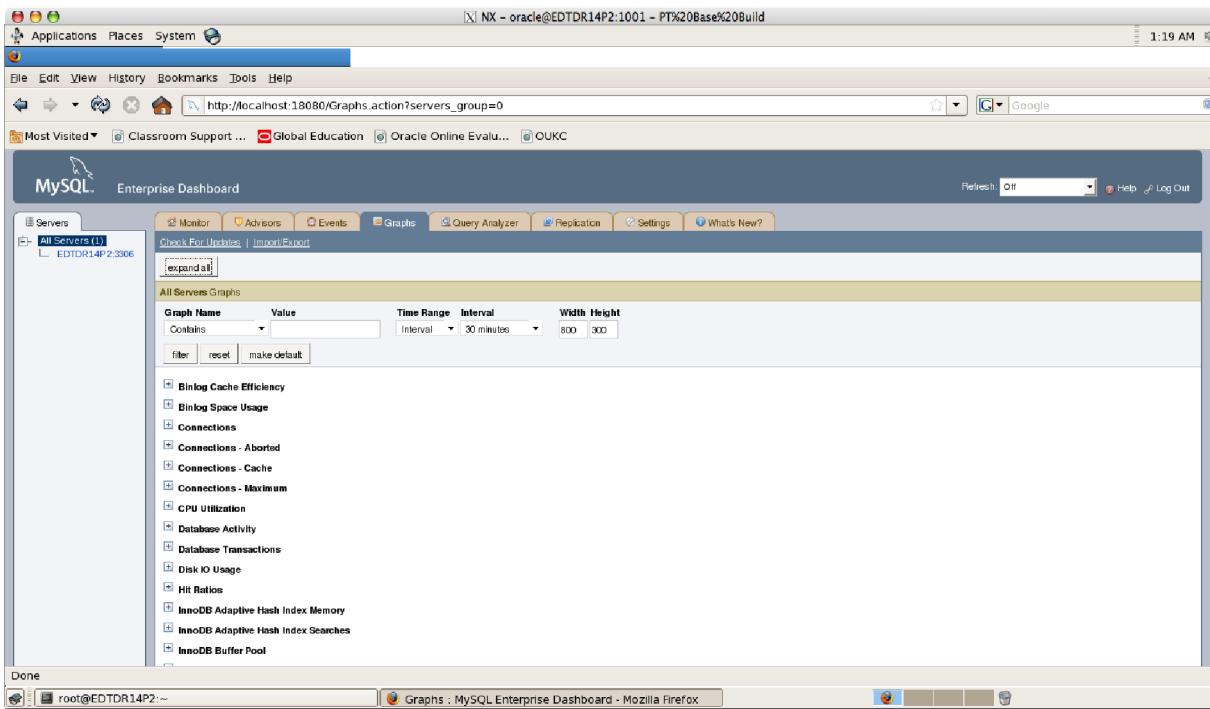
24. Along the top of the webpage, click the Events tab.

- This tab provides you a listing of alerts that have been captured by the MySQL Enterprise Monitor. The alerts range from *Critical* (requires immediate attention) to *Info* (issues that do not affect the operation of your server). In addition, alerts also appear on the Monitor screen in order of severity.



25. Along the top of the webpage, click the Graphs tab.

- This tab provides you with a complete listing of all graphs available for you to view. You can select any one of the graphs to view simply by selecting the plus symbol to the left of the graph to view. You can also view all the graphs by clicking the expand all button.



26. Along the top of the webpage, click the Query Analyzer tab.

- This tab lets you monitor the statements being executed on a monitored server and retrieve information about the query, number of executions, and the execution times of each query.

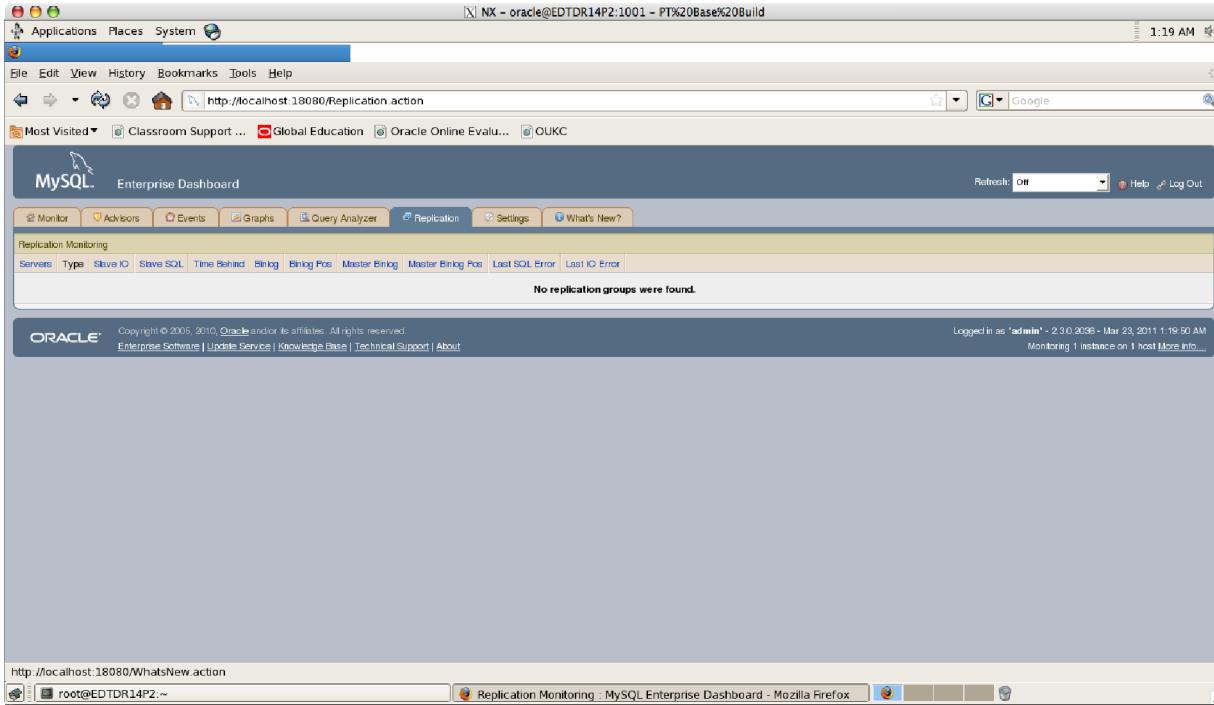
Note: In the classroom environment, this function has not been enabled.

The screenshot shows the MySQL Enterprise Dashboard interface. At the top, there's a navigation bar with links for Applications, Places, System, File, Edit, View, History, Bookmarks, Tools, and Help. Below that is a toolbar with icons for Home, Refresh, Stop, and others. The main content area is titled "MySQL Enterprise Dashboard" and shows the "Enterprise Dashboard". On the left, there's a sidebar titled "Servers" with a tree view showing "All Servers (1)" and "EDTDR14P2:3306". The main panel has tabs for Monitor, Advisors, Events, Graphs, **Query Analyzer**, Replicat, Settings, and What's New? The "Query Analyzer" tab is selected. A message says "The selected server 'EDTDR14P2:3306' does not have query analyzer enabled. Enable Query Analyzer." Below this is a search and filter section with fields for Statement Text, Value, Statement Type, DB Name, Time Range, Interval, and Limit. A table below shows "No queries found in the last 30 minutes." The table has columns for Query, Database, Counts (Exec, Err, Warn, Total, Max, Avg), Rows (Total, Max, Avg), Bytes (Total, Max, Avg), and First Seen. The bottom of the dashboard includes copyright information for Oracle and a log-in status message: "Logged in as 'admin' - 2.3.0.2036 - Mar 23, 2011 1:19:31 AM".

27. Along the top of the webpage, click the Replication tab.

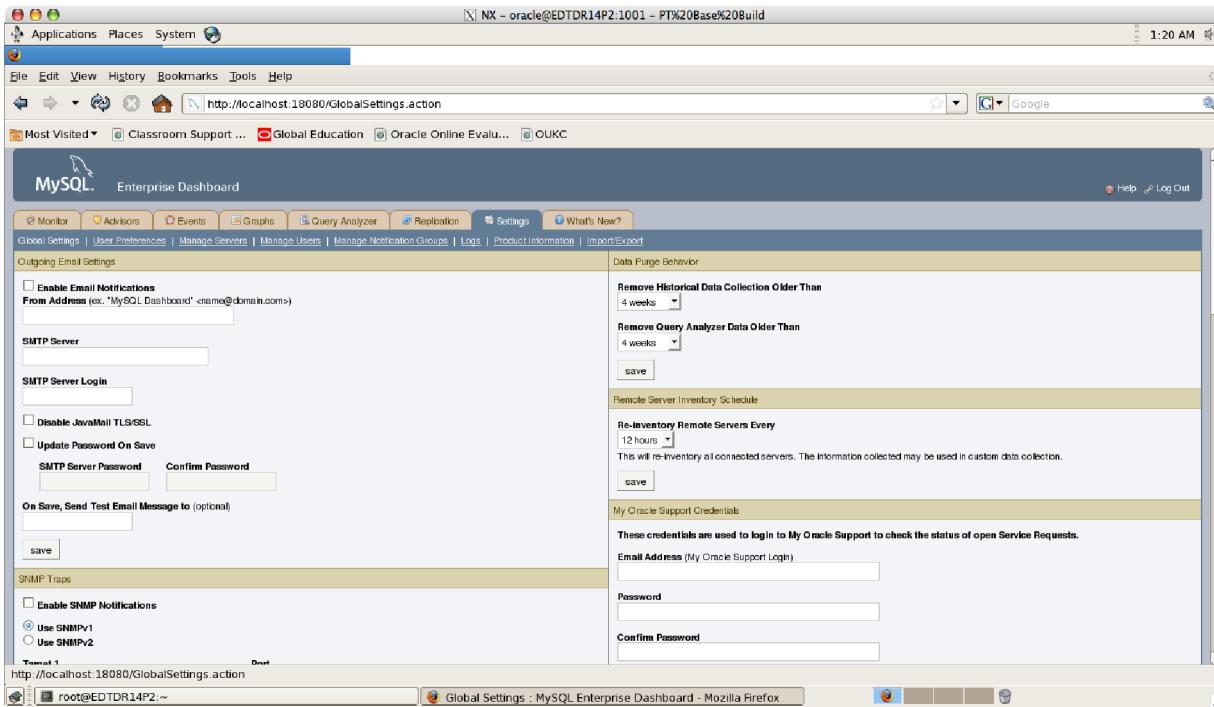
- This tab provides you a quick summary view of the state of your replication servers or, if you wish, you can drill down and determine specifics about any master or slave.

Note: In the classroom environment, there is a single instance of the MySQL server running and results in no replication groups being found by MySQL Enterprise Monitor.



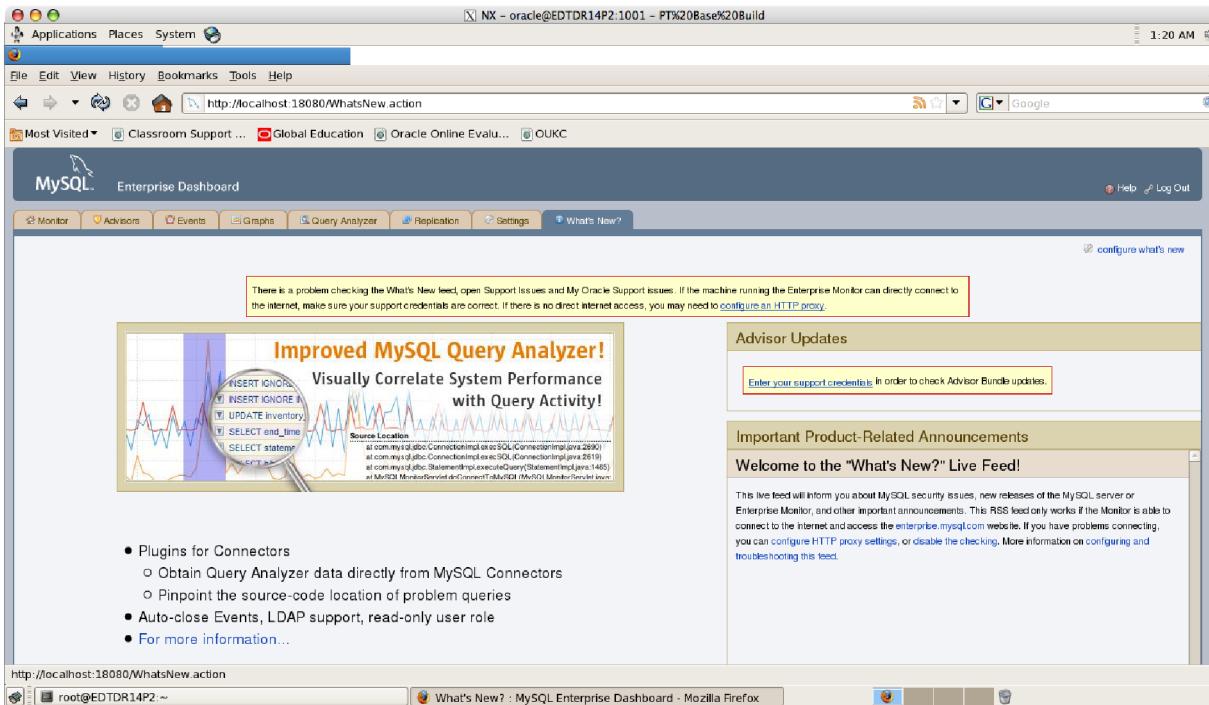
28. Along the top of the webpage, click the Settings tab.

- This tab explores the configuration settings in more detail, and shows you how to manage servers, users, notification groups, Simple Network Management Protocol (SNMP) traps, log files, and the product information screen.



29. Along the top of the webpage, choose the What's New? tab.

- This tab provides you with a simplified interface to view updates and news related to your MySQL Enterprise Subscription.



Note: In the classroom environment there is limited access to the internet. This results in an error on the *What's New?* tab of the Enterprise Monitor.

30. In the top right-hand corner of the webpage, click the Log Out link.

31. Exit the browser and return to the terminal window that was originally opened in step 1.

32. As root, stop the MySQL Enterprise Monitor Agent by entering the following command:

```
sys> /etc/init.d/mysql-monitor-agent stop
```

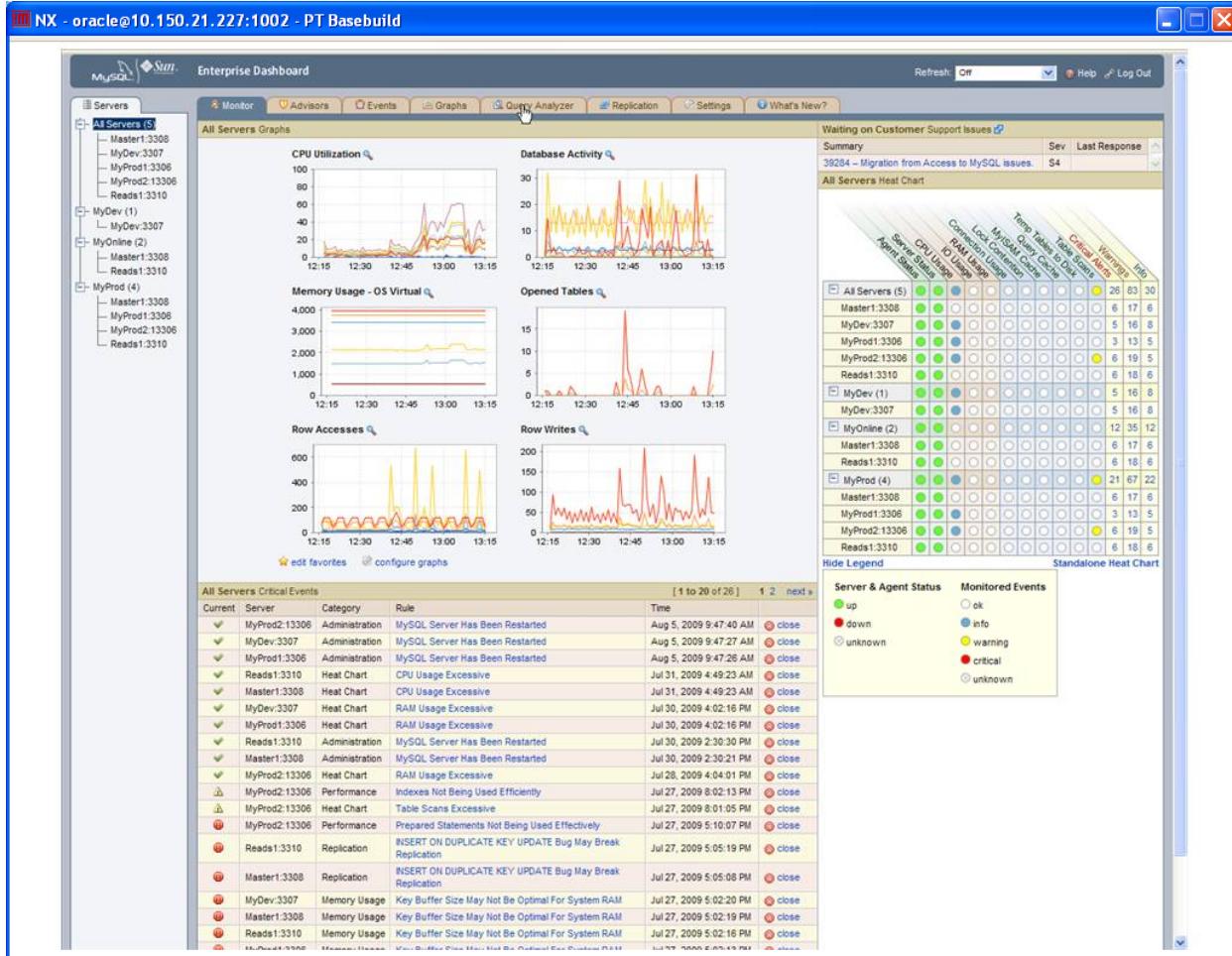
33. As root, stop the MySQL Enterprise Service Manager by entering the following command:

```
sys> /etc/init.d/mysql-monitor-server stop
Using CATALINA_BASE:      /opt/mysql/enterprise/monitor/apache-tomcat
Using CATALINA_HOME:       /opt/mysql/enterprise/monitor/apache-tomcat
Using CATALINA_TMPDIR:     /opt/mysql/enterprise/monitor/apache-tomcat/temp
Using JRE_HOME:            /opt/mysql/enterprise/monitor/java
Stopping tomcat service ... [ OK ]
Stopping mysql service .110228 21:38:18 mysqld_safe mysqld from pid file
/opt/mysql/enterprise/monitor/mysql/data/mysqld.pid ended
. [ OK ]
```

34. Open the Firefox browser and enter the following URL:

file:///home/oracle/queryanalyzer_final.swf

Note: This opens the Query Analyzer demo. This demo includes sound. Check with your instructor before viewing to minimize distractions in the classroom environment. To view the entire demo, you may need to choose Full Screen from the View menu.



35. Exit the Firefox browser.

Practice 3-2: Open Source Community Monitoring Tools

Overview

In this practice, you use and compare the different open source community monitoring tools available. To accomplish this objective, you do the following:

- Start and use the **dim_STAT** server and service.
- Review the **mysqlresources** application output.
- Review the multiple tools available in the **Maatkit** toolkit and use a select number of these tools.
- Execute the **mysqlreport** application, perform multiple queries against the MySQL server, and execute the **mysqlreport** application again.
 - The output between the two runs are reviewed and compared to demonstrate the usefulness of the information reported.
- Start the **mytop** application and perform a select number of commands within the application.

Assumptions

- The MySQL server is installed and running.
- The **dim_STAT** application created by Dimitri Kravtchuk is installed.
- The **mysqlresources** perl script created by Giuseppe Maxia is installed.
 - This would include Perl installed and configured to run on the operating system.
- The **Maatkit** toolkit created by Daniel Nichter and Baron Schwartz is installed.
- The **mysqlreport** application created by Daniel Nichter is installed.
- The **mytop** application created by Jeremy Zawodny is installed.
- The **Qcache-queries.sql** sample file is accessible and located in the **/stage/scripts** directory.

Note: Learning each of these tools in great detail requires a considerable amount of time. This activity introduces you to the tools and some of their capabilities; however, the steps in this activity are not designed to make you experts in the use of the tools. After you have seen each of the tools, you can decide which tools you desire to have a greater understanding for. The creators of the tools and their respective websites have been included in the training material to assist you in learning more about each tool.

Duration

- This practice should take 45 minutes to complete.

Tasks

1. Stop the MySQL server as **root**.
 - The first open source community application you use for this lesson is **dim_STAT**. To prevent complications with the MySQL server that **dim_STAT** uses, it is best to shut down the current instance of the MySQL server that is running.
2. Stop the Apache web server as **root**.
 - To prevent complications with the Apache web server instance that **dim_STAT** uses, it is best to shut down the current instance of the Apache web server that is running.

Note: This application may not be running. If an error is displayed when you attempt to shut down the Apache web server, it can be ignored.

3. As root, start the dim_STAT server by executing the following command:

```
sys> /apps/ADMIN/dim_STAT-Server start
```

4. As root, start the dim_STAT service by executing the following command:

```
sys> /etc/STATsrv/STAT-service start
```

5. In your web browser, enter the following website address:

```
http://localhost
```

- This opens the **Dimitri (dim) Tools HOMEPAGE**.
- Click the **Welcome!** link under the heading **dim_STAT Main Page**.

6. On the dim_STAT welcome page that is opened, click the **Start New Collect** link in the menu of options located on the right-hand side of the page.

7. On the dim_STAT Start New Collect(s) page, select the check box to the left of the sun/localhost:5000 host. Click the **Continue** button.

8. On the next page, use the following values to complete the form:

- Collect Basename: **Classroom_Session1**
- New Collect Title: **Collecting Stats for Session 1**
- Collect ID: **1040**
- STAT(s): Choose **Liosstat**, **Lmpstat**, **LnetLOAD**, **LPrclLOAD**, and **Lvmstat**
- Click the **Start STAT(s) collect now!!!** button.

9. On the next page, a listing of the active databases and their utilization is displayed. Click the **[Home]** link in the top right-hand corner to return to the dim_STAT welcome page.

10. On the dim_STAT welcome page, click the **Delete/Recycle Collect(s)** link in the menu of options located on the right-hand side of the page.

11. On the dim_STAT DELETE page, active and stopped collects are listed.

- Only stopped collect(s) can be deleted.
- This is a good location to view the number of records that have been stored for each collect.

12. Click the **[Home]** link in the top right-hand corner to return to the dim_STAT welcome page.

13. On the dim_STAT welcome page, click the **Analyze** link in the menu of options located on the right-hand side of the page.

14. On the dim_STAT Analyze page, use the following values to complete the form:

- Choose the **Single Host Analyze** and **Active Only** options.
- Click the **Analyze** button.

15. On the next page, choose the active collect by selecting the check box to the left of the ID of 1040. Click the **Lvmstat** button.

16. On the Collecting Stats for Session 1 (@localhost) page, use the following values to complete the form:
 - In the Interval table, choose the **All Data** option.
 - Select the check box to the left of the table heading “Top 10 Values.” Choose the **Free(KB)**, **Cache(KB)**, **Usr%_CPU**, and **Sys%_CPU** options.

Note: The value of 10 in the header of this table can be changed to number larger or smaller than 10.
 - Select the check box to the left of the table heading “Table of results.” Choose the **Free(KB)** and **Cache(KB)** options.
 - Select the check box to the left of the table heading “Graphics.” Choose the **Usr%_CPU** and **Sys%CPU** options.
 - If the check box to turn on the Graphics is already selected, it is not necessary to select the check box again.
 - Click the **Start** button.
17. On the next page, there are three distinct outputs:
 - A table displaying the top 10 values for Free(KB), Cache(KB), Usr%_CPU, Sys%_CPU.
 - A table displaying the actual values of the Free(KB) and Cache(KB) that has been collected.
 - A graph displaying the Usr%_CPU and Sys%CPU values.
18. After reviewing the data, click the **[Home]** link in the top right-hand corner to return to the dim_STAT welcome page.
19. On the dim_STAT welcome page, click the **Stop Active Collect(s)** link in the menu of options located on the right-hand side of the page.
20. On the dim_STAT STOP page, choose the active collect by selecting the check box to the left of the ID of 1040. Click the **Stop STAT Collection(s)** button.
21. Exit the browser.
22. As `root`, stop the dim_STAT service by executing the following command:

```
sys> /etc/STATsrv/STAT-service stop
```
23. As `root`, stop the dim_STAT server by executing the following command:

```
sys> /apps/ADMIN/dim_STAT-Server stop
```
24. Start the Apache web server as `root`.
25. Start the MySQL server as `root`.

26. As root, execute the following command to view the output of the mysqlresources script:

```
sys> /stage/mysqlresources -u root -p oracle
```

- The output of the mysqlresources script is based on content from the Linux ps command, the /proc directory, and the Linux ptree command to produce a detailed listing about the CPU and memory consumption of the MySQL server process.
- What is the average percentage of CPU power that the MySQL server is currently consuming?

-
- What is the percentage of memory that the MySQL server is currently consuming?
-

27. Review the Maatkit tools available on this server by displaying the contents of the /stage/Maatkit-7041/bin directory.

Note: The Maatkit application contains multiple tools. For this practice, you review a sampling of these tools to have an introductory understanding of how these tools work.

28. Using the Maatkit find tool (mk-find), find all the tables in the local MySQL server instance that use the InnoDB storage engine by entering the following command:

```
sys> mk-find -uroot -poracle --engine InnoDB
```

- How many tables from the world_innodb database are using the InnoDB storage engine?

29. Using the Maatkit find tool (mk-find), find all the tables in the local MySQL server instance that have a tablesize of 50 MB or greater by entering the following command:

```
sys> mk-find -uroot -poracle --tablesize +50M
```

- How many tables from the local MySQL server instance contain data greater than 50 MB?

30. Using the Maatkit system variable advisor (mk-variable-advisor), review all the possible problems with the current settings of the system variables by entering the following command:

```
sys> mk-variable-advisor -uroot -poracle localhost
```

- How many warnings did the Maatkit variable advisor display? _____
- How many notes did the Maatkit variable advisor display? _____

Note: As with any automated advisor, the potential problems with your systems setting should be reviewed and analyzed for your system needs before implementing any changes suggested.

31. Restart the MySQL server as root.

- Restarting the server resets all the variables you are using in this practice which makes it easier to see the changes taking place.

Note: Restarting the server is acceptable in a classroom and development environment, but should be avoided in production environments unless absolutely necessary.

32. In the /stage/mysqlreport-3.5 directory, execute the mysqlreport application as root and output the content of the report produced to a new file called mysqlreport_class1.txt in the /tmp directory:

```
sys> mysqlreport --user root \
> --password oracle > /tmp/mysqlreport_class1.txt
```

Note: The script execution produces 4 warning messages that start with “Use of uninitialized value ...” These warnings do not interfere with the execution of the script and can be ignored.

33. Review the contents of the /tmp/mysqlreport_class1.txt file.

- Under the Query Cache heading, what is the current % of the Query Cache that is used by the server?

34. Using the mysql client, execute the /stage/scripts/Qcache-queries.sql against the MySQL server.

35. In the /stage/mysqlreport-3.5 directory, execute the mysqlreport application as root and output the content of the report produced to a new file called mysqlreport_class2.txt in the /tmp directory:

```
sys> mysqlreport --user root \
> --password oracle > /tmp/mysqlreport_class2.txt
```

Note: The script execution produces 4 warning messages that start with “Use of uninitialized value ...” These warnings do not interfere with the execution of the script and can be ignored.

36. Review the contents of the /tmp/mysqlreport_class2.txt file.

- Under the Query Cache heading, what is the current % of the Query Cache that is used by the server?

Note: This is a simple example, but the power of comparing the output of the mysqlreport application over a period of time can be invaluable to monitor trends in how your MySQL server is performing.

37. Use the following command to start the `mytop` application:

```
sys> mytop --user root --password oracle -d world
```

- There are two sections of the `mytop` application. The header portion provides valuable information about the MySQL server instance threads:
 - The top line displays the hostname of the server and the uptime of the MySQL server.
 - The second line displays the total number of queries (`Queries :`), the average number of queries per second (`qps :`), the number of slow queries (`Slow :`), and the `Se/In/Up/De (%)` values display the number of `SELECT`, `INSERT`, `UPDATE`, and `DELETE` operations that have taken place.
 - The third line displays the average number of queries per second that the server has handled since the last time the `mytop` application refreshed (`qps now :`), the average number of slow queries per second that the server has handled since the last time the `mytop` application was refreshed (`Slow qps :`), and the number of threads (`Threads :`) connected, running, and cached.
 - The numbers to the right of the threads output displays the number of thread cache hits per second, the number of thread cache hits since the `mytop` application refreshed, the thread cache hit ratio, and the thread cache hit ratio since the `mytop` application refreshed.

Note: The `mytop` application refreshes every five seconds by default. You can change the refresh time by adding the `-s` option followed by the number of seconds between refreshes to the `mytop` command line. You can also press the [S] key to change the refresh rate while the `mytop` application is running.

- The fourth line displays the key efficiency of the queries run against the server. In addition, the bytes per second (`Bps in/out :`) are displayed with the first number being the average number of bytes per second coming into the server, and the second number being the average number of bytes per second going out from the server since the server has been started. The `Now in/out:` output is equivalent to the `Bps in/out:` output with the numbers displayed reflecting the values since the last time the `mytop` application refreshed.
- The lower portion is referred to as the `mytop` thread body. In this portion of the output, the current threads and which statement that thread is executing is displayed. There are multiple commands you can execute concerning this portion of the output:
 - Pressing the [H] key, displays only the `mytop` thread body. Pressing the [H] key again, returns the header information.
 - Pressing the [E] key prompts you for a query id. Enter the query id from the list of running threads to see an explanation of the statement being executed.
 - Pressing the [O] key sorts the threads based on the time that the statement has been executing.

38. Press the [C] key while the `mytop` application is running.
 - The output displayed is the `mytop` command view. The `mytop` command view is made up of five columns:
 - The first column is the command/statement that is running (`Command`).
 - The second column is the total number of times this command/statement has run since the server has started (`Total`).
 - The third column is the percentage of times this command/statement has been run in comparison to all the commands/statements executed against the server since the server has started (`Pct`).
 - The fourth column is the total number of times the command/select statement has run since the last refresh of the `mytop` application (`Last`).
 - The fifth column is the percentage of times this command/statement has been run in comparison to all the commands/statements executed against the server since the last refresh of the `mytop` application (`Pct`).
39. Press the [T] key while the `mytop` application is running to return to the `mytop` thread view.
40. Press the [?] key while the `mytop` application is running to see a complete list of all available shortcut keys for the application.
41. Terminate the running of the `mytop` application.

Solutions 3-2: Open Source Community Monitoring Tools

Tasks

Note: These solutions are sample solutions and are not expected to be the exact output you see in your execution of the steps.

1. Stop the MySQL server as root.

```
sys> /etc/init.d/mysql stop
```

- The first open source community application you use for this lesson is dim_STAT. To prevent complications with the MySQL server that dim_STAT uses, it is best to shut down the current instance of the MySQL server that is running.

2. Stop the Apache web server as root.

```
sys> /etc/init.d/apache2 stop
```

- To prevent complications with the Apache web server instance that dim_STAT uses, it is best to shut down the current instance of the Apache web server that is running.

Note: This application may not be running. If an error is displayed when you attempt to shut down the Apache web server, it can be ignored.

3. As root, start the dim_STAT server by executing the following command:

```
sys> /apps/ADMIN/dim_STAT-Server start
```

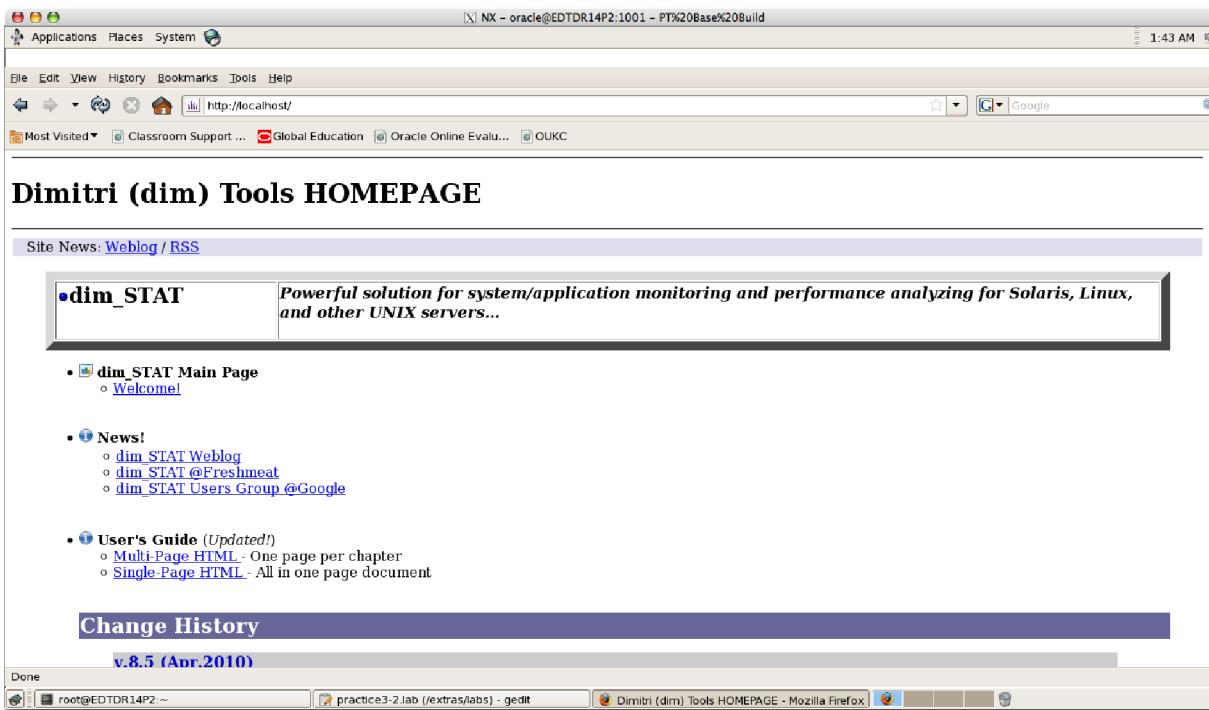
4. As root, start the dim_STAT service by executing the following command:

```
sys> /etc/STATsrv/STAT-service start
```

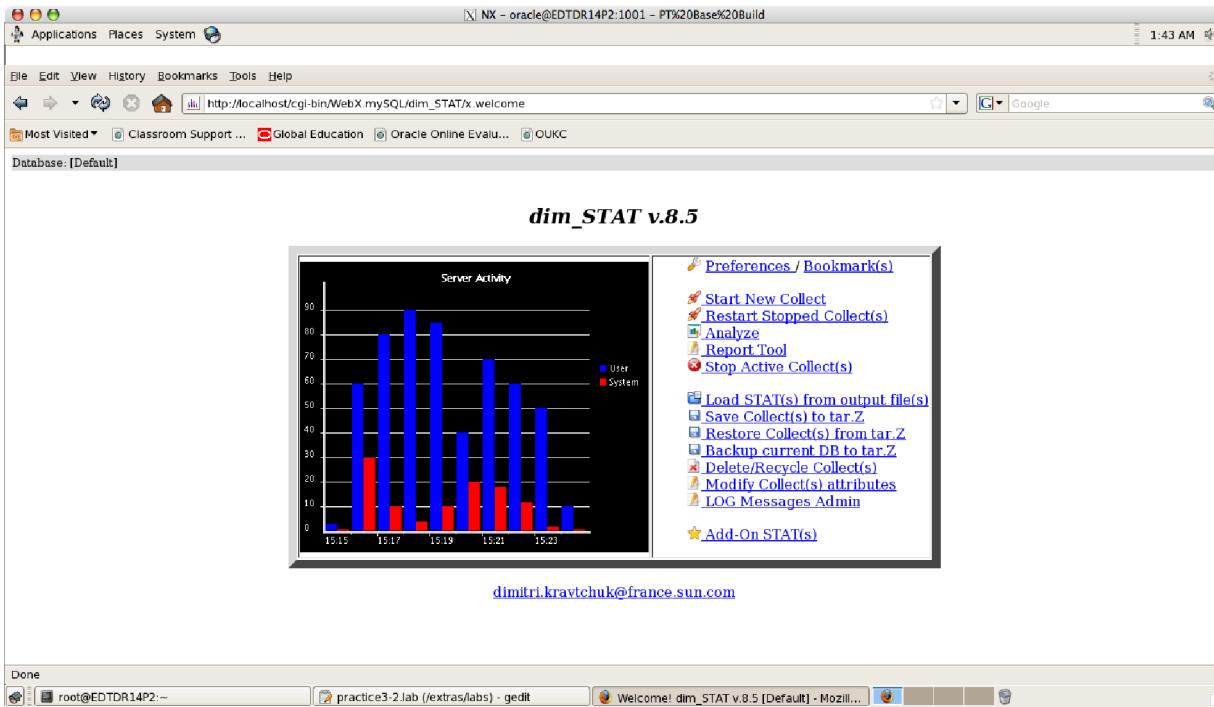
5. In your web browser, enter the following website address:

```
http://localhost
```

- This opens the **Dimitri (dim) Tools HOMEPAGE**.
- Click the **Welcome!** link under the heading **dim_STAT Main Page**.



6. On the dim_STAT welcome page that is opened, click the **Start New Collect** link in the menu of options located on the right-hand side of the page.



7. On the dim_STAT Start New Collect(s) page, select the check box to the left of the sun/localhost:5000 host. Click the **Continue** button.

dim_STAT Start New Collect(s)

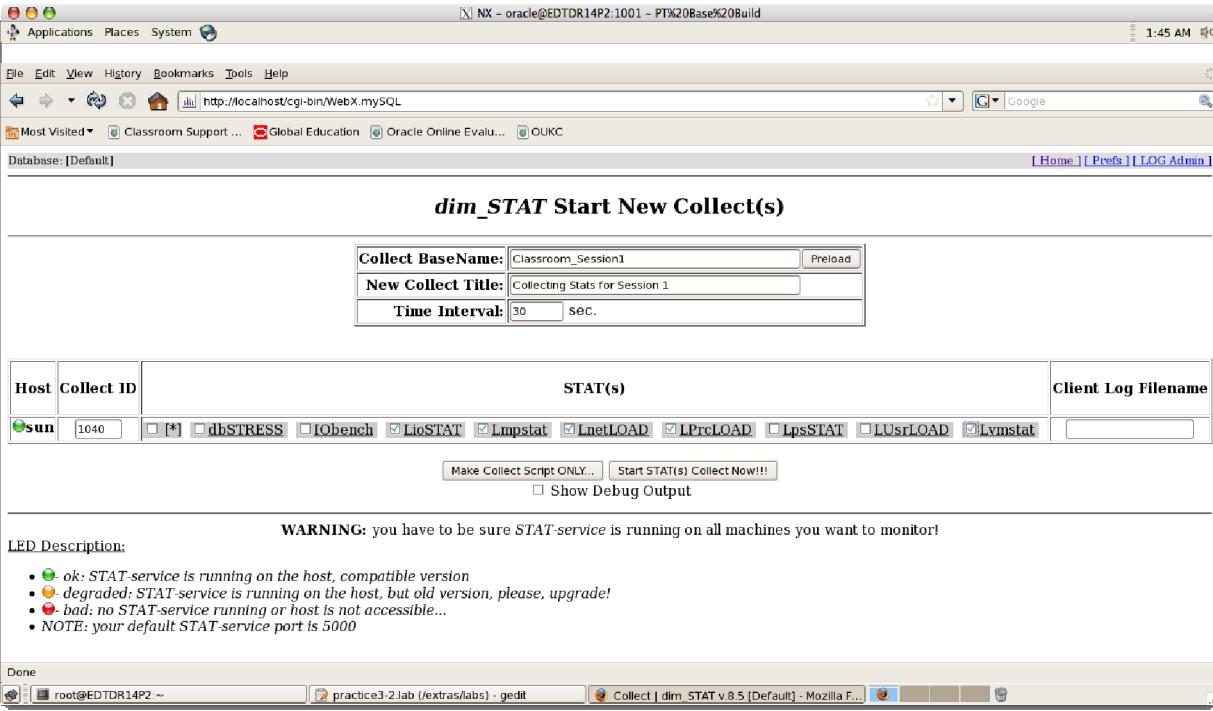
Host(s)

- sun/localhost:5000
- [ALL]
- New:

Preselect All STATS matching...
STATS List: Compact

LED Description:

- ● ok: STAT-service is running on the host, compatible version
- ● degraded: STAT-service is running on the host, but old version, please, upgrade!
- ● bad: no STAT-service running or host is not accessible...
- NOTE: your default STAT-service port is 5000

8. On the next page, use the following values to complete the form:
- Collect Basename: Classroom_Session1
 - New Collect Title: Collecting Stats for Session 1
 - Collect ID: 1040
 - STAT(s): Choose Liostat, Lmpstat, LnetLOAD, LPrcLOAD, and Lvmstat
 - Click the **Start STAT(s) collect now!!!** button.
- 
- The screenshot shows the 'dim_STAT Start New Collect(s)' configuration page. It includes fields for 'Collect BaseName' (Classroom_Session1), 'New Collect Title' (Collecting Stats for Session 1), and 'Time Interval' (30 sec). Below this is a table for hosts and STAT services. A warning message at the bottom states: 'WARNING: you have to be sure STAT-service is running on all machines you want to monitor!' followed by a legend for LED descriptions: ok (green), degraded (orange), and bad (red).
9. On the next page, a listing of the active databases and their utilization is displayed. Click the [\[Home \]](#) link in the top right-hand corner to return to the dim_STAT welcome page.
10. On the dim_STAT welcome page, click the **Delete/Recycle Collect(s)** link in the menu of options located on the right-hand side of the page.

11. On the dim_STAT DELETE page, active and stopped collects are listed.
- Only stopped collect(s) can be deleted.
 - This is a good location to view the number of records that have been stored for each collect.

ID	Host	Title	Started	Time Interval	#Records	State
1	goldgate	Demo collect, just to see it's ok!	1998-12-18 15:28:27	15	8413	Stopped
1040	sun	Collecting Stats for Session 1 (@localhost)	2011-03-23 09:39:04	30	25	Active!

Remove ALL data
 Keep only data of day(s)
 Recently collected
 Firstly collected

Remove Data

WARNING! Delete operation may temporary block all active collects during execution...

Active Databases	
DATABASE	#CONNECTIONS
Default	5

12. Click the **[Home]** link in the top right-hand corner to return to the dim_STAT welcome page.
13. On the dim_STAT welcome page, click the **Analyze** link in the menu of options located on the right-hand side of the page.

14. On the dim_STAT Analyze page, use the following values to complete the form:

- Choose the **Single Host Analyze** and **Active Only** options.
- Click the **Analyze** button.

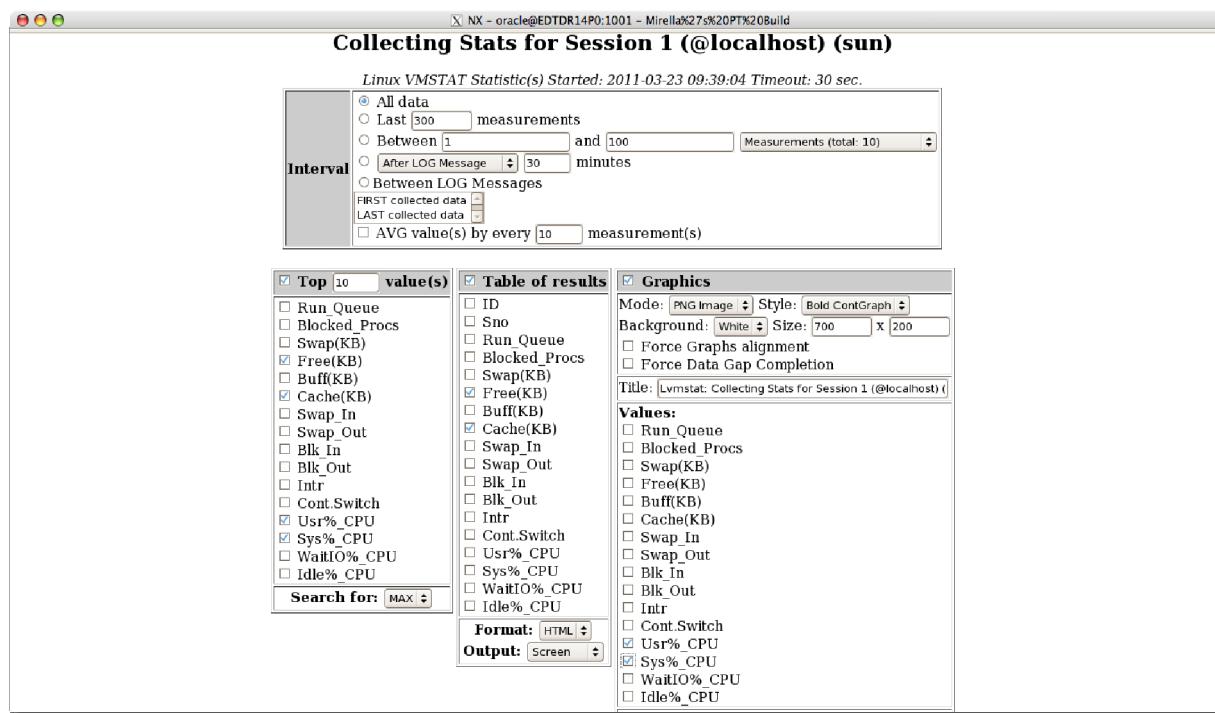
The screenshot shows the 'dim_STAT Analyze' configuration page. A large gray box highlights the 'Single-Host Analyze' radio button and the 'Active ONLY' checkbox. Below these are several other optional checkboxes: 'Preselect Multi-Hosts', 'Show STAT(s) Status', 'Titles Matching:' (with a dropdown menu), 'Hostname Matching:' (with a dropdown menu), and 'LOG Messages Matching:' (with a dropdown menu). At the bottom of the page are 'Analyze' and 'Reset' buttons.

15. On the next page, choose the active collect by selecting the check box to the left of the ID of 1040. Click the **Lvmstat** button.

The screenshot shows the 'dim_STAT Analyzer' page displaying a table of active collects. One row is selected, showing ID 1040, Host sun, Title 'Collecting Stats for Session 1 (@localhost)', Started at 2011-03-23 09:39:04, Interval 30, #Records 158, and State Active!. Below the table is a section for 'Use LogFile Messages from:' with a dropdown set to 'the same host' and a matching input field. At the bottom are several buttons: LIOSTAT, Lmpstat, LnetLOAD, LPrcLOAD, Lvmstat, and a 'Bookmarks' dropdown.

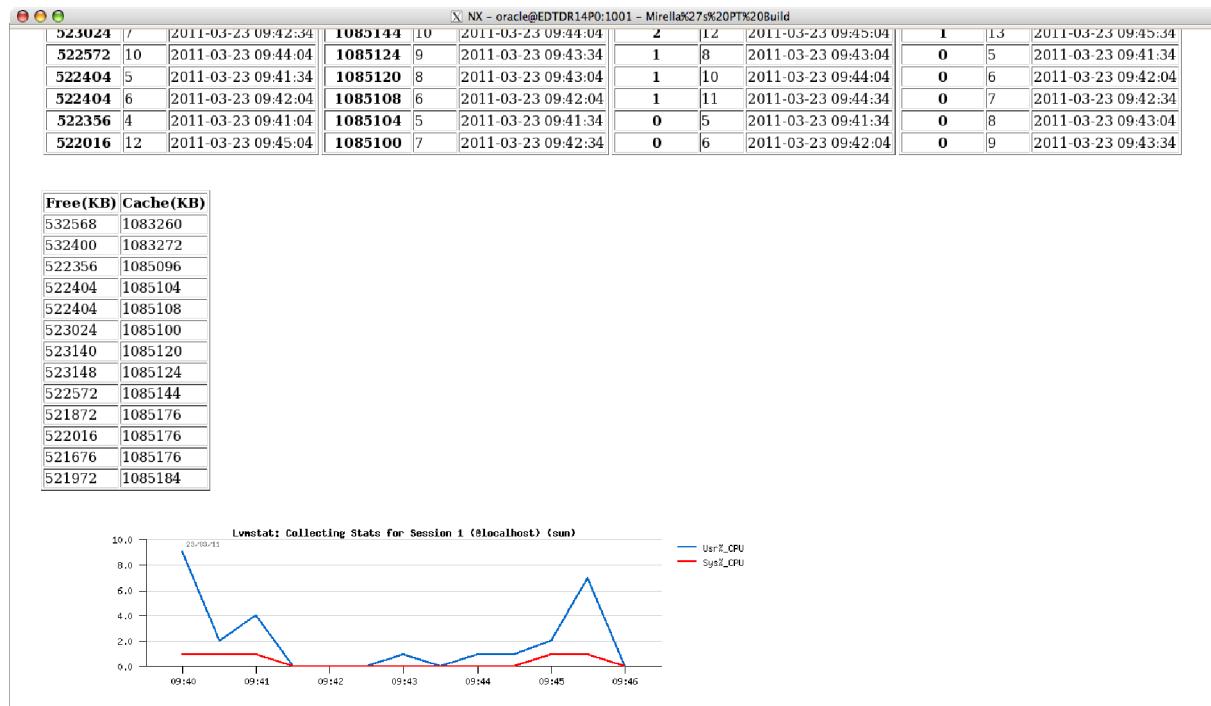
16. On the Collecting Stats for Session 1 (@localhost) page, use the following values to complete the form:

- In the Interval table, choose the **All Data** option.
 - Select the check box to the left of the table heading “Top 10 Values”, choose the **Free(KB)**, **Cache(KB)**, **Usr%_CPU**, and **Sys%_CPU** options.
- Note:** The value of 10 in the header of this table can be changed to number larger or smaller than 10.
- Select the check box to the left of the table heading “Table of results”. Choose the **Free(KB)** and **Cache(KB)** options.
 - Select the check box to the left of the table heading “Graphics.” Choose the **Usr%_CPU** and **Sys%CPU** options.
 - If the check box to turn on the Graphics is already selected, it is not necessary to select the check box again.
 - Click the **Start** button.



17. On the next page, there are three distinct outputs:

- A table displaying the top 10 values for Free(KB), Cache(KB), Usr%_CPU, Sys%_CPU.
- A table displaying the actual values of the Free(KB) and Cache(KB) that has been collected.
- A graph displaying the Usr%_CPU and Sys%CPU values.



18. After reviewing the data, click the **[Home]** link in the top right-hand corner to return to the dim_STAT welcome page.

19. On the dim_STAT welcome page, click the **Stop Active Collect(s)** link in the menu of options located on the right-hand side of the page.

20. On the dim_STAT STOP page, choose the active collect by selecting the check box to the left of the ID of 1040. Click the **Stop STAT Collection(s)** button.

ID	Host	Title	Started	Time Interval	#Records	State
<input checked="" type="checkbox"/> 1040	sun	Collecting Stats for Session 1 (@localhost)	2011-03-23 09:39:04	30	-	Active!

Add Log Message:

Active Databases

DATABASE	#CONNECTIONS
Default:	5
Total:	5
Max:	1014

DATABASE	#Active Collect(s)
Default:	1

21. Exit the browser.
22. As root, stop the dim_STAT service by executing the following command:
- ```
sys> /etc/STATsrv/STAT-service stop
```
23. As root, stop the dim\_STAT server by executing the following command:
- ```
sys> /apps/ADMIN/dim_STAT-Server stop
```
24. Start the Apache web server as root.
- ```
sys> /etc/init.d/apachectl start
```
25. Start the MySQL server as root.
- ```
sys> /etc/init.d/mysql start
```

26. As root, execute the following command to view the output of the mysqlresources script:

```
sys> /stage/mysqlresources -u root -p oracle
The MySQL Resource Locator, version 1.0
(C) 2006 Giuseppe Maxia, Stardata s.r.l.

hostname          : EDTDR14P2
mysql_pid         :        22629
pid_file          : /var/lib/mysql/EDTDR14P2.pid
mysqld_safe       :        22336
open_files         :            33

user
-----
name      :      mysql
UID       :        101

mysql_load
-----
avg_perc_cpu   :      0.35
perc_memory    :      7.20
vmem_size      :    737332
res_set_size   :    77268
data_size       :  727700
pids           :        18
active_pids    :        2

server_load
-----
cpu5          :      1.53
cpu10         :      1.41
cpu15         :      1.36
```

- The output of the mysqlresources script is based on content from the Linux ps command, the /proc directory, and the Linux ptree command to produce a detailed listing about the CPU and memory consumption of the MySQL server process.
- What is the average percentage of CPU power that the MySQL server is currently consuming?

0.35%

- What is the percentage of memory that the MySQL server is currently consuming?

7.20%

27. Review the Maatkit tools available on this server by displaying the contents of the /stage/maatkit-7041/bin directory.

```
sys> ls /stage/Maatkit-7041/bin
mk-archiver          mk-log-player        mk-slave-find
mk-checksum-filter   mk-merge-mqd-results  mk-slave-move
mk-deadlock-logger   mk-parallel-dump     mk-slave-prefetch
mk-duplicate-key-checker  mk-parallel-restore  mk-slave-restart
mk-error-log         mk-profile-compact   mk-table-checksum
mk-fifo-split        mk-purge-logs       mk-table-sync
mk-find              mk-query-advisor    mk-upgrade
mk-heartbeat         mk-query-digest    mk-variable-advisor
mk-index-usage       mk-query-profiler   mk-visual-explain
mk-kill              mk-show-grants    mk-slave-delay
mk-loadavg
```

28. Using the Maatkit find tool (mk-find), find all the tables in the local MySQL server instance that use the InnoDB storage engine by entering the following command:

```
sys> mk-find -uroot -poracle --engine InnoDB
`employees`.`departments`
`employees`.`dept_emp`
`employees`.`dept_manager`
`employees`.`employees`
`employees`.`salaries`
`employees`.`titles`
`isfdb`.`mw_user_groups`
`many_tables`.`BLOB_InnoDB`
`many_tables`.`many_tb1s0`
`many_tables`.`many_tb1s1`
...
`sakila`.`store`
`world_NonNorm`.`world_all`
`world_innodb`.`City`
`world_innodb`.`Country`
`world_innodb`.`CountryLanguage`
```

- How many tables from the world_innodb database are using the InnoDB storage engine?
- 3

29. Using the Maatkit find tool (mk-find), find all the tables in the local MySQL server instance that have a table size of 50 MB or greater by entering the following command:

```
sys> mk-find -uroot -poracle --tablesize +50M
`employees`.`salaries`
`isfdb`.`titles`
`many_tables`.`BLOB_InnoDB`
`many_tables`.`BLOB_MyISAM`
```

- How many tables from the local MySQL server instance contain data greater than 50 MB?

4

30. Using the Maatkit system variable advisor (`mk-variable-advisor`), review all the possible problems with the current settings of the system variables by entering the following command:

```
sys> mk-variable-advisor -uroot -poracle localhost
# WARN delay_key_write: MyISAM index blocks are never flushed until necessary.

# WARN innodb_log_file_size: The InnoDB log file size is set to its default
value, which is not usable on production systems.

# NOTE innodb_max_dirty_pages_pct: The innodb_max_dirty_pages_pct is lower
than the default.

# NOTE log_warnings-2: Log_warnings must be set greater than 1 to log unusual
events such as aborted connections.

# NOTE max_connect_errors: max_connect_errors should probably be set as large
as your platform allows.

# NOTE read_buffer_size-1: The read_buffer_size variable should generally be
left at its default unless an expert determines it is necessary to change it.

# NOTE read_rnd_buffer_size-1: The read_rnd_buffer_size variable should
generally be left at its default unless an expert determines it is necessary
to change it.

# WARN read_rnd_buffer_size-2: The read_rnd_buffer_size variable should not be
larger than 4M.

# WARN slave_net_timeout: This variable is set too high.

# NOTE sort_buffer_size-1: The sort_buffer_size variable should generally be
left at its default unless an expert determines it is necessary to change it.

# NOTE innodb_data_file_path: Auto-extending InnoDB files can consume a lot of
disk space that is very difficult to reclaim later.

# NOTE innodb_flush_method: Most production database servers that use InnoDB
should set innodb_flush_method to O_DIRECT to avoid double-buffering, unless
the I/O system is very low performance.

# WARN myisam_recover_options: myisam_recover_options should be set to some
value such as BACKUP, FORCE to ensure that table corruption is noticed.

# WARN sync_binlog: Binary logging is enabled, but sync_binlog isn't
configured so that every transaction is flushed to the binary log for
durability.
```

- How many warnings did the Maatkit variable advisor display? 6
- How many notes did the Maatkit variable advisor display? 8

Note: As with any automated advisor, the potential problems with your systems setting should be reviewed and analyzed for your system needs before implementing any changes suggested.

31. Restart the MySQL server as root.

```
sys> /etc/init.d/mysql restart
```

32. In the /stage/mysqlreport-3.5 directory, execute the mysqlreport application as root and output the content of the report produced to a new file called mysqlreport_class1.txt in the /tmp:

```
sys> mysqlreport --user root \
> --password oracle > /tmp/mysqlreport_class1.txt
```

Note: The script execution produces 4 warning messages that start with “Use of uninitialized value ...” These warnings do not interfere with the execution of the script and can be ignored.

33. Review the contents of the /tmp/mysqlreport_class1.txt file.

- Under the Query Cache heading, what is the current % of the Query Cache that is used by the server?

0.03%

34. Using the mysql client, execute the /stage/scripts/Qcache-queries.sql against the MySQL server.

```
sys> mysql -uroot -poracle < /stage/scripts/Qcache-queries.sql
```

35. In the /stage/mysqlreport-3.5 directory, execute the mysqlreport application as root and output the content of the report produced to a new file called mysqlreport_class2.txt in the /tmp directory:

```
sys> mysqlreport --user root \
> --password oracle > /tmp/mysqlreport_class2.txt
```

Note: The script execution produces 4 warning messages that start with “Use of uninitialized value ...” These warnings do not interfere with the execution of the script and can be ignored.

36. Review the contents of the /tmp/mysqlreport_class2.txt file.

- Under the Query Cache heading, what is the current % of the Query Cache that is used by the server?

0.64%

37. Use the following command to start the mytop application:

```
sys> mytop --user root --password oracle -d world
MySQL on localhost (5.5.8-log)                               up 0+00:33:20 [13:29:17]
    Queries: 239.0  qps:      0 Slow:      0.0          Se/In/Up/De(%):   140/00/00/00
              qps now:      0 Slow qps: 0.0  Threads:     1 (   1/   1) 00/00/00/00
    Key Efficiency: 100.0%  Bps in/out:   2.6/501.0  Now in/out:   8.0/ 1.6k

      Id      User        Host/IP        DB      Time      Cmd Query or State
      --      ----        -----        --      ---      --- -----
     10      root       localhost      world      0  Query show full process
```

- There are two sections of the mytop application. The header portion provides valuable information about the MySQL server instance threads:
 - The top line displays the hostname of the server and the uptime of the MySQL server.

- The second line displays the total number of queries (Queries :), the average number of queries per second (qps :), the number of slow queries (Slow :), and the Se/In/Up/De (%) values display the number of SELECT, INSERT, UPDATE, and DELETE operations that have taken place.
- The third line displays the average number of queries per second that the server has handled since the last time the mytop application refreshed (qps now :), the average number of slow queries per second that the server has handled since the last time the mytop application was refreshed (Slow qps :), and the number of threads (Threads :) connected, running, and cached.
 - The numbers to the right of the threads output displays the number of thread cache hits per second, the number of thread cache hits since the mytop application refreshed, the thread cache hit ratio, and the thread cache hit ratio since the mytop application refreshed.
- Note:** The mytop application refreshes every five seconds by default. You can change the refresh time by adding the -s option followed by the number of seconds between refreshes to the mytop command line. You can also press the [S] key to change the refresh rate while the mytop application is running.
- The fourth line displays the key efficiency of the queries run against the server. In addition, the bytes per second (Bps in/out :) are displayed with the first number being the average number of bytes per second coming into the server, and the second number being the average number of bytes per second going out from the server since the server has been started. The Now in/out: output is equivalent to the Bps in/out: output with the numbers displayed reflecting the values since the last time the mytop application refreshed.
- The lower portion is referred to as the mytop thread body. In this portion of the output, the current threads and which statement that thread is executing is displayed. There are multiple commands you can execute concerning this portion of the output:
 - Pressing the [H] key, displays only the mytop thread body. Pressing the [H] key again, returns the header information.
 - Pressing the [E] key prompts you for a query id. Enter the query id from the list of running threads to see an explanation of the statement being executed.
 - Pressing the [O] key sorts the threads based on the time that the statement has been executing.

38. Press the [C] key while the `mytop` application is running.

Command	Total	Pct	Last	Pct
show processlist	11	44%	1	16%
show status	11	44%	1	16%
show variables	1	4%	1	16%
set option	1	4%	1	16%
change db	1	4%	1	16%
Compression	0	0%	1	16%

- The output displayed is the `mytop` command view. The `mytop` command view is made up of five columns:
 - The first column is the command/statement that is running (Command).
 - The second column is the total number of times this command/statement has run since the server has started (Total).
 - The third column is the percentage of times this command/statement has been run in comparison to all the commands/statements executed against the server since the server has started (Pct).
 - The fourth column is the total number of times the command/select statement has run since the last refresh of the `mytop` application (Last).
 - The fifth column is the percentage of times this command/statement has been run in comparison to all the commands/statements executed against the server since the last refresh of the `mytop` application (Pct).

39. Press the [T] key while the `mytop` application is running to return to the `mytop` thread view.

40. Press the [?] key while the `mytop` application is running to see a complete list of all available shortcut keys for the application.

```
? - display this screen
# - toggle short/long numbers (not yet implemented)
c - command summary view (based on Com_* counters)
d - show only a specific database
e - explain the query that a thread is running
f - show full query info for a given thread
F - unFilter the dispaly
h - show only a specifc host's connections
H - toggle the mytop header
i - toggle the display of idle (sleeping) threads
I - show innodb status
k - kill a thread
p - pause the display
m - switch [mode] to qps (queries/sec) scrolling view
o - reverse the sort order (toggle)
q - quit
r - reset the status counters (via FLUSH STATUS on your server)
s - change the delay between screen updates
t - switch to thread view (default)
u - show only a specific user
: - enter a command (not yet implemented)
```

41. Terminate the running of the `mytop` application.

- Pressing “q” while `mytop` is running terminates the application.

Practice 3-3: Benchmark Tools

Overview

In this practice, you use and compare the different benchmarking tools available. To accomplish this objective, you do the following:

- Run all the tests of the **sql-bench** benchmark suite against the MySQL server.
 - Review the output of the tests.
- Prepare and run the **mysqlslap** benchmark tool using a predefined script.
 - Review the output of the test.
- Prepare and run the **sysbench** benchmark tool using built-in tests against the MySQL server.
 - Review the output of the test.

Assumptions

- The MySQL server is installed and running.
- The **sql-bench** benchmark suite is installed.
- The **mysqlslap** application is installed.
- The **sysbench** application is installed.
- The `dept_employees.sql` sample file is accessible and located in the `/stage/scripts` directory.

Duration

- This practice should take 30 minutes to complete.

Tasks

1. Restart the MySQL server as `root`.
2. In the `/usr/bin/sql-bench` directory, execute the following command as `root` to run all the tests of the MySQL benchmark suite:

```
sys> perl run-all-tests \
> --small-test \
> --create-options=ENGINE=INNODB \
> --user="root" \
> --password="oracle" \
> | tee /tmp/sql-bench-run-all-tests.txt
```

- The `--small-test` option tells the `sql-bench` tests to speed up the tests by using smaller limits.
 - The `--create-options` option tells the `sql-bench` tests to create all tables with the InnoDB storage engine.
 - The output of the screen is captured to a file called `sql-bench-run-all-tests.txt` in the `/tmp` directory for review.
3. List the files located in the `/usr/bin/sql-bench/output` directory.
 - Each of the test results (nine in total) have been individually stored in files located in the `/usr/bin/sql-bench/output` directory.

4. Review the contents of the file that starts with the word “select”.
 - Locate the Total time line. What is the total time that the execution of the test took to complete?

5. Review the contents of the /tmp/sql-bench-run-all-tests.txt file.
 - What is the total time that the application took to complete all the tests?

6. Using the mysql client, create a database called mysqlslap.

Note: This database is used with the mysqlslap application. If this database did not exist, the mysqlslap application would still work for this practice, but there would be warnings produced during the execution.

7. As root, use the mysqlslap application to create a load on the MySQL server to test the table_open_cache setting by issuing the following command in a terminal window:

```
sys> mysqlslap \
> --user=root \
> --password=oracle \
> --query="/stage/scripts/dept_employees.sql" \
> --iterations=10 \
> --concurrency=10
```

- The --iterations option tells mysqlslap how many times they should execute the script for each connection.
- The --concurrency option tells mysqlslap how many connections it should simulate connecting to the MySQL server.
- The --query options tells mysqlslap which script to execute for the test.
- What was the average number of seconds to run all queries? _____

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

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

- This results in an output similar to the following:

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

No DB drivers specified, using mysql
Creating table 'sbtest'...
Creating 50000 records in table 'sbtest'...
```

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

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

- The run step for the sysbench application executes 10,000 read requests simulating 128 threads against the MySQL server.
- How many transactions were completed per second? _____
- What was the average time that the MySQL server completed each of the events?

10. Drop and recreate the test database using the mysql client.

Note: This removes any tables that were created for running the benchmark tests.

Solutions 3-3: Benchmark Tools

Tasks

Note: These solutions are sample solutions and are not expected to be the exact output you see in your execution of the steps.

1. Restart the MySQL server as root.

```
sys> /etc/init.d/mysql restart
```

2. In the /usr/bin/sql-bench directory, execute the following command as root to run all the tests of the MySQL benchmark suite:

```
sys> cd /usr/bin/sql-bench
sys> perl run-all-tests \
> --small-test \
> --create-options=ENGINE=INNODB \
> --user="root" \
> --password="oracle" \
> | tee /tmp/sql-bench-run-all-tests.txt
```

- The --small-test option tells the sql-bench tests to speed up the tests by using smaller limits.
- The --create-options option tells the sql-bench tests to create all tables with the InnoDB storage engine.
- The output of the screen is captured to a file called sql-bench-run-all-tests.txt in the /tmp directory for review.

3. List the files located in the /usr/bin/sql-bench/output directory.

```
sys> ls /usr/bin/sql-bench/output/
alter-table-mysql-Linux_2.6.18_164.el5_i686
ATIS-mysql-Linux_2.6.18_164.el5_i686
big-tables-mysql-Linux_2.6.18_164.el5_i686
connect-mysql-Linux_2.6.18_164.el5_i686
create-mysql-Linux_2.6.18_164.el5_i686
insert-mysql-Linux_2.6.18_164.el5_i686
select-mysql-Linux_2.6.18_164.el5_i686
transactions-mysql-Linux_2.6.18_164.el5_i686
wisconsin-mysql-Linux_2.6.18_164.el5_i686
```

- Each of the test results (nine in total) have been individually stored in files located in the /usr/bin/sql-bench/output directory.

4. Review the contents of the file that starts with the word "select".

```
sys> more /usr/bin/sql-bench/output/select-mysql-Linux_2.6.18_164.el5_i686
Testing server 'MySQL 5.5.8 log' at 2011-02-02 0:38:51
Testing the speed of selecting on keys that consist of many parts
The test-table has 1000 rows and the test is done with 500 ranges.
Creating table
Inserting 1000 rows
Time to insert (1000): 1 wallclock secs ( 0.04 usr 0.03 sys + 0.00 cusr
0.00 csys = 0.07 CPU)
```

```

Test if the database has a query cache
Time for select_cache (1000): 0 wallclock secs ( 0.12 usr  0.01 sys +  0.00
cusr  0.00 csys =  0.13 CPU)
Time for select_cache2 (1000): 1 wallclock secs ( 0.13 usr  0.01 sys +  0.00
cusr  0.00 csys =  0.14 CPU)
Testing big selects on the table
Time for select_big (7:401): 0 wallclock secs ( 0.00 usr  0.00 sys +  0.00
cusr  0.00 csys =  0.00 CPU)
...
Time for count_distinct_group (100:1000): 0 wallclock secs ( 0.02 usr  0.00
sys +  0.00 cusr  0.00 csys =  0.02 CPU)
Time for count_distinct_big (10:10000): 0 wallclock secs ( 0.04 usr  0.00 sys
+  0.00 cusr  0.00 csys =  0.04 CPU)
Total time: 4 wallclock secs ( 1.87 usr  0.27 sys +  0.00 cusr  0.00 csys =
2.14 CPU)

```

- Locate the Total time line. What is the total time that the execution of the test took to complete?

2.14 seconds

5. Review the contents of the /tmp/sql-bench-run-all-tests.txt file.

```

sys> more /tmp/sql-bench-run-all-tests.txt
Benchmark DBD suite: 2.15
Date of test: 2011-02-02 0:38:29
Running tests on: Linux 2.6.18-164.el5 i686
Arguments: --create-options=ENGINE=INNODB --small-test
Comments:
Limits from:
Server version: MySQL 5.5.8 log
Optimization: None
Hardware:

alter-table: Total time: 2 wallclock secs ( 0.02 usr  0.01 sys +  0.00 cusr
0.00 csys =  0.03 CPU)
...
wisconsin: Total time: 21 wallclock secs ( 1.57 usr  0.74 sys +  0.00 cusr
0.00 csys =  2.31 CPU)

All 9 test executed successfully

Totals per operation:
Operation      seconds    usr     sys     cpu   tests
alter_table_add          1.00    0.01    0.00    0.01    92
alter_table_drop          1.00    0.01    0.00    0.01    46
...
update_rollback           0.00    0.00    0.01    0.01    10
update_with_key           2.00    0.08    0.07    0.15   3000
update_with_key_prefix    0.00    0.06    0.02    0.08   1000
wisc_benchmark            1.00    0.62    0.05    0.67    34
TOTALS                  46.00   6.24   1.67   7.91  78867

```

- What is the total time that the application took to complete all the tests?

46 seconds

6. Using the mysql client, create a database called mysqlslap.

```
sys> mysql -uroot -poracle -e"CREATE DATABASE mysqlslap"
```

7. As root, use the mysqlslap application to create a load on the MySQL server by issuing the following command in a terminal window:

```
sys> mysqlslap \
> --user=root \
> --password=oracle \
> --query="/stage/scripts/dept_employees.sql" \
> --iterations=10 \
> --concurrency=10
```

- The --iterations option tells mysqlslap how many times they should execute the script for each connection.
- The --concurrency option tells mysqlslap how many connections it should simulate connecting to the MySQL server.
- The --query options tells mysqlslap which script to execute for the test.

Benchmark

```
Average number of seconds to run all queries: 0.102 seconds
Minimum number of seconds to run all queries: 0.100 seconds
Maximum number of seconds to run all queries: 0.107 seconds
Number of clients running queries: 10
Average number of queries per client: 200
```

- What was the average number of seconds to run all queries? 0.102 seconds

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

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

- This results in an output similar to the following:

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

No DB drivers specified, using mysql
Creating table 'sbtest'...
Creating 50000 records in table 'sbtest'...
```

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

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

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

```
sysbench 0.4.12: multi-threaded system evaluation benchmark
No DB drivers specified, using mysql
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 "LOCK TABLES WRITE" for starting transactions
Using auto_inc on the id column
Maximum number of requests for OLTP test is limited to 10000
Threads started!
Done.
```

```
OLTP test statistics:
  queries performed:
    read:          1001
    write:         0
    other:        1001
    total:        2002
  transactions:   1001 (917.06 per sec.)
  deadlocks:      0 (0.00 per sec.)
  read/write requests: 1001 (917.06 per sec.)
  other operations: 1001 (917.06 per sec.)

  Test execution summary:
    total time:           1.0915s
    total number of events: 1001
    total time taken by event execution: 51.5137
    per-request statistics:
      min:                 12.85ms
      avg:                 51.46ms
      max:                888.90ms
      approx. 95 percentile: 158.24ms

  Threads fairness:
    events (avg/stddev): 7.8203/11.44
    execution time (avg/stddev): 0.4025/0.37
```

- How many transactions were completed per second? 917.06

- What was the average time that the MySQL server completed each of the events?

51.46ms

10. Drop and recreate the `test` database using the `mysql` client.

```
sys> mysql -uroot -poracle -e"DROP DATABASE test;CREATE \
> DATABASE test;"
```


Practices for Lesson 4: MySQL Server Tuning

Chapter 4

Practices for Lesson 4

Practices Overview

In these practices, you test your knowledge of improving the performance of the MySQL server by modifying server configuration variables.

Practice 4-1: Effects of Thread Caching

Overview

In this practice, you execute a number of random queries against the MySQL server using multiple connections to determine the best setting for the `thread_cache_size` system variable. To accomplish this objective, you do the following:

- Simulate an environment where 128 connections result in approximately 900 transactions running against the MySQL server every second.
 - The number of threads that are cached can have an effect on performance.
- Modify the `thread_cache_size` system variable, simulate the environment, and then calculate the thread cache hit rate.
 - The closer the thread cache hit rate is to 100%, the less the MySQL server needs to create new threads to handle connections. This results in less overhead on the server.

Assumptions

- The MySQL server is installed and running.
- The `sysbench` application is installed.

Duration

This practice should take 45 minutes to complete.

Tasks

Note: A table has been provided at the end of the steps for this practice for you to fill in the values collected during the running of the steps. Entering the data from the steps into this table can assist you in comparing the values collected.

1. Open a terminal window and restart the server as `root`.
2. Prepare the `sysbench` tool for use by issuing the following command in a terminal window:

```
sys> sysbench \
> --test=oltp \
> --mysql-table-engine=myisam \
> --oltp-table-size=1000000 \
> --mysql-socket=/var/lib/mysql/mysql.sock \
> --mysql-user=root \
> --mysql-password=oracle \
> --mysql-db=test \
> prepare
```

3. Using the `mysql` client, set the global `thread_cache_size` system variable to 0 and the `max_connections` system variable to 300.

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

```
sys> sysbench \
> --test=oltp \
> --oltp-table-size=1000000 \
> --mysql-socket=/var/lib/mysql/mysql.sock \
> --oltp-test-mode=simple \
> --oltp-reconnect-mode=query \
> --mysql-user=root \
> --mysql-password=oracle \
> --mysql-db=test \
> --max-requests=1000 \
> --num-threads=128 \
> run
```

- The `run` step for the `sysbench` application executes 10,000 read requests simulating 128 threads against the MySQL server.
5. Using the `mysql` client, view the `Threads_created` status variable.
– What is the value of the `Threads_created` status variable? _____
6. Using the `mysql` client, view the `Connections` status variable.
– What is the value of the `Connections` status variable? _____
7. Use the values recorded in steps 5 and 6 to calculate the thread cache hit rate.
– _____
8. Execute the command from step 4.
9. Using the `mysql` client, view the `Threads_created` status variable.
– What is the value of the `Threads_created` status variable? _____
10. Using the `mysql` client, view the `Connections` status variable.
– What is the value of the `Connections` status variable? _____
11. Use the values recorded in steps 9 and 10 to calculate the thread cache hit rate.
– _____
12. Execute the command from step 4.
13. Using the `mysql` client, view the `Threads_created` status variable.
– What is the value of the `Threads_created` status variable? _____
14. Using the `mysql` client, view the `Connections` status variable.
– What is the value of the `Connections` status variable? _____
15. Use the values recorded in steps 13 and 14 to calculate the thread cache hit rate.
– _____
16. Is the thread cache hit rate getting better or worse as commands are executed against the MySQL server?

17. Restart the server as `root`.
18. Using the `mysql` client, set the global `thread_cache_size` system variable to 8 and the `max_connections` system variable to 300.
19. Execute the command from step 4.

20. Using the `mysql` client, view the `Threads_created` status variable.
 - What is the value of the `Threads_created` status variable? _____
21. Using the `mysql` client, view the `Connections` status variable.
 - What is the value of the `Connections` status variable? _____
22. Use the values recorded in steps 20 and 21 to calculate the thread cache hit rate.
- _____
23. Execute the command from step 4.
24. Using the `mysql` client, view the `Threads_created` status variable.
 - What is the value of the `Threads_created` status variable? _____
25. Using the `mysql` client, view the `Connections` status variable.
 - What is the value of the `Connections` status variable? _____
26. Use the values recorded in steps 24 and 25 to calculate the thread cache hit rate.
- _____
27. Execute the command from step 4.
28. Using the `mysql` client, view the `Threads_created` status variable.
 - What is the value of the `Threads_created` status variable? _____
29. Using the `mysql` client, view the `Connections` status variable.
 - What is the value of the `Connections` status variable? _____
30. Use the values recorded in steps 28 and 29 to calculate the thread cache hit rate.
- _____
31. Is the thread cache hit rate getting better or worse as commands are executed against the MySQL server? _____
32. Restart the server as `root`.
33. Using the `mysql` client, set the global `thread_cache_size` system variable to 16 and the `max_connections` system variable to 300.
34. Execute the command from step 4.
35. Using the `mysql` client, view the `Threads_created` status variable.
 - What is the value of the `Threads_created` status variable? _____
36. Using the `mysql` client, view the `Connections` status variable.
What is the value of the `Connections` status variable? _____
37. Use the values recorded in steps 35 and 36 to calculate the thread cache hit rate.
- _____
38. Execute the command from step 4.
39. Using the `mysql` client, view the `Threads_created` status variable.
 - What is the value of the `Threads_created` status variable? _____
40. Using the `mysql` client, view the `Connections` status variable.
 - What is the value of the `Connections` status variable? _____
41. Use the values recorded in steps 39 and 40 to calculate the thread cache hit rate.
- _____

42. Execute the command from step 4.
43. Using the mysql client, view the Threads_created status variable.
– What is the value of the Threads_created status variable? _____
44. Using the mysql client, view the Connections status variable.
– What is the value of the Connections status variable? _____
45. Use the values recorded in steps 43 and 44 to calculate the thread cache hit rate.
– _____
46. Is the thread cache hit rate getting better or worse as commands are executed against the MySQL server?

Steps	thread_cache_size	max_connections	Threads_created	Connections	Thread Cache Hit Rate
1-7	0	300			
8-11	0	300			
12-15	0	300			
19-22	8	300			
23-26	8	300			
27-30	8	300			
34-37	16	300			
38-41	16	300			
42-45	16	300			

Solutions 4-1: Effects of Thread Caching

Tasks

Note: These solutions are sample solutions and are not expected to be the exact output you see in your execution of the steps.

1. Open a terminal window and restart the server as root.

```
sys> /etc/init.d/mysql restart
```

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

```
sys> sysbench \
> --test=oltp \
> --mysql-table-engine=myisam \
> --oltp-table-size=1000000 \
> --mysql-socket=/var/lib/mysql/mysql.sock \
> --mysql-user=root \
> --mysql-password=oracle \
> --mysql-db=test \
> prepare
```

3. Using the mysql client, set the global `thread_cache_size` system variable to 0 and the `max_connections` system variable to 300.

```
sys> mysql -uroot -poracle -e"SET GLOBAL thread_cache_size=0; \
> SET GLOBAL max_connections=300;"
```

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

```
sys> sysbench \
> --test=oltp \
> --oltp-table-size=1000000 \
> --mysql-socket=/var/lib/mysql/mysql.sock \
> --oltp-test-mode=simple \
> --oltp-reconnect-mode=query \
> --mysql-user=root \
> --mysql-password=oracle \
> --mysql-db=test \
> --max-requests=1000 \
> --num-threads=128 \
> run
```

5. Using the mysql client, view the `Threads_created` status variable.

```
sys> mysql -uroot -poracle -e"SHOW STATUS LIKE \
> 'Threads_created'"
```

Variable_name	Value
Threads_created	1131

6. Using the mysql client, view the connections status variable.

```
sys> mysql -uroot -poracle -e"SHOW STATUS LIKE 'Connections'"  
+-----+  
| Variable_name | Value |  
+-----+  
| Connections | 1133 |  
+-----+
```

7. Use the values recorded in steps 5 and 6 to calculate the thread cache hit rate.

- $100 - (1131/1133)*100 = \text{approximately } 0\%$

8. Execute the command from step 4.

```
sys> sysbench \  
> --test=oltp \  
> --oltp-table-size=1000000 \  
> --mysql-socket=/var/lib/mysql/mysql.sock \  
> --oltp-test-mode=simple \  
> --oltp-reconnect-mode=query \  
> --mysql-user=root \  
> --mysql-password=oracle \  
> --mysql-db=test \  
> --max-requests=1000 \  
> --num-threads=128 \  
> run
```

9. Using the mysql client, view the Threads_created status variable.

```
sys> mysql -uroot -poracle -e"SHOW STATUS LIKE \  
> 'Threads_created'"  
+-----+  
| Variable_name | Value |  
+-----+  
| Threads_created | 2262 |  
+-----+
```

10. Using the mysql client, view the Connections status variable.

```
sys> mysql -uroot -poracle -e"SHOW STATUS LIKE 'Connections'"  
+-----+  
| Variable_name | Value |  
+-----+  
| Connections | 2264 |  
+-----+
```

11. Use the values recorded in steps 9 and 10 to calculate the thread cache hit rate.

- $100 - (2262/2264)*100 = \text{approximately } 0\%$

12. Execute the command from step 4.

```
sys> sysbench \
> --test=oltp \
> --oltp-table-size=1000000 \
> --mysql-socket=/var/lib/mysql/mysql.sock \
> --oltp-test-mode=simple \
> --oltp-reconnect-mode=query \
> --mysql-user=root \
> --mysql-password=oracle \
> --mysql-db=test \
> --max-requests=1000 \
> --num-threads=128 \
> run
```

13. Using the mysql client, view the Threads_created status variable.

```
sys> mysql -uroot -poracle -e"SHOW STATUS LIKE \
> 'Threads_created'"
```

Variable_name	Value
Threads_created	3393

14. Using the mysql client, view the Connections status variable.

```
sys> mysql -uroot -poracle -e"SHOW STATUS LIKE 'Connections'"
```

Variable_name	Value
Connections	3395

15. Use the values recorded in steps 13 and 14 to calculate the thread cache hit rate.

- $100 - (3393/3395)*100 = \text{approximately } 0\%$

16. Is the thread cache hit rate getting better or worse as commands are executed against the MySQL server? Worse, and will continue to remain ineffective.

17. Restart the server as root.

```
sys> /etc/init.d/mysql restart
```

18. Using the mysql client, set the global thread_cache_size system variable to 8 and the max_connections system variable to 300.

```
sys> mysql -uroot -poracle -e"SET GLOBAL thread_cache_size=8; \
> SET GLOBAL max_connections=300;"
```

19. Execute the command from step 4.

```
sys> sysbench \
> --test=oltp \
> --oltp-table-size=1000000 \
> --mysql-socket=/var/lib/mysql/mysql.sock \
> --oltp-test-mode=simple \
> --oltp-reconnect-mode=query \
> --mysql-user=root \
> --mysql-password=oracle \
> --mysql-db=test \
> --max-requests=1000 \
> --num-threads=128 \
> run
```

20. Using the mysql client, view the Threads_created status variable.

```
sys> mysql -uroot -poracle -e"SHOW STATUS LIKE \
> 'Threads_created'"
```

Variable_name	Value
Threads_created	489

21. Using the mysql client, view the Connections status variable.

```
sys> mysql -uroot -poracle -e"SHOW STATUS LIKE 'Connections'"
```

Variable_name	Value
Connections	1133

22. Use the values recorded in steps 20 and 21 to calculate the thread cache hit rate.

$$- 100 - (487/1132)*100 = \text{approximately } 57\%$$

23. Execute the command from step 4.

```
sys> sysbench \
> --test=oltp \
> --oltp-table-size=1000000 \
> --mysql-socket=/var/lib/mysql/mysql.sock \
> --oltp-test-mode=simple \
> --oltp-reconnect-mode=query \
> --mysql-user=root \
> --mysql-password=oracle \
> --mysql-db=test \
> --max-requests=1000 \
> --num-threads=128 \
> run
```

24. Using the `mysql` client, view the `Threads_created` status variable.

```
sys> mysql -uroot -poracle -e"SHOW STATUS LIKE \
> 'Threads_created'"
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Threads_created | 804 |
+-----+-----+
```

25. Using the `mysql` client, view the `Connections` status variables.

```
sys> mysql -uroot -poracle -e"SHOW STATUS LIKE 'Connections'""
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Connections | 2266 |
+-----+-----+
```

26. Use the values recorded in steps 24 and 25 to calculate the thread cache hit rate.

- $100 - (804/2266)*100 = \text{approximately } 64\%$

27. Execute the command from step 4.

```
sys> sysbench \
> --test=oltp \
> --oltp-table-size=1000000 \
> --mysql-socket=/var/lib/mysql/mysql.sock \
> --oltp-test-mode=simple \
> --oltp-reconnect-mode=query \
> --mysql-user=root \
> --mysql-password=oracle \
> --mysql-db=test \
> --max-requests=1000 \
> --num-threads=128 \
> run
```

28. Using the `mysql` client, view the `Threads_created` status variable.

```
sys> mysql -uroot -poracle -e"SHOW STATUS LIKE \
> 'Threads_created'"
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Threads_created | 1038 |
+-----+-----+
```

29. Using the `mysql` client, view the `Connections` status variables.

```
sys> mysql -uroot -poracle -e"SHOW STATUS LIKE 'Connections'""
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Connections | 3397 |
+-----+-----+
```

30. Use the values recorded in steps 28 and 29 to calculate the thread cache hit rate.

- $100 - (1038/3397)*100 = \text{approximately } 69\%$

31. Is the thread cache hit rate getting better or worse as commands are executed against the MySQL server? In the first few runs, a steady growth of improvement was seen, but any further runs would continue to remain at the same rate as long as the number of threads connected continues to remain the same.

32. Restart the server as root.

```
sys> /etc/init.d/mysql restart
```

33. Using the mysql client, set the global `thread_cache_size` system variable to 16 and the `max_connections` system variable to 300.

```
sys> mysql -uroot -poracle -e"SET GLOBAL thread_cache_size=16; \  
> SET GLOBAL max_connections=300;"
```

34. Execute the command from step 4.

```
sys> sysbench \  
> --test=oltp \  
> --oltp-table-size=1000000 \  
> --mysql-socket=/var/lib/mysql/mysql.sock \  
> --oltp-test-mode=simple \  
> --oltp-reconnect-mode=query \  
> --mysql-user=root \  
> --mysql-password=oracle \  
> --mysql-db=test \  
> --max-requests=1000 \  
> --num-threads=128 \  
> run
```

35. Using the mysql client, view the `Threads_created` status variable.

```
sys> mysql -uroot -poracle -e"SHOW STATUS LIKE \  
> 'Threads_created'"  
+-----+-----+  
| Variable_name | Value |  
+-----+-----+  
| Threads_created | 551 |  
+-----+-----+
```

36. Using the mysql client, view the `Connections` status variable.

```
sys> mysql -uroot -poracle -e"SHOW STATUS LIKE 'Connections'"  
+-----+-----+  
| Variable_name | Value |  
+-----+-----+  
| Connections | 1131 |  
+-----+-----+
```

37. Use the values recorded in steps 35 and 36 to calculate the thread cache hit rate.

– $100 - (551/1131)*100 = \text{approximately } 51\%$

38. Execute the command from step 4.

```
sys> sysbench \
> --test=oltp \
> --oltp-table-size=1000000 \
> --mysql-socket=/var/lib/mysql/mysql.sock \
> --oltp-test-mode=simple \
> --oltp-reconnect-mode=query \
> --mysql-user=root \
> --mysql-password=oracle \
> --mysql-db=test \
> --max-requests=1000 \
> --num-threads=128 \
> run
```

39. Using the mysql client, view the Threads_created status variable.

```
sys> mysql -uroot -poracle -e"SHOW STATUS LIKE \
'Threads_created'"
```

Variable_name	Value
Threads_created	668

40. Using the mysql client, view the Connections status variables.

```
sys> mysql -uroot -poracle -e"SHOW STATUS LIKE 'Connections'"
```

Variable_name	Value
Connections	2260

41. Use the values recorded in steps 39 and 40 to calculate the thread cache hit rate.

$$- 100 - (668/2260)*100 = \text{approximately } 70\%$$

42. Execute the command from step 4.

```
sys> sysbench \
> --test=oltp \
> --oltp-table-size=1000000 \
> --mysql-socket=/var/lib/mysql/mysql.sock \
> --oltp-test-mode=simple \
> --oltp-reconnect-mode=query \
> --mysql-user=root \
> --mysql-password=oracle \
> --mysql-db=test \
> --max-requests=1000 \
> --num-threads=128 \
> run
```

43. Using the mysql client, view the Threads_created status variable.

```
sys> mysql -uroot -poracle -e"SHOW STATUS LIKE \
> 'Threads_created'"
```

Variable_name	Value
Threads_created	817

44. Using the mysql client, view the Connections status variable.

```
sys> mysql -uroot -poracle -e"SHOW STATUS LIKE 'Connections'"
```

Variable_name	Value
Connections	3389

45. Use the values recorded in steps 43 and 44 to calculate the thread cache hit rate.

$$- 100 - (817/3389)*100 = \text{approximately } 76\%$$

46. Is the thread cache hit rate getting better or worse as commands are executed against the MySQL server? Slightly better. The thread cache hit rate will continue to remain in the mid 70% range as long as the number of threads connected continues to remain the same.

Steps	thread_cache_size	max_connections	Threads_created	Connections	Thread Cache Hit Rate
1-7	0	300	1131	1133	~ 0%
8-11	0	300	2262	2264	~ 0%
12-15	0	300	3393	3395	~ 0%
19-22	8	300	489	1133	~ 57%
23-26	8	300	804	2266	~ 64%
27-30	8	300	1038	3397	~ 69%
34-37	16	300	551	1131	~ 51%
38-41	16	300	668	2260	~ 70%
42-45	16	300	817	3389	~ 76%

Practice 4-2: Table Caching

Overview

In this practice, you execute a sampling of queries from your logs against the MySQL server to determine the best setting for the `table_open_cache` system variable. To accomplish this objective, you do the following:

- Execute the `mysqlslap` application to simulate 5 separate connections executing a sampling of 2,600 queries executed against the `employees` database.
 - Each query executed requires at least 3 tables being open in the `employees` database.
 - Each connection runs the 2,600 queries 20 times.
- Modify the `table_open_cache` system variable, simulate the environment, and then calculate the effectiveness of the `table_open_cache` setting.
 - Setting the `table_open_cache` too low results in numerous I/O events to disk, while setting the `table_open_cache` too high results in wasted memory that could be used for other purposes.

Assumptions

- The MySQL server is installed and running.
- The `PT_Stress.php` script and `dept_employees_large.sql` sample files are accessible and located in the `/stage/scripts` directory.

Duration

This practice should take 30 minutes to complete.

Task

Note: A table has been provided at the end of the steps for this practice for you to fill in the values collected during the running of the steps. Entering the data from the steps into this table can assist you in comparing the values collected.

1. Open a terminal window and shut down the MySQL server as `root`.

Note: You can use the existing terminal window from the previous lab. However, at this point, you should have only one terminal window open to minimize confusion as more terminal windows are opened up.
2. As `root`, start the MySQL server with the `table_open_cache` system variable set to 1 and the `query_cache_type` system variable set to 0.

Note: The minimum setting for the `table_open_cache` system variable is 1. Turning off the query cache by setting the `query_cache_type` to 0 prevents the query cache from affecting the outcome of the tests.
3. In the `/stage/scripts` directory, execute the following command to review the purpose and options available for use with the `PT_Stress.php` script:

```
sys> ./PT_Stress.php -h
```

Note: There are times when it may be necessary to create your own scripts to use with the MySQL server. Having an understanding of what questions you need to ask and how the MySQL server can provide the answers is a valuable tool in your tool chest.

4. In the /stage/scripts directory, use the `PT_Stress.php` script to create a load on the MySQL server to test the `table_open_cache` setting by issuing the following command in a terminal window:

```
sys> ./PT_Stress.php \
> -u root \
> -p oracle \
> -s "dept_employees_large.sql" \
> -i 20 \
> -c 5 \
> -t \
> -v "open%table%"
```

- What was the average number of seconds to run all queries? _____
 - What is the *end* value of the `open_tables` status variable? _____
 - What is the *end* value of the `opened_tables` status variable? _____
 - Using these numbers, calculate the effectiveness of the `table_open_cache` server setting:

5. As `root`, shut down the MySQL server.
6. As `root`, start the MySQL server with the `table_open_cache` system variable set to 8 and the `query_cache_type` system variable set to 0.
7. Execute the command from step 2.
- What was the average number of seconds to run all queries? _____
 - What is the *end* value of the `open_tables` status variable? _____
 - What is the *end* value of the `opened_tables` status variable? _____
 - Using these numbers, calculate the effectiveness of the `table_open_cache` server setting:

8. As `root`, shut down the MySQL server.
9. As `root`, start the MySQL server with the `table_open_cache` system variable set to 16 and the `query_cache_type` system variable set to 0.
10. Execute the command from step 2.
- What was the average number of seconds to run all queries? _____
 - What is the *end* value of the `open_tables` status variable? _____
 - What is the *end* value of the `opened_tables` status variable? _____
 - Using these numbers, calculate the effectiveness of the `table_open_cache` server setting:

11. As `root`, shut down the MySQL server.
12. As `root`, start the MySQL server with the `table_open_cache` system variable set to 32 and the `query_cache_type` system variable set to 0.

13. Execute the command from step 2.

- What was the average number of seconds to run all queries? _____
 - What is the *end* value of the `open_tables` status variable? _____
 - What is the *end* value of the `opened_tables` status variable? _____
 - Using these numbers, calculate the effectiveness of the `table_open_cache` server setting:
-

14. As `root`, shut down the MySQL server.

15. As `root`, start the MySQL server with the `table_open_cache` system variable set to 48 and the `query_cache_type` system variable set to 0.

16. Execute the command from step 2.

- What was the average number of seconds to run all queries? _____
 - What is the *end* value of the `open_tables` status variable? _____
 - What is the *end* value of the `opened_tables` status variable? _____
 - Using these numbers, calculate the effectiveness of the `table_open_cache` server setting:
-

17. As `root`, shut down the MySQL server.

18. As `root`, start the MySQL server with the `table_open_cache` system variable set to 64 and the `query_cache_type` system variable set to 0.

19. Execute the command from step 2.

- What was the average number of seconds to run all queries? _____
 - What is the *end* value of the `open_tables` status variable? _____
 - What is the *end* value of the `opened_tables` status variable? _____
 - Using these numbers, calculate the effectiveness of the `table_open_cache` server setting:
-

Step	<code>table_open_cache</code>	Seconds/Avg.	<code>open_tables</code>	<code>opened_tables</code>	Effectiveness
2-4	1				
6-7	8				
9-10	16				
12-13	32				
15-16	48				
18-19	64				

Solutions 4-2: Table Caching

Tasks

Note: These solutions are sample solutions and are not expected to be the exact output you see in your execution of the steps.

1. Open a terminal window and shut down the MySQL server as root.

```
sys> /etc/init.d/mysql stop
```

Note: You can use the existing terminal window from the previous lab. However, at this point, you should have only one terminal window open to minimize confusion as more terminal windows are opened up.

2. As root, start the MySQL server with the `table_open_cache` system variable set to 1 and the `query_cache_type` system variable set to 0.

```
sys> /etc/init.d/mysql start --table-open-cache=1 \
> --query_cache_type=0
```

3. In the `/stage/scripts` directory, execute the following command to review the purpose and options available for use with the `PT_Stress.php` script:

```
sys> ./PT_Stress.php -h
PT_Stress.php Ver 1.0 written for use with the MySQL Performance Tuning
course.

Copyright (c) 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Runs a query multiple times against the server to include output of status
variables requested.

Usage: PT_Stress.php [OPTIONS]

The following options are available:

-u [user_name]           The user name to log into the local
                         MySQL server with.

-p [password]            The password associated with the MySQL
                         user name.

-c #                     The number of connections the script
                         should simulate connecting to the local
                         MySQL server with (default is 1).

-s "script"              The location and file name of the script
                         to be run (default is "SELECT 1+1").

-i #                     The number of times each connection should
                         run the script (default is 1).

-r #                     The maximum number to be used to create a
                         random number. If a script includes the
                         text {RANDOM}, this text will be replaced
                         with a random number no larger than this
                         number.

-v "[status vars]"       The MySQL Status Variables that should be
                         output with the script. Multiple MySQL
                         Status Variables should be entered with
                         commas. Wildcards understood by the MySQL
                         Server can be used.

-t                       Execute the script without prompting the
                         user to verify the entries.

-q                       Execute the script without a visual output
                         showing that it is running.
```

```
Example:
PT_Stress.php -u root -p oracle -c 10 \
-s "Qcache-queries.sql" -i 100 \
-r 40000 -v "Qcache%, Com_%" -t -q
Simulating 10 connection(s) against the Qcache-queries.sql
script repeating 100 times!
```

	Execution Times
Minimum:	140.814089
Maximum:	155.956133
Average:	153.282811
	Start End
Qcache_free_blocks	1 1
Qcache_free_memory	16768392 16562056
Qcache_hits	0 3215592
Qcache_inserts	55 256
Qcache_lowmem_prunes	0 0
Qcache_not_cached	55340 55547
Qcache_queries_in_cache	0 201
Qcache_total_blocks	1 404
Com_select	55765 56173

Note: There are times when it may be necessary to create your own scripts to use with the MySQL server. Having an understanding of what questions you need to ask and how the MySQL server can provide the answers is a valuable tool in your tool chest.

- In the /stage/scripts directory, use the `PT_Stress.php` script to create a load on the MySQL server to test the `table_open_cache` setting by issuing the following command in a terminal window:

```
sys> ./PT_Stress.php \
> -u root \
> -p oracle \
> -s "dept_employees_large.sql" \
> -i 20 \
> -c 5 \
> -t \
> -v "open%table%"

Execution Times
Minimum: 61.519466
Maximum: 70.538582
Average: 66.160441

Start                  End
Open_table_definitions         33                  36
    Open_tables                 1                  1
Opened_table_definitions         33                  36
    Opened_tables               35                  739842
```

- What was the average number of seconds to run all queries? 66.16 seconds
- What is the value of the `open_tables` status variable? 1
- What is the value of the `opened_tables` status variable? 739842
- Using these numbers, calculate the effectiveness of the `table_open_cache` server setting:

$$(1/739842)*100 = 0\% \text{ effective}$$

- As root shut down the MySQL server.

```
sys> /etc/init.d/mysql stop
```

6. As root, start the MySQL server with the `table_open_cache` system variable set to 8 and the `query_cache_type` system variable set to 0.

```
sys> /etc/init.d/mysql start --table-open-cache=8 \
> --query_cache_type=0
```

7. In the `/stage/scripts` directory, execute the command from step 2.

```
sys> ./PT_Stress.php \
> -u root \
> -p oracle \
> -s "dept_employees_large.sql" \
> -i 20 \
> -c 5 \
> -t \
> -v "open%table%"
```

Execution Times
Minimum: 50.019625
Maximum: 60.294774
Average: 56.937018

	Start	End
Open_table_definitions	33	36
Open_tables	8	8
Opened_table_definitions	33	36
Opened_tables	35	226473

- What was the average number of seconds to run all queries? 56.94 seconds
- What is the `end` value of the `open_tables` status variable? 8
- What is the `end` value of the `opened_tables` status variable? 226473
- Using these numbers, calculate the effectiveness of the `table_open_cache` server setting:

$$(8/226473)*100 = 0\% \text{ effective}$$

8. As root, shut down the MySQL server.

```
sys> /etc/init.d/mysql stop
```

9. As root, start the MySQL server with the `table_open_cache` system variable set to 16 and the `query_cache_type` system variable set to 0.

```
sys> /etc/init.d/mysql start --table-open-cache=16 \
> --query_cache_type=0
```

10. In the /stage/scripts directory, execute the command from step 2.

```
sys> ./PT_Stress.php \
> -u root \
> -p oracle \
> -s "dept_employees_large.sql" \
> -i 20 \
> -c 5 \
> -t \
> -v "open%table%"
```

Execution Times
Minimum: 50.808101
Maximum: 55.680517
Average: 53.019976

	Start	End
Open_table_definitions	33	36
Open_tables	16	16
Opened_table_definitions	33	36
Opened_tables	35	50

- What was the average number of seconds to run all queries? 53.02 seconds
- What is the *end* value of the `open_tables` status variable? 16
- What is the *end* value of the `opened_tables` status variable? 50
- Using these numbers, calculate the effectiveness of the `table_open_cache` server setting:

$$(16/50)*100 = 32\% \text{ effective}$$

11. As root, shut down the MySQL server.

```
sys> /etc/init.d/mysql stop
```

12. As root, start the MySQL server with the `table_open_cache` system variable set to 32 and the `query_cache_type` system variable set to 0.

```
sys> /etc/init.d/mysql start --table-open-cache=32 \
> --query_cache_type=0
```

13. In the /stage/scripts directory, execute the command from step 2.

```
sys> ./PT_Stress.php \
> -u root \
> -p oracle \
> -s "dept_employees_large.sql" \
> -i 20 \
> -c 5 \
> -t \
> -v "open%table%"

Execution Times
Minimum: 33.730711
Maximum: 55.841082
Average: 49.088413

Start           End
Open_table_definitions    33      36
          Open_tables        26      32
Opened_table_definitions  33      36
          Opened_tables       33      48
```

- What was the average number of seconds to run all queries? 49.09 seconds
- What is the value of the `open_tables` status variable? 32
- What is the value of the `opened_tables` status variable? 48
- Using these numbers, calculate the effectiveness of the `table_open_cache` server setting:

$$(32/48)*100 = 67\% \text{ effective}$$

14. As root, shut down the MySQL server.

```
sys> /etc/init.d/mysql stop
```

15. As root, start the MySQL server with the `table_open_cache` system variable set to 48 and the `query_cache_type` system variable set to 0.

```
sys> /etc/init.d/mysql start --table-open-cache=48 \
> --query_cache_type=0
```

16. In the /stage/scripts directory, execute the command from step 2.

```
sys> ./PT_Stress.php \
> -u root \
> -p oracle \
> -s "dept_employees_large.sql" \
> -i 20 \
> -c 5 \
> -t \
> -v "open%table%"

Execution Times
Minimum: 50.698227
Maximum: 55.221367
Average: 53.519993

Start           End
Open_table_definitions    33      36
          Open_tables        26      41
Opened_table_definitions  33      36
          Opened_tables       33      48
```

- What was the average number of seconds to run all queries? 53.52 seconds
- What is the value of the `open_tables` status variable? 41
- What is the value of the `opened_tables` status variable? 48
- Using these numbers, calculate the effectiveness of the `table_open_cache` server setting:

$$(41/48)*100 = 85\% \text{ effective}$$

17. As root, shut down the MySQL server.

```
sys> /etc/init.d/mysql stop
```

18. As root, start the MySQL server with the `table_open_cache` system variable set to 64 and the `query_cache_type` system variable set to 0.

```
sys> /etc/init.d/mysql start --table-open-cache=64 \
> --query_cache_type=0
```

19. In the /stage/scripts directory, execute the command from step 2.

```

sys> ./PT_Stress.php \
> -u root \
> -p oracle \
> -s "dept_employees_large.sql" \
> -i 20 \
> -c 5 \
> -t \
> -v "open%table%"

Execution Times
Minimum: 51.945315
Maximum: 55.878493
Average: 54.672053

          Start      End
Open_table_definitions    33      36
          Open_tables     26      41
Opened_table_definitions   33      36
          Opened_tables    33      48

```

- What was the average number of seconds to run all queries? 54.67 seconds
- What is the value of the `open_tables` status variable? 41
- What is the value of the `opened_tables` status variable? 48
- Using these numbers, calculate the effectiveness of the `table_open_cache` server setting:

$$(41/48) * 100 = 85\% \text{ effective}$$

Step	table_open_cache	Seconds/Avg.	open_tables	opened_tables	Effectiveness
2-4	1	66.16	1	739842	~ 0%
6-7	8	56.94	8	226473	~ 0%
9-10	16	53.02	16	50	~ 32%
12-13	32	49.09	32	48	~ 67%
15-16	48	53.52	41	48	~ 85%
18-19	64	54.67	41	48	~ 85%

Practice 4-3: Setting `open_files_limit`

Overview

In this practice, you set the `open_files_limit` system variable based on the operating system settings. To accomplish this objective, you do the following:

- Review the memory details for the hardware being used.
 - The two most important memory details to review are the total memory for the hardware and the total memory that is free.
- Review the number of file descriptors that the O/S allows to be open concurrently.
- Update the `open_files_limit` system variable equal to the `fs.file-max` setting for the server MySQL is running on.

Assumptions

- The MySQL server is running.

Duration

This practice should take five minutes to complete.

Tasks

1. Open a terminal window and execute the following command in a terminal window to display memory details for the O/S:

```
sys> cat /proc/meminfo
```

- Make a note of the following output values of the command:

– MemTotal: _____

– MemFree: _____

Note: You can use the existing terminal window from the previous lab. However, at this point, you should have only one terminal window open to minimize confusion as more terminal windows are opened up.

2. As root, execute the following command in a terminal window to display the maximum number of concurrently open file descriptors the O/S allows:

```
sys> sysctl fs.file-max
```

- What is the default value of `fs.file-max` assigned by the operating system?

3. Using the `mysql` client, view the current MySQL global `open_files_limit` system variable setting.
4. As root, increase the `open_files_limit` in the `/etc/my.cnf` file to the value of `fs.file-max` in step 2.
5. Restart the server as root.
6. Using the `mysql` client, view the MySQL global `open_files_limit` system variable setting again.

Solutions 4-3: Setting `open_files_limit`

Tasks

Note: These solutions are sample solutions and are not expected to be the exact output you see in your execution of the steps.

1. Open a terminal window and execute the following command in a terminal window to display memory details for the O/S:

```
sys> cat /proc/meminfo
```

- Make a note of the following output values of the command:
 - MemTotal: 2065880 kb
 - MemFree: 104924 kb

Note: You can use the existing terminal window from the previous lab. However, at this point, you should have only one terminal window open to minimize confusion as more terminal windows are opened up.

2. As root, execute the following command in a terminal window to display the maximum number of concurrently open file descriptors the O/S allows:

```
sys> sysctl fs.file-max
```

- What is the default value of `fs.file-max` assigned by the operating system?
360413

3. Using the `mysql` client, view the current MySQL global `open_files_limit` system variable setting.

```
sys> mysql -uroot -poracle -e"SELECT @@global.open_files_limit"
+-----+
| @@global.open_files_limit |
+-----+
|          1024           |
+-----+
```

4. As root, increase the `open_files_limit` in the `/etc/my.cnf` file to the value off `fs.file-max` in step 2.

```
sys> vi /etc/my.cnf
```

- Add the following line under the `[mysqld]` section of the file:

```
open_files_limit = 360413
```

- Close and save the file.

5. Restart the server as root.

```
sys> /etc/init.d/mysql restart
```

6. Using the `mysql` client, view the MySQL global `open_files_limit` system variable setting again.

```
sys> mysql -uroot -poracle -e"SELECT @@global.open_files_limit"
+-----+
| @@global.open_files_limit |
+-----+
|          360413           |
+-----+
```

Practice 4-4: Setting max_connections

Overview

In this practice, you review the response from the MySQL server when an attempt is made to open more connections than the setting of the `max_connections` system variable. To accomplish this objective, you do the following:

- View the current `max_connections` system variable setting.
- Execute a test against the MySQL server which produces connections greater than the `max_connections` system variable setting.
- Increase the `max_connections` system variable value and attempt again the test issued.

Assumptions

- The MySQL server is running.
- The `sysbench` application is installed.

Duration

This practice should take 30 minutes to complete.

Tasks

1. Open a terminal window and use the `mysql` client to view the maximum number of connections that the MySQL server allows.

- What is the value assigned to the `max_connections` system variable? _____

Note: You can use the existing terminal window from the previous lab. However, at this point, you should have only one terminal window open to minimize confusion as more terminal windows are opened up.

2. Using a terminal window, execute the following command to produce 200 connections against the MySQL server:

```
sys> sysbench \
> --test=oltp \
> --oltp-table-size=1000000 \
> --mysql-socket=/var/lib/mysql/mysql.sock \
> --oltp-test-mode=simple \
> --oltp-reconnect-mode=query \
> --mysql-user=root \
> --mysql-password=oracle \
> --mysql-db=test \
> --max-requests=1000 \
> --num-threads=200 \
> run
```

- Does the script produce any errors? If the script produced errors, what did the error(s) identify?

3. Using the `mysql` client, review the maximum number of connections MySQL has had open at the same time since the server was last restarted.
4. Using the `mysql` client, increase the number of connections that the MySQL server can handle by increasing the value of `max_connections` to 225.
5. Execute the command from step 3.
 - Does the script produce any errors? If the script produced errors, what did the error(s) identify?

6. Using the `mysql` client, review the maximum number of connections MySQL has had open at the same time since the server was last restarted.
7. Restart the MySQL server as `root`.
8. Execute the command from step 3.
 - Does the script produce any errors? If the script produced errors, what did the error(s) identify?

9. Using the `mysql` client, review the maximum number of connections MySQL has had open at the same time since the server was last restarted.
10. As `root`, edit the `/etc/my.cnf` file and add the `max_connections` system variable with a value of 225.
11. Restart the MySQL server as `root`.
12. Execute the command from step 3.
 - Does the script produce any errors? If the script produced errors, what did the error(s) identify?

13. Using the `mysql` client, review the maximum number of connections MySQL has had open at the same time since the server was last restarted.

Solutions 4-4: Setting `max_connections`

Tasks

Note: These solutions are sample solutions and are not expected to be the exact output you see in your execution of the steps.

1. Open a terminal window and use the `mysql` client to view the maximum number of connections that the MySQL server allows.

```
sys> mysql -uroot -poracle \
> -e"SHOW VARIABLES LIKE 'max_connect%'";
+-----+-----+
| Variable_name      | Value   |
+-----+-----+
| max_connect_errors | 10      |
| max_connections    | 151     |
+-----+-----+
```

- What is the value assigned to the `max_connections` system variable? [151](#)

Note: You can use the existing terminal window from the previous lab. However, at this point, you should have only one terminal window open to minimize confusion as more terminal windows are opened up.

2. Using a terminal window, execute the following command to produce 200 connections against the MySQL server:

```
sys> sysbench \
> --test=oltp \
> --oltp-table-size=1000000 \
> --mysql-socket=/var/lib/mysql/mysql.sock \
> --oltp-test-mode=simple \
> --oltp-reconnect-mode=query \
> --mysql-user=root \
> --mysql-password=oracle \
> --mysql-db=test \
> --max-requests=1000 \
> --num-threads=200 \
> run

No DB drivers specified, using mysql
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? If the script produced errors, what did the error(s) identify? The script produced an error preventing additional connections to the MySQL server.

3. Using the `mysql` client, review the maximum number of connections MySQL has had open at the same time since the server was last restarted.

```
sys> mysql -uroot -poracle \  
> -e"SHOW STATUS LIKE 'Max_used_connections';  
+-----+-----+  
| Variable_name      | Value |  
+-----+-----+  
| Max_used_connections | 152   |  
+-----+-----+
```

4. Using the `mysql` client, increase the number of connections that the MySQL server can handle by increasing the value of `max_connections` to 225.

```
sys> mysql -uroot -poracle \  
> -e"SET GLOBAL max_connections = 225";
```

5. Execute the command from step 3.

```
sys> sysbench \
> --test=oltp \
> --oltp-table-size=1000000 \
> --mysql-socket=/var/lib/mysql/mysql.sock \
> --oltp-test-mode=simple \
> --oltp-reconnect-mode=query \
> --mysql-user=root \
> --mysql-password=oracle \
> --mysql-db=test \
> --max-requests=1000 \
> --num-threads=200 \
> run

No DB drivers specified, using mysql
Running the test with following options:
Number of threads: 200
Random number generator seed is 0 and will be ignored
Doing OLTP test.
Running simple OLTP test
Using Special distribution (12 iterations, 1 pct of values are returned in 75
pct cases)
Using "LOCK TABLES WRITE" 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      (820.70 per sec.)
    deadlocks:                  0          (0.00 per sec.)
    read/write requests:        1000      (820.70 per sec.)
    other operations:           1000      (820.70 per sec.)

Test execution summary:
    total time:                 1.2185s
    total number of events:     1000
    total time taken by event execution: 161.9980
    per-request statistics:
        min:                      13.48ms
        avg:                      162.00ms
        max:                      1063.78ms
        approx. 95 percentile:    792.23ms

Threads fairness:
    events (avg/stddev) :      5.0000/10.01
    execution time (avg/stddev): 0.8100/0.23
```

- Does the script produce any errors? If the script produced errors, what did the error(s) identify? The script ran without errors.

6. Using the `mysql` client, review the maximum number of connections MySQL has had open at the same time since the server was last restarted.

```
sys> mysql -uroot -poracle \
> -e"SHOW STATUS LIKE 'Max_used_connections'";
+-----+-----+
| Variable_name      | Value |
+-----+-----+
| Max_used_connections | 204 |
+-----+-----+
```

7. Restart the MySQL server as root.

```
sys> /etc/init.d/mysql restart
```

8. Execute the command from step 3.

```
sys> sysbench \
> --test=oltp \
> --oltp-table-size=1000000 \
> --mysql-socket=/var/lib/mysql/mysql.sock \
> --oltp-test-mode=simple \
> --oltp-reconnect-mode=query \
> --mysql-user=root \
> --mysql-password=oracle \
> --mysql-db=test \
> --max-requests=1000 \
> --num-threads=200 \
> run

No DB drivers specified, using mysql
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? If the script produced errors, what did the error(s) identify? The script produced an error preventing additional connections to the MySQL server. After the MySQL server is restarted, the max_connections setting reverts to its default value.

9. Using the `mysql` client, review the maximum number of connections MySQL has had open at the same time since the server was last restarted.

```
sys> mysql -uroot -poracle \
> -e"SHOW STATUS LIKE 'Max_used_connections'";
+-----+-----+
| Variable_name      | Value |
+-----+-----+
| Max_used_connections | 152 |
+-----+-----+
```

10. As root, edit the `/etc/my.cnf` file and add the `max_connections` system variable with a value of 225.

```
sys> vi /etc/my.cnf
```

- Add the following lines under the `[mysqld]` section:

```
max_connections = 225
```

11. Restart the MySQL server as root.

```
sys> /etc/init.d/mysql restart
```

12. Execute the command from step 3.

```
sys> sysbench \
> --test=oltp \
> --oltp-table-size=1000000 \
> --mysql-socket=/var/lib/mysql/mysql.sock \
> --oltp-test-mode=simple \
> --oltp-reconnect-mode=query \
> --mysql-user=root \
> --mysql-password=oracle \
> --mysql-db=test \
> --max-requests=1000 \
> --num-threads=200 \
> run

No DB drivers specified, using mysql
Running the test with following options:
Number of threads: 200
Random number generator seed is 0 and will be ignored
Doing OLTP test.
Running simple OLTP test
Using Special distribution (12 iterations, 1 pct of values are returned in 75
pct cases)
Using "LOCK TABLES WRITE" 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  (837.01 per sec.)
    deadlocks:                  0   (0.00 per sec.)
    read/write requests:        1000  (837.01 per sec.)
    other operations:           1000  (837.01 per sec.)

Test execution summary:
    total time:                 1.1947s
    total number of events:     1000
    total time taken by event execution: 200.0404
    per-request statistics:
        min:                      28.86ms
        avg:                      200.04ms
        max:                      367.67ms
        approx. 95 percentile:    272.60ms

Threads fairness:
    events (avg/stddev):      5.0000/0.92
    execution time (avg/stddev): 1.0002/0.09
```

- Does the script produce any errors? If the script produced errors, what did the error(s) identify? The script ran without errors.

13. Using the mysql client, review the maximum number of connections MySQL has had open at the same time since the server was last restarted.

```
sys> mysql -uroot -poracle \
> -e"SHOW STATUS LIKE 'Max_used_connections'";
+-----+-----+
| Variable_name      | Value   |
+-----+-----+
| Max_used_connections | 206    |
+-----+-----+
```

Practice 4-5: Evaluating the Effects of Numerous Connections

Overview

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

- Set the `max_connections` system variable to be able to handle 1,024 connections.
- Execute 100,000 requests against the MySQL server using a progressive number of connections.
 - The requests are split up between the connections.
- Evaluate the amount of virtual (swapped and physical) and physical (non-swapped) memory used by each execution.

Assumptions

- The MySQL server is running.
- The `sysbench` application is installed and has been prepared with a MyISAM table containing 1,000,000 records.

Duration

This practice should take 45 minutes to complete.

Tasks

Note: A table has been provided at the end of the steps for this practice for you to fill in the values collected during the running of the steps. Entering the data from the steps into this table can assist you in comparing the values collected.

1. Open a terminal window and restart the MySQL server as `root`.

Note: You can use the existing terminal window from the previous lab. However, at this point, you should have only one terminal window open to minimize confusion as more terminal windows are opened up.

2. Using the `mysql` client, increase the number of connections that the MySQL server can handle by increasing the value of `max_connections` to 1024.

```
sys> mysql -uroot -poracle \
> -e "SET GLOBAL max_connections = 1024";
```

3. Using a terminal window, execute the following command to produce 1,000,000 queries using 8 connections against the MySQL server:

```
sys> sysbench \
> --test=oltp \
> --oltp-table-size=1000000 \
> --mysql-socket=/var/lib/mysql/mysql.sock \
> --oltp-test-mode=simple \
> --oltp-reconnect-mode=query \
> --mysql-user=root \
> --mysql-password=oracle \
> --mysql-db=test \
> --max-requests=100000 \
> --num-threads=8 \
> run | grep "total time"
```

- Each connection handles 125,000 queries.

Note: The complete output of the sysbench application is not necessary for this lab. By placing the grep "total time" option at the end of the script, only those values that start with "total time" are displayed at the end of the run.

- In a separate terminal window, execute the following command to obtain a dynamic real time view of the mysqld application:

```
sys> top -n 8 | grep mysqld
```

- The output you see is similar to the output below:

30530	mysql	15	0	761m	116m	4324	S	65.1	5.8	2:03.00	mysqld
30530	mysql	15	0	761m	116m	4324	S	68.1	5.8	2:05.05	mysqld
30530	mysql	15	0	761m	116m	4324	S	68.7	5.8	2:07.12	mysqld
30530	mysql	15	0	761m	116m	4324	S	68.1	5.8	2:09.17	mysqld
30530	mysql	15	0	761m	116m	4324	S	73.7	5.8	2:11.39	mysqld
30530	mysql	15	0	761m	116m	4324	S	70.7	5.8	2:13.52	mysqld
30530	mysql	15	0	761m	116m	4324	S	68.1	5.8	2:15.57	mysqld
30530	mysql	15	0	761m	116m	4324	S	67.4	5.8	2:17.60	mysqld

- The columns of most interest to you for this lab include:
 - Column 5: VIRT – This output value represents the virtual size of a process which is the sum of memory the process is actually using.
 - Column 6: RES – This output value represents the actual physical memory a process is consuming.
- What is the approximate value of the Virtual memory (VIRT column) of the top output on your system?

- What is the approximate value of the Resident memory (RES Column) of the top output on your system?

- Return to the first terminal (step 3) window and terminate the sysbench application execution if it is still running.

Note: The amount of time that the sysbench application runs is not important to demonstrate the memory consumption for this lab. Terminating the application prior to its completion prevents the output values from being displayed.

- Execute the command from step 3 using 32 connections.
 - Each connection handles 31,750 queries.
- Execute the command from step 4.
 - What is the approximate value of the Virtual memory (VIRT column) of the top output on your system?

 - What is the approximate value of the Resident memory (RES Column) of the top output on your system?
- Return to the first terminal window (step 3) and terminate the sysbench application execution if it is still running.

9. Execute the command from step 3 using 64 connections.
 - Each connection handles 15,625 queries.
10. Execute the command from step 4.
 - What is the approximate value of the Virtual memory (VIRT column) of the `top` output on your system?

 - What is the approximate value of the Resident memory (RES Column) of the `top` output on your system?

11. Return to the first terminal window (step 3) and terminate the `sysbench` application execution if it is still running.
12. Execute the command from step 3 using 128 connections.
 - Each connection handles 7,813 queries.
13. Execute the command from step 4.
 - What is the approximate value of the Virtual memory (VIRT column) of the `top` output on your system?

 - What is the approximate value of the Resident memory (RES Column) of the `top` output on your system?

14. Return to the first terminal window (step 3) and terminate the `sysbench` application execution if it is still running.
15. Execute the command from step 3 using 256 connections.
 - Each connection handles 3,907 queries.
16. Execute the command from step 4.
 - What is the approximate value of the Virtual memory (VIRT column) of the `top` output on your system?

 - What is the approximate value of the Resident memory (RES Column) of the `top` output on your system?

17. Return to the first terminal window (step 3) and terminate the `sysbench` application execution if it is still running.
18. Execute the command from step 3 using 512 connections.
 - Each connection handles 1,953 queries.

19. Execute the command from step 4.
- What is the approximate value of the Virtual memory (VIRT column) of the `top` output on your system?

 - What is the approximate value of the Resident memory (RES Column) of the `top` output on your system?

20. Return to the first terminal window (step 3) and terminate the `sysbench` application execution if it is still running.
21. Execute the command from step 3 using 1,000 connections.
- Each connection handles 1,000 queries.
Note: Using 1,000 connections (versus 1,024) prevents the Linux kernel from terminating the application for too many threads.
22. Execute the command from step 4.
- What is the approximate value of the Virtual memory (VIRT column) of the `top` output on your system?

 - What is the approximate value of the Resident memory (RES Column) of the `top` output on your system?

23. Return to the first terminal window (step 3) and terminate the `sysbench` application execution if it is still running.
24. What do the results tell you about the relationship between the number of threads running against the MySQL server and the memory consumed by `mysqld`.
-
-
-

Steps	Connection #	VIRT	RES
3-4	8		
6-7	32		
9-10	64		
12-13	128		
15-16	256		
18-19	512		
21-22	1000		

Solutions 4-5: Evaluating the Effects of Numerous Connections

Tasks

Note: These solutions are sample solutions and are not expected to be the exact output you see in your execution of the steps.

1. Open a terminal window and restart the MySQL server as root.

```
sys> /etc/init.d/mysql restart
```

Note: You can use the existing terminal window from the previous lab. However, at this point, you should have only one terminal window open to minimize confusion as more terminal windows are opened up.

2. Using the mysql client, increase the number of connections that the MySQL server can handle by increasing the value of max_connections to 1024.

```
sys> mysql -uroot -poracle \
> -e"SET GLOBAL max_connections = 1024";
```

3. Using a terminal window, execute the following command to produce 1,000,000 queries using 8 connections against the MySQL server:

```
sys> sysbench \
> --test=oltp \
> --oltp-table-size=1000000 \
> --mysql-socket=/var/lib/mysql/mysql.sock \
> --oltp-test-mode=simple \
> --oltp-reconnect-mode=query \
> --mysql-user=root \
> --mysql-password=oracle \
> --mysql-db=test \
> --max-requests=100000 \
> --num-threads=8 \
> run | grep "total time"
```

4. In a separate terminal window, execute the following command to obtain a dynamic real time view of the mysqld application:

```
sys> top -n 8 | grep mysqld
30530 mysql      15   0  761m 116m 4324 S 65.1  5.8  2:03.00 mysqld
30530 mysql      15   0  761m 116m 4324 S 68.1  5.8  2:05.05 mysqld
30530 mysql      15   0  761m 116m 4324 S 68.7  5.8  2:07.12 mysqld
30530 mysql      15   0  761m 116m 4324 S 68.1  5.8  2:09.17 mysqld
30530 mysql      15   0  761m 116m 4324 S 73.7  5.8  2:11.39 mysqld
30530 mysql      15   0  761m 116m 4324 S 70.7  5.8  2:13.52 mysqld
30530 mysql      15   0  761m 116m 4324 S 68.1  5.8  2:15.57 mysqld
30530 mysql      15   0  761m 116m 4324 S 67.4  5.8  2:17.60 mysqld
```

- What is the approximate value of the Virtual memory (VIRT column) of the top output on your system? 761m
- What is the approximate value of the Resident memory (RES Column) of the top output on your system? 116m

5. Return to the first terminal window (step 3) and terminate the `sysbench` application execution if it is still running.
 - Pressing the [CTRL] and [C] keys together terminates the execution of the `sysbench` application.
 - In the event that the `sysbench` application has completed prior to your terminating the execution, the following will be displayed:

```
total time: 168.7924s
total time taken by event execution: 1349.9336
```

Note: The total time taken by event execution: output is equivalent to the individual threads time accumulated together.

6. Execute the command from step 3 using 32 connections.

```
sys> sysbench \
> --test=oltp \
> --oltp-table-size=1000000 \
> --mysql-socket=/var/lib/mysql/mysql.sock \
> --oltp-test-mode=simple \
> --oltp-reconnect-mode=query \
> --mysql-user=root \
> --mysql-password=oracle \
> --mysql-db=test \
> --max-requests=100000 \
> --num-threads=32 \
> run | grep "total time"
```

7. Execute the command from step 4.

```
sys> top -n 8 | grep mysqld
30530 mysql      25  0  763m 119m 4324 R 133.3  5.9  4:09.43 mysqld
30530 mysql      25  0  762m 119m 4324 R 131.4  5.9  4:13.39 mysqld
30530 mysql      25  0  762m 119m 4324 S 133.1  5.9  4:17.40 mysqld
30530 mysql      25  0  762m 119m 4324 R 132.1  5.9  4:21.38 mysqld
30530 mysql      25  0  763m 119m 4324 R 132.7  5.9  4:25.38 mysqld
30530 mysql      25  0  762m 119m 4324 R 132.4  5.9  4:29.37 mysqld
30530 mysql      25  0  762m 119m 4324 R 132.4  5.9  4:33.36 mysqld
30530 mysql      25  0  762m 119m 4324 R 130.1  5.9  4:37.28 mysqld
```

- What is the approximate value of the Virtual memory (VIRT column) of the `top` output on your system? 762m
- What is the approximate value of the Resident memory (RES Column) of the `top` output on your system? 119m

8. Return to the first terminal window (step 3) and terminate the `sysbench` application execution if it is still running.

- Pressing the [CTRL] and [C] keys together terminates the execution of the `sysbench` application.
- In the event that the `sysbench` application has completed prior to your terminating the execution, the following will be displayed:

```
total time: 112.5607s
total time taken by event execution: 3600.9343
```

9. Execute the command from step 3 using 64 connections.

```
sys> sysbench \
> --test=oltp \
> --oltp-table-size=1000000 \
> --mysql-socket=/var/lib/mysql/mysql.sock \
> --oltp-test-mode=simple \
> --oltp-reconnect-mode=query \
> --mysql-user=root \
> --mysql-password=oracle \
> --mysql-db=test \
> --max-requests=100000 \
> --num-threads=64 \
> run | grep "total time"
```

10. Execute the command from step 4.

```
sys> top -n 8 | grep mysqld
30530 mysql      25  0  763m 121m 4324 R 138.5  6.0    7:51.06 mysqld
30530 mysql      25  0  763m 121m 4324 R 140.7  6.0    7:55.30 mysqld
30530 mysql      25  0  764m 122m 4324 R 138.4  6.0    7:59.47 mysqld
30530 mysql      25  0  763m 121m 4324 R 140.7  6.0    8:03.71 mysqld
30530 mysql      25  0  764m 121m 4324 S 140.4  6.0    8:07.94 mysqld
30530 mysql      25  0  765m 122m 4324 S 136.7  6.1    8:12.06 mysqld
30530 mysql      25  0  763m 121m 4324 R 139.4  6.0    8:16.26 mysqld
30530 mysql      25  0  763m 121m 4324 R 136.0  6.0    8:20.36 mysqld
```

- What is the approximate value of the Virtual memory (VIRT column) of the `top` output on your system? 764m
- What is the approximate value of the Resident memory (RES Column) of the `top` output on your system? 121m

11. Return to the first terminal window (step 3) and terminate the `sysbench` application execution if it is still running.

- Pressing the [CTRL] and [C] keys together terminates the execution of the `sysbench` application.
- In the event that the `sysbench` application has completed prior to your terminating the execution, the following will be displayed:

```
total time:                      107.5977s
total time taken by event execution: 6870.0500
```

12. Execute the command from step 3 using 128 connections.

```
sys> sysbench \
> --test=oltp \
> --oltp-table-size=1000000 \
> --mysql-socket=/var/lib/mysql/mysql.sock \
> --oltp-test-mode=simple \
> --oltp-reconnect-mode=query \
> --mysql-user=root \
> --mysql-password=oracle \
> --mysql-db=test \
> --max-requests=100000 \
> --num-threads=128 \
> run | grep "total time"
```

13. Execute the command from step 4.

```
sys> top -n 8 | grep mysqld
30530 mysql      25  0  766m 123m 4324 R 138.3  6.1   9:23.65 mysqld
30530 mysql      25  0  766m 123m 4324 S 142.0  6.1   9:27.93 mysqld
30530 mysql      25  0  764m 122m 4324 R 140.6  6.1   9:32.16 mysqld
30530 mysql      25  0  764m 122m 4324 R 139.0  6.1   9:36.35 mysqld
30530 mysql      25  0  765m 122m 4324 R 142.1  6.1   9:40.63 mysqld
30530 mysql      25  0  766m 123m 4324 R 141.4  6.1   9:44.89 mysqld
30530 mysql      25  0  765m 122m 4324 R 141.0  6.1   9:49.14 mysqld
30530 mysql      25  0  765m 122m 4324 S 139.7  6.1   9:53.35 mysqld
```

- What is the approximate value of the Virtual memory (VIRT column) of the `top` output on your system? 765m
- What is the approximate value of the Resident memory (RES Column) of the `top` output on your system? 123m

14. Return to the first terminal window (step 3) and terminate the `sysbench` application execution if it is still running.

- Pressing the [CTRL] and [C] keys together terminates the execution of the `sysbench` application.
- In the event that the `sysbench` application has completed prior to your terminating the execution, the following will be displayed:

```
total time:                      107.6731s
total time taken by event execution: 13709.8795
```

15. Execute the command from step 3 using 256 connections.

```
sys> sysbench \
> --test=oltp \
> --oltp-table-size=1000000 \
> --mysql-socket=/var/lib/mysql/mysql.sock \
> --oltp-test-mode=simple \
> --oltp-reconnect-mode=query \
> --mysql-user=root \
> --mysql-password=oracle \
> --mysql-db=test \
> --max-requests=100000 \
> --num-threads=256 \
> run | grep "total time"
```

16. Execute the command from step 4.

```
sys> top -n 8 | grep mysqld
30530 mysql      25  0  774m 128m 4324 S 136.9  6.4 12:06.88 mysqld
30530 mysql      25  0  773m 127m 4324 R 142.0  6.3 12:11.16 mysqld
30530 mysql      25  0  773m 128m 4324 R 140.4  6.4 12:15.39 mysqld
30530 mysql      25  0  772m 126m 4324 R 139.7  6.3 12:19.60 mysqld
30530 mysql      25  0  772m 126m 4324 R 139.7  6.3 12:23.81 mysqld
30530 mysql      25  0  772m 127m 4324 R 141.6  6.3 12:28.08 mysqld
30530 mysql      25  0  771m 126m 4324 R 140.4  6.3 12:32.31 mysqld
30530 mysql      25  0  772m 126m 4324 R 142.7  6.3 12:36.61 mysqld
```

- What is the approximate value of the Virtual memory (VIRT column) of the `top` output on your system? 772m
- What is the approximate value of the Resident memory (RES Column) of the `top` output on your system? 127m

17. Return to the first terminal window (step 3) and terminate the `sysbench` application execution if it is still running.

- Pressing the [CTRL] and [C] keys together terminates the execution of the `sysbench` application.
- In the event that the `sysbench` application has completed prior to your terminating the execution, the following will be displayed:

```
total time:                      108.9555s
total time taken by event execution: 27551.8230
```

18. Execute the command from step 3 using 512 connections.

```
sys> sysbench \
> --test=oltp \
> --oltp-table-size=1000000 \
> --mysql-socket=/var/lib/mysql/mysql.sock \
> --oltp-test-mode=simple \
> --oltp-reconnect-mode=query \
> --mysql-user=root \
> --mysql-password=oracle \
> --mysql-db=test \
> --max-requests=100000 \
> --num-threads=512 \
> run | grep "total time"
```

19. Execute the command from step 4.

```
sys> top -n 8 | grep mysqld
30530 mysql      25  0  780m 133m 4324 R 137.8  6.6 14:24.14 mysqld
30530 mysql      25  0  779m 133m 4324 R 140.3  6.6 14:28.37 mysqld
30530 mysql      25  0  779m 133m 4324 R 141.7  6.6 14:32.64 mysqld
30530 mysql      25  0  780m 133m 4324 R 142.4  6.6 14:36.93 mysqld
30530 mysql      25  0  778m 132m 4324 R 139.7  6.6 14:41.14 mysqld
30530 mysql      25  0  778m 132m 4324 R 141.4  6.6 14:45.40 mysqld
30530 mysql      25  0  779m 133m 4324 R 139.7  6.6 14:49.61 mysqld
30530 mysql      25  0  779m 133m 4324 R 140.0  6.6 14:53.83 mysqld
```

- What is the approximate value of the Virtual memory (VIRT column) of the `top` output on your system? 779m
- What is the approximate value of the Resident memory (RES Column) of the `top` output on your system? 133m

20. Return to the first terminal window (step 3) and terminate the `sysbench` application execution if it is still running.

- Pressing the [CTRL] and [C] keys together terminates the execution of the `sysbench` application.
- In the event that the `sysbench` application has completed prior to your terminating the execution, the following will be displayed:

```
total time:                      110.2585s
total time taken by event execution: 55893.1398
```

21. Execute the command from step 3 using 1,000 connections.

```
sys> sysbench \
> --test=oltp \
> --oltp-table-size=1000000 \
> --mysql-socket=/var/lib/mysql/mysql.sock \
> --oltp-test-mode=simple \
> --oltp-reconnect-mode=query \
> --mysql-user=root \
> --mysql-password=oracle \
> --mysql-db=test \
> --max-requests=100000 \
> --num-threads=1000 \
> run | grep "total time"
```

22. Execute the command from step 4.

```
sys> top -n 8 | grep mysqld
30530 mysql      25  0  810m 148m 4324 S 135.9  7.4 17:23.14 mysqld
30530 mysql      25  0  807m 147m 4324 R 141.4  7.3 17:27.40 mysqld
30530 mysql      25  0  807m 147m 4324 R 142.6  7.3 17:31.70 mysqld
30530 mysql      25  0  808m 148m 4324 R 139.7  7.4 17:35.91 mysqld
30530 mysql      25  0  807m 147m 4324 R 141.3  7.3 17:40.17 mysqld
30530 mysql      25  0  807m 148m 4324 R 141.4  7.3 17:44.43 mysqld
30530 mysql      25  0  806m 147m 4324 R 138.7  7.3 17:48.61 mysqld
30530 mysql      25  0  806m 147m 4324 R 142.1  7.3 17:52.89 mysqld
```

- What is the approximate value of the Virtual memory (VIRT column) of the `top` output on your system? 808m
- What is the approximate value of the Resident memory (RES Column) of the `top` output on your system? 148m

23. Return to the first terminal window (step 3) and terminate the `sysbench` application execution if it is still running.

- Pressing the [CTRL] and [C] keys together terminates the execution of the `sysbench` application.
- In the event that the `sysbench` application has completed prior to your terminating the execution, the following will be displayed:

```
total time:                      110.1285s
total time taken by event execution: 108691.6597
```

24. What do the results tell you about the relationship between the number of threads running against the MySQL server and the memory consumed by mysqld.

- Even though the speed of the processing can be more efficient using multiple threads, the amount of memory that is consumed can have an adverse affect on the performance of the system.

Steps	Connection #	VIRT	RES
3-4	8	761MB	116MB
6-7	32	762MB	119MB
9-10	64	764MB	121MB
12-13	128	765MB	123MB
15-16	256	772MB	127MB
18-19	512	779MB	133MB
21-22	1000	808MB	148MB

Practice 4-6: Total Thread Memory Usage

Overview

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

Assumptions

- The MySQL server is running.

Duration

This practice should take 15 minutes to complete.

Tasks

- Open a terminal window and then open the `/etc/my.cnf` file and make a note of the following settings:

- `read_buffer_size`: _____
- `read_rnd_buffer_size`: _____
- `sort_buffer_size`: _____
- `max_connections`: _____

Note: You can use the existing terminal window from the previous lab. However, at this point, you should have only one terminal window open to minimize confusion as more terminal windows are opened up.

Note: If the values requested are not located in the `my.cnf` file, use the process in the next step to obtain the values.

- Using the `mysql` client, record the values for the following variables:

- `join_buffer_size`: _____
- `thread_stack`: _____
- `max_heap_table_size`: _____
- `tmp_table_size`: _____

- Using the values recorded in step 1 and 2, calculate the total thread memory usage by using the following formula:

```
- max_connections x (
    sort_buffer_size +
    read_rnd_buffer_size +
    join_buffer_size +
    read_buffer_size +
    thread_stack +
    (smaller of tmp_table_size or max_heap_table_size)
)
```

- Divide the total thread memory usage calculated in step 3 by 4 to determine a more realistic thread memory usage:

- _____ /4 = _____

Solutions 4-6: Total Thread Memory Usage

Tasks

Note: These solutions are sample solutions and are not expected to be the exact output you see in your execution of the steps.

1. Open a terminal window and then open the /etc/my.cnf file and make a note of the following settings:

- read_buffer_size: 2M
- read_rnd_buffer_size: 8M
- sort_buffer_size: 2M
- max_connections: 225

Note: You can use the existing terminal window from the previous lab. However, at this point, you should have only one terminal window open to minimize confusion as more terminal windows are opened up.

Note: If the values requested are not located in the my.cnf file, use the process in the next step to obtain the values.

2. Using the mysql client, record the values for the following variables:

```
sys> mysql -uroot -poracle \
> -e "SHOW VARIABLES LIKE 'join_buffer_size'";
```

- join_buffer_size:

131072

```
sys> mysql -uroot -poracle \
> -e "SHOW VARIABLES LIKE 'thread_stack'";
```

- thread_stack:

196608

```
sys> mysql -uroot -poracle \
> -e "SHOW VARIABLES LIKE 'max_heap_table_size'";
```

- max_heap_table_size:

16777216

```
sys> mysql -uroot -poracle \
> -e "SHOW VARIABLES LIKE 'tmp_table_size'";
```

- tmp_table_size:

16777216

3. Using the values recorded in step 1 and 2, calculate the total thread memory usage by using the following formula:

- `max_connections x (sort_buffer_size + read_rnd_buffer_size + join_buffer_size + read_buffer_size + thread_stack + (smaller of tmp_table_size or max_heap_table_size))`

$$- 225 \times (2M + 8M + 131K + 2M + 196K + 16M) = 6,373,575,000$$

4. Divide the total thread memory usage calculated in step 3 by 4 to determine a more realistic thread memory usage:

$$- 6,373,575,000 / 4 = 1,593,393,750$$

Practice 4-7: Sort Queries

Overview

In this practice, you determine the most effective setting for the `sort_buffer_size` system variable based on a sample of data collected. To accomplish this objective, you do the following:

- Set the `sort_buffer_size` system variable using settings ranging from 32k to 32m.
- Execute a sample collection of SQL data using 20 connections against the MySQL server.
- Evaluate the average time that the connections to the server took to complete, the number of sort merge passes the server had to perform along with the amount of virtual (swapped and physical) and physical (non-swapped) memory used by each execution.

Assumptions

- The MySQL server is running.
- The `PT_Stress.php` and `sort.read` files are accessible and located in the `/stage/scripts` directory.

Duration

This practice should take 45 minutes to complete.

Tasks

Note: A table has been provided at the end of the steps for this practice for you to fill in the values collected during the running of the steps. Entering the data from the steps into this table can assist you in comparing the values collected.

1. Open a terminal window and restart the MySQL server as `root` to reset the status variables.
Note: You can use the existing terminal window from the previous lab. However, at this point, you should have only one terminal window open to minimize confusion as more terminal windows are opened up.
2. Using the `mysql` client, set the global `sort_buffer_size` system variable to 32,768 and the global `query_cache_type` to 0.
Note: The minimum value that the `sort_buffer_size` system variable can be set to is 32,768. The system automatically applies the minimum if the value attempted to be set is lower than the minimum.
3. In the `/stage/scripts` directory, execute the following command to force MySQL to perform a sort operation using 20 connections with each connection running the `sort.read` file 1 time:

```
sys> ./PT_Stress.php \
> -u root \
> -p oracle \
> -c 20 \
> -s "sort.read" \
> -i 1 \
> -v "Sort%" \
> -t
```

4. In a separate terminal window, execute the following command to obtain a dynamic real time view of the mysqld application:

```
sys> top | grep mysqld
```

- The output you see is similar to the output below:

5451	mysql	15	0	741m	140m	3960	S	194.8	6.9	1:03.70	mysqld
5451	mysql	15	0	737m	136m	3972	S	198.7	6.7	1:09.69	mysqld
5451	mysql	15	0	738m	134m	3972	S	195.5	6.7	1:15.58	mysqld
5451	mysql	15	0	738m	136m	3972	S	198.8	6.8	1:21.57	mysqld
5451	mysql	15	0	735m	136m	3972	S	197.4	6.8	1:27.52	mysqld
5451	mysql	15	0	738m	136m	3972	S	198.1	6.8	1:33.49	mysqld
5451	mysql	15	0	736m	136m	3972	S	197.0	6.8	1:39.43	mysqld
5451	mysql	15	0	734m	134m	3972	S	192.5	6.7	1:45.23	mysqld
...											

- The columns of most interest to you for this lab include:
 - Column 5: VIRT – This output value represents the virtual size of a process which is the sum of memory the process is actually using.
 - Column 6: RES – This output value represents the actual physical memory a process is consuming.
- What is the maximum value of the Virtual memory (VIRT column) of the `top` output on your system?

- What is the max value of the Resident memory (RES Column) of the `top` output on your system?

5. In the first terminal window (step 3), view the output from the `PT_Stress.php` script execution.

- What is the average time that the executions took to complete? _____
- What are the output values for the `Sort_merge_passes` status variable?

Start value _____ End value _____

6. In the first terminal window (step 3) and using the `mysql` client, set the global `sort_buffer_size` system variable to 524,288 (512KB).

7. In the first terminal window (step 3), execute the command from step 3.

8. In the second terminal window (opened from step 4), view the output of the `top` command.

- What is the maximum value of the Virtual memory (VIRT column) of the `top` output on your system?

-
- What is the max value of the Resident memory (RES Column) of the `top` output on your system?
-

9. In the first terminal window, view the output from the `PT_Stress.php` script execution (step 7).

- What is the average time that the executions took to complete? _____
- What are the output values for the `Sort_merge_passes` status variable?

Start value _____ End value _____

10. In the first terminal window (step 3) and using the `mysql` client, set the global `sort_buffer_size` system variable to 1,048,576 (1MB):

11. In the first terminal window (step 3), execute the command from step 3.

12. In the second terminal window (step 4), view the output of the `top` command.

- What is the maximum value of the Virtual memory (VIRT column) of the `top` output on your system?

- What is the max value of the Resident memory (RES Column) of the `top` output on your system?

13. In the first terminal window, view the output from the `PT_Stress.php` script execution (step 11).

- What is the average time that the executions took to complete? _____
- What are the output values for the `Sort_merge_passes` status variable?

Start value _____ End value _____

14. In the first terminal window (step 3) and using the `mysql` client, set the global `sort_buffer_size` system variable to 4,194,304 (4MB).

15. In the first terminal window (step 3), execute the command from step 3.

16. In the second terminal window (step 4), view the output of the `top` command.

- What is the maximum value of the Virtual memory (VIRT column) of the `top` output on your system?

- What is the max value of the Resident memory (RES Column) of the `top` output on your system?

17. In the first terminal window, view the output from the `PT_Stress.php` script execution (step 15).

- What is the average time that the executions took to complete? _____
- What are the output values for the `Sort_merge_passes` status variable?

Start value _____ End value _____

18. In the first terminal window (step 15) and using the `mysql` client, set the global `sort_buffer_size` system variable to 8,388,608 (8MB).

19. In the first terminal window (step 3), execute the command from step 3.
20. In the second terminal window (step 4), view the output of the `top` command.
 - What is the maximum value of the Virtual memory (VIRT column) of the `top` output on your system?

 - What is the max value of the Resident memory (RES Column) of the `top` output on your system?

21. In the first terminal window, view the output from the `PT_Stress.php` script execution (step 19).
 - What is the average time that the executions took to complete? _____
 - What are the output values for the `Sort_merge_passes` status variable?
Start value _____ End value _____
22. In the first terminal window (step 3) and using the `mysql` client, set the global `sort_buffer_size` system variable to 16,777,216 (16MB).
23. In the first terminal window (step 3), execute the command from step 3.
24. In the second terminal window (step 4), view the output of the `top` command.
 - What is the maximum value of the Virtual memory (VIRT column) of the `top` output on your system?

 - What is the max value of the Resident memory (RES Column) of the `top` output on your system?

25. In the first terminal window, view the output from the `PT_Stress.php` script execution (step 23).
 - What is the average time that the executions took to complete? _____
 - What are the output values for the `Sort_merge_passes` status variable?
Start value _____ End value _____
26. In the first terminal window (step 3) and using the `mysql` client, set the global `sort_buffer_size` system variable to 33,554,432 (32MB).
27. In the first terminal window (step 3), execute the command from step 3.

28. In the second terminal window (step 4), view the output of the `top` command.

- What is the maximum value of the Virtual memory (VIRT column) of the `top` output on your system?

-
- What is the max value of the Resident memory (RES Column) of the `top` output on your system?

29. In the first terminal window, view the output from the `PT_Stress.php` script execution (step 27).

- What is the average time that the executions took to complete? _____
- What are the output values for the `Sort_merge_passes` status variable?

Start value _____ End value _____

30. Reviewing the data collected, answer the following questions:

- Which `sort_buffer_size` setting produced the lowest number of sort merge passes?

- Which `sort_buffer_size` setting produced the highest number of sort merge passes?

- Which `sort_buffer_size` setting required the highest usage of memory?

- Which `sort_buffer_size` setting required the lowest usage of memory?

- Which `sort_buffer_size` setting had the best average execution time?

- Which `sort_buffer_size` setting had the worst average execution time?

- Which `sort_buffer_size` setting performed the best in relation to memory usage and execution time? Provide an explanation for your answer.

Steps	<code>sort_buffer_size</code>	VIRT	RES	Avg Time	<code>Sort_merge_passes</code>
2-5	32KB				Start
					End
6-9	512KB				Start
					End
10-13	1MB				Start
					End
14-17	4MB				Start
					End
18-21	8MB				Start
					End
22-25	16MB				Start
					End
26-29	32MB				Start
					End

Solutions 4-7: Sort Queries

Tasks

Note: These solutions are sample solutions and are not expected to be the exact output you see in your execution of the steps.

1. Open a terminal window and restart the MySQL server as `root` to reset the status variables.

Note: You can use the existing terminal window from the previous lab. However, at this point, you should have only one terminal window open to minimize confusion as more terminal windows are opened up.

```
sys> /etc/init.d/mysql restart
```

2. Using the `mysql` client, set the global `sort_buffer_size` system variable to 32,768 and the global `query_cache_type` to 0.

```
sys> mysql -uroot -poracle \
> -e"SET GLOBAL sort_buffer_size=32768;" \
> SET GLOBAL query_cache_type=0;"
```

Note: The minimum value that the `sort_buffer_size` system variable can be set to is 32,768. The system automatically applies the minimum if the value attempted to be set is lower than the minimum.

3. In the `/stage/scripts` directory, execute the following command to force MySQL to perform a sort operation using 20 connections with each connection running the `sort.read` file 1 time:

```
sys> ./PT_Stress.php \
> -u root \
> -p oracle \
> -c 20 \
> -s "sort.read" \
> -i 1 \
> -v "Sort%" \
> -t
```

4. In a separate terminal window, execute the following command to obtain a dynamic real time view of the `mysqld` application:

```
sys> top | grep mysqld
```

- The output you see is similar to the output below:

5451	mysql	15	0	741m	140m	3960	S	194.8	6.9	1:03.70	mysqld
5451	mysql	15	0	737m	136m	3972	S	198.7	6.7	1:09.69	mysqld
5451	mysql	15	0	738m	134m	3972	S	195.5	6.7	1:15.58	mysqld
5451	mysql	15	0	738m	136m	3972	S	198.8	6.8	1:21.57	mysqld
5451	mysql	15	0	735m	136m	3972	S	197.4	6.8	1:27.52	mysqld
5451	mysql	15	0	738m	136m	3972	S	198.1	6.8	1:33.49	mysqld
5451	mysql	15	0	736m	136m	3972	S	197.0	6.8	1:39.43	mysqld
5451	mysql	15	0	734m	134m	3972	S	192.5	6.7	1:45.23	mysqld
...											

- The columns of most interest to you for this lab include:

- Column 5: VIRT – This output value represents the virtual size of a process which is the sum of memory the process is actually using.

- Column 6: RES – This output value represents the actual physical memory a process is consuming.
 - What is the maximum value of the Virtual memory (VIRT column) of the top output on your system? 738m
 - What is the max value of the Resident memory (RES Column) of the top output on your system? 136m
5. In the first terminal window (step 3), view the output from the PT_Stress.php script execution.

Execution Times		
	Start	End
Minimum:	37.798122	
Maximum:	63.410322	
Average:	48.146330	
Sort_merge_passes	0	9800
Sort_range	0	0
Sort_rows	0	500000
Sort_scan	0	100

- What is the average time that the executions took to complete? 48.14 seconds
- What are the output values for the Sort_merge_passes status variable?
Start value 0 End value 9800

6. In the first terminal window (step 3) and using the mysql client, set the global sort_buffer_size system variable to 524,288 (512KB).

```
sys> mysql -uroot -poracle -e"SET GLOBAL \
> sort_buffer_size=524288"
```

7. In the first terminal window (step 3), in the /stage/scripts directory, execute the command from step 3.

```
sys> ./PT_Stress.php \
> -u root \
> -p oracle \
> -c 20 \
> -s "sort.read" \
> -i 1 \
> -v "Sort%" \
> -t
```

8. In the second terminal window (step 4), view the output of the top command.

- The output you see is similar to the output below:

```
...
5451 mysql      18   0  710m 110m 4112 S 196.7  5.5  3:27.10 mysqld
5451 mysql      18   0  710m 110m 4112 S 198.1  5.5  3:33.07 mysqld
5451 mysql      18   0  710m 110m 4112 S 196.0  5.5  3:38.98 mysqld
5451 mysql      18   0  710m 110m 4112 S 197.4  5.5  3:44.93 mysqld
5451 mysql      18   0  710m 110m 4112 S 197.7  5.5  3:50.89 mysqld
5451 mysql      18   0  710m 110m 4112 S 197.4  5.5  3:56.84 mysqld
5451 mysql      18   0  710m 110m 4112 S 196.7  5.5  4:02.77 mysqld
...
```

- What is the maximum value of the Virtual memory (VIRT column) of the top output on your system? 710m

- What is the max value of the Resident memory (RES Column) of the `top` output on your system? 110m
9. In the first terminal window, view the output from the `PT_Stress.php` script execution (step 7).

Execution Times		
	Start	End
Sort_merge_passes	9800	10500
Sort_range	0	0
Sort_rows	500000	1000000
Sort_scan	100	200

- What is the average time that the executions took to complete? 45.10 seconds
- What are the output values for the `Sort_merge_passes` status variable?
Start value 9800 End value 10500

10. In the first terminal window (step 3) and using the `mysql` client, set the global `sort_buffer_size` system variable to 1,048,576 (1MB).

```
sys> mysql -uroot -poracle -e"SET GLOBAL \
> sort_buffer_size=1048576"
```

11. In the first terminal window (step 3), in the `/stage/scripts` directory, execute the command from step 3.

```
sys> ./PT_Stress.php \
> -u root \
> -p oracle \
> -c 20 \
> -s "sort.read" \
> -i 1 \
> -v "Sort%" \
> -t
```

12. In the second terminal window (step 4), view the output of the `top` command.

- The output you see is similar to the output below:

```
...
5451 mysql      15   0  719m 119m 4112 S 198.4  5.9  5:43.19 mysqld
5451 mysql      15   0  721m 120m 4112 S 197.8  6.0  5:49.15 mysqld
5451 mysql      15   0  718m 118m 4112 S 196.3  5.9  5:55.07 mysqld
5451 mysql      15   0  718m 117m 4112 S 198.0  5.8  6:01.04 mysqld
5451 mysql      15   0  717m 117m 4112 S 192.4  5.8  6:06.84 mysqld
5451 mysql      15   0  716m 117m 4112 S 193.8  5.8  6:12.68 mysqld
5451 mysql      15   0  717m 118m 4112 S 198.3  5.9  6:18.66 mysqld
...
```

- What is the maximum value of the Virtual memory (VIRT column) of the `top` output on your system? 718m
- What is the max value of the Resident memory (RES Column) of the `top` output on your system? 117m

13. In the first terminal window, view the output from the `PT_Stress.php` script execution (step 11).

Execution Times		
	Start	End
Sort_merge_passes	10500	10900
Sort_range	0	0
Sort_rows	1000000	1500000
Sort_scan	200	300

- What is the average time that the executions took to complete? 50.45 seconds
- What are the output values for the `Sort_merge_passes` status variable?
Start value 10500 End value 10900

14. In the first terminal window (step 3) and using the `mysql` client, set the global `sort_buffer_size` system variable to 4,194,304.

```
sys> mysql -uroot -poracle -e"SET GLOBAL \
> sort_buffer_size= 4194304"
```

15. In the first terminal window (step 3), in the `/stage/scripts` directory, execute the command from step 3.

```
sys> ./PT_Stress.php \
> -u root \
> -p oracle \
> -c 20 \
> -s "sort.read" \
> -i 1 \
> -v "Sort%" \
> -t
```

16. In the second terminal window (step 4), view the output of the `top` command.

- The output you see is similar to the output below:

```
...
5451 mysql      15   0  778m 178m 4112 S 198.4  8.8  8:29.80 mysqld
5451 mysql      15   0  780m 181m 4112 S 195.7  9.0  8:35.70 mysqld
5451 mysql      15   0  781m 178m 4112 S 198.0  8.8  8:41.67 mysqld
5451 mysql      15   0  782m 179m 4112 S 194.4  8.9  8:47.53 mysqld
5451 mysql      15   0  777m 170m 4112 S 198.1  8.4  8:53.50 mysqld
5451 mysql      15   0  782m 181m 4112 S 199.1  9.0  8:59.50 mysqld
5451 mysql      15   0  782m 182m 4112 S 199.0  9.0  9:05.50 mysqld
5451 mysql      15   0  770m 167m 4112 S 197.4  8.3  9:11.45 mysqld
...
```

- What is the maximum value of the Virtual memory (VIRT column) of the `top` output on your system? 782m
- What is the max value of the Resident memory (RES Column) of the `top` output on your system? 181m

17. In the first terminal window, view the output from the `PT_Stress.php` script execution (step 15).

Execution Times		
	Start	End
Sort_merge_passes	10900	11000
Sort_range	0	0
Sort_rows	1500000	2000000
Sort_scan	300	400

- What is the average time that the executions took to complete? 87.72 seconds
- What are the output values for the `Sort_merge_passes` status variable?
Start value 10900 End value 11000

18. In the first terminal window (step 3) and using the `mysql` client, set the global `sort_buffer_size` system variable to 8,388,608 (8MB):

```
sys> mysql -uroot -poracle -e"SET GLOBAL \
> sort_buffer_size= 8388608"
```

19. In the first terminal window (step 3), in the `/stage/scripts` directory, execute the command from step 3.

```
sys> ./PT_Stress.php \
> -u root \
> -p oracle \
> -c 20 \
> -s "sort.read" \
> -i 1 \
> -v "Sort%" \
> -t
```

20. In the second terminal window (step 4), view the output of the `top` command.

- The output you see is similar to the output below:

```
...
5451 mysql      16   0  848m 195m 4112 S 197.5  9.7 11:00.62 mysqld
5451 mysql      16   0  848m 202m 4112 S 197.8 10.0 11:06.58 mysqld
5451 mysql      16   0  857m 234m 4112 S 199.1 11.6 11:12.58 mysqld
5451 mysql      16   0  841m 228m 4112 S 198.8 11.3 11:18.57 mysqld
5451 mysql      16   0  833m 222m 4112 S 197.8 11.0 11:24.57 mysqld
5451 mysql      16   0  833m 221m 4112 S 199.0 11.0 11:30.57 mysqld
5451 mysql      16   0  856m 219m 4112 S 198.1 10.9 11:36.54 mysqld
...
```

- What is the maximum value of the Virtual memory (VIRT column) of the `top` output on your system? 841m
- What is the max value of the Resident memory (RES Column) of the `top` output on your system? 228m

21. In the first terminal window, view the output from the `PT_Stress.php` script execution (step 19).

Execution Times		
	Start	End
Sort_merge_passes	11000	11100
Sort_range	0	0
Sort_rows	2000000	2500000
Sort_scan	400	500

- What is the average time that the executions took to complete? 43.79 seconds
- What are the output values for the `Sort_merge_passes` status variable?
Start value 11000 End value 11100

22. In the first terminal window (step 3) and using the `mysql` client, set the global `sort_buffer_size` system variable to 16,777,216 (16MB).

```
sys> mysql -uroot -poracle -e"SET GLOBAL \
> sort_buffer_size= 16777216"
```

23. In the first terminal window (step 3), in the `/stage/scripts` directory, execute the command from step 3.

```
sys> ./PT_Stress.php \
> -u root \
> -p oracle \
> -c 20 \
> -s "sort.read" \
> -i 1 \
> -v "Sort%" \
> -t
```

24. In the second terminal window (step 4), view the output of the `top` command.

- The output you see is similar to the output below:

```
...
5451 mysql      15   0  842m 232m 4112 S 198.4 11.5 13:03.54 mysqld
5451 mysql      15   0  857m 241m 4112 S 196.8 12.0 13:09.47 mysqld
5451 mysql      16   0  859m 233m 4112 S 198.1 11.6 13:15.44 mysqld
5451 mysql      16   0  859m 232m 4112 S 198.5 11.5 13:21.42 mysqld
5451 mysql      16   0  920m 272m 4112 S 198.4 13.5 13:27.40 mysqld
5451 mysql      16   0  890m 264m 4112 S 196.4 13.1 13:33.32 mysqld
5451 mysql      16   0  889m 254m 4112 S 198.5 12.6 13:39.30 mysqld
...
```

- What is the maximum value of the Virtual memory (VIRT column) of the `top` output on your system? 900m
- What is the max value of the Resident memory (RES Column) of the `top` output on your system? 250m

25. In the first terminal window, view the output from the `PT_Stress.php` script execution (step 23).

Execution Times		
	Start	End
Sort_merge_passes	11100	11200
Sort_range	0	0
Sort_rows	2500000	3000000
Sort_scan	500	600

- What is the average time that the executions took to complete? 46.05 seconds
- What are the output values for the `Sort_merge_passes` status variable?
Start value 11100 End value 11200

26. In the first terminal window (step 3) and using the `mysql` client, set the global `sort_buffer_size` system variable to 33554432 (32MB).

```
sys> mysql -uroot -poracle -e"SET GLOBAL \
> sort_buffer_size= 33554432"
```

27. In the first terminal window (step 3), in the `/stage/scripts` directory, execute the command from step 3.

```
sys> ./PT_Stress.php \
> -u root \
> -p oracle \
> -c 20 \
> -s "sort.read" \
> -i 1 \
> -v "Sort%" \
> -t
```

28. In the second terminal window (step 4), view the output of the `top` command.

- The output you see is similar to the output below:

```
...
5451 mysql      16   0 1286m 270m 4112 S 198.1 13.4 16:44.99 mysqld
5451 mysql      16   0 1225m 337m 4112 S 197.4 16.7 16:50.94 mysqld
5451 mysql      16   0 1134m 331m 4112 S 199.1 16.4 16:56.94 mysqld
5451 mysql      16   0 1256m 272m 4112 S 197.1 13.5 17:02.88 mysqld
5451 mysql      16   0 1226m 301m 4112 S 197.4 14.9 17:08.83 mysqld
5451 mysql      16   0 1257m 260m 4112 S 197.4 12.9 17:14.78 mysqld
5451 mysql      16   0 1287m 290m 4112 S 194.5 14.4 17:20.64 mysqld
...
```

- What is the maximum value of the Virtual memory (VIRT column) of the `top` output on your system? 1256m
- What is the max value of the Resident memory (RES Column) of the `top` output on your system? 301m

29. In the first terminal window, view the output from the `PT_Stress.php` script execution (step 27).

Execution Times		
	Start	End
Sort_merge_passes	11200	11200
Sort_range	0	0
Sort_rows	3000000	3500000
Sort_scan	600	700

- What is the average time that the executions took to complete? 37.23 seconds
- What are the output values for the `Sort_merge_passes` status variable?
Start value 11200 End value 11200

30. Reviewing the data collected, answer the following questions:

- Which `sort_buffer_size` setting produced the lowest number of sort merge passes? 32MB
- Which `sort_buffer_size` setting produced the highest number of sort merge passes? 32MB
- Which `sort_buffer_size` setting required the highest usage of memory? 32MB
- Which `sort_buffer_size` setting required the lowest usage of memory? 32KB
- Which `sort_buffer_size` setting had the best average execution time? 32MB
- Which `sort_buffer_size` setting had the worst average execution time? 4MB
- Which `sort_buffer_size` setting performed the best in relation to memory usage and execution time? Provide an explanation for your answer.

The setting of 512KB (using the data in the solution set), produced the best time with a manageable amount of memory usage.

Steps	<code>sort_buffer_size</code>	VIRT	RES	Avg Time	Sort_merge_passes
2-5	32KB	738m	136m	48.14	Start 0
					End 9800
6-9	512KB	710m	110m	45.1	Start 9800
					End 10500
10-13	1MB	718m	117m	50.45	Start 10500
					End 10900
14-17	4MB	782m	181m	87.72	Start 10900
					End 11000
18-21	8MB	841m	228m	43.79	Start 11000
					End 11100
22-25	16MB	900m	250m	46.04	Start 11100
					End 11200
26-29	32MB	1256m	301m	37.23	Start 11200
					End 11200

Practices for Lesson 5: MySQL Query Cache

Chapter 5

Practices for Lesson 5

Practices Overview

In this practice, you test your knowledge of improving the performance of the MySQL server using the MySQL query cache.

Practice 5-1: Query Cache

Overview

In this practice, you determine the most effective setting for `query_cache_size` system variable based on a sample of data collected. To accomplish this objective, you do the following:

- Set the `query_cache_size` system variable using settings ranging from 0 to 512 MB.
- Execute a sample collection of SQL data using 10 connections against the MySQL server.
- Evaluate the maximum time that the connections to the server took to complete along with the amount of virtual (swapped and physical) memory used by each execution.
- Calculate the query cache utilization rate for each setting.

Assumptions

- The MySQL server is running.
- The `many_tables` database is installed.
- The `PT_stress.php` and `qcache.read` files are located in the `/stage/scripts` directory.

Duration

This practice should take 40 minutes to complete.

Tasks

Note: A table has been provided at the end of the steps for this practice for you to fill in the values collected during the running of the steps. Entering the data from the steps into this table can assist you in comparing the values collected.

1. Open a terminal window and restart the server as `root` to reset the status variables.
2. Using the `mysql` client, execute the following command to view the system variable that lists the optional installed features of the MySQL instance being used:

```
sys> mysql -uroot -poracle -e"SHOW VARIABLES LIKE 'have%'"
```

- The output looks similar to the output below:

Variable_name	Value
have_compress	YES
have_crypt	YES
...	
have_query_cache	YES
have_rtree_keys	YES
have_ssl	DISABLED
have_symlink	YES

- What is the value for the `have_query_cache` system variable? _____
 - YES – States that the `mysqld` application contains the code for the query cache.
 - NO – States that the `mysqld` application does not contain the code for the query cache.

Note: The MySQL server can be compiled from source without support for the query cache by using the configure option `--without-query-cache`.

3. Using the `mysql` client, set the `query_cache_type` system variable to 1 (ON).
4. Using the `mysql` client, set the `query_cache_size` system variable to 0.
 - Setting the `query_cache_size` to 0 effectively disables the query cache.
5. Execute the following command to have MySQL simulate 10 connections running the `qcache.read` file five times each:

```
sys> ./PT_Stress.php \
> -u root \
> -p oracle \
> -c 10 \
> -s "qcache.read" \
> -i 5 \
> -v "QCache_hits,Com_select" \
> -t
```

6. In a separate terminal window, execute the following command to obtain a dynamic real-time view of the `mysqld` application:

```
sys> top | grep mysqld
```

- The output you see is similar to the output below:

```
...
1022 mysql      22    0  705m  83m 4176 S 195.4  2.4  1:02.79 mysqld
1022 mysql      22    0  705m  83m 4176 S 195.1  2.4  1:08.67 mysqld
1022 mysql      22    0  705m  83m 4176 S 196.1  2.4  1:14.58 mysqld
1022 mysql      22    0  705m  83m 4176 S 197.8  2.4  1:20.54 mysqld
1022 mysql      22    0  705m  83m 4176 S 195.8  2.4  1:26.44 mysqld
1022 mysql      22    0  705m  83m 4176 S 195.8  2.4  1:32.34 mysqld
1022 mysql      22    0  705m  83m 4176 S 195.4  2.4  1:38.23 mysqld
1022 mysql      22    0  705m  83m 4176 S 196.8  2.4  1:44.16 mysqld
1022 mysql      22    0  705m  83m 4176 S 196.4  2.4  1:50.08 mysqld
...
```

- What is the maximum value of the virtual memory (VIRT column) of the `top` output on your system?
-
7. In the first terminal window (step 2), view the output from the `PT_stress.php` script execution.
 - What is the maximum time that the executions took to complete? _____
 - What are the output values for the `Qcache_hits` status variable?

Start value _____ End value _____

- What are the output values for the `Com_select` status variable?

Start value _____ End value _____

- What is the query cache utilization rate? _____

8. Using the `mysql` client, set the `query_cache_size` system variable to 524,288 (512 KB).
9. Execute the command from step 5.
10. In the second terminal window (step 6), view the output of the `top` command.
 - What is the maximum value of the virtual memory (VIRT column) of the `top` output on your system:

11. In the first terminal window (step 2), view the output from the `PT_stress.php` script execution.
 - What is the maximum time that the executions took to complete? _____
 - What are the output values for the `Qcache_hits` status variable?
Start value _____ End value _____
 - What are the output values for the `Com_select` status variable?
Start value _____ End value _____
 - What is the query cache utilization rate? _____
12. Using the `mysql` client, set the `query_cache_size` system variable to 1,048,576 (1 MB).
13. Execute the command from step 5.
14. In the second terminal window (step 6), view the output of the `top` command.
 - What is the maximum value of the virtual memory (VIRT column) of the `top` output on your system:

15. In the first terminal window (step 2), view the output from the `PT_stress.php` script execution.
 - What is the maximum time that the executions took to complete? _____
 - What are the output values for the `Qcache_hits` status variable?
Start value _____ End value _____
 - What are the output values for the `Com_select` status variable?
Start value _____ End value _____
 - What is the query cache utilization rate? _____
16. Using the `mysql` client, set the `query_cache_size` system variable to 4,194,304 (4 MB).
17. Execute the command from step 5.
18. In the second terminal window (step 6), view the output of the `top` command.
 - What is the maximum value of the virtual memory (VIRT column) of the `top` output on your system:

19. In the first terminal window (step 2), view the output from the `PT_stress.php` script execution.

- What is the maximum time that the executions took to complete? _____
- What are the output values for the `Qcache_hits` status variable?

Start value _____ End value _____

- What are the output values for the `Com_select` status variable?

Start value _____ End value _____

- What is the query cache utilization rate? _____

20. Using the `mysql` client, set the `query_cache_size` system variable to 16,777,216 (16 MB).

21. Execute the command from step 5.

22. In the second terminal window (step 6), view the output of the `top` command.

- What is the maximum value of the virtual memory (VIRT column) of the `top` output on your system:
- _____

23. In the first terminal window (step 2), view the output from the `PT_stress.php` script execution.

- What is the maximum time that the executions took to complete? _____
- What are the output values for the `Qcache_hits` status variable?

Start value _____ End value _____

- What are the output values for the `Com_select` status variable?

Start value _____ End value _____

- What is the query cache utilization rate? _____

24. Using the `mysql` client, set the `query_cache_size` system variable to 67,108,864 (64 MB).

25. Execute the command from step 5.

26. In the second terminal window (step 6), view the output of the `top` command.

- What is the maximum value of the virtual memory (VIRT column) of the `top` output on your system:
- _____

27. In the first terminal window (step 2), view the output from the `PT_stress.php` script execution.

- What is the maximum time that the executions took to complete? _____
- What are the output values for the `Qcache_hits` status variable?

Start value _____ End value _____

- What are the output values for the `Com_select` status variable?

Start value _____ End value _____

- What is the query cache utilization rate? _____

28. Using the `mysql` client, set the `query_cache_size` system variable to 134,217,728 (128 MB).

29. Execute the command from step 5.

30. In the second terminal window (step 6), view the output of the `top` command.

- What is the maximum value of the virtual memory (VIRT column) of the `top` output on your system:
- _____

31. In the first terminal window (step 2), view the output from the `PT_stress.php` script execution.

- What is the maximum time that the executions took to complete? _____
- What are the output values for the `Qcache_hits` status variable?

Start value _____ End value _____

- What are the output values for the `Com_select` status variable?

Start value _____ End value _____

- What is the query cache utilization rate? _____

32. Using the `mysql` client, set the `query_cache_size` system variable to 268,435,456 (256 MB).

33. Execute the command from step 5.

34. In the second terminal window (step 6), view the output of the `top` command.

- What is the maximum value of the virtual memory (VIRT column) of the `top` output on your system:
- _____

35. In the first terminal window (step 2), view the output from the `PT_stress.php` script execution.

- What is the maximum time that the executions took to complete? _____
- What are the output values for the `Qcache_hits` status variable?

Start value _____ End value _____

- What are the output values for the `Com_select` status variable?

Start value _____ End value _____

- What is the query cache utilization rate? _____

36. Using the `mysql` client, set the `query_cache_size` system variable to 536,870,912 (512 MB).

37. Execute the command from step 5.

38. In the second terminal window (step 6), view the output of the `top` command.

- What is the maximum value of the virtual memory (VIRT column) of the `top` output on your system:
- _____

39. In the first terminal window (step 2), view the output from the `PT_stress.php` script execution.

- What is the maximum time that the executions took to complete? _____
- What are the output values for the `Qcache_hits` status variable?

Start value _____ End value _____

- What are the output values for the `Com_select` status variable?

Start value _____ End value _____

- What is the query cache utilization rate? _____

40. Reviewing the data collected, answer the following questions:

- Which `query_cache_size` setting required the highest usage of memory?
- _____

- Which `query_cache_size` setting required the lowest usage of memory?
- _____

- Which `query_cache_size` setting had the best average execution time?
- _____

- Which `query_cache_size` setting had the worst average execution time?
- _____

- Which `query_cache_size` setting performed the best in relation to memory usage and execution time? Provide an explanation for your answer.
-
-

Steps	<code>query_cache_size</code>	VIRT	Max Time	Qcache_hits	Com_select	Query Cache Utilization Rate
4-7	0			Start	Start	
				End	End	
8-11	512KB			Start	Start	
				End	End	
12-15	1MB			Start	Start	
				End	End	
16-19	4MB			Start	Start	
				End	End	
20-23	16MB			Start	Start	
				End	End	
24-27	64MB			Start	Start	
				End	End	
28-31	128MB			Start	Start	
				End	End	
32-35	256MB			Start	Start	
				End	End	
36-39	512MB			Start	Start	
				End	End	

41. As `root`, modify the `/etc/my.cnf` file to turn off the MySQL query cache.

42. As `root`, restart the MySQL server.

Note: The remaining practices in this course do not use the query cache. Turning the query cache off now prevents it from having an effect on the results of the practices.

Solutions 5-1: Query Cache

Tasks

Note: These solutions are sample solutions and are not expected to be the exact output you see in your execution of the steps.

1. Open a terminal window and restart the server as `root` to reset the status variables.

```
sys> /etc/init.d/mysql restart
```

2. Using the `mysql` client, execute the following command to view the system variable that lists the optional installed features of the MySQL instance being used:

```
sys> mysql -uroot -poracle -e"SHOW VARIABLES LIKE 'have%'"
```

- The output looks similar to the output below:

Variable_name	Value
have_compress	YES
have_crypt	YES
...	
have_query_cache	YES
have_rtree_keys	YES
have_ssl	DISABLED
have_symlink	YES

- What is the value for the `have_query_cache` system variable? YES
 - YES – States that the `mysqld` application contains the code for the query cache.
 - NO – States that the `mysqld` application does not contain the code for the query cache.

Note: The MySQL server can be compiled from source without support for the query cache by using the configure option `--without-query-cache`.

3. Using the `mysql` client, set the `query_cache_type` system variable to 1 (ON).

```
sys> mysql -uroot -poracle -e"SET GLOBAL \
> query_cache_type = 1"
```

4. Using the `mysql` client, set the `query_cache_size` system variable to 0.

```
sys> mysql -uroot -poracle -e"SET GLOBAL \
> query_cache_size = 0"
```

- Setting the `query_cache_size` to 0 effectively disables the query cache.

5. Execute the following command to have MySQL simulate 10 connections running the `qcache.read` file five times each:

```
sys> ./PT_Stress.php \
> -u root \
> -p oracle \
> -c 10 \
> -s "qcache.read" \
> -i 5 \
> -v "QCache_hits,Com_select" \
> -t
```

6. In a separate terminal window, execute the following command to obtain a dynamic real-time view of the mysqld application:

```
sys> top | grep mysqld
```

- The output you see is similar to the output below:

```
...
13889 mysql      18    0  701m  80m 4152 S 197.2  2.3  0:14.52 mysqld
13889 mysql      18    0  701m  80m 4160 S 197.2  2.3  0:20.46 mysqld
13889 mysql      18    0  701m  80m 4160 S 196.5  2.3  0:26.38 mysqld
13889 mysql      18    0  701m  80m 4160 S 191.6  2.3  0:32.15 mysqld
13889 mysql      18    0  701m  80m 4160 S 181.9  2.3  0:37.63 mysqld
13889 mysql      18    0  701m  80m 4160 S 183.6  2.3  0:43.16 mysqld
...
```

- What is the maximum value of the virtual memory (VIRT column) of the `top` output on your system: 701m

7. In the first terminal window (step 2), view the output from the `PT_stress.php` script execution.

Execution Times	
Minimum:	90.443334
Maximum:	152.759345
Average:	139.341257

Start	End
Qcache_hits	0
Com_select	3
	49953

- What is the maximum time that the executions took to complete? 152.76 seconds
- What are the output values for the `Qcache_hits` status variable?
Start value 0 End value 0
- What are the output values for the `Com_select` status variable?
Start value 3 End value 49953
- What is the query cache utilization rate? 0%

8. Using the `mysql` client, set the `query_cache_size` system variable to 524,288 (512 KB).

```
sys> mysql -uroot -poracle -e"SET GLOBAL \
> query_cache_size = 524288"
```

9. Execute the command from step 5.

```
sys> ./PT_Stress.php \
> -u root \
> -p oracle \
> -c 10 \
> -s "qcache.read" \
> -i 5 \
> -v "QCache_hits,Com_select" \
> -t
```

10. In the second terminal window (step 6), view the output of the `top` command.

- The output you see is similar to the output below:

```
...
13889 mysql      15    0  702m  81m 4208 S 197.2  2.3  8:00.62 mysqld
13889 mysql      15    0  702m  81m 4208 S 196.8  2.3  8:06.55 mysqld
13889 mysql      15    0  702m  81m 4208 S 196.6  2.3  8:12.47 mysqld
13889 mysql      15    0  702m  81m 4208 S 191.3  2.3  8:18.23 mysqld
...
```

- What is the maximum value of the virtual memory (VIRT column) of the `top` output on your system: 702m

11. In the first terminal window (step 2), view the output from the `PT_stress.php` script execution.

Execution Times		
Minimum:	79.245844	
Maximum:	108.732350	
Average:	97.324099	
	Start	End
Qcache_hits	0	12909
Com_select	49954	86995

- What is the maximum time that the executions took to complete? 108.99 seconds
- What are the output values for the `Qcache_hits` status variable?
Start value 0 End value 12909
- What are the output values for the `Com_select` status variable?
Start value 49954 End value 86995
- What is the query cache utilization rate?

$$((12909-0)/(86995-49954+(12909-0)) * 100) = 25.8\%$$

12. Using the `mysql` client, set the `query_cache_size` system variable to 1,048,576 (1 MB).

```
sys> mysql -uroot -poracle -e"SET GLOBAL \
> query_cache_size = 1048576"
```

13. Execute the command from step 5.

```
sys> ./PT_Stress.php \
> -u root \
> -p oracle \
> -c 10 \
> -s "qcache.read" \
> -i 5 \
> -v "QCache_hits,Com_select" \
> -t
```

14. In the second terminal window (step 6), view the output of the `top` command.

- The output you see is similar to the output below:

```
...
13889 mysql      15    0  702m  80m 4208 S 132.3  2.3  8:41.96 mysqld
13889 mysql      15    0  702m  81m 4208 S 193.3  2.3  8:47.78 mysqld
13889 mysql      23    0  702m  81m 4208 S 129.1  2.3  8:51.67 mysqld
13889 mysql      17    0  702m  81m 4208 S   7.0  2.3  8:51.88 mysqld
...
```

- What is the maximum value of the virtual memory (VIRT column) of the `top` output on your system: 702m

15. In the first terminal window (step 2), view the output from the `PT_stress.php` script execution.

Execution Times		
	Start	End
Minimum:	4.694027	
Maximum:	8.241787	
Average:	6.203766	
Qcache_hits	12909	60902
Com_select	86996	88953

- What is the maximum time that the executions took to complete? 8.24 seconds
- What are the output values for the `Qcache_hits` status variable?
Start value 12909 End value 60902
- What are the output values for the `Com_select` status variable?
Start value 86996 End value 88953
- What is the query cache utilization rate?

$$((60902-12909)/(88953-86996+(60902-12909)) * 100) = 96.1\%$$

16. Using the `mysql` client, set the `query_cache_size` system variable to 4,194,304 (4 MB).

```
sys> mysql -uroot -poracle -e"SET GLOBAL \
> query_cache_size = 4194304"
```

17. Execute the command from step 5.

```
sys> ./PT_Stress.php \
> -u root \
> -p oracle \
> -c 10 \
> -s "qcache.read" \
> -i 5 \
> -v "QCache_hits,Com_select" \
> -t
```

18. In the second terminal window (step 6), view the output of the `top` command.

- The output you see is similar to the output below:

```
...
13889 mysql      15    0  705m  81m 4208 S 188.5  2.3  8:57.55 mysqld
13889 mysql      21    0  705m  81m 4208 S 193.2  2.3  9:03.37 mysqld
13889 mysql      17    0  705m  81m 4208 S   73.7  2.3  9:05.59 mysqld
...
```

- What is the maximum value of the virtual memory (VIRT column) of the `top` output on your system: 705m

19. In the first terminal window (step 2), view the output from the `PT_stress.php` script execution.

Execution Times		
	Start	End
Minimum:	1.992539	
Maximum:	8.138601	
Average:	6.455585	
<code>Qcache_hits</code>	60902	108886
<code>Com_select</code>	88954	90920

- What is the maximum time that the executions took to complete? 8.14 seconds
- What are the output values for the `Qcache_hits` status variable?
Start value 60902 End value 108886
- What are the output values for the `Com_select` status variable?
Start value 88954 End value 90920
- What is the query cache utilization rate?

$$((108886-60902)/(90920-88954+(108886-60902)) * 100) = 96.1\%$$

20. Using the `mysql` client, set the `query_cache_size` system variable to 16,777,216 (16 MB).

```
sys> mysql -uroot -poracle -e"SET GLOBAL \
> query_cache_size = 16777216"
```

21. Execute the command from step 5.

```
sys> ./PT_Stress.php \
> -u root \
> -p oracle \
> -c 10 \
> -s "qcache.read" \
> -i 5 \
> -v "QCache_hits,Com_select" \
> -t
```

22. In the second terminal window (step 6), view the output of the `top` command.

- The output you see is similar to the output below:

```
...
13889 mysql      24    0  717m  80m 4208 S  69.8  2.3  9:07.69 mysqld
13889 mysql      24    0  717m  81m 4208 S 184.2  2.3  9:13.24 mysqld
13889 mysql      17    0  717m  81m 4208 S 147.7  2.3  9:17.69 mysqld
13889 mysql      15    0  717m  81m 4208 S  36.2  2.3  9:18.78 mysqld
...
```

- What is the maximum value of the virtual memory (VIRT column) of the `top` output on your system: 717m

23. In the first terminal window (step 2), view the output from the `PT_stress.php` script execution.

Execution Times		
	Start	End
Minimum:	1.390881	
Maximum:	7.972511	
Average:	5.108484	
<code>Qcache_hits</code>	108886	156981
<code>Com_select</code>	90921	92776

- What is the maximum time that the executions took to complete? 7.97 seconds
- What are the output values for the `Qcache_hits` status variable?
Start value 108886 End value 156981
- What are the output values for the `Com_select` status variable?
Start value 90921 End value 92776
- What is the query cache utilization rate?

$$((156981-108886)/(92776-90921+(156981-108886)) * 100) = 96.3\%$$

24. Using the `mysql` client, set the `query_cache_size` system variable to 67,108,864 (64 MB).

```
sys> mysql -uroot -poracle -e"SET GLOBAL \
> query_cache_size = 67108864"
```

25. Execute the command from step 5.

```
sys> ./PT_Stress.php \
> -u root \
> -p oracle \
> -c 10 \
> -s "qcache.read" \
> -i 5 \
> -v "QCache_hits,Com_select" \
> -t
```

26. In the second terminal window (step 6), view the output of the `top` command.

- The output you see is similar to the output below:

```
...
13889 mysql      15      0  765m  81m 4208 S 173.2  2.3   9:23.99 mysqld
13889 mysql      15      0  765m  81m 4208 S 194.2  2.3   9:29.84 mysqld
13889 mysql      15      0  765m  81m 4208 S  91.6  2.3   9:32.60 mysqld
...
```

- What is the maximum value of the virtual memory (VIRT column) of the `top` output on your system: 765m

27. In the first terminal window (step 2), view the output from the `PT_stress.php` script execution.

```
Execution Times
Minimum: 5.606014
Maximum: 8.276349
Average: 6.742483
```

	Start	End
<code>Qcache_hits</code>	156981	204972
<code>Com_select</code>	92777	94736

- What is the maximum time that the executions took to complete? 8.27 seconds
- What are the output values for the `Qcache_hits` status variable?
Start value 156981 End value 204972
- What are the output values for the `Com_select` status variable?
Start value 92777 End value 94736
- What is the query cache utilization rate?

$$((204972-156981)/(94736-92777 + (204972-156981)) * 100) = 96.1\%$$

28. Using the `mysql` client, set the `query_cache_size` system variable to 134,217,728 (128 MB).

```
sys> mysql -uroot -poracle -e"SET GLOBAL \
> query_cache_size = 134217728"
```

29. Execute the command from step 5.

```
sys> ./PT_Stress.php \
> -u root \
> -p oracle \
> -c 10 \
> -s "qcache.read" \
> -i 5 \
> -v "QCache_hits,Com_select" \
> -t
```

30. In the second terminal window (step 6), view the output of the `top` command.

- The output you see is similar to the output below:

```
...
13889 mysql      19    0  829m  80m 4208 S  16.3  2.3   9:33.09 mysqld
13889 mysql      18    0  829m  81m 4208 S 196.2  2.3   9:39.00 mysqld
13889 mysql      17    0  829m  81m 4208 S 185.9  2.3   9:44.60 mysqld
13889 mysql      20    0  829m  81m 4208 S  59.4  2.3   9:46.39 mysqld
...
```

- What is the maximum value of the virtual memory (VIRT column) of the `top` output on your system: 829m

31. In the first terminal window (step 2), view the output from the `PT_stress.php` script execution.

```
Execution Times
```

```
Minimum: 3.353318
```

```
Maximum: 8.113228
```

```
Average: 6.068488
```

	Start	End
Qcache_hits	204972	252974
Com_select	94737	96685

- What is the maximum time that the executions took to complete? 8.11 seconds
- What are the output values for the `Qcache_hits` status variable?
Start value 204972 End value 252974
- What are the output values for the `Com_select` status variable?
Start value 94737 End value 96685
- What is the query cache utilization rate?

$$((252974-204972)/(96685-94737+(252974-204972)) * 100) = 25.8\%$$

32. Using the `mysql` client, set the `query_cache_size` system variable to 268,435,456 (256 MB).

```
sys> mysql -uroot -poracle -e"SET GLOBAL \
> query_cache_size = 268435456"
```

33. Execute the command from step 5.

```
sys> ./PT_Stress.php \
> -u root \
> -p oracle \
> -c 10 \
> -s "qcache.read" \
> -i 5 \
> -v "QCache_hits,Com_select" \
> -t
```

34. In the second terminal window (step 6), view the output of the `top` command.

- The output you see is similar to the output below:

```
...
13889 mysql      17    0  957m  81m 4208 S 108.0  2.3   9:49.64 mysqld
13889 mysql      17    0  957m  81m 4208 S 194.2  2.3   9:55.49 mysqld
13889 mysql      17    0  957m  81m 4208 S 142.1  2.3   9:59.77 mysqld
13889 mysql      20    0  957m  81m 4208 S  18.3  2.3  10:00.32 mysqld
...
```

- What is the maximum value of the virtual memory (VIRT column) of the `top` output on your system: 957m

35. In the first terminal window (step 2), view the output from the `PT_stress.php` script execution.

Execution Times		
	Start	End
Minimum:	6.858047	
Maximum:	8.333686	
Average:	7.727609	
<code>Qcache_hits</code>	252974	300949
<code>Com_select</code>	96686	98661

- What is the maximum time that the executions took to complete? 8.33 seconds
- What are the output values for the `Qcache_hits` status variable?
Start value 252974 End value 300949
- What are the output values for the `Com_select` status variable?
Start value 96686 End value 98661
- What is the query cache utilization rate?

$$\frac{(300949 - 252974)}{(98661 - 96686 + (300949 - 252974))} * 100 = 96.0\%$$

36. Using the `mysql` client, set the `query_cache_size` system variable to 536,870,912 (512 MB).

```
sys> mysql -uroot -poracle -e"SET GLOBAL \
> query_cache_size = 536870912"
```

37. Execute the command from step 5.

```
sys> ./PT_Stress.php \
> -u root \
> -p oracle \
> -c 10 \
> -s "qcache.read" \
> -i 5 \
> -v "QCache_hits,Com_select" \
> -t
```

38. In the second terminal window (step 6), view the output of the `top` command.

- The output you see is similar to the output below:

```
...
13889 mysql      16    0 1213m  81m 4208 S 153.9  2.3 10:04.95 mysqld
13889 mysql      16    0 1213m  81m 4208 S 186.9  2.3 10:10.58 mysqld
13889 mysql      16    0 1213m  81m 4208 S 111.9  2.3 10:13.95 mysqld
13889 mysql      18    0 1213m  81m 4208 S   4.3  2.3 10:14.08 mysqld
...
```

- What is the maximum value of the virtual memory (VIRT column) of the `top` output on your system: 1213m

39. In the first terminal window (step 2), view the output from the `PT_stress.php` script execution.

Execution Times
Minimum: 7.361715
Maximum: 8.629994
Average: 8.034780
Start End
Qcache_hits 300949 348967
Com_select 98662 100594

- What is the maximum time that the executions took to complete? 8.62 seconds
- What are the output values for the `Qcache_hits` status variable?
Start value 300949 End value 348967
- What are the output values for the `Com_select` status variable?
Start value 98662 End value 100594
- What is the query cache utilization rate?

$$\frac{(348967 - 300949)}{(100594 - 98662 + (348967 - 300949))} * 100 = 96.1\%$$

40. Reviewing the data collected, answer the following questions:

- Which `query_cache_size` setting required the highest usage of memory? 512 MB
- Which `query_cache_size` setting required the lowest usage of memory? 0
- Which `query_cache_size` setting had the best average execution time? 16 MB
- Which `query_cache_size` setting had the worst average execution time? 0
- Which `query_cache_size` setting performed the best in relation to memory usage and execution time? Provide an explanation for your answer.

The setting of 16 MB (using the data in the solution set), provided the most flexibility without having a large difference in the amount of memory that was used.

Steps	query_cache_size	VIRT	Max Time	Qcache_hits	Com_select	Query Cache Utilization Rate
4-7	0	701m	152.76	Start 0	Start 3	0%
				End 0	End 49953	
8-11	512KB	702m	108.73	Start 0	Start 49954	25.8%
				End 12909	End 86995	
12-15	1MB	702m	8.24	Start 12909	Start 86996	96.1%
				End 60902	End 88953	
16-19	4MB	705m	8.14	Start 60902	Start 88954	96.1%
				End 108886	End 90920	
20-23	16MB	717m	7.97	Start 108886	Start 90921	96.3%
				End 156981	End 92776	
24-27	64MB	765m	8.27	Start 156981	Start 92777	96.1%
				End 204972	End 94736	
28-31	128MB	829m	8.11	Start 204972	Start 94737	96.1%
				End 252974	End 96685	
32-35	256MB	957m	8.33	Start 252974	Start 96686	96.0%
				End 300949	End 98661	
36-39	512MB	1213m	8.62	Start 300949	Start 98662	96.1%
				End 348967	End 100594	

41. As root, modify the `/etc/my.cnf` file to turn off the mysql query cache.

```
sys> vi /etc/my.cnf
```

- Add the following lines under the `[mysqld]` section:

```
query_cache_type = 0
```

42. As root, restart the MySQL server.

```
sys> /etc/init.d/mysql restart
```

Note: The remaining practices in this course do not use the query cache. Turning the query cache off now prevents it from having an effect on the results of the practices.

Practices for Lesson 6: InnoDB

Chapter 6

Practices for Lesson 6

Practices Overview

In these practices, you test your knowledge of improving the performance of queries by using the InnoDB storage engine.

Practice 6-1: Committing Transactions

Overview

In this practice, you evaluate the effect of the frequency of commits on performance. To accomplish this objective, you do the following:

- Evaluate a MySQL stored procedure called “innodb_inserts”.
- Use the many_tables database to execute multiple instances of the innodb_inserts MySQL stored procedure.

Assumptions

- The MySQL server is installed and running.
- The many_tables database is installed.

Duration

This practice should take 20 minutes to complete.

Prerequisite Task

1. Open a terminal window, start the mysql client, and select the many_tables database.
2. To prevent a warning from being displayed in the following steps, execute the following in the mysql client to tell the MySQL server to use the MIXED binary log format:

```
mysql> SET binlog_format = 'MIXED';
```

Tasks

Note: A table has been provided at the end of the steps for this practice for you to fill in the values collected during the running of the steps. Entering the data from the steps into this table can assist you in comparing the values collected.

1. Execute the following statement in the mysql client to review the structure of the innodb_inserts MySQL stored procedure:

```
mysql> SELECT ROUTINE_NAME, ROUTINE_DEFINITION FROM  
      -> INFORMATION_SCHEMA.ROUTINES  
      -> WHERE ROUTINE_NAME LIKE 'innodb_inserts'\G
```

- Reading the script of the innodb_inserts MySQL stored procedure, what is its purpose?

Note: Understanding and using MySQL stored procedures can assist you in multiple DBA tasks, especially performance tuning.

2. Execute the following statement in the mysql client:

```
mysql> CALL innodb_inserts(100000,1);
```

- This execution of the innodb_inserts MySQL stored procedure creates a table and inserts 100,000 records into it committing after each insert. How long did it take to perform this action?

3. Execute the following statement in the mysql client:

```
mysql> CALL innodb_inserts(100000,5);
```

- This execution of the innodb_inserts MySQL stored procedure creates a table and inserts 100,000 records into it committing after every 5 inserts. How long did it take to perform this action?

4. Execute the following statement in the mysql client:

```
mysql> CALL innodb_inserts(100000,10);
```

- This execution of the innodb_inserts MySQL stored procedure creates a table and inserts 100,000 records into it committing after every 10 inserts. How long did it take to perform this action?

5. Execute the following statement in the mysql client:

```
mysql> CALL innodb_inserts(100000,100);
```

- This execution of the innodb_inserts MySQL stored procedure creates a table and inserts 100,000 records into it committing after every 100 inserts. How long did it take to perform this action?

6. Execute the following statement in the mysql client:

```
mysql> CALL innodb_inserts(100000,1000);
```

- This execution of the innodb_inserts MySQL stored procedure creates a table and inserts 100,000 records into it committing after every 1,000 inserts. How long did it take to perform this action?

7. Execute the following statement in the mysql client:

```
mysql> CALL innodb_inserts(100000,5000);
```

- This execution of the innodb_inserts MySQL stored procedure creates a table and inserts 100,000 records into it committing after every 5,000 inserts. How long did it take to perform this action?

8. Execute the following statement in the mysql client:

```
mysql> CALL innodb_inserts(100000,10000);
```

- This execution of the `innodb_inserts` MySQL stored procedure creates a table and inserts 100,000 records into it committing after every 10,000 inserts. How long did it take to perform this action?

9. Execute the following statement in the mysql client:

```
mysql> CALL innodb_inserts(100000,50000);
```

- This execution of the `innodb_inserts` MySQL stored procedure creates a table and inserts 100,000 records into it committing after every 50,000 inserts. How long did it take to perform this action?

10. Execute the following statement in the mysql client:

```
mysql> CALL innodb_inserts(100000,100000);
```

- This execution of the `innodb_inserts` MySQL stored procedure creates a table and inserts 100,000 records only committing when all records have been inserted. How long did it take to perform this action?

11. Reviewing the data obtained, which statement executed the best?

12. Reviewing the data obtained, which statement executed the worst?

13. What is the danger of not committing often with `INSERT`, `DELETE`, and `UPDATE` operations?

Note: Balancing performance with the usefulness of the data is a task that you need to consider when determining how often to commit the transactions running against your MySQL server.

14. Quit the mysql client.

Insert Statements	Number of Statements when Commit Occurs	Execution Time
100,000	1	
100,000	5	
100,000	10	
100,000	100	
100,000	1,000	
100,000	5,000	
100,000	10,000	
100,000	50,000	
100,000	100,00	

Solutions 6-1: Committing Transactions

Prerequisite Task

1. Open a terminal window, start the mysql client, and select the many_tables database.
2. To prevent a warning from being displayed in the following steps, execute the following in the mysql client to tell the MySQL server to use the MIXED binary log format:

```
mysql> SET binlog_format = 'MIXED';
```

Tasks

Note: These solutions are sample solutions and are not expected to be the exact output you see in your execution of the steps.

1. Open a terminal window, start the mysql client, and select the many_tables database.

```
sys> mysql -uroot -poracle  
mysql> USE many_tables;
```

2. Execute the following statement in the mysql client to review the structure of the innodb_inserts MySQL stored procedure:

```
mysql> SELECT ROUTINE_NAME, ROUTINE_DEFINITION FROM  
      -> INFORMATION_SCHEMA.ROUTINES  
      -> WHERE ROUTINE_NAME LIKE 'innodb_inserts'\G  
*****  
ROUTINE_NAME: innodb_inserts  
ROUTINE_DEFINITION: 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  
1 row in set (0.01 sec)
```

- Reading the script of the innodb_inserts MySQL stored procedure, what is its purpose?

This MySQL stored procedure creates a table then runs multiple inserts against the table based on user input. The frequency that the inserts commit is also controlled by the end user.

Note: Understanding and using MySQL stored procedures can assist you in multiple DBA tasks, especially performance tuning.

3. Execute the following statement in the mysql client:

```
mysql> CALL innodb_inserts(100000,1);  
Query OK, 0 rows affected(51.53 sec)
```

- This execution of the innodb_inserts MySQL stored procedure creates a table and inserts 100,000 records into it committing after each insert. How long did it take to perform this action?

51.5 seconds

4. Execute the following statement in the mysql client:

```
mysql> CALL innodb_inserts(100000,5);  
Query OK, 1 rows affected(16.66 sec)
```

- This execution of the innodb_inserts MySQL stored procedure creates a table and inserts 100,000 records into it committing after every 5 inserts. How long did it take to perform this action?

16.7 seconds

5. Execute the following statement in the mysql client:

```
mysql> CALL innodb_inserts(100000,10);  
Query OK, 0 rows affected(12.51 sec)
```

- This execution of the innodb_inserts MySQL stored procedure creates a table and inserts 100,000 records into it committing after every 10 inserts. How long did it take to perform this action?

12.5 seconds

6. Execute the following statement in the mysql client:

```
mysql> CALL innodb_inserts(100000,100);  
Query OK, 0 rows affected(8.24 sec)
```

- This execution of the innodb_inserts MySQL stored procedure creates a table and inserts 100,000 records into it committing after every 100 inserts. How long did it take to perform this action?

8.2 seconds

7. Execute the following statement in the mysql client:

```
mysql> CALL innodb_inserts(100000,1000);  
Query OK, 0 rows affected(7.49 sec)
```

- This execution of the innodb_inserts MySQL stored procedure creates a table and inserts 100,000 records into it committing after every 1,000 inserts. How long did it take to perform this action?

7.5 seconds

8. Execute the following statement in the mysql client:

```
mysql> CALL innodb_inserts(100000,5000);  
Query OK, 0 rows affected(7.63 sec)
```

- This execution of the innodb_inserts MySQL stored procedure creates a table and inserts 100,000 records into it committing after every 5,000 inserts. How long did it take to perform this action?

7.6 seconds

9. Execute the following statement in the mysql client:

```
mysql> CALL innodb_inserts(100000,10000);  
Query OK, 0 rows affected(7.67 sec)
```

- This execution of the innodb_inserts MySQL stored procedure creates a table and inserts 100,000 records into it committing after every 10,000 inserts. How long did it take to perform this action?

7.6 seconds

10. Execute the following statement in the mysql client:

```
mysql> CALL innodb_inserts(100000,50000);  
Query OK, 0 rows affected(7.17 sec)
```

- This execution of the innodb_inserts MySQL stored procedure creates a table and inserts 100,000 records into it committing after every 50,000 inserts. How long did it take to perform this action?

7.2 seconds

11. Execute the following statement in the mysql client:

```
mysql> CALL innodb_inserts(100000,100000);  
Query OK, 0 rows affected(7.33 sec)
```

- This execution of the innodb_inserts MySQL stored procedure creates a table and inserts 100,000 records only committing when all records have been inserted. How long did it take to perform this action?

7.3 seconds

12. Reviewing the data obtained, which statement executed the best?

Committing after 50,000 records were inserted.

13. Reviewing the data obtained, which statement executed the worst?

Committing after each insert.

14. What is the danger of not committing often with INSERT, DELETE, and UPDATE operations?

Having statements run for any extended length of time without committing can produce deadlocks (small transactions are less prone to collisions), the data can be inconsistent to end users until the transaction commits, holding log space, preventing purging, etc.

Note: Balancing performance with the usefulness of the data is a task that you need to consider when determining how often to commit the transactions running against your MySQL server.

15. Quit the mysql client.

```
mysql> exit;
```

Insert Statements	Number of Statements when Commit Occurs	Execution Time
100,000	1	51.53
100,000	5	16.66
100,000	10	12.51
100,000	100	8.24
100,000	1,000	7.49
100,000	5,000	7.63
100,000	10,000	7.67
100,000	50,000	7.17
100,000	100,00	7.33

Practice 6-2: SHOW ENGINE INNODB STATUS

Overview

In this practice, you evaluate the output of the SHOW ENGINE INNODB STATUS command. To accomplish this objective, you do the following:

- Use the `many_tables` database to create a large table that is used with this practice.
- Execute the `mysqlslap` application running multiple iterations of the `innodb_query.sql` script using multiple concurrent connections.
- Execute and answer questions related to the `SHOW ENGINE INNODB STATUS` output.

Assumptions

- The MySQL server is installed and running.
- The `many_tables` database is installed.
- The `mysqlslap` application is installed.
- The `innodb_query.sql` file is located in the `/stage/scripts` directory.

Duration

This practice should take 40 minutes to complete.

Tasks

1. Open a terminal window and restart the MySQL server as `root`.

Note: You can use the existing terminal window from the previous practice. However, at this point, you should have only one terminal window open to minimize confusion as more terminal windows are opened up.

2. Start the `mysql` client and select the `many_tables` database.
3. Execute the following statement in the `mysql` client to review the structure of the `create_city_huge` MySQL stored procedure:

```
mysql> SELECT ROUTINE_NAME, ROUTINE_DEFINITION FROM
    -> INFORMATION_SCHEMA.ROUTINES
    -> WHERE ROUTINE_NAME LIKE 'create_city_huge'\G
```

- Reading the script definition, what is the purpose of this script?

4. Execute the following statement in the `mysql` client to review the structure of the `add_data_to_city_huge` MySQL stored procedure:

```
mysql> SELECT ROUTINE_NAME, ROUTINE_DEFINITION FROM  
-> INFORMATION_SCHEMA.ROUTINES  
-> WHERE ROUTINE_NAME LIKE 'add_data_to_city_huge'\G
```

- Reading the script definition, what is the purpose of this script?

5. Execute the following statement in the `mysql` client:

```
mysql> CALL create_city_huge(100);
```

- This execution of the `create_city_huge` MySQL stored procedure creates a table and inserts 400,000 records into it.

6. Modify the `city_huge` table created in step 5 to use the InnoDB storage engine.
7. Open a second terminal window and execute the following command as `root` to create a load against the MySQL server and the `city_huge` table created in step 5.

```
sys> mysqlslap -uroot -poracle \  
> -q /stage/scripts/innodb_query.sql \  
> --create-schema=many_tables -i 20 -c 20
```

Note: This script runs for approximately 6 minutes. This is necessary to have a good sampling of data for the next few steps and to provide a more realistic InnoDB status report. Continue on with the remaining steps while the `mysqlslap` command is executing.

8. About 30 seconds after the `mysqlslap` command has begun executing in step 7, return to the first terminal window (step 1) and execute the following command to display the InnoDB status report:

```
mysql> SHOW ENGINE INNODB STATUS\G
```

- How many seconds were used to calculate the *per second averages* for the report?

Note: This is first line in the InnoDB status report of consequence and can be found directly under the `INNODB MONITOR OUTPUT` heading. Due to the fact that many of the values are based on per second, it is important to have a sampling of data of at least 20-30 seconds. Anything less is unusable and should be discarded.

- The `SEMAPHORES` section of the output assists you with determining how well MySQL is handling context switching. `reservation count` and `signal count` displays how often InnoDB uses the internal sync array. The ratios between these two values represent how frequently the InnoDB storage engine needs to use the O/S wait functions.
 - What is the reservation count value? _____
 - What is the signal count value? _____

- Spin locks are less costly than O/S waits, but still can be consuming a large amount of CPU resources.
- What is the Mutex spin waits value? _____
- What is the Mutex spin rounds value? _____

Note: Spin waits and spin rounds identify a problem if both are high because it is a sign that significant CPU resources are being wasted. Reducing the system variable `innodb_sync_spin_loops` can minimize the amount of CPU wasted by doing spin loops. However, reducing this setting can increase the amount of CPU resources that is used for context switching.

- The TRANSACTIONS section of the output provides helpful information determining if your application has lock contention and the reasons for transaction deadlocks. The `Trx id` counter is a hexadecimal number that is incremented for each transaction.
 - What is the `Trx id` counter value? _____
- The Purge done for `trx's n:0` is the hexadecimal number of the transaction that was last purged. The `undo n:0` is the hexadecimal number of the transaction currently processing. If the value is 0, InnoDB is currently not executing any transactions.
 - What is the Purge done for `trx's n:0` value? _____
 - What is the `undo n:0` value? _____

Note: InnoDB only purges old versions of transactions when an event takes place to let InnoDB know that it is completed. Uncommitted transactions that are old or stale may block the purge process. This can lead to system resources being consumed. By comparing the `Trx id` counter and Purge done for `trx's n:0` values, you can spot uncommitted transactions taking up resources.

- The History list length is the number of unpurged transactions in undo space. This value is increased as updated transactions are committed and decreased as purges take place. The isolation mode being used for the transactions has a large effect on how large this list can become.
 - What is the History list length value? _____
- The LIST OF TRANSACTIONS FOR EACH SESSION: displays all the transactions (if the number of transactions is relatively small) or a portion of the transactions (if the number of transactions is large). The information contained in the transaction lines can provide you with detailed information on the state of the transaction.
 - What is the status of the last transaction that appears in this list? _____

-
- What is the value of the os thread id for this transaction? _____
 - Which MySQL thread is running this transaction? _____
 - What is the value of the query id for this transaction? _____
 - Who is the user that this transaction is running under? _____

Note: The os thread id, MySQL thread id, and query id values can be used to identify the query for administration and troubleshooting purposes.

- 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).

- How many transactions are waiting for fsync (Pending flushes (fsync) log)?

- How many transactions are in the buffer pool? _____

Note: Calling fsync() on modified files is used by InnoDB to ensure data makes it to the disk (simply passing it to the OS cache is not enough). Consistently high values for Pending flushes or the buffer pool can be an indication of an I/O bound workload.

- The total number of I/O operations is displayed along with the computed averages for those operations. These values can be used for monitoring and graphing to evaluate trends in your systems use.

- What is the value of the os file reads? _____

- What is the value of the os file writes? _____

- What is the value of the os fsyncs? _____

- What is the average number of reads per second for this reporting period?

- What is the average bytes read per second for this reporting period?

- What is the average number of writes per second for this reporting period?

- What is the average number of fsync() operations for this reporting period?

- The INSERT BUFFER AND ADAPTIVE HASH INDEX section of the report displays the status of the insert buffer (Ibuf) to include the segment size and the free list along with the any records that are located in the insert buffer. This is followed by how many inserts were done in the insert buffer, how many records were merged and how many merges took place.

- For the merged operations, how many insert's took place during this reporting period? _____

- For the merged operations, how many delete mark's took place during this reporting period? _____

Note: When a page is in the buffer pool, it is updated directly. When a page is loaded to the buffer pool, any buffered changes are merged to it, so that users never see unmerged changes. The insert buffer bitmap keeps track of the available space on pages and prevents overflows when buffering inserts. Delete-marking records can always be buffered, because the flag is updated in place.

- Calculating the ratio of delete mark's and insert's, what is the insert buffer efficiency for this reporting period?

- The last section displays the hash table size, number of used cells and number of buffers used by adaptive hash index.

- What is the average number of *hash* searches that have taken place during this reporting period?

- What is the average number of *non-hash* searches that have taken place during this reporting period?

- The LOG section of the report displays the current log sequence number (which is the amount of bytes InnoDB has written in log files since system tablespace creation), up to which point logs have been flushed, and when the last checkpoint was performed.
 - What is the value of the Log sequence number? _____
 - What is the value of the Log flushed up to? _____
 - What is the percentage difference between these two values?

Note: A difference that is greater than 30% indicates a potential problem with the `innodb_log_buffer_size` system variable. Increasing the size of this system variable can reduce this difference.

- How log writes are pending? _____
- How many checkpoint writes are pending? _____
- How many log i/o's have taken place during this reporting period?

- What is the average number of log i/o's per second that have taken place during this reporting period?

Note: The log writes performance is primarily based on the `innodb_flush_logs_at trx_commit` system variable. This system variable controls when the log buffer is written to and logs files are flushed. Changing this value can improve the performance of your log i/o but raises the possibility of losing up to a second of transactions.

- The BUFFER POOL AND MEMORY section of the report displays the status of the information to help you determine if the buffer pool is sized properly for your system.
 - What is the Total memory allocated for InnoDB? _____
 - What is the value of the Buffer pool size? _____

Note: The buffer pool size listed in the SHOW ENGINE INNODB STATUS report is presented in pages of 16,384 bytes. For an innodb_buffer_pool_size of 134217728 would be equivalent to a buffer pool size of 8192.

- What is the value of the Free buffers? _____
- Note:** If your system continually has a high percentage of pages free (in comparison to the buffer pool size), it is a sign that your active database size may be smaller than the allocated buffer pool size. In this case, you can consider decreasing the innodb_buffer_pool_size system variable to redeem system resources for other processes.
- What is the Buffer pool hit rate? _____

Note: A buffer pool hit rate of 1000/1000 corresponds to 100% efficiency. A high efficiency identifies that the MySQL server was able to complete the page reads from the memory; a lower efficiency identifies that the MySQL server needed to read from disk more often to complete the page reads. A good buffer pool hit rate is based on the workload of the system; however, an efficiency of 95% or less should be researched to improve efficiency.

- The ROW OPERATIONS section of the report displays activity on the row basics and some miscellaneous system information. These values can be used to create graphs that show the InnoDB load over time.
 - How many rows have been inserted since system startup? _____
 - How many rows have been updated since system startup? _____
 - How many rows have been deleted since system startup? _____
 - How many rows have been read since system startup? _____

Note: The number of rows can be an inefficient means to track the load on the MySQL server due to the potential for different size rows being accessed (a 2 byte row is cheaper than accessing a 2 MB blob). However, the number of rows is more efficient than the number of queries (which can have an even greater disparity).

9. In the first terminal window (step 1), quit the mysql client.

10. Close out the second terminal window (step 7).

Solutions 6-2: SHOW ENGINE INNODB STATUS

Tasks

Note: These solutions are sample solutions and are not expected to be the exact output you see in your execution of the steps.

1. Open a terminal window and restart the MySQL server as root.

```
sys> /etc/init.d/mysql restart
```

Note: You can use the existing terminal window from the previous practice. However, at this point, you should have only one terminal window open to minimize confusion as more terminal windows are opened up.

2. Start the mysql client and select the many_tables database.

```
sys> mysql -uroot -poracle  
mysql> USE many_tables;
```

3. Execute the following statement in the mysql client to review the structure of the create_city_huge MySQL stored procedure:

```
mysql> SELECT ROUTINE_NAME, ROUTINE_DEFINITION FROM  
-> INFORMATION_SCHEMA.ROUTINES  
-> WHERE ROUTINE_NAME LIKE 'create_city_huge'\G  
*****  
ROUTINE_NAME: create_city_huge  
ROUTINE_DEFINITION: BEGIN  
    DROP TABLE IF EXISTS city_huge;  
    CREATE TABLE city_huge LIKE City;  
    ALTER TABLE city_huge ADD INDEX (CountryCode), ADD INDEX  
(Name), ADD INDEX (Population);  
    CALL add_data_to_city_huge(size);  
END  
1 row in set (0.00 sec)
```

- Reading the script definition, what is the purpose of this script?

This script creates a table called city_huge based on the City table structure. The table is then altered to add indexes on the CountryCode, Name, and Population fields. The script then calls another script called add_data_to_city_huge and passes on a value passed into this script.

4. Execute the following statement in the `mysql` client to review the structure of the `add_data_to_city_huge` MySQL stored procedure:

```
mysql> SELECT ROUTINE_NAME, ROUTINE_DEFINITION FROM
-> INFORMATION_SCHEMA.ROUTINES
-> WHERE ROUTINE_NAME LIKE 'add_data_to_city_huge'\G
***** 1. row *****
ROUTINE_NAME: add_data_to_city_huge
ROUTINE_DEFINITION: BEGIN
    DECLARE loops INT default 0;
    WHILE (loops < size) DO
        INSERT INTO city_huge (Name, CountryCode, District,
Population)
        SELECT Name, CountryCode, District, Population FROM City;
        SET loops = loops + 1;
    END WHILE;
END
1 row in set (0.00 sec)
```

- Reading the script definition, what is the purpose of this script?

This script inserts into the `city_huge` table all the records from the `City` table.
This process repeats the number of times identified by the `size` value that was
passed from the `create_city_huge` script.

5. Execute the following statement in the `mysql` client:

```
mysql> CALL create_city_huge(100);
Query OK, 4079 rows affected (16.16 sec)
```

- This execution of the `create_city_huge` MySQL stored procedure creates a table and inserts 400,000 records into it.

6. Modify the `city_huge` table created in step 5 to use the InnoDB storage engine.

```
mysql> ALTER TABLE city_huge ENGINE=INNODB;
Query OK, 407900 rows affected (28.43 sec)
Records: 407900  Duplicates: 0  Warnings: 0
```

7. Open a second terminal window and execute the following command as `root` to create a load against the MySQL server and the `city_huge` table created in step 5.

```
sys> mysqlslap -uroot -poracle \
> -q /stage/scripts/innodb_query.sql \
> --create-schema=many_tables -i 20 -c 20
Benchmark
Average number of seconds to run all queries: 8.288 seconds
Minimum number of seconds to run all queries: 6.311 seconds
Maximum number of seconds to run all queries: 9.003 seconds
Number of clients running queries: 10
Average number of queries per client: 3745
```

Note: This script runs for approximately 6 minutes. This is necessary to have a good sampling of data for the next few steps and to provide a more realistic InnoDB status report. Continue on with the remaining steps while this `mysqlslap` command is executing.

8. About 30 seconds after the `mysqlslap` command has begun executing in step 7, return to the first terminal window opened in step1 and execute the following command to display the InnoDB status report:

```
mysql> SHOW ENGINE INNODB STATUS\G
***** 1. row *****
  Type: InnoDB
  Name:
Status:
=====
110204 16:22:35 INNODB MONITOR OUTPUT
=====
Per second averages calculated from the last 46 seconds
```

- How many seconds were used to calculate the *per second averages* for the report?

46 seconds

Note: This is first line in the InnoDB status report of consequence and can be found directly under the `INNODB MONITOR OUTPUT` heading. Due to the fact that many of the values are based on per second, it is important to have a sampling of data of at least 20-30 seconds. Anything less is unusable and should be discarded.

```
-----
BACKGROUND THREAD
-----
srv_master_thread loops: 11 1_second, 11 sleeps, 1 10_second, 2 background, 2
flush
srv_master_thread log flush and writes: 15
-----
SEMAPHORES
-----
OS WAIT ARRAY INFO: reservation count 6403, signal count 3048
--Thread 2403597200 has waited at /export/home/pb2/build/sb_0-2629600-
1291400836.29/rpm/BUILD/mysql-5.5.8/mysql-5.5.8/storage/innobase/log/log0log.c
line 2049 for 0.00 seconds the semaphore:
S-lock on RW-latch at 0x99f1468 created in file /export/home/pb2/build/sb_0-
2629600-1291400836.29/rpm/BUILD/mysql-5.5.8/mysql-
5.5.8/storage/innobase/log/log0log.c line 832
a writer (thread id 2403597200) has reserved it in mode exclusive
number of readers 0, waiters flag 1, lock_word: 0
Last time read locked in file /export/home/pb2/build/sb_0-2629600-
1291400836.29/rpm/BUILD/mysql-5.5.8/mysql-5.5.8/storage/innobase/log/log0log.c
line 2049
Last time write locked in file /export/home/pb2/build/sb_0-2629600-
1291400836.29/rpm/BUILD/mysql-5.5.8/mysql-5.5.8/storage/innobase/log/log0log.c
line 1846
Mutex spin waits 37582, rounds 597068, OS waits 4631
RW-shared spins 999, rounds 27530, OS waits 450
RW-excl spins 1224, rounds 37292, OS waits 552
Spin rounds per wait: 15.89 mutex, 27.56 RW-shared, 30.47 RW-excl
```

- The SEMAPHORES section of the output assists you with determining how well MySQL is handling context switching. `reservation count` and `signal count` displays how often InnoDB uses the internal sync array. The ratios between these two values represent how frequently the InnoDB storage engine needs to use the O/S wait functions.
 - What is the reservation count value? 6403
 - What is the signal count value? 3048

- Spin locks are less costly than O/S waits, but still can be consuming a large amount of CPU resources.
- What is the Mutex spin waits value? 37582
- What is the Mutex spin rounds value? 597068

Note: Spin waits and spin rounds identify a problem if both are high because it is a sign that significant CPU resources are being wasted. Reducing the system variable `innodb_sync_spin_loops` can minimize the amount of CPU wasted by doing spin loops. However, reducing this setting can increase the amount of CPU resources that is used for context switching.

```
-----
TRANSACTIONS
-----
Trx id counter FA5E470
Purge done for trx's n:o < FA5E427 undo n:o < 0
History list length 142
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 0, not started, process no 13115, OS thread id 2409286544
MySQL thread id 1, query id 59824 localhost root
SHOW ENGINE INNODB STATUS

---TRANSACTION FA5E46F, ACTIVE 0 sec, process no 13115, OS thread id
2408082320 setting auto-inc lock
mysql tables in use 1, locked 1
LOCK WAIT 1 lock struct(s), heap size 320, 0 row lock(s)
MySQL thread id 7, query id 59800 localhost root Sending data
INSERT INTO city_huge (Name, CountryCode, District, Population) SELECT Name,
CountryCode, District, Population FROM City LIMIT 1
----- TRX HAS BEEN WAITING 0 SEC FOR THIS LOCK TO BE GRANTED:
TABLE LOCK table `many_tables`.`city_huge` trx id FA5E46F lock mode AUTO-INC
waiting
...
---TRANSACTION FA5E439, ACTIVE (PREPARED) 0 sec, process no 13115, OS thread
id 2406476688 preparing
7 lock struct(s), heap size 1024, 2 row lock(s), undo log entries 2
MySQL thread id 15, query id 59813 localhost root
COMMIT
Trx read view will not see trx with id >= FA5E470, sees < FA5E444
```

- The TRANSACTIONS section of the output provides helpful information determining if your application has lock contention and the reasons for transaction deadlocks. The Trx id counter is a hexadecimal number that is incremented for each transaction.
 - What is the Trx id counter value? FA5E470 (or 262,530,160)
- The Purge done for trx's n:0 is the hexadecimal number of the transaction that was last purged. The undo n:0 is the hexadecimal number of the transaction currently processing. If the value is 0, InnoDB is currently not executing any transactions.
 - What is the Purge done for trx's n:0 value? FA5E427 (or 262,530,087)
 - What is the undo n:0 value? 0

Note: InnoDB only purges old versions of transactions when an event takes place to let InnoDB know that it is completed. Uncommitted transactions that are old or stale may block the purge process. This can lead to system resources being consumed. By comparing the Trx id counter and Purge done for trx's n:0 values, you can spot uncommitted transactions taking up resources.

- The History list length is the number of unpurged transactions in undo space. This value is increased as updated transactions are committed and decreased as purges take place. The isolation mode being used for the transactions has a large effect on how large this list can become.

- What is the History list length value? 142

Note: A large value in the History list length can indicate that the undo space contains a large amount of “garbage” that can have a degraded effect on performance and an increase in disk usage.

- The LIST OF TRANSACTIONS FOR EACH SESSION: displays all the transactions (if the number of transactions is relatively small) or a portion of the transactions (if the number of transactions is large). The information contained in the transaction lines can provide you with detailed information on the state of the transaction.

- What is the status of the last transaction that appears in this list?

ACTIVE (PREPARED) 0 sec

- What is the value of the OS thread id for this transaction? 2406476688
- Which MySQL thread is running this transaction? 15
- What is the value of the query id for this transaction? 59813
- Who is the user that this transaction is running under? root

Note: The OS thread id, MySQL thread id, and query id values can be used to identify the query for administration and troubleshooting purposes.

```
-----
FILE I/O
-----
I/O thread 0 state: waiting for completed aio requests (insert buffer thread)
I/O thread 1 state: complete io for log (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: waiting for completed aio requests (write thread)
Pending normal aio reads: 0 [0, 0, 0, 0] , aio writes: 0 [0, 0, 0, 0] ,
ibuf aio reads: 0, log i/o's: 0, sync i/o's: 0
Pending flushes (fsync) log: 2; buffer pool: 0
2940 OS file reads, 10271 OS file writes, 6783 OS fsyncs
63.91 reads/s, 17076 avg bytes/read, 223.28 writes/s, 147.45 fsyncs/s
```

- 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).
- How many transactions are waiting for fsync (Pending flushes (fsync) log)?

2

- How many transactions are in the buffer pool? 0
- Note:** Calling `fsync()` on modified files is used by InnoDB to ensure data makes it to the disk (simply passing it to the OS cache is not enough). Consistently high values for Pending flushes or the buffer pool can be an indication of an I/O bound workload.
- The total number of I/O operations is displayed along with the computed averages for those operations. These values can be used for monitoring and graphing to evaluate trends in your systems use.
 - What is the value of the OS file reads? 2940
 - What is the value of the OS file writes? 10271
 - What is the value of the OS fsyncs? 6783
 - What is the average number of reads per second for this reporting period?
63.91
 - What is the average bytes read per second for this reporting period?
17076
 - What is the average number of writes per second for this reporting period?
223.28
 - What is the average number of `fsync()` operations for this reporting period?
147.45

```
-----
INSERT BUFFER AND ADAPTIVE HASH INDEX
-----
Ibuf: size 102, free list len 55, seg size 158, 41 merges
merged operations:
  insert 727, delete mark 466, delete 247
discarded operations:
  insert 0, delete mark 0, delete 0
Hash table size 553193, node heap has 20 buffer(s)
243.02 hash searches/s, 2347.17 non-hash searches/s
```

- The INSERT BUFFER AND ADAPTIVE HASH INDEX section of the report displays the status of the insert buffer (`Ibuf`) to include the segment size and the free list along with the any records that are located in the insert buffer. This is followed by how many inserts were done in the insert buffer, how many records were merged and how many merges took place.
 - For the merged operations, how many `insert`'s took place during this reporting period? 727
 - For the merged operations, how many `delete mark`'s took place during this reporting period? 466

Note: When a page is in the buffer pool, it is updated directly. When a page is loaded to the buffer pool, any buffered changes are merged to it, so that users never see unmerged changes. The insert buffer bitmap keeps track of the available space on pages and prevents overflows when buffering inserts. Delete-marking records can always be buffered, because the flag is updated in place.

- Calculating the ratio of delete mark's and insert's, what is the insert buffer efficiency for this reporting period?

466/727 = 0.641 or 64% effective

- The last section displays the hash table size, number of used cells and number of buffers used by adaptive hash index.
- What is the average number of *hash* searches that have taken place during this reporting period?

243.02

- What is the average number of *non-hash* searches that have taken place during this reporting period?

2347.17

- The number of hash index lookups and number of non-hash index lookups indicate the hash index efficiency. What is the hash index efficiency for this reporting period?

243.02/2347.17 = 0.104 or 10.4%

```
---
LOG
---
Log sequence number 16006290052
Log flushed up to 16006280142
Last checkpoint at 15998752796
1 pending log writes, 1 pending chkp writes
6708 log i/o's done, 145.82 log i/o's/second
```

- The LOG section of the report displays the current log sequence number (which is the amount of bytes InnoDB has written in log files since system tablespace creation), up to which point logs have been flushed, and when the last checkpoint was performed.
 - What is the value of the Log sequence number? 16006290052
 - What is the value of the Log flushed up to? 16006280142
 - What is the percentage difference between these two values?

$(16006290052 - 16006280142) / 16006290052 = 0.00000062$ or insignificant

Note: A difference that is greater than 30% indicates a potential problem with the `innodb_log_buffer_size` system variable. Increasing the size of this system variable can reduce this difference.

- How log writes are pending? 1
- How many checkpoint writes are pending? 1
- How many log i/o's have taken place during this reporting period? 6708
- What is the average number of log i/o's per second that have taken place during this reporting period?

145.82

Note: The log writes performance is primarily based on the `innodb_flush_logs_at_trx_commit` system variable. This system variable controls when the log buffer is written to and logs files are flushed. Changing this value can improve the performance of your log i/o but raises the possibility of losing up to one second of transactions.

```
--  
-----  
BUFFER POOL AND MEMORY  
-----  
Total memory allocated 136675328; in additional pool allocated 0  
Dictionary memory allocated 25568  
Buffer pool size     8191  
Free buffers        4777  
Database pages      3394  
Old database pages  1271  
Modified db pages   2303  
Pending reads       0  
Pending writes: LRU 0, flush list 0, single page 0  
Pages made young 4, not young 0  
0.09 youngs/s, 0.00 non-youngs/s  
Pages read 2929, created 465, written 3500  
63.67 reads/s, 10.11 creates/s, 76.09 writes/s  
Buffer pool hit rate 997 / 1000, young-making rate 0 / 1000 not 0 / 1000  
Pages read ahead 0.00/s, evicted without access 0.00/s  
LRU len: 3394, unzip_LRU len: 0  
I/O sum[0]:cur[35], unzip sum[0]:cur[0]
```

- The BUFFER POOL AND MEMORY section of the report displays the status of the information to help you determine if the buffer pool is sized properly for your system.
 - What is the Total memory allocated for InnoDB? 136675328 or 137 MB
 - What is the value of the Buffer pool size? 8191

Note: The buffer pool size listed in the `SHOW ENGINE INNODB STATUS` report is presented in pages of 16,384 bytes. For an `innodb_buffer_pool_size` of 134217728 would be equivalent to a buffer pool size of 8192.

- What is the value of the Free buffers? 4777
- Note:** If your system continually has a high percentage of pages free (in comparison to the buffer pool size), it is a sign that your active database size may be smaller than the allocated buffer pool size. In this case, you can consider decreasing the `innodb_buffer_pool_size` system variable to redeem system resources for other processes.
- What is the Buffer pool hit rate? 997/1000
- Note:** A buffer pool hit rate of 1000/1000 corresponds to 100% efficiency. A high efficiency identifies that the MySQL server was able to complete the page reads from the memory; a lower efficiency identifies that the MySQL server needed to read from disk more often to complete the page reads. A good buffer pool hit rate is based on the workload of the system; however, an efficiency of 95% or less should be researched to improve efficiency.

```
-----  
ROW OPERATIONS  
-----  
0 queries inside InnoDB, 0 queries in queue  
12 read views open inside InnoDB  
Main thread process no. 13115, id 2403597200, state: making checkpoint  
Number of rows inserted 3985, updated 11964, deleted 0, read 39865  
86.63 inserts/s, 260.08 updates/s, 0.00 deletes/s, 866.61 reads/s  
-----  
END OF INNODB MONITOR OUTPUT  
=====  
1 row in set (0.00 sec)
```

- The ROW OPERATIONS section of the report displays activity on the row basics and some miscellaneous system information. These values can be used to create graphs that show the InnoDB load over time.
 - How many rows have been inserted since system startup? 3985
 - How many rows have been updated since system startup? 11964
 - How many rows have been deleted since system startup? 0
 - How many rows have been read since system startup? 39865

Note: The number of rows can be an inefficient means to track the load on the MySQL server due to the potential for different size rows being accessed (a 2 byte row is cheaper than accessing a 2 MB blog). However, the number of rows is more efficient than the number of queries (which can have an even greater disparity).

9. In the first terminal window (step 1), quit the mysql client.

```
mysql> exit;
```

10. Close out the second terminal window (step 7).

Practice 6-3: InnoDB Monitors

Overview

In this practice, you evaluate the output of the additional InnoDB monitors available. To accomplish this objective, you do the following:

- Make a backup copy of the mysql error log, delete the error log, and restart the server.
- Create a table that is the trigger to output the InnoDB data dictionary information on the tablespace to the mysql error log.
 - After creating the table, you wait a specified amount of time and then delete the table. This stops the output of the InnoDB data dictionary information on the tablespace being printed to the error log.
 - Open the error log and answer questions related to the tablespace output.
- Make a backup copy of the mysql error log, delete the error log, and restart the server.
- Create a table that is the trigger to output the contents of the InnoDB internal data dictionary.
 - After creating the table, you wait a specified amount of time and then delete the table. This stops the output of the InnoDB internal data dictionary being printed to the error log.
 - Open the error log and answer questions related to the tablespace output.
- Execute and answer questions related to the table output.

Assumptions

- The MySQL server is installed and running.
- The `many_tables` database is installed.

Duration

This practice should take 20 minutes to complete.

Tasks

1. Open a terminal window and list all `*.err` files in the `/var/lib/mysql` directory.
 - What is the name of the file that was listed? _____

Note: You can use the existing terminal window from the previous practice. However, at this point, you should have only one terminal window open to minimize confusion as more terminal windows are opened up.
2. As `root`, make a copy of the error log listed in step 1.
3. As `root`, remove the file listed in step 1.
4. Restart the MySQL server as `root`.
5. In a second terminal window, start the `mysql` client and use the `many_tables` database.
6. Execute the following statement in the `mysql` client to create an InnoDB table called `innodb_monitor` with one integer column:

```
mysql> CREATE TABLE innodb_monitor (A INT) ENGINE=INNODB;
```

- By creating this table, the MySQL server writes the output of the `SHOW ENGINE INNODB STATUS` command to the mysql error log.

7. Wait one minute and then using the `mysql` client delete the `innodb_monitor` table.

8. In the first terminal window, review the contents of the mysql error log.
 - How many times did the SHOW ENGINE INNODB STATUS report output appear in the mysql error log?

9. As root, make another copy of the error log listed in step 1.

10. As root, remove the file listed in step 1.

11. Restart the MySQL server as root.

12. In the second terminal window (with the mysql client open), execute the following statement to create an InnoDB table called innodb_tablespace_monitor with one integer column:

```
mysql> CREATE TABLE innodb_tablespace_monitor \
> (A INT) ENGINE=INNODB;
Query OK, 0 rows affected (0.00 sec)
```

- By creating this table, the MySQL server writes the data dictionary information on the tablespace to the mysql error log.

13. Wait one minute and then using the mysql client delete the innodb_tablespace_monitor table.

14. In the first terminal window, review the contents of the mysql error log.

- Review the first SEGMENT output of the INNODB TABLESPACE MONITOR OUTPUT and answer the following questions:

- What is the identifier for the segment (include id, space, and page)?

- What are the values for the res and used outputs?

Note: res means how many pages is allocated for the segment and used means how many of these pages are actually used by the segment at the moment.

- What are the values for the full ext and fragm pages outputs?

Note: full ext identifies the number of extents that are completely used and fragm pages identifies the number of pages that InnoDB first allocates for the segment.

- What are the values for the free extents and not full extents outputs?

Note: free extents identifies the number of extents that are allocated for the segment, but where none of the pages are used. not full extents identifies the number of extents that are allocated for the segment.

- What is the value for the pages output? _____

Note: pages identifies how many pages are used within the extents for the segment.

15. As root, make another copy of the error log listed in step 1.

16. As `root`, remove the file listed in step 1.
17. Restart the MySQL server as `root`.
18. In the second terminal window (with the `mysql` client open), execute the following statement to create an InnoDB table called `innodb_table_monitor` with one integer column:

```
mysql> CREATE TABLE innodb_table_monitor \
> (A INT) ENGINE=INNODB;
Query OK, 0 rows affected (0.00 sec)
```

- By creating this table, the MySQL server writes the data dictionary information on the tables to the `mysql` error log.
19. Wait one minute and then using the `mysql` client delete the `innodb_table_monitor` table.
20. In the first terminal window, review the contents of the `mysql` error log.

- Locate the table `sakila/address` in the INNODB TABLE MONITOR OUTPUT and answer the following questions:

- How many columns does the table contain? _____

Note: The table was created with eight columns and InnoDB added three internal columns: `DB_ROW_ID` (row ID), `DB_TRX_ID` (transaction ID), and `DB_ROLL_PTR` (a pointer to the rollback/undo data).

- How many indexes does the table contain?

- _____
How many approximate records does the table contain?

- _____
Under `COLUMNS` field, what information about the `address_id` column can be learned?

- _____
Under `COLUMNS` field, what information about the `address` column can be learned?

- _____
What is the `fields` value for the `PRIMARY` index?

Note: The first value in the `fields` output identifies the user-defined indexes. The second value includes all the fields associated with the index, including those added internally.

- What is the `uniq` value for the PRIMARY index?

Note: The `uniq` output identifies the number of leading fields that are enough to determine index values uniquely.

- What is the `type` value for the PRIMARY index?

Note: The `type` output identifies the index type for the index. This is a bit field with four possible values: 1 identifies a clustered index, 2 identifies a unique index, 3 identifies a clustered index which contains unique values, and 0 identifies neither a clustered or unique index.

- What is the `appr.key values` value for the PRIMARY index?

Note: The `appr. key values` identifies the approximate index cardinality.

- What is the value for the `leaf pages` and `size pages` output for the PRIMARY index?

Note: The `leaf pages` output identifies the approximate number of leaf pages that the index in the index and `size pages` identifies the approximate total number of pages in the index.

- What is the `FIELDS` output for the PRIMARY index?

Note: The `FIELDS` output identifies the name(s) of the columns in the index. The values `DB_TRX_ID` and `DB_ROLL_PTR` are always added internally to the clustered index, following the fields that comprise the primary key. The remaining output displays the remainder of the fields contained within the index.

- How many `FOREIGN KEY` constraints are associated with the table?

Note: The `FOREIGN KEY` definitions that apply to the table appear whether the table is a referencing or referenced table.

21. Quit the `mysql` client.

Solutions 6-3: InnoDB Monitors

Tasks

Note: These solutions are sample solutions and are not expected to be the exact output you see in your execution of the steps.

1. Open a terminal window and list all *.err files in the /var/lib/mysql directory.

```
sys> ls /var/lib/mysql/*.err  
/var/lib/mysql/EDTDR14P2.err
```

- What is the name of the file that was listed? EDTDR14P2.err

Note: You can use the existing terminal window from the previous practice. However, at this point, you should have only one terminal window open to minimize confusion as more terminal windows are opened up.

2. As root, make a copy of the error log listed in step 1.

```
sys> cp /var/lib/mysql/EDTDR14P2.err \  
> /var/lib/mysql/EDTDR14P2.err-bu1
```

3. As root, remove the file listed in step 1.

```
sys> rm -rf /var/lib/mysql/EDTDR14P2.err
```

4. Restart the MySQL server as root.

```
sys> /etc/init.d/mysql restart
```

5. In a second terminal window, start the mysql client and use the many_tables database.

```
sys> mysql -uroot -poracle  
mysql> USE many_tables;
```

6. Execute the following statement in the mysql client to create an InnoDB table called innodb_monitor with one integer column:

```
mysql> CREATE TABLE innodb_monitor (A INT) ENGINE=INNODB;  
Query OK, 0 rows affected (0.00 sec)
```

- By creating this table, the MySQL server writes the output of the SHOW ENGINE INNODB STATUS command to the mysql error log.

7. Wait one minute and then using the mysql client delete the innodb_monitor table.

```
mysql> DROP TABLE innodb_monitor;  
Query OK, 0 rows affected (0.00 sec)
```

8. In the first terminal window, review the contents of the mysql error log.

```
sys> more EDTDR14P2.err
```

- How many times did the SHOW ENGINE INNODB STATUS report output appear in the mysql error log?

3

9. As root, make another copy of the error log listed in step 1.

```
sys> cp /var/lib/mysql/EDTDR14P2.err \  
> /var/lib/mysql/EDTDR14P2.err-bu2
```

10. As root, remove the file listed in step 1.

```
sys> rm -rf /var/lib/mysql/EDTDR14P2.err
```

11. Restart the MySQL server as root.

```
sys> /etc/init.d/mysql restart
```

12. In the second terminal window (with the mysql client open), execute the following statement to create an InnoDB table called innodb_tablespace_monitor with one integer column:

```
mysql> CREATE TABLE innodb_tablespace_monitor \
> (A INT) ENGINE=INNODB;
Query OK, 0 rows affected (0.00 sec)
```

- By creating this table, the MySQL server writes the data dictionary information on the tablespace to the mysql error log.

13. Wait one minute and then using the mysql client delete the innodb_tablespace_monitor table.

```
mysql> DROP TABLE innodb_tablespace_monitor;
Query OK, 0 rows affected (0.00 sec)
```

14. In the first terminal window, review the contents of the mysql error log.

```
sys> more EDTDR14P2.err
=====
110207  0:52:19  INNODB TABLESPACE MONITOR OUTPUT
=====
FILE SPACE INFO: id 0
size 160896, free limit 159104, free extents 1945
not full frag extents 50: used pages 1993, full frag extents 11
first seg id not used 838792
SEGMENT id 1 space 0; page 2; res 450 used 390; full ext 6
fragm pages 2; free extents 0; not full extents 1: pages 4
...
NUMBER of file segments: 1193
Validating tablespace
Validation ok
-----
END OF INNODB TABLESPACE MONITOR OUTPUT
=====
```

- Review the first SEGMENT output of the INNODB TABLESPACE MONITOR OUTPUT and answer the following questions:

- What is the identifier for the segment (include id, space, and page)?

id 1 space 0; page 2

- What are the values for the res and used outputs? 450 and 390

Note: res means how many pages is allocated for the segment and used means how many of these pages are actually used by the segment at the moment.

- What are the values for the full ext and fragm pages outputs? 6 and 2
Note: full ext identifies the number of extents that are completely used and fragm pages identifies the number of pages that InnoDB first allocates for the segment.
- What are the values for the free extents and not full extents outputs? 0 and 1
Note: free extents identifies the number of extents that are allocated for the segment, but where none of the pages are used. not full extents identifies the number of extents that are allocated for the segment.
- What is the value for the pages output? 4
Note: pages identifies how many pages are used within the extents for the segment.

15. As root, make another copy of the error file listed in step 1.

```
sys> cp /var/lib/mysql/EDTDR14P2.err \
> /var/lib/mysql/EDTDR14P2.err-bu3
```

16. As root, remove the file listed in step 1.

```
sys> rm -rf /var/lib/mysql/EDTDR14P2.err
```

17. Restart the MySQL server as root.

```
sys> /etc/init.d/mysql restart
```

18. In the second terminal window (with the mysql client open), execute the following statement to create an InnoDB table called innodb_table_monitor with one integer column:

```
mysql> CREATE TABLE innodb_table_monitor \
> (A INT) ENGINE=INNODB;
Query OK, 0 rows affected (0.00 sec)
```

- By creating this table, the MySQL server writes the data dictionary information on the tables to the mysql error log.

19. Wait one minute and then using the mysql client delete the innodb_table_monitor table.

```
mysql> DROP TABLE innodb_table_monitor;
Query OK, 0 rows affected (0.00 sec)
```

20. In the first terminal window, review the contents of the mysql error log.

```
sys> more EDTDR14P2.err
=====
110207  1:29:18 INNODB TABLE MONITOR OUTPUT
=====
...
-----
TABLE: name sakila/address, id 30473, flags 1, columns 11, indexes 2, appr.rows 549
  COLUMNS: address_id: DATA_INT DATA_UNSIGNED DATA_BINARY_TYPE DATA_NOT_NULL len 2;
address: DATA_VARTEXT DATA_NOT_NULL len 150; address2: DATA_VARTEXT len 150; district:
DATA_VARTEXT DATA_NOT_NULL len 60; city_id: DATA_INT DATA_UNSIGNED DATA_BINARY_TYPE
DATA_NOT_NULL len 2; postal_code: DATA_VARTEXT len 30; phone: DATA_VARTEXT
DATA_NOT_NULL len 60; last_update: DATA_INT DATA_UNSIGNED DATA_BINARY_TYPE DATA_NOT_NULL
len 4; DB_ROW_ID: DATA_SYS prtype 256 len 6; DB_TRX_ID: DATA_SYS prtype 257 len 6;
DB_ROLL_PTR: DATA_SYS prtype 258 len 7;
  INDEX: name PRIMARY, id 180694, fields 1/10, uniq 1, type 3
    root page 8558, appr.key vals 549, leaf pages 4, size pages 5
    FIELDS: address_id DB_TRX_ID DB_ROLL_PTR address address2 district city_id
postal_code phone last_update
  INDEX: name idx_fk_city_id, id 180695, fields 1/2, uniq 2, type 0
    root page 8559, appr.key vals 599, leaf pages 1, size pages 1
    FIELDS: city_id address_id
  FOREIGN KEY CONSTRAINT sakila/fk_address_city: sakila/address ( city_id )
    REFERENCES sakila/city ( city_id )
  FOREIGN KEY CONSTRAINT sakila/fk_customer_address: sakila/customer ( address_id )
    REFERENCES sakila/address ( address_id )
  FOREIGN KEY CONSTRAINT sakila/fk_staff_address: sakila/staff ( address_id )
    REFERENCES sakila/address ( address_id )
  FOREIGN KEY CONSTRAINT sakila/fk_store_address: sakila/store ( address_id )
    REFERENCES sakila/address ( address_id )
-----
...
END OF INNODB TABLE MONITOR OUTPUT
=====
```

- Locate the table `sakila/address` in the INNODB TABLE MONITOR OUTPUT and answer the following questions:

- How many columns does the table contain? 11

Note: The table was created with eight columns and InnoDB added three internal columns: `DB_ROW_ID` (row ID), `DB_TRX_ID` (transaction ID), and `DB_ROLL_PTR` (a pointer to the rollback/undo data).

- How many indexes does the table contain? 2
- How many approximate records does the table contain? 549
- Under `COLUMNS` field, what information about the `address_id` column can be learned?

Internally this column is seen as a raw (binary) not null unsigned integer with length of 2 bytes.

- Under `COLUMNS` field, what information about the `address` column can be learned?

Internally this column is seen as a not null variable string with a length of 150 bytes.

- What is the `fields` value for the PRIMARY index? 1/10
Note: The first value in the `fields` output identifies the user-defined indexes. The second value includes all the fields associated with the index, including those added internally.
- What is the `uniq` value for the PRIMARY index? 1
Note: The `uniq` output identifies the number of leading fields that are enough to determine index values uniquely.
- What is the `type` value for the PRIMARY index? 3
Note: The `type` output identifies the index type for the index. This is a bit field with four possible values: 1 identifies a clustered index, 2 identifies a unique index, 3 identifies a clustered index which contains unique values, and 0 identifies neither a clustered or unique index.
- What is the `appr.key values` value for the PRIMARY index? 549
Note: The `appr. key values` identifies the approximate index cardinality.
- What is the value for the `leaf pages` and `size pages` output for the PRIMARY index?
4 and 5
Note: The `leaf pages` output identifies the approximate number of leaf pages that the index in the index and `size pages` identifies the approximate total number of pages in the index.
- What is the `FIELDS` output for the PRIMARY index?
address id DB TRX ID DB ROLL PTR address address2
district city id postal code phone last update
Note: The `FIELDS` output identifies the name(s) of the columns in the index. The values `DB_TRX_ID` and `DB_ROLL_PTR` are always added internally to the clustered index, following the fields that comprise the primary key. The remaining output displays the remainder of the fields contained within the index.
- How many FOREIGN KEY constraints are associated with the table? 4
Note: The FOREIGN KEY definitions that apply to the table appear whether the table is a referencing or referenced table.

21. Quit the mysql client.

```
mysql> exit;
```

Practice 6-4: InnoDB Settings

Overview

In this practice, you modify innodb server settings and evaluate the effect on scripts that place a load on the MySQL server. To accomplish this objective, you do the following:

- View the current innodb server settings.
- Using the `mysqlslap` application, create a load against the MySQL server.
 - Modify the `innodb_flush_log_at_trx_commit` system variable between runs and make note of any changes in performance.
- Using the `mysqlslap` application, create a load against the MySQL server.
 - Modify the `innodb_buffer_pool_size` system variable between runs and make note of any changes in performance.
 - In addition, while the `mysqlslap` application is running, view the amount of virtual and resident memory that the MySQL server is consuming.

Assumptions

- The MySQL server is installed and running.
- The `many_tables` database is installed.
- The `mysqlslap` application is installed.
- The `innodb_query.sql` and `innodb_buffer.sql` files are located in the `/stage/scripts` directory.

Duration

This practice should take 20 minutes to complete.

Tasks

Note: A table has been provided at the end of the steps for this practice for you to fill in the values collected during the running of the steps. Entering the data from the steps into this table can assist you in comparing the values collected.

1. Open a terminal window and use the `mysql` client to list all the system variables that begin with the word “`innodb`”.

Note: You can use the existing terminal window from the previous practice. However, at this point, you should have only one terminal window open to minimize confusion as more terminal windows are opened up.

- What is the current setting for the `innodb_additional_mem_pool` system variable?

- What is the current setting for the `innodb_autoextend_increment` system variable?

- What is the current setting for the `innodb_buffer_pool_size` system variable?

 - What is the current setting for the `innodb_file_per_table` system variable?

 - What is the current setting for the `innodb_flush_log_at_trx_commit` system variable?

 - What is the current setting for the `innodb_flush_method` system variable?

 - What is the current setting for the `innodb_log_files_in_group` system variable?

 - What is the current setting for the `innodb_log_file_size` system variable?

 - What is the current setting for the `innodb_thread_concurrency` system variable?

2. Using the `mysqlslap` application, execute the following command as `root` to create a load against the MySQL server and the `city_huge` table created in an earlier practice:
- ```
sys> mysqlslap -uroot -poracle \
> -q /stage/scripts/innodb_query.sql \
> --create-schema=many_tables -i 4 -c 10
```
- What is the value for the average number of seconds to run all queries output?  
\_\_\_\_\_
3. Using the `mysql` client, set the `innodb_flush_log_at_trx_commit` system variable to 2.
- Note:** Setting the `innodb_flush_log_at_trx_commit` system variable to 2 can result in an improved performance but at the cost of losing one second or more of transactions on a system crash or power down.
4. Execute the command from step 2 as `root`.
- What is the value for the average number of seconds to run all queries output?  
\_\_\_\_\_

5. Using the mysql client, set the `innodb_flush_log_at_trx_commit` system variable to 0.

**Note:** Setting the `innodb_flush_log_at_trx_commit` system variable to 0 can result in a best performance but at the cost of losing up to one second of transactions on any mysqld process crash.

6. Execute the command from step 2 as root.
  - What is the value for the average number of seconds to run all queries output?

- 
7. Restart the MySQL server as root and set the `innodb_buffer_pool_size` server setting to 0.

**Note:** The minimum value that the `innodb_buffer_pool_size` can be set to is 5242880. Setting this system variable to 0 will cause the MySQL server to assign the minimum value.

8. Using the `mysqlslap` application, execute the following command as root to create a load against the MySQL server using a predefined script:

```
mysqlslap -uroot -poracle \
> -q /stage/scripts/innodb_buffer.sql -i 5 -c 1
```

- While this is executing, open a second terminal window and complete the task in the next step.

**Note:** This command runs for approximately 4 minutes.

9. In a separate terminal window, execute the following command to obtain a dynamic real-time view of the mysqld application:

```
sys> top -n 8 | grep mysqld
```

- The columns of most interest to you for this practice include:
  - Column 5: VIRT – The total amount of virtual memory used by the task. This value includes the resident memory the task is using along with the memory used by swapping out pages.
  - Column 6: RES – The non-swapped physical memory used by the task.
- What is the approximate value of the virtual memory (VIRT column) of the `top` output on your system?
  
- What is the approximate value of the resident memory (RES Column) of the `top` output on your system?

- 
10. Return to the first terminal window and view the results of the `mysqlslap` application.

- What is the value for the average number of seconds to run all queries output?
-

11. Restart the MySQL server as `root` and set the `innodb_buffer_pool_size` server setting to 12M.
  12. Execute the command from step 9 as `root`.
    - While this is executing, go to the second terminal window opened in step 10 and complete the task in the next step.
- Note:** This command runs for approximately one minute.
13. In a separate terminal window, execute the following command to obtain a dynamic real-time view of the `mysqld` application:

```
sys> top -n 8 | grep mysqld
```

    - What is the approximate value of the virtual memory (VIRT column) of the `top` output on your system?
  
    - What is the approximate value of the resident memory (RES Column) of the `top` output on your system?
  14. Return to the first terminal window and view the results of the `mysqlslap` application.
    - What is the value for the average number of seconds to run all queries output?
  15. Restart the MySQL server as `root` and set the `innodb_buffer_pool_size` server setting to 24M.
  16. Execute the command from step 9 as `root`.
    - While this is executing, go to the second terminal window opened in step 10 and complete the task in the next step.
- Note:** This command runs for approximately 30 seconds.
17. In a separate terminal window, execute the following command to obtain a dynamic real-time view of the `mysqld` application:

```
sys> top -n 8 | grep mysqld
```

    - What is the approximate value of the virtual memory (VIRT column) of the `top` output on your system?
  
    - What is the approximate value of the resident memory (RES Column) of the `top` output on your system?
  18. Return to the first terminal window and view the results of the `mysqlslap` application.
    - What is the value for the average number of seconds to run all queries output?

19. Restart the MySQL server as `root` and set the `innodb_buffer_pool_size` server setting to 36M.
20. Execute the command from step 9 as `root`.
  - While this is executing, go to the second terminal window opened in step 10 and complete the task in the next step.

**Note:** This command runs for approximately 30 seconds.

21. In a separate terminal window, execute the following command to obtain a dynamic real-time view of the `mysqld` application:

```
sys> top -n 8 | grep mysqld
```

- What is the approximate value of the virtual memory (VIRT column) of the `top` output on your system?
  
- What is the approximate value of the resident memory (RES Column) of the `top` output on your system?

22. Return to the first terminal window and view the results of the `mysqlslap` application.

- What is the value for the average number of seconds to run all queries output?

23. Which `innodb_buffer_pool_size` server setting produced the best results for the load placed upon this server?

---



---



---

| Step(s) | <code>innodb_flush_log_at_trx_commit</code> | <code>innodb_buffer_pool_size</code> | Avg. Seconds | VIRT | RES |
|---------|---------------------------------------------|--------------------------------------|--------------|------|-----|
| 2       | Default (1)                                 | Default                              |              |      |     |
| 3-4     | 2                                           | Default                              |              |      |     |
| 5-6     | 0                                           | Default                              |              |      |     |
| 8-11    | Default (1)                                 | 0                                    |              |      |     |
| 12-15   | Default (1)                                 | 12M                                  |              |      |     |
| 16-19   | Default (1)                                 | 24M                                  |              |      |     |
| 20-23   | Default (1)                                 | 36M                                  |              |      |     |

## Solutions 6-4: InnoDB Settings

### Tasks

**Note:** These solutions are sample solutions and are not expected to be the exact output you see in your execution of the steps.

1. Using the mysql client, list all the system status variables that begin with the word "innodb".

**Note:** You can use the existing terminal window from the previous practice. However, at this point, you should have only one terminal window open to minimize confusion as more terminal windows are opened up.

```
sys> mysql -uroot -poracle -e"SHOW VARIABLES LIKE 'innodb%'"
+-----+-----+
| Variable_name | Value |
+-----+-----+
innodb_adaptive_flushing	ON
innodb_adaptive_hash_index	ON
innodb_additional_mem_pool_size	8388608
innodb_autoextend_increment	8
innodb_autoinc_lock_mode	1
innodb_buffer_pool_instances	1
innodb_buffer_pool_size	134217728
innodb_change_buffering	all
innodb_checksums	ON
innodb_commit_concurrency	0
innodb_concurrency_tickets	500
innodb_data_file_path	ibdata1:10M:autoextend
innodb_data_home_dir	
innodb_doublewrite	ON
innodb_fast_shutdown	1
innodb_file_format	Antelope
innodb_file_format_check	ON
innodb_file_format_max	Antelope
innodb_file_per_table	OFF
innodb_flush_log_at_trx_commit	1
innodb_flush_method	
innodb_force_recovery	0
innodb_io_capacity	200
innodb_lock_wait_timeout	50
innodb_locks_unsafe_for_binlog	OFF
innodb_log_buffer_size	8388608
innodb_log_file_size	5242880
innodb_log_files_in_group	2
innodb_log_group_home_dir	./
innodb_max_dirty_pages_pct	75
innodb_max_purge_lag	0
innodb_mirrored_log_groups	1
innodb_old_blocks_pct	37
innodb_old_blocks_time	0
innodb_open_files	300
innodb_purge_batch_size	20
innodb_purge_threads	0
```

|                             |       |
|-----------------------------|-------|
| innodb_read_ahead_threshold | 56    |
| innodb_read_io_threads      | 4     |
| innodb_replication_delay    | 0     |
| innodb_rollback_on_timeout  | OFF   |
| innodb_spin_wait_delay      | 6     |
| innodb_stats_on_metadata    | ON    |
| innodb_stats_sample_pages   | 8     |
| innodb_strict_mode          | OFF   |
| innodb_support_xa           | ON    |
| innodb_sync_spin_loops      | 30    |
| innodb_table_locks          | ON    |
| innodb_thread_concurrency   | 0     |
| innodb_thread_sleep_delay   | 10000 |
| innodb_use_native_aio       | ON    |
| innodb_use_sys_malloc       | ON    |
| innodb_version              | 1.1.4 |
| innodb_write_io_threads     | 4     |

- What is the current setting for the `innodb_additional_mem_pool` system variable?

8388608

- What is the current setting for the `innodb_autoextend_increment` system variable?

8

- What is the current setting for the `innodb_buffer_pool_size` system variable?

134217728

- What is the current setting for the `innodb_file_per_table` system variable?

OFF

- What is the current setting for the `innodb_flush_log_at_trx_commit` system variable?

1

- What is the current setting for the `innodb_flush_method` system variable?

not set

- What is the current setting for the `innodb_log_files_in_group` system variable?

2

- What is the current setting for the `innodb_log_file_size` system variable?

5242880

- What is the current setting for the `innodb_thread_concurrency` system variable?

0

2. Using the `mysqlslap` application, execute the following command as `root` to create a load against the MySQL server and the `city_huge` table created in an earlier practice.

```
sys> mysqlslap -uroot -poracle \
> -q /stage/scripts/innodb_query.sql \
> --create-schema=many_tables -i 4 -c 10
Benchmark
 Average number of seconds to run all queries: 9.252 seconds
 Minimum number of seconds to run all queries: 6.760 seconds
 Maximum number of seconds to run all queries: 10.441 seconds
 Number of clients running queries: 10
 Average number of queries per client: 3745
```

- What is the value for the average number of seconds to run all queries output?

9.252 seconds

3. Using the `mysql` client, set the `innodb_flush_log_at_trx_commit` system variable to 2.

```
sys> mysql -uroot -poracle -e"SET GLOBAL \
> innodb_flush_log_at_trx_commit=2"
```

- Setting the `innodb_flush_log_at_trx_commit` system variable to 2 can result in an improved performance but at the cost of losing one second or more of transactions on a system crash or power down.

4. Execute the command from step 2 as `root`.

```
sys> mysqlslap -uroot -poracle \
> -q /stage/scripts/innodb_query.sql \
> --create-schema=many_tables -i 4 -c 10
Benchmark
 Average number of seconds to run all queries: 7.644 seconds
 Minimum number of seconds to run all queries: 5.638 seconds
 Maximum number of seconds to run all queries: 8.689 seconds
 Number of clients running queries: 10
 Average number of queries per client: 3745
```

- What is the value for the average number of seconds to run all queries output?

7.644 seconds

5. Using the `mysql` client, set the `innodb_flush_log_at_trx_commit` system variable to 0.

```
sys> mysql -uroot -poracle -e"SET GLOBAL \
> innodb_flush_log_at_trx_commit=0"
```

- Setting the `innodb_flush_log_at_trx_commit` system variable to 0 can result in a best performance but at the cost of losing up to one second of transactions on any mysqld process crash.

6. Execute the command from step 2 as root.

```
sys> mysqlslap -uroot -poracle \
> -q /stage/scripts/innodb_query.sql \
> --create-schema=many_tables -i 4 -c 10
Benchmark
Average number of seconds to run all queries: 7.501 seconds
Minimum number of seconds to run all queries: 5.363 seconds
Maximum number of seconds to run all queries: 8.485 seconds
Number of clients running queries: 10
Average number of queries per client: 3745
```

- What is the value for the average number of seconds to run all queries output?

7.501 seconds

7. Restart the MySQL server as root and set the `innodb_buffer_pool_size` server setting to 0.

```
sys> /etc/init.d/mysql restart --innodb_buffer_pool_size=0
Shutting down MySQL.. [OK]
Starting MySQL.. [OK]
```

8. Using the `mysqlslap` application, execute the following command as root to create a load against the MySQL server using a predefined script:

```
sys> mysqlslap -uroot -poracle \
> -q /stage/scripts/innodb_buffer.sql -i 5 -c 1
```

- While this is executing, open a second terminal window and complete the task in the next step.

**Note:** This command runs for approximately 4 minutes.

9. In a separate terminal window, execute the following command to obtain a dynamic real-time view of the `mysqld` application:

```
sys> top -n 8 | grep mysqld
```

- The output you see is similar to the output below:

|       |       |    |   |      |     |      |   |      |     |         |        |
|-------|-------|----|---|------|-----|------|---|------|-----|---------|--------|
| 19953 | mysql | 17 | 0 | 559m | 70m | 4560 | S | 87.7 | 3.5 | 0:11.95 | mysqld |
| 19953 | mysql | 17 | 0 | 559m | 70m | 4560 | S | 93.6 | 3.5 | 0:14.77 | mysqld |
| 19953 | mysql | 17 | 0 | 559m | 70m | 4560 | S | 95.3 | 3.5 | 0:17.64 | mysqld |
| 19953 | mysql | 17 | 0 | 559m | 70m | 4560 | S | 96.3 | 3.5 | 0:20.54 | mysqld |
| 19953 | mysql | 17 | 0 | 559m | 70m | 4560 | S | 93.3 | 3.5 | 0:23.35 | mysqld |
| 19953 | mysql | 17 | 0 | 559m | 70m | 4560 | S | 89.3 | 3.5 | 0:26.04 | mysqld |
| 19953 | mysql | 17 | 0 | 559m | 70m | 4560 | S | 88.6 | 3.5 | 0:28.71 | mysqld |
| 19953 | mysql | 17 | 0 | 559m | 70m | 4560 | S | 98.3 | 3.5 | 0:31.67 | mysqld |

- The columns of most interest to you for this practice include:

- Column 5: VIRT – The total amount of virtual memory used by the task. This value includes the resident memory the task is using along with the memory used by swapping out pages.
- Column 6: RES – The non-swapped physical memory used by the task.
- What is the approximate value of the virtual memory (VIRT column) of the `top` output on your system?

559m

- What is the approximate value of the resident memory (RES Column) of the `top` output on your system?

70m

10. Return to the first terminal window and view the results of the `mysqlslap` application.

Benchmark

```
Average number of seconds to run all queries: 46.497 seconds
Minimum number of seconds to run all queries: 43.409 seconds
Maximum number of seconds to run all queries: 48.179 seconds
Number of clients running queries: 1
Average number of queries per client: 5
```

- What is the value for the average number of seconds to run all queries output?

46.497 seconds

11. Restart the MySQL server as `root` and set the `innodb_buffer_pool_size` server setting to 12M.

```
sys> /etc/init.d/mysql restart --innodb_buffer_pool_size=12M
Shutting down MySQL.. [OK]
Starting MySQL.. [OK]
```

12. Execute the command from step 9 as `root`.

```
sys> mysqlslap -uroot -poracle \
> -q /stage/scripts/innodb_buffer.sql -i 5 -c 1
```

- While this is executing, go to the second terminal window opened in step 10 and complete the task in the next step.

**Note:** This command runs for approximately one minute.

13. In a separate terminal window, execute the following command to obtain a dynamic real-time view of the `mysqld` application:

```
sys> top -n 8 | grep mysqld
```

- The output you see is similar to the output below:

|       |       |    |   |      |     |      |   |      |     |         |        |
|-------|-------|----|---|------|-----|------|---|------|-----|---------|--------|
| 20347 | mysql | 25 | 0 | 579m | 81m | 4416 | S | 81.8 | 4.0 | 0:04.65 | mysqld |
| 20347 | mysql | 25 | 0 | 579m | 81m | 4416 | S | 88.0 | 4.0 | 0:07.30 | mysqld |
| 20347 | mysql | 18 | 0 | 579m | 81m | 4448 | S | 82.3 | 4.0 | 0:09.78 | mysqld |
| 20347 | mysql | 18 | 0 | 579m | 81m | 4448 | S | 82.0 | 4.0 | 0:12.25 | mysqld |
| 20347 | mysql | 18 | 0 | 579m | 81m | 4448 | S | 86.3 | 4.0 | 0:14.85 | mysqld |
| 20347 | mysql | 15 | 0 | 579m | 81m | 4448 | S | 79.7 | 4.0 | 0:17.25 | mysqld |
| 20347 | mysql | 15 | 0 | 579m | 81m | 4448 | S | 90.0 | 4.0 | 0:19.96 | mysqld |
| 20347 | mysql | 15 | 0 | 579m | 81m | 4448 | S | 81.4 | 4.0 | 0:22.41 | mysqld |

- What is the approximate value of the virtual memory (VIRT column) of the `top` output on your system?

579m

- What is the approximate value of the resident memory (RES Column) of the `top` output on your system?

81m

14. Return to the first terminal window and view the results of the `mysqlslap` application.

```
Benchmark
```

```
Average number of seconds to run all queries: 9.550 seconds
Minimum number of seconds to run all queries: 8.398 seconds
Maximum number of seconds to run all queries: 10.524 seconds
Number of clients running queries: 1
Average number of queries per client: 5
```

- What is the value for the average number of seconds to run all queries output?

9.550 seconds

15. Restart the MySQL server as `root` and set the `innodb_buffer_pool_size` server setting to 24M.

```
sys> /etc/init.d/mysql restart --innodb_buffer_pool_size=24M
Shutting down MySQL.. [OK]
Starting MySQL.. [OK]
```

16. Execute the command from step 9 as `root`.

```
sys> mysqlslap -uroot -poracle \
> -q /stage/scripts/innodb_buffer.sql -i 5 -c 1
```

- While this is executing, go to the second terminal window opened in step 10 and complete the task in the next step.

**Note:** This command runs for approximately 30 seconds.

17. In a separate terminal window, execute the following command to obtain a dynamic real-time view of the `mysqld` application:

```
sys> top -n 8 | grep mysqld
```

- The output you see is similar to the output below:

|       |       |    |   |      |     |      |   |      |     |         |        |
|-------|-------|----|---|------|-----|------|---|------|-----|---------|--------|
| 20733 | mysql | 18 | 0 | 581m | 91m | 4396 | S | 63.1 | 4.5 | 0:02.05 | mysqld |
| 20733 | mysql | 18 | 0 | 581m | 94m | 4396 | S | 72.3 | 4.7 | 0:04.23 | mysqld |
| 20733 | mysql | 18 | 0 | 581m | 94m | 4428 | S | 72.1 | 4.7 | 0:06.40 | mysqld |
| 20733 | mysql | 18 | 0 | 582m | 94m | 4440 | S | 76.7 | 4.7 | 0:08.71 | mysqld |
| 20733 | mysql | 18 | 0 | 582m | 94m | 4440 | S | 75.7 | 4.7 | 0:10.99 | mysqld |
| 20733 | mysql | 18 | 0 | 582m | 94m | 4440 | S | 71.3 | 4.7 | 0:13.14 | mysqld |

- What is the approximate value of the virtual memory (VIRT column) of the `top` output on your system?

582m

- What is the approximate value of the resident memory (RES Column) of the `top` output on your system?

94m

18. Return to the first terminal window and view the results of the `mysqlslap` application.

```
Benchmark
```

```
Average number of seconds to run all queries: 5.708 seconds
Minimum number of seconds to run all queries: 5.243 seconds
Maximum number of seconds to run all queries: 5.940 seconds
Number of clients running queries: 1
Average number of queries per client: 5
```

- What is the value for the average number of seconds to run all queries output?

5.708 seconds

19. Restart the MySQL server as `root` and set the `innodb_buffer_pool_size` server setting to 36M.

```
sys> /etc/init.d/mysql restart --innodb_buffer_pool_size=36M
Shutting down MySQL.. [OK]
Starting MySQL.. [OK]
```

20. Execute the command from step 9 as `root`.

```
sys> mysqlslap -uroot -poracle \
> -q /stage/scripts/innodb_buffer.sql -i 5 -c 1
```

- While this is executing, go to the second terminal window opened in step 10 and complete the task in the next step.

**Note:** This command runs for approximately 30 seconds.

21. In a separate terminal window, execute the following command to obtain a dynamic real-time view of the `mysqld` application:

```
sys> top -n 8 | grep mysqld
```

- The output you see is similar to the output below:

|       |       |    |   |      |      |      |   |      |     |         |        |
|-------|-------|----|---|------|------|------|---|------|-----|---------|--------|
| 21112 | mysql | 18 | 0 | 605m | 106m | 4428 | S | 74.0 | 5.3 | 0:04.66 | mysqld |
| 21112 | mysql | 18 | 0 | 605m | 106m | 4428 | S | 72.3 | 5.3 | 0:06.84 | mysqld |
| 21112 | mysql | 18 | 0 | 605m | 106m | 4440 | S | 75.3 | 5.3 | 0:09.11 | mysqld |
| 21112 | mysql | 18 | 0 | 605m | 106m | 4440 | S | 76.3 | 5.3 | 0:11.41 | mysqld |
| 21112 | mysql | 18 | 0 | 605m | 106m | 4440 | S | 70.4 | 5.3 | 0:13.53 | mysqld |
| 21112 | mysql | 18 | 0 | 605m | 106m | 4440 | S | 76.3 | 5.3 | 0:15.83 | mysqld |
| 21112 | mysql | 18 | 0 | 605m | 106m | 4440 | S | 66.8 | 5.3 | 0:17.84 | mysqld |
| 21112 | mysql | 18 | 0 | 605m | 106m | 4440 | S | 85.3 | 5.3 | 0:20.41 | mysqld |

- What is the approximate value of the virtual memory (VIRT column) of the `top` output on your system?

605m

- What is the approximate value of the resident memory (RES Column) of the `top` output on your system?

106m

22. Return to the first terminal window and view the results of the `mysqlslap` application.

```
Benchmark
Average number of seconds to run all queries: 5.632 seconds
Minimum number of seconds to run all queries: 5.289 seconds
Maximum number of seconds to run all queries: 5.940 seconds
Number of clients running queries: 1
Average number of queries per client: 5
```

- What is the value for the average number of seconds to run all queries output?

5.632 seconds

23. Which `innodb_buffer_pool_size` server setting produced the best results for the load placed upon this server?

Comparing the amount of memory consumed and the response time for the execution of the queries, setting the innodb buffer pool size to 24M provided the best solution for the activity on this server. Benchmarking your server using a similar approach would be crucial to find the best setting for your servers workload.

| Step(s) | <code>innodb_flush_log_at_trx_commit</code> | <code>innodb_buffer_pool_size</code> | Avg. Seconds | VIRT | RES  |
|---------|---------------------------------------------|--------------------------------------|--------------|------|------|
| 2       | Default (1)                                 | Default                              | 9.25         | N/A  | N/A  |
| 3-4     | 2                                           | Default                              | 7.64         | N/A  | N/A  |
| 5-6     | 0                                           | Default                              | 7.5          | N/A  | N/A  |
| 8-11    | Default (1)                                 | 0                                    | 46.5         | 559m | 70m  |
| 12-15   | Default (1)                                 | 12M                                  | 9.55         | 579m | 81m  |
| 16-19   | Default (1)                                 | 24M                                  | 5.71         | 582m | 94m  |
| 20-23   | Default (1)                                 | 36M                                  | 5.63         | 605m | 106m |



## **Practices for Lesson 7: MyISAM**

**Chapter 7**

## Practices for Lesson 7

---

### Practices Overview

In these practices, you test your knowledge of improving the performance of queries executed against the MySQL server.

## Practice 7-1: Optimizing MyISAM

### Overview

In this practice, you evaluate the effect of optimizing a MyISAM table. To accomplish this objective, you do the following:

- Create 2 million records and load them into a table in the `many_tables` database.
- Delete a large number of records in a MyISAM table and record if the sizes of the table and indexes on disk were affected.
- Optimize the MyISAM table with deleted records and record if the sizes of the table and indexes on disk were affected.

### Assumptions

- The MySQL server is installed and running.
- The `many_tables` database is installed.
- The `city_huge.sql` and `delete_city_huge.sql` files are located in the `/stage/scripts` directory.

### Duration

This practice should take 15 minutes to complete.

### Tasks

**Note:** A table has been provided at the end of the steps for this practice for you to fill in the values collected during the running of the steps. Entering the data from the steps into this table can assist you in comparing the values collected.

1. Using the `mysql` client, execute the following command to create the `city_huge` table in the `many_table` database, change the `city_huge` table storage engine to MyISAM, disable the keys, load 2 million records into the `city_huge` table, and then reenable the keys:

```
sys> mysql -uroot -poracle -e"USE many_tables;\
> CALL create_city_huge(0);\
> ALTER TABLE city_huge ENGINE=MYISAM;\
> ALTER TABLE city_huge DISABLE KEYS;\
> SOURCE /stage/scripts/city_huge.sql;\
> ALTER TABLE city_huge ENABLE KEYS;"
```

**Note:** This command should take approximately one minute to complete.

2. Using the `mysql` client, view the number of records in the `many_tables.city_huge` table.
  - How many records does the `many_tables.city_huge` table contain?

---

3. In the /var/lib/mysql/many\_table directory, as root view the size of the city\_huge.MYD and city\_huge.MYI files.

– What is the size of the city\_huge.MYD file?

---

– What is the size of the city\_huge.MYI file?

4. Using the mysql client, execute the following command to use the many\_tables database and delete a selection of records that creates “holes” in the city\_huge table:

```
sys> mysql -uroot -poracle -e"USE many_tables;SOURCE \
> /stage/scripts/delete_city_huge.sql;"
```

5. Using the mysql client, view the number of records in the many\_tables.city\_huge table after the delete has taken place.

– How many records does the many\_tables.city\_huge table contain?

6. As root, review the size of the files from step 5 to see if there is any noticeable change in the size of the files.

– What is the size of the city\_huge.MYD file?

---

– What is the size of the city\_huge.MYI file?

7. Is there a difference between the sizes of the city\_huge.MYD and city\_huge.MYI files before and after the deletion of records?

---

8. Using the mysql client, run the OPTIMIZE TABLE command against the many\_tables.city\_huge table.

9. As root, review the size of the files from step 5 to see if there is any noticeable change in the size of the files after the table has been optimized.

– What is the size of the city\_huge.MYD file?

---

– What is the size of the city\_huge.MYI file?

---

10. Is there a difference between the sizes of the `city_huge.MYD` and `city_huge.MYI` files before and after the optimization of the table?
- 

| <b>Step(s)</b> | <b><code>many_tables.city_huge</code> records</b> | <b><code>city_huge.MYD</code> file size</b> | <b><code>city_huge.MYI</code> file size</b> |
|----------------|---------------------------------------------------|---------------------------------------------|---------------------------------------------|
| 2-3            |                                                   |                                             |                                             |
| 4-6            |                                                   |                                             |                                             |
| 7-9            | N/A                                               |                                             |                                             |

## Solutions 7-1: Optimizing MyISAM

---

### Tasks

**Note:** These solutions are sample solutions and are not expected to be the exact output you see in your execution of the steps.

- Using the `mysql` client, execute the following command to create the `city_huge` table in the `many_table` database, change the `city_huge` table storage engine to MyISAM, disable the keys, load 2 million records into the `city_huge` table, and then reenable the keys:

```
sys> mysql -uroot -poracle -e"USE many_tables; \
> CALL create_city_huge(1); \
> ALTER TABLE city_huge ENGINE=MYISAM; \
> ALTER TABLE city_huge DISABLE KEYS; \
> SOURCE /stage/scripts/city_huge.sql; \
> ALTER TABLE city_huge ENABLE KEYS;"
```

**Note:** This command should take approximately one minute to complete.

- Using the `mysql` client, view the number of records in the `many_tables.city_huge` table.

```
sys> mysql -uroot -poracle -e"SELECT COUNT(*) FROM \
> many_tables.city_huge;"
```

| COUNT(*) |
|----------|
| 2082237  |

- How many records does the `many_tables.city_huge` table contain?

2,082,237 records

- In the `/var/lib/mysql/many_tables` directory, as root view the size of the `city_huge.MYD` and `city_huge.MYI` files.

```
sys> ls -l city_huge.*
```

|            |   |       |       |           |        |       |               |
|------------|---|-------|-------|-----------|--------|-------|---------------|
| -rw-rw---- | 1 | mysql | mysql | 8710      | Feb 17 | 10:43 | city_huge.frm |
| -rw-rw---- | 1 | mysql | mysql | 139509879 | Feb 18 | 00:35 | city_huge.MYD |
| -rw-rw---- | 1 | mysql | mysql | 76743680  | Feb 18 | 00:36 | city_huge.MYI |

- What is the size of the `city_huge.MYD` file?

139,509,879 bytes

- What is the size of the `city_huge.MYI` file?

76,743,680 bytes

- Using the `mysql` client, execute the following command to use the `many_tables` database and delete a selection of records that creates “holes” in the `city_huge` table:

```
sys> mysql -uroot -poracle -e"USE many_tables;SOURCE \
> /stage/scripts/delete_city_huge.sql;"
```

5. Using the mysql client, view the number of records in the many\_tables.city\_huge table after the delete has taken place.

```
sys> mysql -uroot -poracle -e"SELECT COUNT(*) FROM
many_tables.city_huge;"
```

| COUNT(*) |
|----------|
| 1836389  |

- How many records does the many\_tables.city\_huge table contain?

1,836,389 records

6. As root, review the size of the files from step 5 to see if there is any noticeable change in the size of the files.

```
sys> ls -l city_huge.*
```

|            |   |       |       |           |        |       |               |
|------------|---|-------|-------|-----------|--------|-------|---------------|
| -rw-rw---- | 1 | mysql | mysql | 8710      | Feb 17 | 10:43 | city_huge.frm |
| -rw-rw---- | 1 | mysql | mysql | 139509879 | Feb 18 | 00:35 | city_huge.MYD |
| -rw-rw---- | 1 | mysql | mysql | 76743680  | Feb 18 | 00:36 | city_huge.MYI |

- What is the size of the city\_huge.MYD file?

139,509,879 bytes

- What is the size of the city\_huge.MYI file?

76,743,680 bytes

7. Is there a difference between the sizes of the city\_huge.MYD and city\_huge.MYI files before and after the deletion of records?

No, the size of the files remained the same, but the number of records is much smaller after the deletion of the records in step 6.

8. Using the mysql client, run the OPTIMIZE TABLE command against the many\_tables.city\_huge table.

```
sys> mysql -uroot -poracle -e"OPTIMIZE TABLES \
> many_tables.city_huge;"
```

| Table                 | Op       | Msg_type | Msg_text |
|-----------------------|----------|----------|----------|
| many_tables.city_huge | optimize | status   | OK       |

9. As root, review the size of the files from step 5 to see if there is any noticeable change in the size of the files after the table has been optimized.

```
sys> ls -l city_huge.*
-rw-rw---- 1 mysql mysql 8710 Feb 17 10:43 city_huge.frm
-rw-rw---- 1 mysql mysql 123038063 Feb 18 00:35 city_huge.MYD
-rw-rw---- 1 mysql mysql 67561472 Feb 18 00:36 city_huge.MYI
```

- What is the size of the `city_huge.MYD` file?

123,038,063 bytes

- What is the size of the `city_huge.MYI` file?

67,561,472 bytes

10. Is there a difference between the sizes of the `city_huge.MYD` and `city_huge.MYI` files before and after the optimization of the table?

Yes, the size of both files have been reduced to reflect that the MySQL server was able to reclaim free space that was being taken up by the “holes” in the `city_huge` table.

| Step(s) | many_tables.<br>city_huge<br>records | city_huge.MYD<br>file size | city_huge.MYI<br>file size |
|---------|--------------------------------------|----------------------------|----------------------------|
| 2-3     | 2,082,237                            | 139,509,879                | 76,743,680                 |
| 4-6     | 1,836,389                            | 139,509,879                | 76,743,680                 |
| 7-9     | N/A                                  | 123,038,063                | 67,561,472                 |

## Practice 7-2: MyISAM Table Locks

### Overview

In this practice, you evaluate the effects of MyISAM table locking on delete, insert, and select operations. To accomplish this objective, you do the following:

- Execute delete, insert, and select operations simultaneously to evaluate the effect of lock contention on the operations.

### Assumptions

- The MySQL server is installed and running.
- The `PT_Stress.php` script is installed.
- The `many_tables` database is installed.
- The `city_huge.sql`, `delete_city_huge_32K.sql`,  
`delete_city_huge_8K_1.sql`, `delete_city_huge_8K_2.sql`,  
`delete_city_huge_8K_3.sql`, and the `delete_city_huge_8K_4.sql` scripts  
are located in the `/stage/scripts/` directory.

### Duration

This practice should take 40 minutes to complete.

### Tasks

**Note:** A table has been provided at the end of the steps for this practice for you to fill in the values collected during the running of the steps. Entering the data from the steps into this table can assist you in comparing the values collected.

1. Using the `mysql` client, execute the following command to clean up the `city_huge` table:

```
sys> mysql -uroot -poracle -e"USE many_tables;TRUNCATE \
> city_huge;ALTER TABLE city_huge DISABLE KEYS;SOURCE \
> /stage/scripts/city_huge.sql; ALTER TABLE city_huge ENABLE \
> KEYS;"
```

**Note:** This process takes approximately one minute to complete.

**Note:** For the next few steps, you need to have four separate terminal windows opened. The first terminal window is running the `mysql` client. The second terminal window is running delete operations against the MySQL server. The third terminal window is running insert operations against the MySQL server. The fourth terminal window is running the select operations.

2. In the first terminal window, open the `mysql` client and execute a `FLUSH STATUS` command to clear the status variables.
3. In the first terminal window (the `mysql` client), view the processes currently running by the MySQL server.
4. In the first terminal window (the `mysql` client), view the values of the global status variables that are associated with table locks.

**Note:** The status variable `Table_locks_immediate` displays the number of times that a request for a table lock could be granted immediately. The status variable `Table_locks_waited` displays the number of times that a request for a table lock could not be granted immediately and a wait was needed.

5. In the second terminal window, execute the following command to delete 32,000 records in the `city_huge` table:

```
sys> time mysql -uroot -poracle -e"USE many_tables;SOURCE \
> /stage/scripts/delete_city_huge_32K.sql;"
```

**Note:** While this command is running, open the third terminal window and complete the next step.

6. In the third terminal window, execute the following command to insert 100,000 records into the `city_huge` table:

```
sys> time mysql -uroot -poracle -e"USE many_tables;SOURCE \
> /stage/scripts/insert_city_huge_100K.sql;"
```

**Note:** While this command is running, open the fourth terminal window and complete the next step.

7. In the fourth terminal window, in the `/stage/script` directory, execute the following command to select 100,000 random records from the `city_huge` table in the `many_tables` database:

```
sys> ./PT_Stress.php -u root -p oracle -t -q \
> -s "Q:SELECT * FROM many_tables.city_huge WHERE ID={RANDOM}" \
> -i 100000 -r 2000000
```

**Note:** The `PT_Stress.php` script has the capability to produce random numbers to a maximum of the `-r` argument (in this case 2,000,000). The random number produced replaces the text `{RANDOM}` when executed.

**Note:** While this command is running, return to the first terminal window and complete the next step.

8. In the first terminal window (the `mysql` client), view the processes currently running by the MySQL server.

**Note:** You may have to execute the `SHOW FULL PROCESSLIST` command a few times until you are able to find a `State` output that identifies that the `INSERT` operation is waiting on a table level lock.

9. In the second terminal window, review the time that the MySQL server took to complete the delete operation.

- What is the approximate time that it took the `mysql` client to complete the task?

- 
10. In the third terminal window, review the time that the MySQL server took to complete the insert operation.

- What is the approximate time that it took the `mysql` client to complete the task?

- 
11. In the fourth terminal window, review the time that the `PT_Stress.php` application took to complete the select operation.

- What is the approximate time that it took the `PT_Stress.php` application to complete the task?
-

12. In the first terminal window (the `mysql` client), view the values of the global status variables that are associated with table locks.

- What was the value of the `Table_locks_immediate` status variable?

---

- What was the value of the `Table_locks_waited` status variable?

---

13. In the first terminal window (the `mysql` client), execute a `FLUSH STATUS` command to clear the status variables.

14. In the second terminal window, execute the command from step 1 to clean up the `city_huge` table.

15. In the second terminal window, execute the following command that deletes 32,000 records in the `city_huge` table in increments of 8,000 records at a time with a 30 second pause between each incremental delete:

```
sys> time mysql -uroot -poracle -e"USE many_tables;\
> SOURCE /stage/scripts/delete_city_huge_8K_1.sql;\
> SELECT SLEEP(30);\
> SOURCE /stage/scripts/delete_city_huge_8K_2.sql;\
> SELECT SLEEP(30);\
> SOURCE /stage/scripts/delete_city_huge_8K_3.sql;\
> SELECT SLEEP(30);\
> SOURCE /stage/scripts/delete_city_huge_8K_4.sql;"
```

**Note:** This command is similar to the earlier delete step but uses the `SLEEP` command to execute a 30-second pause between each incremental run. The command `SLEEP(30)` is designed to simulate a natural "pause" that you would find in a real world environment.

**Note:** While this command is running, return to the third terminal window and complete the next step.

16. In the third terminal window, execute the command from step 6 to insert 100,000 records in the the `city_huge` table.

**Note:** While this command is running, return to the fourth terminal window and complete the next step.

17. In the fourth terminal window, execute the command from step 7 to select 100,000 random records from the `city_huge` table in the `many_tables` database.

18. In the second terminal window, review the time that the MySQL server took to complete the delete operation.

- What is the approximate time that it took the `mysql` client to complete the task?

---

19. In the third terminal window, review the time that the MySQL server took to complete the insert operation.

- What is the approximate time that it took the `mysql` client to complete the task?
-

20. In the fourth terminal window, review the time that the `PT_Stress.php` application took to complete the select operation.

- What is the approximate time that it took the `PT_Stress.php` application to complete the task?

---

21. In the first terminal window (the `mysql` client), view the values of the global status variables that are associated with table locks.

- What was the value of the `Table_locks_immediate` status variable?

- 
- What was the value of the `Table_locks_waited` status variable?

22. In the first terminal window (the `mysql` client), execute a `FLUSH STATUS` command to clear the status variables.

23. In the second terminal window, execute the command from step 1 to clean up the `city_huge` table.

24. In the third terminal window, execute the command from step 6 to insert 100,000 records in the `city_huge` table.

**Note:** While this command is running, return to the fourth terminal window and complete the next step.

25. In the fourth terminal window, execute the command from step 7 to select 100,000 random records from the `city_huge` table in the `many_tables` database.

26. In the third terminal window, review the time that the MySQL server took to complete the insert operation.

- What is the approximate time that it took the `mysql` client to complete the task?

---

27. In the fourth terminal window, review the time that the `PT_Stress.php` application took to complete the select operation.

- What is the approximate time that it took the `PT_Stress.php` application to complete the task?

---

28. In the first terminal window (the `mysql` client), view the values of the global status variables that are associated with table locks.

- What was the value of the `Table_locks_immediate` status variable?

- 
- What was the value of the `Table_locks_waited` status variable?

29. Comparing the outputs from the different approaches to completing delete, insert, and select operations together, what do the output values tell you?

---

---

---

| Step(s) | Delete Operation Time | Insert Operation Time | PT_Stress.php Time | Table locks%          |                    |
|---------|-----------------------|-----------------------|--------------------|-----------------------|--------------------|
|         |                       |                       |                    | Table locks immediate | Table locks waited |
| 4-12    |                       |                       |                    | Table locks immediate |                    |
|         |                       |                       |                    | Table locks waited    |                    |
| 13-21   |                       |                       |                    | Table locks immediate |                    |
|         |                       |                       |                    | Table locks waited    |                    |
| 22-28   | N/A                   |                       |                    | Table locks immediate |                    |
|         |                       |                       |                    | Table locks waited    |                    |

## Solutions 7-2: MyISAM Table Locks

### Tasks

**Note:** These solutions are sample solutions and are not expected to be the exact output you see in your execution of the steps.

1. Using the `mysql` client, execute the following command to clean up the `city_huge` table:

```
sys> mysql -uroot -poracle -e"USE many_tables;TRUNCATE \
> city_huge;ALTER TABLE city_huge DISABLE KEYS;SOURCE \
> /stage/scripts/city_huge.sql; ALTER TABLE city_huge ENABLE \
> KEYS;"
```

**Note:** This process takes approximately one minute to complete.

**Note:** For the next few steps, you need to have four separate terminal windows opened. The first terminal window is running the `mysql` client. The second terminal window is running delete operations against the MySQL server. The third terminal window is running insert operations against the MySQL server. The fourth terminal window is running the select operations.

2. In the first terminal window, open the `mysql` client and execute a `FLUSH STATUS` command to clear the status variables.

```
sys> mysql -uroot -poracle
mysql> FLUSH STATUS;
```

3. In the first terminal window (the `mysql` client), view the processes currently running by the MySQL server.

```
mysql> SHOW FULL PROCESSLIST\G
***** 1. row *****
 Id: 52695
 User: root
 Host: localhost
 db: NULL
 Command: Query
 Time: 0
 State: NULL
 Info: SHOW FULL PROCESSLIST
1 row in set (0.00 sec)
```

4. In the first terminal window (the `mysql` client), view the values of the global status variables that are associated with table locks.

```
mysql> SHOW GLOBAL STATUS LIKE 'Table_locks%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Table_locks_immediate | 0 |
| Table_locks_waited | 0 |
+-----+-----+
2 rows in set (0.00 sec)
```

**Note:** The status variable `Table_locks_immediate` displays the number of times that a request for a table lock could be granted immediately. The status variable `Table_locks_waited` displays the number of times that a request for a table lock could not be granted immediately and a wait was needed.

5. In the second terminal window, execute the following command to delete 32,000 records in the `city_huge` table:

```
sys> time mysql -uroot -poracle -e"USE many_tables;SOURCE \
> /stage/scripts/delete_city_huge_32K.sql;"
```

**Note:** While this command is running, open the third terminal window and complete the next step.

6. In the third terminal window, execute the following command to insert 100,000 records into the `city_huge` table:

```
sys> time mysql -uroot -poracle -e" USE many_tables;SOURCE \
> /stage/scripts/insert_city_huge_100K.sql;"
```

**Note:** While this command is running, open the fourth terminal window and complete the next step.

7. In the fourth terminal window, change to the `/stage/scripts` directory and execute the following command to select 100,000 random records from the `city_huge` table in the `many_tables` database:

```
sys> ./PT_Stress.php -u root -p oracle -t -q \
> -s "Q:SELECT * FROM many_tables.city_huge WHERE ID={RANDOM}" \
> -i 100000 -r 2000000
```

**Note:** The `PT_Stress.php` script has the capability to produce random numbers to a maximum of the `-r` argument (in this case 2,000,000). The random number produced replaces the text `{RANDOM}` when executed.

**Note:** While this command is running, return to the first terminal window and complete the next step.

8. In the first terminal window (the mysql client), view the processes currently running by the MySQL server.

```
mysql> SHOW FULL PROCESSLIST\G
***** 1. row *****
Id: 53212
User: root
Host: localhost
db: NULL
Command: Query
Time: 0
State: NULL
Info: SHOW FULL PROCESSLIST
***** 2. row *****
Id: 53213
User: root
Host: localhost
db: many_tables
Command: Query
Time: 1
State: updating
Info: DELETE FROM city_huge WHERE ID = 389449
***** 3. row *****
Id: 53214
User: root
Host: localhost
db: many_tables
Command: Query
Time: 1
State: Waiting for table level lock
Info: INSERT INTO `city_huge` (Name, CountryCode, District, Population) VALUES ('Owo', 'NGA', 'Ondo & Ekiti', 183500)
***** 4. row *****
Id: 53215
User: root
Host: localhost
db: NULL
Command: Query
Time: 1
State: Waiting for table level lock
Info: SELECT * FROM many_tables.city_huge WHERE ID=1587035
4 rows in set (0.00 sec)
```

**Note:** You may have to execute the SHOW FULL PROCESSLIST command a few times until you are able to find a State output that identifies that the INSERT operation is waiting on a table level lock.

9. In the second terminal window, review the time that the MySQL server took to complete the delete operation.

```
real 4m32.081s
user 0m0.809s
sys 0m0.649s
```

- What is the approximate time that it took the mysql client to complete the task?

4 minutes 32 seconds

10. In the third terminal window, review the time that the MySQL server took to complete the insert operation.

```
real 6m55.209s
user 0m2.902s
sys 0m1.962s
```

- What is the approximate time that it took the mysql client to complete the task?

6 minutes 55 seconds

11. In the fourth terminal window, review the time that the PT\_Stress.php application took to complete the select operation.

```
Execution Times
Minimum: 339.200426
Maximum: 339.200426
Average: 339.200426
```

- What is the approximate time that it took the PT\_Stress.php application to complete the task?

339 seconds or 6 minutes 39 seconds

12. In the first terminal window (the mysql client), view the values of the global status variables that are associated with table locks.

```
mysql> SHOW GLOBAL STATUS LIKE 'Table_locks%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Table_locks_immediate | 208312 |
| Table_locks_waited | 23686 |
+-----+-----+
2 rows in set (0.00 sec)
```

- What was the value of the Table\_locks\_immediate status variable?

208,312

- What was the value of the Table\_locks\_waited status variable?

23,686

13. In the first terminal window (the mysql client), execute a FLUSH STATUS command to clear the status variables.

```
mysql> FLUSH STATUS;
```

14. In the second terminal window, execute the command from step 1 to clean up the `city_huge` table.

```
sys> mysql -uroot -poracle -e"USE many_tables;TRUNCATE \
> city_huge;ALTER TABLE city_huge DISABLE KEYS;SOURCE \
> /stage/scripts/city_huge.sql; ALTER TABLE city_huge ENABLE \
> KEYS;"
```

15. In the second terminal window, execute the following command that deletes 32,000 records in the `city_huge` table in increments of 8,000 records at a time with a 30 second pause between each incremental delete:

```
sys> time mysql -uroot -poracle -e"USE many_tables; \
> SOURCE /stage/scripts/delete_city_huge_8K_1.sql; \
> SELECT SLEEP(30); \
> SOURCE /stage/scripts/delete_city_huge_8K_2.sql; \
> SELECT SLEEP(30); \
> SOURCE /stage/scripts/delete_city_huge_8K_3.sql; \
> SELECT SLEEP(30); \
> SOURCE /stage/scripts/delete_city_huge_8K_4.sql;"
```

**Note:** This command is similar to the earlier delete step but uses the `SLEEP` command to execute a 30 second pause between each incremental run. The command `SLEEP(30)` is designed to simulate a natural "pause" that you would find in a real world environment.

**Note:** While this command is running, return to the third terminal window and complete the next step.

16. In the third terminal window, execute the command from step 6 to insert 100,000 records in the `city_huge` table.

```
sys> time mysql -uroot -poracle -e" USE many_tables;SOURCE \
> /stage/scripts/insert_city_huge_100K.sql;"
```

**Note:** While this command is running, return to the fourth terminal window and complete the next step.

17. In the fourth terminal window, change to the `/stage/scripts` directory and execute the command from step 7 to select 100,000 random records from the `city_huge` table in the `many_tables` database.

```
sys> ./PT_Stress.php -u root -p oracle -t -q \
> -s "Q:SELECT * FROM many_tables.city_huge WHERE ID={RANDOM}" \
> -i 100000 -r 2000000
```

18. In the second terminal window, review the time that the MySQL server took to complete the delete operation.

```
real 4m34.643s
user 0m0.825s
sys 0m0.649ss
```

- What is the approximate time that it took the `mysql` client to complete the task?

4 minutes 35 seconds

19. In the third terminal window, review the time that the MySQL server took to complete the insert operation.

```
real 5m38.313s
user 0m2.602s
sys 0m1.948s
```

- What is the approximate time that it took the mysql client to complete the task?

5 minutes 38 seconds

20. In the fourth terminal window, review the time that the PT\_stress.php application took to complete the select operation.

| Execution Times |            |
|-----------------|------------|
| Minimum:        | 306.845381 |
| Maximum:        | 306.845381 |
| Average:        | 306.845381 |

- What is the approximate time that it took the PT\_Stress.php application to complete the task?

306 seconds or 6 minutes 06 seconds

21. In the first terminal window (the mysql client), view the values of the global status variables that are associated with table locks.

```
mysql> SHOW GLOBAL STATUS LIKE 'Table_locks%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Table_locks_immediate | 198511 |
| Table_locks_waited | 33584 |
+-----+-----+
2 rows in set (0.00 sec)
```

- What was the value of the Table\_locks\_immediate status variable?

198,511

- What was the value of the Table\_locks\_waited status variable?

33,584

22. In the first terminal window (the mysql client), execute a FLUSH STATUS command to clear the status variables.

```
mysql> FLUSH STATUS;
```

23. In the second terminal window, execute the command from step 1 to clean up the city\_huge table.

```
sys> mysql -uroot -poracle -e"USE many_tables;TRUNCATE \
> city_huge;ALTER TABLE city_huge DISABLE KEYS;SOURCE \
> /stage/scripts/city_huge.sql; ALTER TABLE city_huge ENABLE \
> KEYS;"
```

24. In the third terminal window, execute the command from step 6 to insert 100,000 records in the the `city_huge` table.

```
sys> time mysql -uroot -poracle -e" USE many_tables;SOURCE \
> /stage/scripts/insert_city_huge_100K.sql;"
```

**Note:** While this command is running, return to the fourth terminal window and complete the next step.

25. In the fourth terminal window, change to the `/stage/scripts` directory and execute the command from step 7 to select 100,000 random records from the `city_huge` table in the `many_tables` database.

```
sys> ./PT_Stress.php -u root -p oracle -t -q \
> -s "SELECT * FROM many_tables.city_huge WHERE ID={RANDOM}" \
> -i 100000 -r 2000000
```

26. In the third terminal window, review the time that the MySQL server took to complete the insert operation.

```
real 1m54.007s
user 0m2.553s
sys 0m1.885s
```

- What is the approximate time that it took the `mysql` client to complete the task?

one minutes 54 seconds

27. In the fourth terminal window, review the time that the `PT_Stress.php` application took to complete the select operation.

| Execution Times    |
|--------------------|
| Minimum: 89.191879 |
| Maximum: 89.191879 |
| Average: 89.191879 |

- What is the approximate time that it took the `PT_Stress.php` application to complete the task?

89 seconds or one minute 29 seconds

28. In the first terminal window (the `mysql` client), view the values of the global status variables that are associated with table locks.

```
mysql> SHOW GLOBAL STATUS LIKE 'Table_locks%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Table_locks_immediate | 200095 |
| Table_locks_waited | 0 |
+-----+-----+
2 rows in set (0.00 sec)
```

- What was the value of the `Table_locks_immediate` status variable?

200,095

- What was the value of the `Table_locks_waited` status variable?

0

29. Comparing the outputs from the different approaches to completing delete, insert, and select operations together, what do the output values tell you?

Breaking up the delete operation into smaller increments with one minute and 30 seconds of the MySQL server “sleeping” improved the insert and select operations. Removing the delete operations or running them at non-working hours (if possible) are better solutions.  
Opportunities where you can break up delete operations or run them at non-working hours can improve the total processes and allow the clients to work more effectively together. If the delete operations require a greater priority, this may not be an acceptable solution.

**Note:** The results described here are specific to the MyISAM storage engine. The InnoDB storage engine does not have these limitations.

| <b>Step(s)</b> | <b>Delete Operation Time</b> | <b>Insert Operation Time</b> | <b>PT_Stress.php Time</b> | <b>Table locks%</b>   |                    |
|----------------|------------------------------|------------------------------|---------------------------|-----------------------|--------------------|
|                |                              |                              |                           | Table locks immediate | Table locks waited |
| 4-12           | 4m 32s                       | 6m 55s                       | 6m 39s                    | Table locks immediate | 208,312            |
|                |                              |                              |                           | Table locks waited    | 23,686             |
| 13-21          | 4m 35s                       | 5m 38s                       | 6m 6s                     | Table locks immediate | 198,511            |
|                |                              |                              |                           | Table locks waited    | 33,584             |
| 22-28          | N/A                          | 1m 54s                       | 1m 29s                    | Table locks immediate | 200,095            |
|                |                              |                              |                           | Table locks waited    | 0                  |

## Practice 7-3: Key Cache Effectiveness

### Overview

In this practice, you evaluate the key cache utilization by monitoring the `key_%` status variables. To accomplish this objective, you do the following:

- Modify the `key_cache_buffer_size` server variable to a comparatively small value.
- Execute multiple simple `SELECT` queries and calculate the effectiveness of the key cache by viewing the `key_blocks_unused` status variable.

### Assumptions

- The MySQL server is installed and running.
- The `world` database is installed.

### Duration

This practice should take 15 minutes to complete.

### Tasks

**Note:** A table has been provided at the end of the steps for this practice for you to fill in the values collected during the running of the steps. Entering the data from the steps into this table can assist you in comparing the values collected.

1. Restart the MySQL server as `root`.
2. Using the `mysql` client, reset the key cache by setting the global `key_buffer_size` server variable to 16K.
3. Using the `mysql` client, view the value that is assigned for the `key_cache_block_size` global server variable.
4. Using the `mysql` client, review the key cache state by viewing all the global status variables that start with the word `Key`.
  - What is the value of the `Key_blocks_unused` status variable?  
\_\_\_\_\_
  - What is the value of the `Key_blocks_used` status variable?  
\_\_\_\_\_
  - What is the value of the `Key_read_requests` status variable?  
\_\_\_\_\_
  - What is the value of the `Key_reads` status variable?  
\_\_\_\_\_
5. Using the `mysql` client, select `Name` field from the `Country` table in the `world` database all the records that have the value `SWE` in the `Code` field.

6. Using the `mysql` client, review the key cache state by viewing all the global status variables that start with the word `Key`.
  - What is the value of the `Key_blocks_unused` status variable?  
\_\_\_\_\_
  - What is the value of the `Key_blocks_used` status variable?  
\_\_\_\_\_
  - What is the value of the `Key_read_requests` status variable?  
\_\_\_\_\_
  - What is the value of the `Key_reads` status variable?  
\_\_\_\_\_
  - Using these values, calculate the key cache utilization by using the following formula:  
$$1 - ((\text{Key\_blocks\_unused} * \text{key\_cache\_block\_size}) / \text{key\_buffer\_size})$$
.

---

7. Using the `mysql` client, select `Name` field from the `Country` table in the `world` database all the records that have the value `SWZ` in the `Code` field.
8. Using the `mysql` client, review the key cache state by viewing all the global status variables that start with the word `Key`.
  - What is the value of the `Key_blocks_unused` status variable?  
\_\_\_\_\_
  - What is the value of the `Key_blocks_used` status variable?  
\_\_\_\_\_
  - What is the value of the `Key_read_requests` status variable?  
\_\_\_\_\_
  - What is the value of the `Key_reads` status variable?  
\_\_\_\_\_
  - Using these values, calculate the key cache utilization by using the following formula:  
$$1 - ((\text{Key\_blocks\_unused} * \text{key\_cache\_block\_size}) / \text{key\_buffer\_size})$$
.

---

9. Using the `mysql` client, select `Name` field from the `Country` table in the `world` database all the records that have the value `FIN` in the `Code` field.

10. Using the `mysql` client, review the key cache state by viewing all the global status variables that start with the word `Key`.

- What is the value of the `Key_blocks_unused` status variable?

---

- What is the value of the `Key_blocks_used` status variable?

---

- What is the value of the `Key_read_requests` status variable?

---

- What is the value of the `Key_reads` status variable?

---

- Using these values, calculate the key cache utilization by using the following formula:  
$$1 - ((\text{Key\_blocks\_unused} * \text{key\_cache\_block\_size}) / \text{key\_buffer\_size})$$

---

| Step(s) | <code>Key_blocks_unused</code> | <code>Key_blocks_used</code> | <code>Key_read_request</code> | <code>Key_reads</code> | <code>Key Cache Utilization</code> |
|---------|--------------------------------|------------------------------|-------------------------------|------------------------|------------------------------------|
| 4       |                                |                              |                               |                        |                                    |
| 5-6     |                                |                              |                               |                        |                                    |
| 7-8     |                                |                              |                               |                        |                                    |
| 9-10    |                                |                              |                               |                        |                                    |

## Solutions 7-3: Key Cache Effectiveness

### Tasks

**Note:** These solutions are sample solutions and are not expected to be the exact output you see in your execution of the steps.

1. Restart the MySQL server as root.

```
sys> /etc/init.d/mysql restart
```

2. Using the mysql client, reset the key cache by setting the global key\_buffer\_size server variable to 16K.

```
sys> mysql -uroot -poracle -e"SET GLOBAL \
> key_buffer_size=16*1024;"
```

3. Using the mysql client, view the value that is assigned for the key\_cache\_block\_size global server variable.

```
sys> mysql -uroot -poracle -e"SHOW GLOBAL VARIABLES LIKE \
> 'key_cache_block_size';"

+-----+-----+
| Variable_name | Value |
+-----+-----+
| key_cache_block_size | 1024 |
+-----+-----+
```

4. Using the mysql client, review the key cache state by viewing all the global status variables that start with the word Key.

```
sys> mysql -uroot -poracle -e"SHOW GLOBAL STATUS LIKE 'Key%';"

+-----+-----+
| Variable_name | Value |
+-----+-----+
| Key_blocks_not_flushed | 0
| Key_blocks_unused | 14
| Key_blocks_used | 0
| Key_read_requests | 0
| Key_reads | 0
| Key_write_requests | 0
| Key_writes | 0
+-----+-----+
```

- What is the value of the Key\_blocks\_unused status variable?

14

- What is the value of the Key\_blocks\_used status variable?

0

- What is the value of the Key\_read\_requests status variable?

0

- What is the value of the Key\_reads status variable?

0

5. Using the mysql client, select Name field from the Country table in the world database all the records that have the value SWE in the Code field.

```
sys> mysql -uroot -poracle -e"SELECT Name FROM world.Country\
> WHERE Code = 'SWE';"

+-----+
| Name |
+-----+
| Sweden |
+-----+
```

6. Using the mysql client, review the key cache state by viewing all the global status variables that start with the word Key.

```
sys> mysql -uroot -poracle -e"SHOW GLOBAL STATUS LIKE 'Key%';"

+-----+-----+
| Variable_name | Value |
+-----+-----+
| Key_blocks_not_flushed | 0
| Key_blocks_unused | 12
| Key_blocks_used | 2
| Key_read_requests | 2
| Key_reads | 2
| Key_write_requests | 0
| Key_writes | 0
+-----+-----+
```

- What is the value of the Key\_blocks\_unused status variable?

12

- What is the value of the Key\_blocks\_used status variable?

2

- What is the value of the Key\_read\_requests status variable?

2

- What is the value of the Key\_reads status variable?

2

- Using these values, calculate the key cache utilization by using the following formula:  
 $1 - ((\text{Key\_blocks\_unused} * \text{key\_cache\_block\_size}) / \text{key\_buffer\_size})$ .

$1 - ((12 * 1024) / 16384) = .25 \text{ or } 25\% \text{ utilization}$

7. Using the mysql client, select Name field from the Country table in the world database all the records that have the value SWZ in the Code field.

```
sys> mysql -uroot -poracle -e"SELECT Name FROM world.Country\
> WHERE Code = 'SWZ';"

+-----+
| Name |
+-----+
| Swaziland |
+-----+
```

8. Using the mysql client, review the key cache state by viewing all the global status variables that start with the word Key.

```
sys> mysql -uroot -poracle -e"SHOW GLOBAL STATUS LIKE 'Key%';"

+-----+-----+
| Variable_name | Value |
+-----+-----+
| Key_blocks_not_flushed | 0
| Key_blocks_unused | 12
| Key_blocks_used | 2
| Key_read_requests | 4
| Key_reads | 2
| Key_write_requests | 0
| Key_writes | 0
+-----+-----+
```

- What is the value of the `Key_blocks_unused` status variable?

12

- What is the value of the `Key_blocks_used` status variable?

2

- What is the value of the `Key_read_requests` status variable?

4

- What is the value of the `Key_reads` status variable?

2

- Using these values, calculate the key cache utilization by using the following formula:  
 $1 - ((\text{Key_blocks_unused} * \text{key_cache_block_size}) / \text{key_buffer_size})$ .

$1 - ((12 * 1024) / 16384) = .25 \text{ or } 25\% \text{ utilization}$

9. Using the `mysql` client, select `Name` field from the `Country` table in the `world` database all the records that have the value `FIN` in the `Code` field.

```
sys> mysql -uroot -poracle -e"SELECT Name FROM world.Country\
> WHERE Code = 'FIN';"

+-----+
| Name |
+-----+
| Finland |
+-----+
```

10. Using the `mysql` client, review the key cache state by viewing all the global status variables that start with the word `Key`.

```
sys> mysql -uroot -poracle -e"SHOW GLOBAL STATUS LIKE 'Key%';"

+-----+-----+
| Variable_name | Value |
+-----+-----+
| Key_blocks_not_flushed | 0
| Key_blocks_unused | 11
| Key_blocks_used | 3
| Key_read_requests | 6
| Key_reads | 3
| Key_write_requests | 0
| Key_writes | 0
+-----+-----+
```

- What is the value of the `Key_blocks_unused` status variable?

11

- What is the value of the `Key_blocks_used` status variable?

3

- What is the value of the `Key_read_requests` status variable?

6

- What is the value of the `Key_reads` status variable?

3

Using these values, calculate the key cache utilization by using the following formula:  
 $1 - ((\text{Key\_blocks\_unused} * \text{key\_cache\_block\_size}) / \text{key\_buffer\_size})$ .

$$1 - ((11 * 1024) / 16384) = .3125 \text{ or } 31\% \text{ utilization}$$

| Step(s) | Key_blocks_unused | Key_blocks_used | Key_read_request | Key_reads | Key Cache Utilization |
|---------|-------------------|-----------------|------------------|-----------|-----------------------|
| 4       | 14                | 0               | 0                | 0         | 25%                   |
| 5-6     | 12                | 2               | 2                | 2         | 25%                   |
| 7-8     | 12                | 2               | 4                | 2         | 25%                   |
| 9-10    | 11                | 3               | 6                | 3         | 31%                   |

## Practice 7-4: Full-Text Indexing

### Overview

In this practice, you create a fulltext index and compare the difference in using NATURAL LANGUAGE and BOOLEAN mode. To accomplish this objective, you do the following:

- Create a table that contains a full-text index.
  - This table is loaded with 2 million records.
- Search against the `fulltext` table using NATURAL LANGUAGE and BOOLEAN mode using multiple `SELECT` statements.

### Assumptions

- The MySQL server is installed and running.
- The `many_tables` database is installed.
- The `PT_Stress.php` and `city_huge.sql` files are located in the `/stage/scripts` directory.

### Duration

This practice should take 15 minutes to complete.

### Tasks

1. Using the `mysql` client, execute the following to create a table with 2 million records that contains a full-text index:

```
sys> mysql -uroot -poracle -e"USE many_tables;\
> DROP TABLE IF EXISTS city_fulltext;\
> CREATE TABLE city_fulltext LIKE city_huge;\
> ALTER TABLE city_fulltext DROP INDEX CountryCode,\
> DROP INDEX Name, DROP INDEX Population;\
> ALTER TABLE city_fulltext ADD FULLTEXT(Name,District);\
> INSERT INTO city_fulltext SELECT * FROM city_huge;"
```

**Note:** This command runs for approximately one minute.

2. Using the `mysql` client, execute the following to search for all the records in the `city_fulltext` table `Name` column that start with the word 'San ' (includes a space):

```
sys> mysql -uroot -poracle -e"SELECT COUNT(*) FROM \
> many_tables.city_fulltext WHERE Name LIKE 'San %';"
```

- How many records did the query return?

3. Using the `mysql` client, execute the following to search for all the records in the `city_fulltext` table that contain the word 'San' using the full-text index created in step 1:

```
sys> mysql -uroot -poracle -e"SELECT COUNT(*) FROM \
> many_tables.city_fulltext WHERE MATCH(Name,District) \
> AGAINST('San') ;"
```

- How many records did the query return?  
\_\_\_\_\_
- Is there a difference in the number of records returned? If so, why do you believe the values are different?  
\_\_\_\_\_
- \_\_\_\_\_
- \_\_\_\_\_

4. Edit the `/etc/init.d/my.cnf` file as `root` and add the following line to the end of the server variables in the `[mysqld]` section:

```
ft_min_word_len = 3
```

- 5. Save and close out the `/etc/init.d/my.cnf` file.
- 6. Restart the MySQL server as `root`.
- 7. Using the `mysql` client, complete the tasks from step 1 to re-create the full-text index.
- 8. In the `/stage/scripts` directory, use the `PT_Stress.php` script to execute the following to select the first thousand records that contain the word 'San' in either the `Name` or `District` columns using the natural language mode (default mode) with the `SELECT` statement repeating 100 times:

```
sys> ./PT_Stress.php -u root -p oracle -s \
> "Q:SELECT * FROM many_tables.city_fulltext WHERE \
> MATCH(Name,District) AGAINST ('San') LIMIT 1000" \
> -t -i 100 -c 10 -q
```

- What is the **average** amount of time it took to complete the task?  
\_\_\_\_\_
- 9. In the `/stage/scripts` directory, use the `PT_Stress.php` script to execute the following to select the first thousand records that contain the word 'San' in either the `Name` or `District` columns using the boolean language mode with the `SELECT` statement repeating 100 times:

```
sys> ./PT_Stress.php -u root -p oracle -s \
> "Q:SELECT * FROM many_tables.city_fulltext WHERE \
> MATCH(Name,District) AGAINST ('San' IN BOOLEAN MODE) LIMIT \
> 1000" -t -i 100 -c 10 -q
```

- What is the **average** amount of time it took to complete the task?  
\_\_\_\_\_

## Solutions 7-4: Full-Text Indexing

### Tasks

**Note:** These solutions are sample solutions and are not expected to be the exact output you see in your execution of the steps.

- Using the `mysql` client, execute the following to create a table with 2 million records that contains a full-text index:

```
sys> mysql -uroot -poracle -e"USE many_tables; \
> DROP TABLE IF EXISTS city_fulltext; \
> CREATE TABLE city_fulltext LIKE city_huge; \
> ALTER TABLE city_fulltext DROP INDEX CountryCode, \
> DROP INDEX Name, DROP INDEX Population; \
> ALTER TABLE city_fulltext ADD FULLTEXT(Name,District); \
> INSERT INTO city_fulltext SELECT * FROM city_huge;"
```

**Note:** This command runs for approximately one minute.

- Using the `mysql` client, execute the following to search for all the records in the `city_fulltext` table `Name` column that start with the word 'San' (includes a space):

```
sys> mysql -uroot -poracle -e"SELECT COUNT(*) FROM \
> many_tables.city_fulltext WHERE Name LIKE 'San %';"
+-----+
| COUNT(*) |
+-----+
| 30320 |
+-----+
```

- How many records did the query return?

30,320

- Using the `mysql` client, execute the following to search for all the records in the `city_fulltext` table that contain the word 'San' using the full-text index created in step 1:

```
sys> mysql -uroot -poracle -e"SELECT COUNT(*) FROM \
> many_tables.city_fulltext WHERE MATCH(Name,District) \
> AGAINST('San') ;"
+-----+
| COUNT(*) |
+-----+
| 0 |
+-----+
```

- How many records did the query return?

0

- Is there a difference in the number of records returned? If so, why do you believe the values are different?

Yes, there is a difference, no records were returned. The default minimum length for words that are included in a full-text index is four characters in length. Because the search was against a word three characters in length, the full-text index would not include this word.

4. Edit the /etc/init.d/my.cnf file as root and add the following line to the end of the server variables in the [mysqld] section:

```
ft_min_word_len = 3
```

5. Save and close out the /etc/init.d/my.cnf file.  
6. Restart the MySQL server as root.

```
/etc/init.d/mysql restart
```

7. Using the mysql client, execute the following to create a table with 2 million records that contains a full-text index:

```
sys> mysql -uroot -poracle -e"USE many_tables;\
> DROP TABLE IF EXISTS city_fulltext;\
> CREATE TABLE city_fulltext LIKE city_huge;\
> ALTER TABLE city_fulltext DROP INDEX CountryCode, \
> DROP INDEX Name, DROP INDEX Population; \
> ALTER TABLE city_fulltext ADD FULLTEXT(Name,District); \
> INSERT INTO city_fulltext SELECT * FROM city_huge;"
```

**Note:** This command runs for approximately one minute.

8. In the /stage/scripts directory, use the PT\_Stress.php script to execute the following to select the first thousand records that contain the word 'San' in either the Name or District columns using the natural language mode (default mode) with the SELECT statement repeating 100 times using 10 connections:

```
sys> ./PT_Stress.php -u root -p oracle -s \
> "Q:SELECT * FROM many_tables.city_fulltext WHERE \
> MATCH(Name,District) AGAINST ('San') LIMIT 1000" \
> -t -i 100 -c 10 -q
```

|                 |           |
|-----------------|-----------|
| Execution Times |           |
| Minimum:        | 35.560963 |
| Maximum:        | 52.177719 |
| Average:        | 45.893405 |

- What is the average amount of time it took to complete the task?

45.89 seconds

9. In the /stage/scripts directory, use the PT\_Stress.php script to execute the following to select the first thousand records that contain the word 'San' in either the Name or District columns using the boolean language mode with the SELECT statement repeating 100 times using 10 connections:

```
sys> ./PT_Stress.php -u root -p oracle -s \
> "Q:SELECT * FROM many_tables.city_fulltext WHERE \
> MATCH(Name,District) AGAINST ('San' IN BOOLEAN MODE) LIMIT \
> 1000" -t -i 100 -c 10 -q
```

|                 |           |
|-----------------|-----------|
| Execution Times |           |
| Minimum:        | 10.380442 |
| Maximum:        | 17.433387 |
| Average:        | 14.583824 |

- What is the average amount of time it took to complete the task?

14.58 seconds

# **Practices for Lesson 8: Other MySQL Storage Engines and Issues**

## **Chapter 8**

## Practices for Lesson 8

---

### Practices Overview

In these practices, you evaluate the effect that storage engines have on the storage and retrieval of large objects. In addition, you evaluate the MEMORY storage engine for performance and memory consumption.

## Practice 8-1: Large Objects

In this practice, you evaluate the effects of large objects on InnoDB and MyISAM storage engines. To accomplish this objective, you do the following:

- Use the `PT_Stress.php` script to execute a query that displays a large object column for every record in an InnoDB and MyISAM and compare the results.
- Use the `PT_Stress.php` script to execute a query that displays a variable character column for every record (that also includes a large object column) in an InnoDB and MyISAM and compare the results.

### Assumptions

- The MySQL server is installed and running.
- The `BLOB_InnoDB` and `BLOB_MyISAM` tables are located in the `many_tables` database.
- The `PT_Stress.php` script is accessible and located in the `/stage/script` directory.

### Duration

This practice should take 15 minutes to complete.

### Tasks

**Note:** A table has been provided at the end of the steps for this practice for you to fill in the values collected during the running of the steps. Entering the data from the steps into this table can assist you in comparing the values collected.

1. Open a terminal window, change into the `/stage/scripts` directory, and use the `PT_Stress.php` script to issue the following command to evaluate the time it would take to list every picture column in the `BLOB_InnoDB` table in the `many_tables` database:

```
sys> ./PT_Stress.php -u root -p oracle -s \
> "Q:SELECT picture FROM many_tables.BLOB_InnoDB" -t
```

**Note:** The picture column in the `BLOB_InnoDB` table is a Large Object data type.

The `BLOB_InnoDB` table uses the InnoDB storage engine.

- What is the amount of time that this task took to complete?

- 
2. In the `/stage/scripts`, use the `PT_Stress.php` script to issue the following command to evaluate the time it would take to list every picture column in the `BLOB_MyISAM` table in the `many_tables` database:

```
sys> ./PT_Stress.php -u root -p oracle -s \
> "Q:SELECT picture FROM many_tables.BLOB_MyISAM" -t
```

**Note:** The picture column in the `BLOB_MyISAM` table is a Large Object data type.

The `BLOB_MyISAM` table uses the MyISAM storage engine.

- What is the amount of time that this task took to complete?
-

3. In the /stage/scripts, use the `PT_Stress.php` script to issue the following command to evaluate the time it would take to list every `last_name` column in the `BLOB_InnoDB` table in the `many_tables` database (with an iteration of 10 times with 10 connections running the script):

```
sys> ./PT_Stress.php -u root -p oracle -s \
> "Q:SELECT last_name FROM many_tables.BLOB_InnoDB" \
> -i 10 -c 10 -t
```

**Note:** The `last_name` column in the `BLOB_InnoDB` table is a variable character data type.

- What is the average amount of time that each iteration took to complete?

4. In the /stage/scripts, use the `PT_Stress.php` script to issue the following command to evaluate the time it would take to list every `last_name` column in the `BLOB_MyISAM` table in the `many_tables` database (with an iteration of 10 times with 10 connections running the script):

```
sys> ./PT_Stress.php -u root -p oracle -s \
> "Q:SELECT last_name FROM many_tables.BLOB_MyISAM" \
> -i 10 -c 10 -t
```

**Note:** The `last_name` column in the `BLOB_MyISAM` table is a variable character data type.

- What is the average amount of time that each iteration took to complete?

5. Why did the InnoDB table perform better when the `last_name` column was the only column output for the query in step 3 over the same query executed against the MyISAM table in step 4?

| Step(s) | Command Executed                  | BLOB_InnoDB time | BLOB_MyISAM time |
|---------|-----------------------------------|------------------|------------------|
| 1-2     | <code>SELECT picture ...</code>   |                  |                  |
| 3-4     | <code>SELECT last_name ...</code> |                  |                  |

## Solutions 8-1: Large Objects

### Tasks

**Note:** These solutions are sample solutions and are not expected to be the exact output you see in your execution of the steps.

1. Open a terminal window, change into the /stage/scripts directory, and use the PT\_Stress.php script to issue the following command to evaluate the time it would take to list every picture column in the BLOB\_InnoDB table in the many\_tables database:

```
sys> ./PT_Stress.php -u root -p oracle -s \
> "Q:SELECT picture FROM many_tables.BLOB_InnoDB" -t
Simulating 1 connection(s) against the SELECT picture FROM
many_tables.BLOB_InnoDB script repeating 1 times!
Execution Times
Minimum: 26.574632
Maximum: 26.574632
Average: 26.574632
```

**Note:** The picture column in the BLOB\_InnoDB table is a Large Object data type. The BLOB\_InnoDB table uses the InnoDB storage engine.

- What is the amount of time that this task took to complete?

26.57 seconds

2. In the /stage/scripts, use the PT\_Stress.php script to issue the following command to evaluate the time it would take to list every picture column in the BLOB\_MyISAM table in the many\_tables database:

```
sys> ./PT_Stress.php -u root -p oracle -s \
> "Q:SELECT picture FROM many_tables.BLOB_MyISAM" -t
Simulating 1 connection(s) against the SELECT picture FROM
many_tables.BLOB_MyISAM script repeating 1 times!
Execution Times
Minimum: 13.964781
Maximum: 13.964781
Average: 13.964781
```

**Note:** The picture column in the BLOB\_MyISAM table is a Large Object data type. The BLOB\_MyISAM table uses the MyISAM storage engine.

- What is the amount of time that this task took to complete?

13.96 seconds

3. In the /stage/scripts, use the PT\_Stress.php script to issue the following command to evaluate the time it would take to list every last\_name column in the BLOB\_InnoDB table in the many\_tables database (with an iteration of 10 times with 10 connections running the script):

```
sys> ./PT_Stress.php -u root -p oracle -s \
> "Q:SELECT last_name FROM many_tables.BLOB_InnoDB" \
> -i 10 -c 10 -t
Simulating 10 connection(s) against the SELECT last_name FROM
many_tables.BLOB_InnoDB script repeating 10 times!
Execution Times
Minimum: 2.844935
Maximum: 3.673456
Average: 3.349234
```

**Note:** The last\_name column in the BLOB\_InnoDB table is a variable character data type.

- What is the average amount of time that each iteration took to complete?

3.35 seconds

4. In the /stage/scripts, use the PT\_Stress.php script to issue the following command to evaluate the time it would take to list every last\_name column in the BLOB\_MyISAM table in the many\_tables database (with an iteration of 10 times with 10 connections running the script):

```
sys> ./PT_Stress.php -u root -p oracle -s \
> "Q:SELECT last_name FROM many_tables.BLOB_MyISAM" \
> -i 10 -c 10 -t
Simulating 10 connection(s) against the SELECT last_name FROM
many_tables.BLOB_MyISAM script repeating 10 times!
Execution Times
Minimum: 20.339358
Maximum: 22.597595
Average: 21.043681
```

**Note:** The last\_name column in the BLOB\_MyISAM table is a variable character data type.

- What is the average amount of time that each iteration took to complete?

21.04 seconds

5. Why did the InnoDB table perform better when the `last_name` column was the only column output for the query in step 3 over the same query executed against the MyISAM table in step 4?

InnoDB skips BLOB columns if not requested in SELECT statement where MyISAM tables will read the entire record (including BLOBS).

| Step(s) | Command Executed                  | BLOB_InnoDB time | BLOB_MyISAM time |
|---------|-----------------------------------|------------------|------------------|
| 1-2     | <code>SELECT picture ...</code>   | 26.57s           | 13.96s           |
| 3-4     | <code>SELECT last_name ...</code> | 3.35s            | 21.04s           |

## Practice 8-2: MEMORY Storage Engine

In this practice, you evaluate the MEMORY storage engine for performance and memory consumption. To accomplish this objective, you do the following:

- Create the `city_huge` database based on the MyISAM storage engine and run scripts against it to test performance and memory consumption.
- Create the `city_huge` database based on the MEMORY storage engine and run scripts against it to test performance and memory consumption.

### Assumptions

- The MySQL server is installed and running.
- The `many_tables` database is installed.
- The `PT_Stress.php` script and `city_huge.sql` files are located in the `/stage/scripts` directory.

### Duration

This practice should take 20 minutes to complete.

### Pre-requisite Task

1. Open a terminal window, start the `mysql` client, and select the `many_tables` database.
2. To prevent a warning from being displayed in the following steps, execute the following in the `mysql` client to tell the MySQL server to use the `MIXED` binary log format:

```
mysql> SET binlog_format = 'MIXED';
```

### Tasks

**Note:** A table has been provided at the end of the steps for this practice for you to fill in the values collected during the running of the steps. Entering the data from the steps into this table can assist you in comparing the values collected.

1. Open a terminal window and use the `mysql` client to set the `max_heap_table_size` server setting to 1 GB.

**Note:** You can use the existing terminal window from the previous practice. However, at this point, you should have only one terminal window open to minimize confusion as more terminal windows are opened up.

2. Using the `mysql` client, issue the following command to remove all the records in the `city_huge` table in the `many_table` database, alter the storage engine to use the MyISAM engine, and then load 2 million records into the `city_huge` table:

```
sys> time mysql -uroot -poracle -e"USE many_tables;\
> TRUNCATE city_huge;\
> ALTER TABLE city_huge ENGINE=MYISAM;\
> ALTER TABLE city_huge DISABLE KEYS;\
> SOURCE /stage/scripts/city_huge.sql;\
> ALTER TABLE city_huge ENABLE KEYS;"
```

**Note:** The `time` command precedes the running of the `mysql` command to provide you an approximation of the time that the server took to complete the tasks.

- What is the approximate time that it took the `mysql` client to complete the tasks?
- 
3. In the `/stage/scripts` directory use the `PT_Stress.php` script to issue the following command to evaluate the time it would take to list every `picture` column in the `BLOB_InnoDB` table in the `many_tables` database: Using the `PT_Stress.php` script, execute 10 threads running a single SQL script using a random number 25,000 times. The random number can not exceed 2,000,000:
- ```
sys> /stage/scripts/PT_Stress.php -u root -p oracle -s \
> "Q:SELECT id FROM many_tables.city_huge WHERE ID={RANDOM};" \
> -r 2000000 -t -i 100000 -c 10 -q
```
- While this is executing, open a second terminal window and complete the task in the next step.
- Note:** This command runs for approximately 3 minutes.
4. In a separate terminal window, execute the following command to obtain a dynamic real-time view of the `mysqld` application:
- ```
sys> top | grep mysqld
```
- What is the approximate value of the virtual memory (VIRT column) of the `top` output on your system:

---

  - What is the approximate value of the resident memory (RES Column) of the `top` output on your system:
- 
5. In the first terminal window, view the output of the `PT_Stress.php` script.
- What is the average execution time for the `PT_Stress.php` script?
- 
6. Using the `mysql` client, issue the following command to remove all the records in the `city_huge` table in the `many_table` database, alter the storage engine to use the `MEMORY` engine, and then load 2 million records into the `city_huge` table:
- ```
sys> time mysql -uroot -poracle -e"USE many_tables; \
> TRUNCATE city_huge; \
> ALTER TABLE city_huge ENGINE=MEMORY; \
> ALTER TABLE city_huge DISABLE KEYS; \
> SOURCE /stage/scripts/city_huge.sql; \
> ALTER TABLE city_huge ENABLE KEYS;"
```
- What is the approximate time that it took the `mysql` client to complete the tasks?
-
7. Using the `PT_Stress.php` script, execute 10 threads running a single SQL script using a random number 25,000 times. The random number can not exceed 2,000,000.
- ```
sys> /stage/scripts/PT_Stress.php -u root -p oracle -s \
> "Q:SELECT id FROM many_tables.city_huge WHERE ID={RANDOM};" \
> -r 2000000 -t -i 100000 -c 10 -q
```

- While this is executing, open a second terminal window and complete the task in the next step.

**Note:** This command runs for approximately 3 minutes.

- In a separate terminal window, execute the following command to obtain a dynamic real-time view of the mysqld application:

```
sys> top | grep mysqld
```

- What is the approximate value of the virtual memory (VIRT column) of the top output on your system:

- 
- What is the approximate value of the resident memory (RES Column) of the top output on your system:

- In the first terminal window, view the output of the PT\_Stress.php script.

- What is the average execution time for the PT\_Stress.php script?

---

| Step(s) | Storage Engine | Avg. Time | VIRT | RES |
|---------|----------------|-----------|------|-----|
| 2-5     | MyISAM         |           |      |     |
| 6-9     | MEMORY         |           |      |     |

## Solution 8-2: MEMORY Storage Engine

### Tasks

**Note:** These solutions are sample solutions and are not expected to be the exact output you see in your execution of the steps.

1. Open a terminal window and use the mysql client to set the max\_heap\_table\_size server setting to 1 GB.

```
sys> mysql -uroot -poracle -e"SET GLOBAL max_heap_table_size = \
> 100000 * 1024"
```

**Note:** You can use the existing terminal window from the previous practice. However, at this point, you should have only one terminal window open to minimize confusion as more terminal windows are opened up.

2. Using the mysql client, issue the following command to remove all the records in the city\_huge table in the many\_table database, alter the storage engine to use the MyISAM engine, and then load 2 million records into the city\_huge table:

```
sys> time mysql -uroot -poracle -e"USE many_tables; \
> TRUNCATE city_huge; \
> ALTER TABLE city_huge ENGINE=MYISAM; \
> ALTER TABLE city_huge DISABLE KEYS; \
> SOURCE /stage/scripts/city_huge.sql; \
> ALTER TABLE city_huge ENABLE KEYS;"

real 0m50.572s
user 0m2.785s
sys 0m0.204s
```

**Note:** The time command precedes the running of the mysql command to provide you an approximation of the time that the server took to complete the tasks.

- What is the approximate time that it took the mysql client to complete the tasks?

51 seconds

3. Using the PT\_Stress.php script, execute 10 threads running a single SQL script using a random number 25,000 times. The random number can not exceed 2,000,000.

```
sys> /stage/scripts/PT_Stress.php -u root -p oracle -s \
> "Q:SELECT id FROM many_tables.city_huge WHERE ID={RANDOM}; " \
> -r 2000000 -t -i 100000 -c 10 -q
```

- While this is executing, open a second terminal window and complete the task in the next step.

**Note:** This command runs for approximately 3 minutes.

4. In a separate terminal window, execute the following command to obtain a dynamic real-time view of the mysqld application:

```
sys> top | grep mysqld
29866 mysql 15 0 722m 120m 4360 S 136.9 3.4 17:04.39 mysqld
29866 mysql 15 0 722m 120m 4360 S 140.2 3.4 17:08.62 mysqld
29866 mysql 15 0 722m 120m 4360 S 138.2 3.4 17:12.79 mysqld
29866 mysql 15 0 722m 120m 4360 S 135.5 3.4 17:16.88 mysqld
29866 mysql 15 0 722m 120m 4360 S 138.2 3.4 17:21.05 mysqld
29866 mysql 15 0 722m 120m 4360 S 137.8 3.4 17:25.21 mysqld
29866 mysql 15 0 722m 120m 4360 S 128.6 3.4 17:29.09 mysqld
29866 mysql 15 0 722m 120m 4360 S 128.9 3.4 17:32.98 mysqld
```

- What is the approximate value of the virtual memory (VIRT column) of the top output on your system:

722m

- What is the approximate value of the resident memory (RES Column) of the top output on your system:

120m

5. In the first terminal window, view the output of the PT\_Stress.php script.

```
Simulating 10 connection(s) against the SELECT id FROM many_tables.city_huge
WHERE ID={RANDOM}; script repeating 100000 times!
```

```
Execution Times
Minimum: 206.292713
Maximum: 211.079379
Average: 209.588483
```

- What is the average execution time for the PT\_Stress.php script?

209.59 seconds or 3 ½ minutes

6. Using the mysql client, issue the following command to remove all the records in the city\_huge table in the many\_table database, alter the storage engine to use the MEMORY engine, and then load 2 million records into the city\_huge table:

```
sys> time mysql -uroot -poracle -e"USE many_tables; \
> TRUNCATE city_huge; \
> ALTER TABLE city_huge ENGINE=MEMORY; \
> ALTER TABLE city_huge DISABLE KEYS; \
> SOURCE /stage/scripts/city_huge.sql; \
> ALTER TABLE city_huge ENABLE KEYS;" \
real 0m24.333s
user 0m2.776s
sys 0m0.207s
```

- What is the approximate time that it took the mysql client to complete the tasks?

24 seconds

7. Using the `PT_Stress.php` script, execute 10 threads running a single SQL script using a random number 25,000 times. The random number can not exceed 2,000,000.

```
sys> /stage/scripts/PT_Stress.php -u root -p oracle -s \
> "Q:SELECT id FROM many_tables.city_huge WHERE ID={RANDOM};"
> -r 2000000 -t -i 100000 -c 10 -q
```

- While this is executing, open a second terminal window and complete the task in the next step.

**Note:** This command runs for approximately 3 minutes.

8. In a separate terminal window, execute the following command to obtain a dynamic real-time view of the `mysqld` application:

```
sys> top | grep mysqld
29866 mysql 18 0 906m 303m 4360 S 129.6 8.6 19:42.65 mysqld
29866 mysql 18 0 906m 303m 4360 S 131.3 8.6 19:46.61 mysqld
29866 mysql 18 0 906m 303m 4360 S 131.3 8.6 19:50.57 mysqld
29866 mysql 18 0 906m 303m 4360 S 131.9 8.6 19:54.55 mysqld
29866 mysql 18 0 906m 303m 4360 S 129.2 8.6 19:58.45 mysqld
29866 mysql 18 0 906m 303m 4360 S 129.6 8.6 20:02.36 mysqld
29866 mysql 18 0 906m 303m 4360 S 130.6 8.6 20:06.30 mysqld
29866 mysql 18 0 906m 303m 4360 S 127.6 8.6 20:10.15 mysqld
```

- What is the approximate value of the virtual memory (VIRT column) of the `top` output on your system:

906m

- What is the approximate value of the resident memory (RES Column) of the `top` output on your system:

303m

9. In the first terminal window, view the output of the `PT_Stress.php` script.

```
Simulating 10 connection(s) against the SELECT id FROM many_tables.city_huge
WHERE ID={RANDOM}; script repeating 100000 times!
Execution Times
Minimum: 148.629331
Maximum: 184.222227
Average: 175.655009
```

- What is the average execution time for the `PT_Stress.php` script?

175.66 seconds or 3 minutes

| Step(s) | Storage Engine | Avg. Time | VIRT | RES  |
|---------|----------------|-----------|------|------|
| 2-5     | MyISAM         | 209.59s   | 722m | 120m |
| 6-9     | MEMORY         | 175.66s   | 906m | 303m |



## **Practices for Lesson 9: Schema Design and Performance**

**Chapter 9**

## Practices for Lesson 9

---

### Practices Overview

In these practices, you evaluate the effects of schema design on the performance of your databases.

## Practice 9-1: Schema Design

In this practice, you evaluate the effects of normalized and non-normalized data in a variety of scenarios. To accomplish this objective, you do the following:

- Use the `PT_Stress.php` script to execute queries that retrieve data from normalized and non-normalized tables that do not require the joining of normalized tables.
- Use the `PT_Stress.php` script to execute queries that retrieve data from normalized and non-normalized tables that require the joining of normalized tables.

### Assumptions

- The MySQL server is installed and running.
- The `world_all` table is located in the `world_NonNorm` database.
- The `PT_Stress.php` script is accessible and located in the `/stage/script` directory.

### Duration

This practice should take 15 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.
  - The `world_all` table is a combination of the `City`, `Country`, and `CountryLanguage` tables in a non-normalized form.
2. In the `/stage/scripts` directory, use the `PT_Stress.php` script to issue the following command to evaluate the time it would take to execute a `SELECT` statement against the `world_all` table in the `world_NonNorm` database that randomly selects records based on their population size. This `SELECT` statement runs five threads, with each thread executing the script 100 times each.

```
sys> ./PT_Stress.php -u root -p oracle \
> -s "Q:SELECT DISTINCT Country_Name, GovernmentForm FROM \
> world_NonNorm.world_all WHERE Country_Population < {RANDOM}" \
> -r 1500000000 -t -i 100 -c 5 -q
```

**Note:** This table is in a non-normalized form and has duplicated data. This duplicated data is eliminated from the output with the `SELECT DISTINCT` option. In addition, the maximum random number that can be created is 1.5 billion to ensure that the largest country population (1.2 billion) is included in the output.

3. In the `/stage/scripts` directory, use the `PT_Stress.php` script to issue the following command to evaluate the time it would take to execute a `SELECT` statement against the `Country` table in the `world` database that randomly selects records based on their population size. This `SELECT` statement runs five threads, with each thread executing the script 100 times each.

```
sys> ./PT_Stress.php -u root -p oracle \
> -s "Q:SELECT Name, GovernmentForm FROM world.Country \
> WHERE Population < {RANDOM}" -r 1500000000 -t -i 100 -c 5 -q
```

**Note:** The Country table in the world database is in a normalized form and contains the data we need to complete the SELECT statement with no duplicates in the output.

- What is the average amount of time that this task took to complete?

- 
4. In the /stage/scripts directory, use the PT\_Stress.php script to issue the following command to evaluate the time it would take to execute a SELECT statement against the world\_all table in the world\_NonNorm database that randomly selects records based on the percentage of the country that speaks a language. This SELECT statement runs two threads, with each thread executing the script 100 times each.

```
sys> ./PT_Stress.php -u root -p oracle -s \
> "Q:SELECT City_Name, Country_Name FROM
> world_NonNorm.world_all\
> WHERE Percentage < {RANDOM}" -r 101 -t -i 100 -c 2 -q
```

**Note:** The maximum random number that can be created is 101 to ensure that all languages (even those where 100% of the population spoke a language) could be included in the output.

- What is the average amount that each thread took to complete?

- 
5. In the /stage/scripts directory, use the PT\_Stress.php script to issue the following command to evaluate the time it would take to execute a SELECT statement against the City, Country, and CountryLanguage tables in the world database that randomly selects records based on the percentage of the country that speaks a language. This SELECT statement runs two threads, with each thread executing the script 100 times each.

```
sys> ./PT_Stress.php -u root -p oracle \
> -s "Q:SELECT City.Name, Country.Name FROM world.City, \
> world.Country, world.CountryLanguage \
> WHERE City.CountryCode = Country.Code \
> AND Country.Code = CountryLanguage.CountryCode \
> AND Percentage < {RANDOM}" -r 101 -t -i 100 -c 2 -q
```

**Note:** The output requires data from the City and Country tables and the WHERE statement requires data from the CountryLanguage table. This requires the three tables to be joined together to produce the correct output.

- What is the average amount that each thread took to complete?

- 
6. Review the values that were produced based on the type of query that was executed against the non-normalized and normalized tables. What do the results tell you?
- 
- 
-

## Solutions 9-1: Schema Design

---

### Tasks

**Note:** These solutions are sample solutions and are not expected to be the exact output you see in your execution of the 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.

```
sys> mysql -uroot -poracle -e"SHOW CREATE TABLE \
> world_NonNorm.world_all\G"
***** 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=30705 DEFAULT CHARSET=latin1
```

- The `world_all` table is a combination of the `City`, `Country`, and `CountryLanguage` tables in a non-normalized form.

2. In the /stage/scripts directory, use the PT\_Stress.php script to issue the following command to evaluate the time it would take to execute a SELECT statement against the world\_all table in the world\_NonNorm database that randomly selects records based on their population size. This SELECT statement runs five threads, with each thread executing the script 100 times each.

```
sys> ./PT_Stress.php -u root -p oracle \
> -s "Q:SELECT DISTINCT Country_Name, GovernmentForm FROM \
> world_NonNorm.world_all WHERE Country_Population < {RANDOM}" \
> -r 1500000000 -t -i 100 -c 5 -q
Simulating 5 connection(s) against the SELECT DISTINCT
Country_Name, GovernmentForm FROM world_NonNorm.world_all WHERE
Country_Population < {RANDOM} script repeating 100 times!
Execution Times
Minimum: 34.553338
Maximum: 38.488400
Average: 36.389518
```

**Note:** This table is in a non-normalized form and has duplicated data. This duplicated data is eliminated from the output with the SELECT DISTINCT option. In addition, the maximum random number that can be created is 1.5 billion to ensure that the largest country population (1.2 billion) is included in the output.

- What is the average amount that each thread took to complete?

36.39 seconds

3. In the /stage/scripts directory, use the PT\_Stress.php script to issue the following command to evaluate the time it would take to execute a SELECT statement against the Country table in the world database that randomly selects records based on their population size. This SELECT statement runs five threads, with each thread executing the script 100 times each.

```
sys> ./PT_Stress.php -u root -p oracle \
> -s "Q:SELECT Name, GovernmentForm FROM world.Country \
> WHERE Population < {RANDOM}" -r 1500000000 -t -i 100 -c 5 -q
Simulating 5 connection(s) against the SELECT Name,
GovernmentForm FROM world.Country WHERE Population < {RANDOM}
script repeating 100 times!
Execution Times
Minimum: 0.135462
Maximum: 0.253514
Average: 0.190530
```

**Note:** The Country table in the world database is in a normalized form with the Country table containing the data we need to complete the SELECT statement with no duplicates in the output.

- What is the average amount of time that this task took to complete?

0.19 seconds

4. In the /stage/scripts directory, use the PT\_Stress.php script to issue the following command to evaluate the time it would take to execute a SELECT statement against the world\_all table in the world\_NonNorm database that randomly selects records based on the percentage of the country that speaks a language. This SELECT statement runs two threads, with each thread executing the script 100 times each.

```
sys> ./PT_Stress.php -u root -p oracle -s \
> "Q:SELECT City_Name, Country_Name FROM \
> world_NonNorm.world_all \
> WHERE Percentage < {RANDOM}" -r 101 -t -i 100 -c 2 -q
Simulating 2 connection(s) against the SELECT City_Name,
Country_Name FROM world_NonNorm.world_all WHERE Percentage <
{RANDOM} script repeating 100 times!
Execution Times
Minimum: 9.819189
Maximum: 10.528301
Average: 10.173745
```

**Note:** The maximum random number that can be created is 101 to ensure that all languages (even those where 100% of the population spoke a language) could be included in the output.

- What is the average amount that each thread took to complete?

10.17 seconds

5. In the /stage/scripts directory, use the PT\_Stress.php script to issue the following command to evaluate the time it would take to execute a SELECT statement against the City, Country and CountryLanguage table in the world database that randomly selects records based on the percentage of the country that speaks a language. This SELECT statement runs two threads, with each thread executing the script 100 times each.

```
sys> ./PT_Stress.php -u root -p oracle \
> -s "Q:SELECT City.Name, Country.Name FROM world.City, \
> world.Country, world.CountryLanguage \
> WHERE City.CountryCode = Country.Code \
> AND Country.Code = CountryLanguage.CountryCode \
> AND Percentage < {RANDOM}" -r 101 -t -i 100 -c 2 -q
Simulating 2 connection(s) against the SELECT City.Name,
Country.Name FROM world.City, world.Country,
world.CountryLanguage WHERE City.CountryCode = Country.Code AND
Country.Code = CountryLanguage.CountryCode AND Percentage <
{RANDOM} script repeating 100 times!
Execution Times
Minimum: 41.789574
Maximum: 42.101719
Average: 41.945646
```

**Note:** The output requires data from the City and Country tables and the WHERE statement requires data from the CountryLanguage table. This requires the three tables to be joined together to produce the correct output.

- What is the average amount that each thread took to complete?

41.95 seconds

6. Review the values that were produced based on the type of query that was executed against the non-normalized and normalized tables. What do the results tell you?

Most of the time, a normalized table produces the best results, provides more flexibility, and eliminates potential for errors to creep into the data. However, when there is a need to join multiple tables together to produce the output desired, a non-normalized table can produce better results.

## Practice 9-2: Data Types

In this practice, you evaluate the effects of data types in a variety of scenarios. To accomplish this objective, you do the following:

- View the size of specific tables in the /var/lib/mysql/isfdb directory.
- Modify a DATE data type column to a DATETIME data type column and determine the effect on the size of the table.
- Create multiple copies of the same table that contain different string data types and determine the effect on the size of the table.
- Use the PT\_Stress.php script to execute queries against the different copies of the table to determine the effect that string data types have on performance.

### Assumptions

- The MySQL server is installed and running.
- The isfdb database is installed.
- The PT\_Stress.php script is accessible and located in the /stage/script directory.

### Duration

This practice should take 30 minutes to complete.

### Tasks

1. Open a terminal window and as root view the file sizes of all the files associated with the authors table in the isfdb database by issuing the following command in a terminal window:

```
sys> ls -alt /var/lib/mysql/isfdb/authors*
```

- What is the size of the authors.MYD file?

---

**Note:** You can use the existing terminal window from the previous practice. However, at this point, you should have only one terminal window open to minimize confusion as more terminal windows are opened up.

2. Open a second terminal window and issue the following command to modify the birthdate column in the authors table in the isfdb database to use the DATETIME data type:

```
sys> mysql -uroot -poracle -e"ALTER TABLE isfdb.authors \
> MODIFY author_birthdate DATETIME DEFAULT NULL"
```

3. In the first terminal window, as root issue the command from step 1 to view all the files associated with the authors table in the isfdb database.

```
sys> ls -alt /var/lib/mysql/isfdb/authors*
```

- What is the size of the authors.MYD file?

4. In the second terminal window, using the mysql client view the table design of the history table in the isfdb database.

5. In the second terminal window, using the `mysql` client view the maximum value of the `history_from` column in the `history` table in the `isfdb` database.

– What is the maximum length of the data in the `history_from` column?

---

6. In the second terminal window, using the `mysql` client view the maximum value of the `history_to` column in the `history` table in the `isfdb` database.

– What is the maximum length of the data in the `history_to` column?

---

7. In the second terminal window, using the `mysql` client issue the following command to create a copy of the `history` table in the `isfdb` database called `history_char` that contains CHAR columns instead of mediumtext columns:

```
sys> mysql -uroot -poracle -e"USE isfdb;\
> CREATE TABLE history_char LIKE history;\
> ALTER TABLE history_char MODIFY history_from CHAR(199);\
> ALTER TABLE history_char MODIFY history_to CHAR(226);\
> INSERT INTO history_char SELECT * FROM history;"
```

8. In the second terminal window, using the `mysql` client issue the following command to create a copy of the `history` table in the `isfdb` database called `history_varchar` that contains CHAR columns instead of mediumtext columns:

```
sys> mysql -uroot -poracle -e"USE isfdb;\
> CREATE TABLE history_varchar LIKE history;\
> ALTER TABLE history_varchar MODIFY history_from VARCHAR(199);\
> ALTER TABLE history_varchar MODIFY history_to VARCHAR(226);\
> INSERT INTO history_varchar SELECT * FROM history;"
```

9. In the second terminal window, using the `mysql` client issue the following command to create a copy of the `history` table in the `isfdb` database called `history_mixed` that contains a CHAR column and a VARCHAR column instead of mediumtext columns:

```
sys> mysql -uroot -poracle -e"USE isfdb;\
> CREATE TABLE history_mixed LIKE history;\
> ALTER TABLE history_mixed MODIFY history_from CHAR(199);\
> ALTER TABLE history_mixed MODIFY history_to VARCHAR(226);\
> INSERT INTO history_mixed SELECT * FROM history;"
```

10. In the first terminal window, as `root` view the file sizes of all the files associated with the `history*` tables in the `isfdb` database by issuing the following command in a terminal window:

```
sys> ls -alt /var/lib/mysql/isfdb/history*
```

- What is the size of the `history_mixed.MYD` file?

---

- What is the size of the `history_char.MYD` file?

---

- What is the size of the `history_varchar.MYD` file?

---

- What is the size of the `history.MYD` file?

11. In the second terminal window, using the `PT_Stress.php` script issue the following command to output the `history_from` and `history_to` columns from the `history_varchar` table in the `isfdb` table. The `SELECT` statement runs two threads, with each thread executing the script 200 times.

```
sys> ./PT_Stress.php -u root -p oracle \
> -s "Q:SELECT history_from, history_to FROM \
> isfdb.history_varchar" -t -i 200 -c 2 -q
```

- What is the average amount that the threads took to complete?

12. In the second terminal window, using the `PT_Stress.php` script issue the following command to output the `history_from` and `history_to` columns from the `history_char` table in the `isfdb` table. The `SELECT` statement runs two threads, with each thread executing the script 200 times.

```
sys> ./PT_Stress.php -u root -p oracle \
> -s "Q:SELECT history_from, history_to FROM \
> isfdb.history_char" -t -i 200 -c 2 -q
```

- What is the average amount that the threads took to complete?

13. In the second terminal window, using the `PT_Stress.php` script issue the following command to output the `history_from` and `history_to` columns from the `history` table in the `isfdb` table. The `SELECT` statement runs two threads, with each thread executing the script 200 times.

```
sys> ./PT_Stress.php -u root -p oracle \
> -s "Q:SELECT history_from, history_to FROM \
> isfdb.history" -t -i 200 -c 2 -q
```

- What is the average amount that the threads took to complete?

14. In the second terminal window, using the `PT_Stress.php` script issue the following command to output the `history_from` and `history_to` columns from the `history_mixed` table in the `isfdb` table. The `SELECT` statement runs two threads, with each thread executing the script 200 times.

```
sys> ./PT_Stress.php -u root -p oracle \
> -s "Q:SELECT history_from, history_to FROM \
> isfdb.history_mixed" -t -i 200 -c 2 -q
```

- What is the average amount that the threads took to complete?

- 
15. Did the size of the table that contained the `CHAR` columns have a negative effect on the execution of the queries executed against the table?

---

---

---

- 
16. In the second terminal window, issue the following command to modify the `birthdate` column in the `authors` table in the `isfdb` database to use the `DATE` data type:

```
sys> mysql -uroot -poracle -e"ALTER TABLE isfdb.authors \
> MODIFY author_birthdate DATE DEFAULT NULL"
```

**Note:** This step ensures that the `authors` table in the `isfdb` database is unaffected by the running of this practice.

## Solutions 9-2: Data Types

### Tasks

**Note:** These solutions are sample solutions and are not expected to be the exact output you see in your execution of the steps.

1. Open a terminal window and as `root` view the file sizes of all the files associated with the `authors` table in the `isfdb` database by issuing the following command in a terminal window:

```
sys> ls -alt /var/lib/mysql/isfdb/authors*
-rw-rw---- 1 mysql mysql 3871628 Jan 22 17:35 authors.MYD
-rw-rw---- 1 mysql mysql 2879488 Jan 22 17:35 authors.MYI
-rw-rw---- 1 mysql mysql 9226 Jan 22 17:35 authors.frm
```

- What is the size of the `authors.MYD` file?

3,871,628 bytes

**Note:** You can use the existing terminal window from the previous practice. However, at this point, you should have only one terminal window open to minimize confusion as more terminal windows are opened up.

2. Open a second terminal window and issue the following command to modify the `birthdate` column in the `authors` table in the `isfdb` database to use the `datetime` data type:

```
sys> mysql -uroot -poracle -e"ALTER TABLE isfdb.authors \
> MODIFY author_birthdate DATETIME DEFAULT NULL"
```

3. In the first terminal window, as `root` issue the command from step 1 to view all the files associated with the `authors` table in the `isfdb` database.

```
sys> ls -alt /var/lib/mysql/isfdb/authors*
-rw-rw---- 1 mysql mysql 2879488 Mar 2 23:14 authors.MYI
-rw-rw---- 1 mysql mysql 3905444 Mar 2 23:14 authors.MYD
-rw-rw---- 1 mysql mysql 9226 Mar 2 23:14 authors.frm
```

- What is the size of the `authors.MYD` file?

3,905,444 bytes

4. In the second terminal window, using the `mysql` client view the table design of the `history` table in the `isfdb` database.

```
sys> mysql -uroot -poracle -e"SHOW CREATE TABLE isfdb.history\G"
***** 1. row *****
 Table: history
Create Table: CREATE TABLE `history` (
 `history_id` int(11) NOT NULL AUTO_INCREMENT,
 `history_time` datetime DEFAULT NULL,
 `history_table` int(11) DEFAULT NULL,
 `history_record` int(11) DEFAULT NULL,
 `history_field` int(11) DEFAULT NULL,
 `history_submission` int(11) DEFAULT NULL,
 `history_submitter` int(11) NOT NULL DEFAULT '0',
 `history_reviewer` int(11) NOT NULL DEFAULT '0',
 `history_from` mediumtext,
 `history_to` mediumtext,
 PRIMARY KEY (`history_id`)
) ENGINE=MyISAM AUTO_INCREMENT=51720 DEFAULT CHARSET=latin1
```

5. In the second terminal window, using the `mysql` client view the maximum value in the `history_from` column in the `history` table in the `isfdb` database.

```
sys> mysql -uroot -poracle -e"SELECT MAX(LENGTH(history_from)) \
> FROM isfdb.history"
+-----+
| MAX(LENGTH(history_from)) |
+-----+
| 199 |
+-----+
```

- What is the maximum length of the data in the `history_from` column?

199

6. In the second terminal window, using the `mysql` client view the maximum value in the `history_to` column in the `history` table in the `isfdb` database.

```
sys> mysql -uroot -poracle -e"SELECT MAX(LENGTH(history_to)) \
> FROM isfdb.history"
+-----+
| MAX(LENGTH(history_to)) |
+-----+
| 226 |
+-----+
```

- What is the maximum length of the data in the `history_to` column?

226

7. In the second terminal window, using the `mysql` client issue the following command to create a copy of the `history` table in the `isfdb` database called `history_char` that contains CHAR columns instead of mediumtext columns:

```
sys> mysql -uroot -poracle -e"USE isfdb;\
> CREATE TABLE history_char LIKE history;\
> ALTER TABLE history_char MODIFY history_from CHAR(199);\
> ALTER TABLE history_char MODIFY history_to CHAR(226);\
> INSERT INTO history_char SELECT * FROM history;"
```

8. In the second terminal window, using the `mysql` client issue the following command to create a copy of the `history` table in the `isfdb` database called `history_varchar` that contains CHAR columns instead of mediumtext columns:

```
sys> mysql -uroot -poracle -e"USE isfdb;\
> CREATE TABLE history_varchar LIKE history;\
> ALTER TABLE history_varchar MODIFY history_from VARCHAR(199);\
> ALTER TABLE history_varchar MODIFY history_to VARCHAR(226);\
> INSERT INTO history_varchar SELECT * FROM history;"
```

9. In the second terminal window, using the `mysql` client issue the following command to create a copy of the `history` table in the `isfdb` database called `history_mixed` that contains a CHAR column and a VARCHAR column instead of mediumtext columns:

```
sys> mysql -uroot -poracle -e"USE isfdb;\
> CREATE TABLE history_mixed LIKE history;\
> ALTER TABLE history_mixed MODIFY history_from CHAR(199);\
> ALTER TABLE history_mixed MODIFY history_to VARCHAR(226);\
> INSERT INTO history_mixed SELECT * FROM history;"
```

10. In the first terminal window, as root view the file sizes of all the files associated with the `history*` tables in the `isfdb` database by issuing the following command in a terminal window:

```
sys> ls -alt /var/lib/mysql/isfdb/history*
-rw-rw---- 1 mysql mysql 9020 Mar 3 01:48 history_mixed.frm
-rw-rw---- 1 mysql mysql 2342352 Mar 3 01:48 history_mixed.MYD
-rw-rw---- 1 mysql mysql 314368 Mar 3 01:48 history_mixed.MYI
-rw-rw---- 1 mysql mysql 314368 Mar 3 01:30 history_char.MYI
-rw-rw---- 1 mysql mysql 9020 Mar 3 01:29 history_char.frm
-rw-rw---- 1 mysql mysql 14089152 Mar 3 01:29 history_char.MYD
-rw-rw---- 1 mysql mysql 314368 Mar 3 01:29 history_varchar.MYI
-rw-rw---- 1 mysql mysql 2321172 Mar 3 01:28 history_varchar.MYD
-rw-rw---- 1 mysql mysql 9020 Mar 3 01:28 history_varchar.frm
-rw-rw---- 1 mysql mysql 9020 Jan 22 09:58 history.frm
-rw-rw---- 1 mysql mysql 2464336 Jan 22 09:58 history.MYD
-rw-rw---- 1 mysql mysql 314368 Jan 22 09:58 history.MYI
```

- What is the size of the `history_mixed.MYD` file?

2,342,352 bytes

- What is the size of the `history_char.MYD` file?

14,089,152 bytes

- What is the size of the `history_varchar.MYD` file?

2,321,172 bytes

- What is the size of the `history.MYD` file?

2,464,336 bytes

11. In the second terminal window, using the `PT_Stress.php` script issue the following command to output the `history_from` and `history_to` columns from the `history_varchar` table in the `isfdb` table. The `SELECT` statement runs two threads, with each thread executing the script 200 times.

```
sys> ./PT_Stress.php -u root -p oracle \
> -s "Q:SELECT history_from, history_to FROM \
> isfdb.history_varchar" -t -i 200 -c 2 -q
Simulating 2 connection(s) against the SELECT history_from,
history_to FROM isfdb.history_varchar script repeating 200
times!
Execution Times
Minimum: 17.655356
Maximum: 17.690923
Average: 17.673139
```

- What is the average amount that the threads took to complete?

17.67 seconds

12. In the second terminal window, using the `PT_Stress.php` script issue the following command to output the `history_from` and `history_to` columns from the `history_char` table in the `isfdb` table. The `SELECT` statement runs two threads, with each thread executing the script 200 times.

```
sys> ./PT_Stress.php -u root -p oracle \
> -s "Q:SELECT history_from, history_to FROM \
> isfdb.history_char" -t -i 200 -c 2 -q
Simulating 2 connection(s) against the SELECT history_from,
history_to FROM isfdb.history_char script repeating 200 times!
Execution Times
Minimum: 18.417700
Maximum: 18.756400
Average: 18.587050
```

- What is the average amount that the threads took to complete?

18.59 seconds

13. In the second terminal window, using the `PT_Stress.php` script issue the following command to output the `history_from` and `history_to` columns from the `history` table in the `isfdb` table. The `SELECT` statement runs two threads, with each thread executing the script 200 times.

```
sys> ./PT_Stress.php -u root -p oracle \
> -s "Q:SELECT history_from, history_to FROM \
> isfdb.history" -t -i 200 -c 2 -q
Simulating 2 connection(s) against the SELECT history_from,
history_to FROM isfdb.history script repeating 200 times!
Execution Times
Minimum: 18.507900
Maximum: 18.520778
Average: 18.514339
```

- What is the average amount that the threads took to complete?

18.51 seconds

14. In the second terminal window, using the `PT_Stress.php` script issue the following command to output the `history_from` and `history_to` columns from the `history_mixed` table in the `isfdb` table. The `SELECT` statement runs two threads, with each thread executing the script 200 times.

```
sys> ./PT_Stress.php -u root -p oracle \
> -s "Q:SELECT history_from, history_to FROM \
> isfdb.history_mixed" -t -i 200 -c 2 -q
Simulating 2 connection(s) against the SELECT history_from,
history_to FROM isfdb.history_mixed script repeating 200 times!
Execution Times
Minimum: 20.753143
Maximum: 20.764785
Average: 20.758964
```

- What is the average amount that the threads took to complete?

20.76 seconds

15. Did the size of the table that contained the `CHAR` columns have a negative effect on the execution of the queries executed against the table?

Yes, but not in proportion to the size of the file in relation to the `history_varchar` and `history` tables. In extremely large tables, having only `CHAR` columns for string fields in a table will make the entire record a fixed size. This can improve performance. However, any mixing of variable and fixed columns in the same table will result in poor performance. A table should have either all fixed strings or all variable strings, mixing should be avoided.

16. In the second terminal window, issue the following command to modify the `birthdate` column in the `authors` table in the `isfdb` database to use the `DATE` data type:

```
sys> mysql -uroot -poracle -e"ALTER TABLE isfdb.authors \
> MODIFY author_birthdate DATE DEFAULT NULL"
```

**Note:** This step ensures that the `authors` table in the `isfdb` database is unaffected by the running of this practice.

## Practice 9-3: Indexes

In this practice, you evaluate the effects of indexes in a variety of scenarios. To accomplish this objective, you do the following:

- View the size of specific tables in the /var/lib/mysql/isfdb directory.
- Create a MEMORY table with both BTREE and HASH indexes.
- Use the `PT_Stress.php` script to run multiple queries against the MEMORY table created to test the performance using the different index types (and no indexes).
- Create a table that uses a prefix index and evaluate the performance along with the size difference between a full index and a prefix index.

### Assumptions

- The MySQL server is installed and running.
- The `many_tables` database is installed.
- The `btree-equal.sql`, `btree-nonequal.sql`, `hash-equal.sql`, `hash-nonequal.sql`, `scan-equal.sql`, `scan-nonequal.sql` files, and `PT_Stress.php` script are accessible and located in the `/stage/script` directory.

### Duration

This practice should take 20 minutes to complete.

### Tasks

1. Open a terminal window and use the `mysql` client, issue the following command to review the structure of the `create_city_memory_huge` MySQL stored procedure:

```
sys> mysql -uroot -poracle -e"SELECT ROUTINE_NAME, \
> ROUTINE_DEFINITION FROM INFORMATION_SCHEMA.ROUTINES \
> WHERE ROUTINE_NAME LIKE 'create_city_memory_huge'\G"
```

- Reading the script of the `innodb_inserts` MySQL stored procedure, what is its purpose?

---

---

---

**Note:** You can use the existing terminal window from the previous practice. However, at this point, you should have only one terminal window open to minimize confusion as more terminal windows are opened up.

2. Using the `mysql` client, issue the following command to run the `create_city_memory_huge` stored procedure in the `many_tables` database:

```
sys> mysql -uroot -poracle -e"USE many_tables; \
> CALL create_city_memory_huge(30); \
> SELECT COUNT(*) FROM city_memory_huge;"
```

- How many records does the `city_memory_huge` table contain?

---

3. Using the mysql client, issue the following command to view the structure of the city\_memory\_huge table created in step 2:

```
sys> mysql -uroot -poracle -e"SHOW CREATE TABLE \
> many_tables.city_memory_huge\G"
```

- How many indexes does the city\_memory\_huge table contain?
- 

4. In the /stage/scripts directory, use the PT\_Stress.php script to issue the following command to execute a script that searches on the name column using the name\_btree index with an exact match WHERE clause:

```
sys> ./PT_Stress.php -u root -p oracle -s btree-equal.sql -t
```

- How long did the script take to complete one iteration of the script?
- 

5. In the /stage/scripts directory, use the PT\_Stress.php script to issue the following command to execute a script that searches on the name column using the name\_hash index with an exact match WHERE clause:

```
sys> ./PT_Stress.php -u root -p oracle -s hash-equal.sql -t
```

- How long did the script take to complete one iteration of the script?
- 

6. In the /stage/scripts directory, use the PT\_Stress.php script to issue the following command to execute a script that searches on the name column not using an indexes with an exact match WHERE clause:

```
sys> ./PT_Stress.php -u root -p oracle -s scan-equal.sql -t
```

- How long did the script take to complete one iteration of the script?
- 

7. In the /stage/scripts directory, use the PT\_Stress.php script to issue the following command to execute a script that searches on the name column using the name\_btree index with a wildcard match WHERE clause (range search):

```
sys> ./PT_Stress.php -u root -p oracle -s btree-nonequal.sql -t
```

- How long did the script take to complete one iteration of the script?
- 

8. In the /stage/scripts directory, use the PT\_Stress.php script to issue the following command to execute a script that searches on the name column using the name\_hash index with a wildcard match WHERE clause (range search):

```
sys> ./PT_Stress.php -u root -p oracle -s hash-nonequal.sql -t
```

- How long did the script take to complete one iteration of the script?
-

9. In the /stage/scripts directory, use the `PT_Stress.php` script to issue the following command to execute a script that searches on the `name` column not using an indexes with a wildcard match `WHERE` clause (range search):

```
sys> ./PT_Stress.php -u root -p oracle -s scan-nonequal.sql -t
- How long did the script take to complete one iteration of the script?
```

- 
10. What do the results received tell you about the different types of indexes used in the previous steps?

---

---

- 
11. Using the `mysql` client, issue the following command to run the `create_city_huge` stored procedure in the `many_tables` database:

```
sys> mysql -uroot -poracle -e"USE many_tables;\n> CALL create_city_huge(30);\n> SELECT COUNT(*) FROM city_huge;"
```

**Note:** The `create_city_huge` stored procedure was reviewed in an earlier practice.

- How many records does the `city_huge` table contain?

- 
12. Using the `mysql` client, issue the following command to view the structure of the `city_huge` table created in step 11:

```
sys> mysql -uroot -poracle \
> -e"SHOW CREATE TABLE many_tables.city_huge\G"
```

- How many indexes does the `city_huge` table contain?

---

---

- 
13. In a separate terminal window, as `root` issue the following command to view the files associated with the `city_huge` table:

```
sys> ls -alt /var/lib/mysql/many_tables/city_huge*
```

- What is the size of the `city_huge.MYI` file?

---

---

14. In the first terminal window, use the mysql client to create a prefix index on the Name column from the city\_huge table in the many\_tables database by issuing the following command:

```
sys> mysql -uroot -poracle \
> -e"ALTER TABLE many_tables.city_huge \
> DROP INDEX Name, ADD INDEX (Name(4))"
```

**Note:** The index associated with the Name column needed to be removed first to eliminate having more than one index on a column.

15. Using the mysql client, issue the following command to view the structure of the city\_huge table modified in step 14:

```
sys> mysql -uroot -poracle -e"SHOW CREATE TABLE \
> many_tables.city_huge\G"
```

- How many indexes does the city\_huge table contain?

---

---

16. In the second terminal window opened in step 13, as root issue the following command to view the files associated with the city\_huge table:

```
sys> ls -alt /var/lib/mysql/many_tables/city_huge*
```

- What is the size of the city\_huge.MYI file?

---

17. In the /stage/scripts directory, use the PT\_Stress.php script to issue the following command to execute a script that searches on the name column using the prefix index created in step 14 with an exact match WHERE clause:

```
sys> ./PT_Stress.php -u root -p oracle -s prefix-equal.sql -t
```

- How long did the script take to complete one iteration of the script?

---

18. In the /stage/scripts directory, use the PT\_Stress.php script to issue the following command to execute a script that searches on the name column using the prefix index created in step 14 with a wildcard match WHERE clause (range search):

```
sys> ./PT_Stress.php -u root -p oracle -s prefix-nonequal.sql -t
```

- How long did the script take to complete one iteration of the script?

---

---

---

19. What do the results received and the size of the index file tell you about prefix indexes?

---

---

---

## Solutions 9-3: Indexes

### Tasks

**Note:** These solutions are sample solutions and are not expected to be the exact output you see in your execution of the steps.

1. Open a terminal window and use the `mysql` client, issue the following command to review the structure of the `create_city_memory_huge` MySQL stored procedure:

```
sys> mysql -uroot -poracle -e"SELECT ROUTINE_NAME, \
> ROUTINE_DEFINITION FROM INFORMATION_SCHEMA.ROUTINES \
> WHERE ROUTINE_NAME LIKE 'create_city_memory_huge'\G"
***** 1. row *****
ROUTINE_NAME: create_city_memory_huge
ROUTINE_DEFINITION: BEGIN
 DECLARE loops INT default 0;
 DROP TABLE IF EXISTS city_memory_huge;
 CREATE TABLE city_memory_huge LIKE City;
 ALTER TABLE city_memory_huge ENGINE=MEMORY, ADD INDEX
name_btree USING BTREE (name), ADD INDEX name_hash USING HASH
(Name);
 WHILE (loops < size) DO
 INSERT INTO city_memory_huge (Name, CountryCode, District,
Population)
 SELECT Name, CountryCode, District, Population FROM City;
 SET loops = loops + 1;
 END WHILE;
END
```

- Reading the script of the `innodb_inserts` MySQL stored procedure, what is its purpose?

This MySQL stored procedure creates a table called `city memory huge` using the `MEMORY` storage engine based on the `City` table structure. The table is then altered and two indexes are created on the `name` column; one `BTREE` index and one `HASH` index. This new table is then loaded with data from the `City` table with the number of copies of the `City` table data loaded into the new table controlled by the end user.

**Note:** You can use the existing terminal window from the previous practice. However, at this point, you should have only one terminal window open to minimize confusion as more terminal windows are opened up.

2. Using the `mysql` client, issue the following command to run the `create_city_memory_huge` stored procedure in the `many_tables` database:

```
sys> mysql -uroot -poracle -e"USE many_tables; \
> CALL create_city_memory_huge(30); \
> SELECT COUNT(*) FROM city_memory_huge;"
```

| COUNT(*) |
|----------|
| 122370   |

- How many records does the `city_memory_huge` table contain?

122,370

- Using the `mysql` client, issue the following command to view the structure of the `city_memory_huge` table created in step 2:

```
sys> mysql -uroot -poracle -e"SHOW CREATE TABLE \
> many_tables.city_memory_huge\G"
***** 1. row *****
 Table: city_memory_huge
Create Table: CREATE TABLE `city_memory_huge` (
 `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 `name_btree` (`Name`) USING BTREE,
 KEY `name_hash` (`Name`) USING HASH
) ENGINE=MEMORY AUTO_INCREMENT=122371 DEFAULT CHARSET=latin1
```

- How many indexes does the `city_memory_huge` table contain?

3; The primary key index, the name\_btree index, and the name\_hash index.

- In the `/stage/scripts` directory, use the `PT_Stress.php` script to issue the following command to execute a script that searches on the `name` column using the `name_btree` index with an exact match `WHERE` clause:

```
sys> ./PT_Stress.php -u root -p oracle -s btree-equal.sql -t
Simulating 1 connection(s) against the btree-equal.sql script
repeating 1 times!
Execution Times
Minimum: 0.111173
Maximum: 0.111173
Average: 0.111173
```

- How long did the script take to complete one iteration of the script?

0.111 seconds

- In the `/stage/scripts` directory, use the `PT_Stress.php` script to issue the following command to execute a script that searches on the `name` column using the `name_hash` index with an exact match `WHERE` clause:

```
sys> ./PT_Stress.php -u root -p oracle -s hash-equal.sql -t
Simulating 1 connection(s) against the hash-equal.sql script
repeating 1 times!
Execution Times
Minimum: 0.115148
Maximum: 0.115148
Average: 0.115148
```

- How long did the script take to complete one iteration of the script?

0.115 seconds

6. In the /stage/scripts directory, use the `PT_Stress.php` script to issue the following command to execute a script that searches on the `name` column not using an indexes with an exact match WHERE clause:

```
sys> ./PT_Stress.php -u root -p oracle -s scan-equal.sql -t
Simulating 1 connection(s) against the scan-equal.sql script
repeating 1 times!
Execution Times
Minimum: 14.853390
Maximum: 14.853390
Average: 14.853390
```

- How long did the script take to complete one iteration of the script?

14.853 seconds

7. In the /stage/scripts directory, use the `PT_Stress.php` script to issue the following command to execute a script that searches on the `name` column using the `name_btree` index with a wildcard match WHERE clause (range search):

```
sys> ./PT_Stress.php -u root -p oracle -s btree-nonequal.sql -t
Simulating 1 connection(s) against the btree-nonequal.sql script
repeating 1 times!
Execution Times
Minimum: 0.123770
Maximum: 0.123770
Average: 0.123770
```

- How long did the script take to complete one iteration of the script?

0.124 seconds

8. In the /stage/scripts directory, use the `PT_Stress.php` script to issue the following command to execute a script that searches on the `name` column using the `name_hash` index with a wildcard match WHERE clause (range search):

```
sys> ./PT_Stress.php -u root -p oracle -s hash-nonequal.sql -t
Simulating 1 connection(s) against the hash-nonequal.sql script
repeating 1 times!
Execution Times
Minimum: 14.548320
Maximum: 14.548320
Average: 14.548320
```

- How long did the script take to complete one iteration of the script?

14.548 seconds

9. In the /stage/scripts directory, use the `PT_Stress.php` script to issue the following command to execute a script that searches on the `name` column not using an indexes with a wildcard match `WHERE` clause (range search):

```
sys> ./PT_Stress.php -u root -p oracle -s scan-nonequal.sql -t
Simulating 1 connection(s) against the scan-nonequal.sql script
repeating 1 times!
 Execution Times
 Minimum: 14.924583
 Maximum: 14.924583
 Average: 14.924583
```

- How long did the script take to complete one iteration of the script?

14.925 seconds

10. What do the results received tell you about the different types of indexes used in the previous steps?

The overall best results were achieved when the BTREE index was used for both exact and range searches; however, the HASH index performed the best when an exact match search was performed. Using a HASH index on a range search was equivalent to using no index at all.

11. Using the `mysql` client, issue the following command to run the `create_city_huge` stored procedure in the `many_tables` database:

```
sys> mysql -uroot -poracle -e"USE many_tables; \
> CALL create_city_huge(30); \
> SELECT COUNT(*) FROM city_huge;"
```

|         |          |
|---------|----------|
| +-----+ |          |
|         | COUNT(*) |
| +-----+ |          |
|         | 122370   |
| +-----+ |          |

**Note:** The `create_city_huge` stored procedure was reviewed in an earlier practice.

- How many records does the `city_huge` table contain?

122,370

12. Using the `mysql` client, issue the following command to view the structure of the `city_huge` table created in step 11:

```
sys> mysql -uroot -poracle \
> -e"SHOW CREATE TABLE many_tables.city_huge\G"
***** 1. row ****
 Table: city_huge
Create Table: CREATE TABLE `city_huge` (
 `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`),
 KEY `Name` (`Name`),
 KEY `Population` (`Population`)
) ENGINE=MyISAM AUTO_INCREMENT=122371 DEFAULT CHARSET=latin1
```

- How many indexes does the `city_huge` table contain?

4; The primary key index, the `CountryCode` index, the `Name` index, and the `Population` index.

13. In a separate terminal window, as `root` issue the following command to view the files associated with the `city_huge` table:

```
sys> ls -alt /var/lib/mysql/many_tables/city_huge*
-rw-rw---- 1 mysql mysql 8198790 Mar 31 00:01 city_huge.MYD
-rw-rw---- 1 mysql mysql 5666816 Mar 31 00:01 city_huge.MYI
-rw-rw---- 1 mysql mysql 8710 Mar 31 00:01 city_huge.frm
```

- What is the size of the `city_huge.MYI` file?

5,666,816 bytes

14. In the first terminal window, use the `mysql` client to create a prefix index on the `Name` column from the `city_huge` table in the `many_tables` database by issuing the following command:

```
sys> mysql -uroot -poracle \
> -e"ALTER TABLE many_tables.city_huge \
> DROP INDEX Name, ADD INDEX (Name(4))"
```

**Note:** The index associated with the `Name` column needed to be removed first to eliminate having more than one index on a column.

15. Using the `mysql` client, issue the following command to view the structure of the `city_huge` table modified in step 14:

```
sys> mysql -uroot -poracle -e"SHOW CREATE TABLE \
> many_tables.city_huge\G"
***** 1. row *****
 Table: city_huge
Create Table: CREATE TABLE `city_huge` (
 `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`),
 KEY `Population` (`Population`),
 KEY `Name` (`Name` (4))
) ENGINE=MyISAM AUTO_INCREMENT=122371 DEFAULT CHARSET=latin1
```

- How many indexes does the `city_huge` table contain?

4; The primary key index, the `CountryCode` index, a prefixed `Name` index, and the `Population` index.

16. In the second terminal window opened in step 13, as root issue the following command to view the files associated with the `city_huge` table:

```
sys> ls -alt /var/lib/mysql/many_tables/city_huge*
- -rw-rw---- 1 mysql mysql 4880384 Mar 31 00:06 city_huge.MYI
- rw-rw---- 1 mysql mysql 8198790 Mar 31 00:06 city_huge.MYD
- rw-rw---- 1 mysql mysql 8710 Mar 31 00:06 city_huge.frm
```

- What is the size of the `city_huge.MYI` file?

4,880,384 bytes

17. In the `/stage/scripts` directory, use the `PT_Stress.php` script to issue the following command to execute a script that searches on the `name` column using the prefix index created in step 14 with an exact match `WHERE` clause:

```
sys> ./PT_Stress.php -u root -p oracle -s prefix-equal.sql -t
Simulating 1 connection(s) against the prefix-equal.sql script
repeating 1 times!
 Execution Times
 Minimum: 0.297449
 Maximum: 0.297449
 Average: 0.297449
```

- How long did the script take to complete one iteration of the script?

0.297 seconds

18. In the /stage/scripts directory, use the PT\_Stress.php script to issue the following command to execute a script that searches on the name column using the prefix index created in step 14 with a wildcard match WHERE clause (range search):

```
sys> ./PT_Stress.php -u root -p oracle -s prefix-nonequal.sql -t
Simulating 1 connection(s) against the prefix-nonequal.sql
script repeating 1 times!
```

| Execution Times   |
|-------------------|
| Minimum: 0.304271 |
| Maximum: 0.304271 |
| Average: 0.304271 |

- How long did the script take to complete one iteration of the script?

0.304 seconds

19. What do the results received and the size of the index file tell you about prefix indexes?

The performance was not as good as a full index on the Name field, but the size of the index file was also smaller. Having an understanding of the type of queries issued against the server can provide you with valuable information to determine if a prefix index would be beneficial in your applications.

## Practice 9-4: Partitioning

In this practice, you evaluate how partitioning can improve performance. To accomplish this objective, you do the following:

- Create a copy of an existing table that uses range partitioning.
- Use the `PT_Stress.php` script to execute queries against the original table and partitioned table to compare performance.

### Assumptions

- The MySQL server is installed and running.
- The `many_tables` database is installed.
- The `PT_Stress.php` script is accessible and located in the `/stage/script` directory.

### Duration

This practice should take 20 minutes to complete.

### Tasks

1. Open a terminal window and use the `mysql` client to issue the following command to create a table in the `many_tables` database called `city_no_partition` based on the `many_tables.city_huge` table:

```
sys> mysql -uroot -poracle \
> -e"CREATE TABLE many_tables.city_no_partition \
> (ID int(11) NOT NULL, \
> 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') \
> ENGINE=MYISAM"
```

**Note:** You can use the existing terminal window from the previous practice. However, at this point, you should have only one terminal window open to minimize confusion as more terminal windows are opened up.

2. Using the `mysql` client, copy the data from the `city_huge` table in the `many_tables` database into the `city_no_partition` table in the `many_tables` database.

```
sys> mysql -uroot -poracle \
> -e"INSERT INTO many_tables.city_no_partition \
> SELECT * FROM many_tables.city_huge"
```

3. Use the mysql client to issue the following command to create a table in the many\_tables database called city\_partition with three different partitions based on range of the population:

```
sys> mysql -uroot -poracle \
> -e"CREATE TABLE many_tables.city_partition \
> (ID int(11) NOT NULL, \
> 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') ENGINE=MYISAM\
> PARTITION BY RANGE(Population) \
> (PARTITION one VALUES LESS THAN (100000), \
> PARTITION two VALUES LESS THAN (1000000), \
> PARTITION three VALUES LESS THAN MAXVALUE)"
```

4. Using the mysql client, copy the data from the city\_huge table in the many\_tables database into the city\_partition table in the many\_tables database.

```
sys> mysql -uroot -poracle \
> -e"INSERT INTO many_tables.city_partition \
> SELECT * FROM many_tables.city_huge"
```

**Note:** This will take approximately 4 minutes to execute. If you include the PRIMARY KEY creation in the CREATE TABLE command, the process would take much longer to perform.

5. Using the mysql client, add a PRIMARY KEY index to the city\_partition table in the many\_tables database.

```
sys> mysql -uroot -poracle -e"ALTER TABLE \
many_tables.city_partition ADD PRIMARY KEY (ID, Population)"
```

6. Use the mysql client to issue the following command to view the number of rows in each partition of the many\_tables.city\_partition table:

```
sys> mysql -uroot -poracle -e"SELECT partition_name, \
table_rows FROM INFORMATION_SCHEMA.PARTITIONS WHERE \
table_name = 'city_partition'"
```

- What is the approximate distribution of rows between the partitions?

---



---



---

7. In the /stage/scripts directory, use the PT\_Stress.php script to issue the following command to evaluate the time it would take to execute a SELECT statement against the city\_huge table in the many\_tables database that selects cities that have a population of greater than 1 million. This SELECT statement runs three threads, with each thread executing the script 10 times each.

```
sys> ./PT_Stress.php -u root -p oracle \
> -s "Q:SELECT Name, Population FROM \
many_tables.city_no_partition WHERE Population > 1000000" \
> -t -i 10 -c 3 -q
```

- What is the average amount that each thread took to complete?
- 
8. In the /stage/scripts directory, use the `PT_Stress.php` script to issue the command from step 7 but use the `city_partition` table instead of the `city_no_partition` table in the `many_tables` database.

```
sys> ./PT_Stress.php -u root -p oracle \
> -s "Q:SELECT Name, Population FROM \
> many_tables.city_partition WHERE Population > 1000000" \
> -t -i 10 -c 3 -q
```

- What is the average amount that each thread took to complete?
- 
9. In the /stage/scripts directory, use the `PT_Stress.php` script to issue the following command to evaluate the time it would take to execute a `SELECT` statement that displays all the records from the `city_no_partition` table in the `many_tables` database. This `SELECT` statement runs three threads, with each thread executing the script 10 times each.

```
sys> ./PT_Stress.php -u root -p oracle \
> -s "Q:SELECT Name, Population \
> FROM many_tables.city_no_partition" -t -i 10 -c 3 -q
```

- What is the average amount that each thread took to complete?
- 
10. In the /stage/scripts directory, use the `PT_Stress.php` script to issue the command from step 9 but use the `city_partition` table instead of the `city_huge` table in the `many_tables` database.

```
sys> ./PT_Stress.php -u root -p oracle \
> -s "Q:SELECT Name, Population FROM \
> many_tables.city_partition" -t -i 10 -c 3 -q
```

- What is the average amount that each thread took to complete?
- 
11. What do the results received tell you about partitioning tables?
- 
- 
-

## Solutions 9-4: Partitioning

### Tasks

**Note:** These solutions are sample solutions and are not expected to be the exact output you see in your execution of the steps.

1. Open a terminal window and use the `mysql` client to issue the following command to create a table in the `many_tables` database called `city_no_partition` based on the `many_tables.city_huge` table:

```
sys> mysql -uroot -poracle \
> -e"CREATE TABLE many_tables.city_no_partition \
> (ID int(11) NOT NULL, \
> 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') \
> ENGINE=MYISAM"
```

**Note:** You can use the existing terminal window from the previous practice. However, at this point, you should have only one terminal window open to minimize confusion as more terminal windows are opened up.

2. Using the `mysql` client, copy the data from the `city_huge` table in the `many_tables` database into the `city_no_partition` table in the `many_tables` database.

```
sys> mysql -uroot -poracle \
> -e"INSERT INTO many_tables.city_no_partition \
> SELECT * FROM many_tables.city_huge"
```

3. Use the `mysql` client to issue the following command to create a table in the `many_tables` database called `city_partition` with three different partitions based on range of the population:

```
sys> mysql -uroot -poracle \
> -e"CREATE TABLE many_tables.city_partition \
> (ID int(11) NOT NULL, \
> 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') ENGINE=MYISAM\
> PARTITION BY RANGE(Population) \
> (PARTITION one VALUES LESS THAN (100000), \
> PARTITION two VALUES LESS THAN (1000000), \
> PARTITION three VALUES LESS THAN MAXVALUE) "
```

4. Using the `mysql` client, copy the data from the `city_huge` table in the `many_tables` database into the `city_partition` table in the `many_tables` database.

```
sys> mysql -uroot -poracle \
> -e"INSERT INTO many_tables.city_partition \
> SELECT * FROM many_tables.city_huge"
```

**Note:** This will take approximately 4 minutes to execute. If you include the PRIMARY KEY creation in the CREATE TABLE command, the process would take much longer to perform.

5. Using the mysql client, add a PRIMARY KEY index to the city\_partition table in the many\_tables database.

```
sys> mysql -uroot -poracle -e"ALTER TABLE \
> many_tables.city_partition ADD PRIMARY KEY (ID, Population)"
```

6. Use the mysql client to issue the following command to view the number of rows in each partition of the many\_tables.city\_partition table:

```
sys> mysql -uroot -poracle -e"SELECT partition_name, \
> table_rows FROM INFORMATION_SCHEMA.PARTITIONS WHERE \
> table_name = 'city_partition'"
```

| partition_name | table_rows |
|----------------|------------|
| one            | 232746     |
| two            | 1496459    |
| three          | 107184     |

- What is the approximate distribution or rows between the partitions?

The records with 10,000 or less population make up approximately 13% of the records, the records with 100,000 or less population make up approximately 81% of the records, and those records with 100,000 or greater population make up approximately 6% of the records.

7. In the /stage/scripts directory, use the PT\_Stress.php script to issue the following command to evaluate the time it would take to execute a SELECT statement against the city\_huge table in the many\_tables database that selects cities that have a population of greater than 1 million. This SELECT statement runs three threads, with each thread executing the script 10 times each.

```
sys> ./PT_Stress.php -u root -p oracle \
> -s "Q:SELECT Name, Population FROM \
> many_tables.city_no_partition WHERE Population > 1000000" \
> -t -i 10 -c 3 -q
```

Simulating 3 connection(s) against the SELECT Name, Population  
FROM many\_tables.city\_no\_partition WHERE Population > 1000000  
script repeating 10 times!

```
Execution Times
Minimum: 10.888965
Maximum: 14.895582
Average: 13.066470
```

- What is the average amount that each thread took to complete?

13.07 seconds

8. In the /stage/scripts directory, use the `PT_Stress.php` script to issue the command from step 7 but use the `city_partition` table instead of the `city_no_partition` table in the `many_tables` database.

```
sys> ./PT_Stress.php -u root -p oracle \
> -s "Q:SELECT Name, Population FROM \
> many_tables.city_partition WHERE Population > 1000000" \
> -t -i 10 -c 3 -q
Simulating 3 connection(s) against the SELECT Name, Population
FROM many_tables.city_partition WHERE Population > 1000000
script repeating 10 times!
```

|                   |
|-------------------|
| Execution Times   |
| Minimum: 2.210520 |
| Maximum: 3.215424 |
| Average: 2.662116 |

- What is the average amount that each thread took to complete?

2.66 seconds

9. In the /stage/scripts directory, use the `PT_Stress.php` script to issue the following command to evaluate the time it would take to execute a `SELECT` statement that displays all the records from the `city_no_partition` table in the `many_tables` database. This `SELECT` statement runs three threads, with each thread executing the script 10 times each.

```
sys> ./PT_Stress.php -u root -p oracle \
> -s "Q:SELECT Name, Population \
> FROM many_tables.city_no_partition" -t -i 10 -c 3 -q
Simulating 3 connection(s) against the SELECT Name, Population
FROM many_tables.city_no_partition WHERE Population > 1000
script repeating 10 times!
```

|                    |
|--------------------|
| Execution Times    |
| Minimum: 46.183742 |
| Maximum: 54.286554 |
| Average: 50.699524 |

- What is the average amount that each thread took to complete?

50.70 seconds

10. In the /stage/scripts directory, use the PT\_Stress.php script to issue the command from step 10 but use the city\_partition table instead of the city\_no\_partition table in the many\_tables database.

```
sys> ./PT_Stress.php -u root -p oracle \
> -s "Q:SELECT Name, Population FROM \
> many_tables.city_partition" -t -i 10 -c 3 -q
Simulating 3 connection(s) against the SELECT Name, Population
FROM many_tables.city_partition WHERE Population > 1000 script
repeating 10 times!
```

| Execution Times    |
|--------------------|
| Minimum: 50.294828 |
| Maximum: 54.967856 |
| Average: 53.164201 |

- What is the average amount that each thread took to complete?

53.16 seconds

11. What do the results received tell you about partitioning tables?

For queries that can use data from only one partition, the performance is substantially better than accessing identical tables without partitioning. However, for queries that must access every partition to retrieve the data, the performance can be worse than accessing identical tables without partitioning.



# **Practices for Lesson 10: MySQL Query Performance**

**Chapter 10**

## Practices for Lesson 10

---

### Practices Overview

In these practices, you test your knowledge of improving the performance of queries executed against the MySQL server.

## Practice 10-1: EXPLAIN Outputs

### Overview

In this practice, you execute a number of `SELECT` statements using the `EXPLAIN` command to review the different outputs that are possible. To accomplish this objective, you do the following:

- Execute multiple `SELECT` statements against the `world` database.
- Add indexes to existing tables.
  - Throughout the running of this practice, there are numerous steps to clean up the `world` database and the tables contained in the `world` database to ensure that the database itself remains unaffected at the end of the practice.

### Assumptions

- The MySQL server is installed and running.
- The `world` database is installed.

### Duration

This practice should take 45 minutes to complete.

### Tasks

1. Open a terminal window and issue the following command to view the `EXPLAIN` output for a query against the `City` table in the `world` database:

```
sys> mysql -uroot -poracle -e"USE world;\
> EXPLAIN SELECT * FROM world.City WHERE ID=3803\G"
```

- What is the select type for this query? \_\_\_\_\_
- What is the type of access strategy that MySQL plans to use for this query?  
\_\_\_\_\_
- Would this access strategy produce good performance or poor performance?  
\_\_\_\_\_

2. Issue the following command to view the `EXPLAIN` output for a query against the `City` table in the `world` database:

```
sys> mysql -uroot -poracle -e"USE world;\
> EXPLAIN SELECT * FROM City WHERE ID=3800+3\G"
```

- Is there any difference in the execution of this query from the query in step 1? \_\_\_\_\_
- Explain your answer.  
\_\_\_\_\_

3. Issue the following statement in the mysql client:

```
sys> mysql -uroot -poracle -e"USE world; \
> EXPLAIN SELECT * FROM world.City WHERE ID=3800\G"
```

- Is there any difference in the execution of this query from the query in step 1? \_\_\_\_\_
- Explain your answer.

---

---

4. Issue the following statement in the mysql client:

```
sys> mysql -uroot -poracle -e"USE world; \
> EXPLAIN SELECT Country.Name \
> FROM Country, City WHERE City.CountryCode=Country.Code\G"
```

- What is the select type for the *first* table being selected? \_\_\_\_\_
- What is the type of access strategy that MySQL plans to use for the *first* table?  
\_\_\_\_\_
- What is the select type for the *second* table being selected? \_\_\_\_\_
- What is the type of access strategy that MySQL plans to use for the *second* table?  
\_\_\_\_\_
- Would this access strategy produce good performance or poor performance?  
\_\_\_\_\_

---

---

5. Add an index on the District field of the City table in the world database.

6. Issue the following statement in the mysql client:

```
sys> mysql -uroot -poracle -e"USE world; \
> EXPLAIN SELECT * FROM City \
> WHERE District='California'\G"
```

- What is the select type for this query? \_\_\_\_\_
- What is the type of access strategy that MySQL plans to use for this query?  
\_\_\_\_\_
- Would this access strategy produce good performance or poor performance?  
\_\_\_\_\_

---

---

7. Remove the index on the District field of the City table in the world database.

8. Add an index on the District field of the City table in the world database.

9. Issue the following statement in the mysql client:

```
sys> mysql -uroot -poracle -e"USE world; \
> EXPLAIN SELECT * FROM Country \
> WHERE IndepYear=1905 OR IndepYear IS NULL\G"
```

- What is the select type for this query? \_\_\_\_\_
- What is the type of access strategy that MySQL plans to use for this query?  
\_\_\_\_\_
- Would this access strategy produce good performance or poor performance?  
\_\_\_\_\_

10. Remove the index on the IndepYear field of the Country table in the world database.

11. Add an index on the District field of the City table in the world database.

12. Issue the following statement in the mysql client:

```
sys> mysql -uroot -poracle -e"USE world; \
EXPLAIN SELECT * FROM City \
> WHERE ID=50 OR District='Michigan'\G"
```

- What is the select type for this query? \_\_\_\_\_
- What is the type of access strategy that MySQL plans to use for this query?  
\_\_\_\_\_
- Would this access strategy produce good performance or poor performance?  
\_\_\_\_\_

13. Remove the index on the District field of the City table in the world database.

14. Issue the following statement in the mysql client:

```
sys> mysql -uroot -poracle -e"USE world; \
> EXPLAIN SELECT * FROM City WHERE CountryCode IN \
> (SELECT code FROM Country WHERE CODE LIKE 'USA')\G"
```

- What is the select type for the *first* table being selected? \_\_\_\_\_
- What is the type of access strategy that MySQL plans to use for the *first* table?  
\_\_\_\_\_
- What is the select type for the *second* table being selected? \_\_\_\_\_
- What is the type of access strategy that MySQL plans to use for the *second* table?  
\_\_\_\_\_
- Would this access strategy produce good performance or poor performance?  
\_\_\_\_\_

15. Add an index on the Population field of the City table in the world database.

16. Issue the following statement in the mysql client:

```
sys> mysql -uroot -poracle -e"USE world;\
> EXPLAIN SELECT * FROM City \
> WHERE population>1000000\G"
```

- What is the select type for this query? \_\_\_\_\_
  - What is the type of access strategy that MySQL plans to use for this query?
- 
- 
- 
- 
- 
- 

17. Remove the index on the Population field of the City table in the world database.

18. Issue the following statement in the mysql client:

```
sys> mysql -uroot -poracle -e"USE world;\
> EXPLAIN SELECT ID FROM City\G"
```

- What is the select type for this query? \_\_\_\_\_
  - What is the type of access strategy that MySQL plans to use for this query?
- 
- 
- 
- 
- 
- 

19. Issue the following statement in the mysql client:

```
sys> mysql -uroot -poracle -e"USE world;\
> EXPLAIN SELECT * FROM City \
> WHERE population>1000000\G"
```

- What is the select type for this query? \_\_\_\_\_
  - What is the type of access strategy that MySQL plans to use for this query?
- 
- 
- 
- 
- 
-

## Solutions 10-1: EXPLAIN Outputs

### Tasks

1. Open a terminal window and issue the following command to view the EXPLAIN output for a query against the City table in the world database.

```
sys> mysql -uroot -poracle -e"USE world;\
> EXPLAIN SELECT * FROM City WHERE ID=3803\G"
***** 1. row *****
 id: 1
select_type: SIMPLE
 table: City
 type: const
possible_keys: PRIMARY
 key: PRIMARY
key_len: 4
 ref: const
 rows: 1
 Extra:
```

- What is the select type for this query? SIMPLE
- What is the type of access strategy that MySQL plans to use for this query? const
- Would this access strategy produce good performance or poor performance?  
The const access strategy type states that MySQL has found only one matching row that can be read at the start of the query. These types of queries can produce good performance because values from this row can be considered constants by the optimizer and are read only once.

2. Issue the following statement in the mysql client:

```
sys> mysql -uroot -poracle -e" USE world;\
> EXPLAIN SELECT * FROM City WHERE ID=3800+3\G"
***** 1. row *****
 id: 1
select_type: SIMPLE
 table: City
 type: const
possible_keys: PRIMARY
 key: PRIMARY
key_len: 4
 ref: const
 rows: 1
 Extra:
```

- Is there any difference in the execution of this query from the query in step 1? NO
- Explain your answer.  
The calculated constant was reduced to a single value prior to the execution of the query.

3. Issue the following statement in the mysql client:

```
sys> mysql -uroot -poracle -e" USE world; \
> EXPLAIN SELECT * FROM City WHERE ID=3800\G"
***** 1. row *****
 id: 1
 select_type: SIMPLE
 table: City
 type: ALL
possible_keys: NULL
 key: NULL
 key_len: NULL
 ref: NULL
 rows: 4079
 Extra: Using where
```

- Is there any difference in the execution of this query from the query in step 1? YES
- Explain your answer.

The calculated constant could not be reduced to a single value prior to the execution of the query.

4. Issue the following statement in the mysql client:

```
sys> mysql -uroot -poracle -e" USE world; \
> EXPLAIN SELECT Country.Name \
> FROM Country, City WHERE City.CountryCode=Country.Code\G"
***** 1. row *****
 id: 1
 select_type: SIMPLE
 table: City
 type: ALL
possible_keys: NULL
 key: NULL
 key_len: NULL
 ref: NULL
 rows: 4079
 Extra:

***** 2. row *****
 id: 1
 select_type: SIMPLE
 table: Country
 type: eq_ref
possible_keys: PRIMARY
 key: PRIMARY
 key_len: 3
 ref: world.City.CountryCode
 rows: 1
 Extra:
```

- What is the select type for the *first* table being selected? SIMPLE
- What is the type of access strategy that MySQL plans to use for the *first* table? ALL
- What is the select type for the *second* table being selected? SIMPLE
- What is the type of access strategy that MySQL plans to use for the *second* table? eq\_ref

- Would this access strategy produce good performance or poor performance?  
The eq\_ref access strategy type states that MySQL plans to read one row from this second table for each combination of rows from the first table. These types of queries can produce good performance because all parts of an index are used by the join and the index is a primary key or UNIQUE NOT NULL index.

5. Add an index on the District field of the City table in the world database.

```
sys> mysql -uroot -poracle -e" USE world; \
> ALTER TABLE City ADD INDEX (District)"
```

6. Issue the following statement in the mysql client:

```
sys> mysql -uroot -poracle -e" USE world; \
> EXPLAIN SELECT * FROM City WHERE District='California'\G"
***** 1. row *****
 id: 1
 select_type: SIMPLE
 table: City
 type: ref
possible_keys: District
 key: District
 key_len: 20
 ref: const
 rows: 62
 Extra: Using where
```

- What is the select type for this query? SIMPLE
- What is the type of access strategy that MySQL plans to use for this query? ref
  - Would this access strategy produce good performance or poor performance?  
The ref access strategy type states that MySQL cannot select a single row based on the key value. These types of queries can produce good performance if the key that is used matches only a few rows.

7. Remove the index on the District field of the City table in the world database.

```
sys> mysql -uroot -poracle -e" USE world; \
> DROP INDEX District ON City"
```

8. Add an index on the IndepYear field of the Country table in the world database.

```
sys> mysql -uroot -poracle -e" USE world; \
> ALTER TABLE Country ADD INDEX (IndepYear)"
```

9. Issue the following statement in the mysql client:

```
sys> mysql -uroot -poracle -e"EXPLAIN SELECT * FROM Country \
> WHERE IndepYear=1905 OR IndepYear IS NULL\G"
***** 1. row *****
 id: 1
 select_type: SIMPLE
 table: Country
 type: ref_or_null
possible_keys: IndepYear
 key: IndepYear
 key_len: 3
 ref: const
 rows: 31
 Extra: Using where
```

- What is the select type for this query? SIMPLE
- What is the type of access strategy that MySQL plans to use for this query? ref or null
  - Would this access strategy produce good performance or poor performance?  
The ref or null access strategy type is similar to the ref access strategy but with the addition that MySQL does an extra search for rows that contain NULL values. These types of queries can produce good performance if the key that is used matches only a few rows.

10. Remove the index on the IndepYear field of the Country table in the world database.

```
sys> mysql -uroot -poracle -e" USE world; \
> DROP INDEX IndepYear ON Country"
```

11. Add an index on the District field of the City table in the world database.

```
sys> mysql -uroot -poracle -e" USE world; \
> ALTER TABLE City ADD INDEX (District)"
```

12. Issue the following statement in the mysql client:

```
sys> mysql -uroot -poracle -e" USE world; \
> EXPLAIN SELECT * FROM City \
> WHERE ID=50 OR District='Michigan'\G"
***** 1. row *****
 id: 1
 select_type: SIMPLE
 table: City
 type: index_merge
possible_keys: PRIMARY,District
 key: PRIMARY,District
 key_len: 4,20
 ref: NULL
 rows: 9
 Extra: Using union(PRIMARY,District); Using where
```

- What is the select type for this query? SIMPLE

- What is the type of access strategy that MySQL plans to use for this query?  
index merge
- Would this access strategy produce good performance or poor performance?  
The index merge access strategy type states that MySQL has found multiple indexes that could be used to complete the task (in this case, the primary key and the created District index). The Index Merge method retrieves rows with several range scans and merges their results into one. These types of queries can produce good performance if there are relatively high selectivity indexes on your table.

13. Remove the index on the District field of the City table in the world database.

```
sys> mysql -uroot -poracle -e" USE world; \
> DROP INDEX District ON City"
```

14. Issue the following statement in the mysql client:

```
sys> mysql -uroot -poracle -e" USE world; \
> EXPLAIN SELECT * FROM City \
> WHERE CountryCode IN \
> (SELECT code FROM Country WHERE CODE LIKE 'USA')\G"
***** 1. row *****
 id: 1
 select_type: PRIMARY
 table: City
 type: ALL
possible_keys: NULL
 key: NULL
 key_len: NULL
 ref: NULL
 rows: 4079
 Extra: Using where
***** 2. row *****
 id: 2
 select_type: DEPENDENT SUBQUERY
 table: Country
 type: unique_subquery
possible_keys: PRIMARY
 key: PRIMARY
 key_len: 3
 ref: func
 rows: 1
 Extra: Using index; Using where
```

- What is the select type for the *first* table being selected? PRIMARY
- What is the type of access strategy that MySQL plans to use for the *first* table? ALL
- What is the select type for the *second* table being selected? DEPENDENT SUBQUERY
- What is the type of access strategy that MySQL plans to use for the *second* table? unique subquery

- Would this access strategy produce good performance or poor performance?  
The unique\_subquery access strategy type states that MySQL creates an index lookup function that replaces the subquery completely for better efficiency. unique\_subquery queries have the potential to perform well because the optimizer can entirely replace the subquery with a set of constants; however, rewriting the query as a join has the potential to produce better performance.

15. Add an index on the Population field of the City table in the world database.

```
sys> mysql -uroot -poracle -e" USE world;\
> ALTER TABLE City ADD INDEX (Population)"
```

16. Issue the following statement in the mysql client:

```
sys> mysql -uroot -poracle -e" USE world;\
> EXPLAIN SELECT * FROM City \
> WHERE population>1000000\G"
***** 1. row *****
 id: 1
 select_type: SIMPLE
 table: City
 type: range
possible_keys: Population
 key: Population
 key_len: 4
 ref: NULL
 rows: 238
 Extra: Using where
```

- What is the select type for this query? SIMPLE
- What is the type of access strategy that MySQL plans to use for this query? range
- Would this access strategy produce good performance or poor performance?  
The range access strategy type states that MySQL retrieves only rows that are in a given range, using an index to select the rows. These types of queries can produce good performance because MySQL contains numerous methods to optimize range operations.

17. Remove the index on the Population field of the City table in the world database.

```
sys> mysql -uroot -poracle -e" USE world;\
> DROP INDEX Population ON City"
```

18. Issue the following statement in the mysql client:

```
sys> mysql -uroot -poracle -e" USE world; \
> EXPLAIN SELECT ID FROM City\G"
***** 1. row *****
 id: 1
 select_type: SIMPLE
 table: City
 type: index
possible_keys: NULL
 key: PRIMARY
 key_len: 4
 ref: NULL
 rows: 4079
 Extra: Using index
```

- What is the select type for this query? SIMPLE
- What is the type of access strategy that MySQL plans to use for this query? index
  - Would this access strategy produce good performance or poor performance?  
The index access strategy type states that MySQL scans only the index tree (not the table itself). These types of queries can be slightly better than a full tables scan, but they still have the potential for poor performance.

19. Issue the following statement in the mysql client:

```
sys> mysql -uroot -poracle -e" USE world; \
> EXPLAIN SELECT * FROM City \
> WHERE population>100000\G"
***** 1. row *****
 id: 1
 select_type: SIMPLE
 table: City
 type: ALL
possible_keys: NULL
 key: NULL
 key_len: NULL
 ref: NULL
 rows: 4079
 Extra: Using where
```

- What is the select type for this query? SIMPLE
- What is the type of access strategy that MySQL plans to use for this query? ALL
  - Would this access strategy produce good performance or poor performance?  
The ALL access strategy type states that MySQL performs a full table scan. These types of queries produce the poorest performance and should be avoided.

## Practice 10-2: Improve Query Executions

### Overview

In this practice, you improve the execution of a sample query. To accomplish this objective, you do the following:

- Use the `world` database to execute the sample `SELECT` statement against.
- Use the `EXPLAIN` command to evaluate the steps needed to reduce the negative performance issues associated with the query.
- Implement changes to the table to improve the performance of the query based on the outcome of your evaluation.

### Assumptions

- The MySQL server is installed and running.
- The `world` database is installed.

### Duration

This practice should take 20 minutes to complete.

### Tasks

1. Open a terminal window, start the `mysql` client, and select the `world` database.

**Note:** You can use the existing terminal window from the previous practice. However, at this point, you should have only one terminal window open to minimize confusion as more terminal windows are opened up.

2. Issue the following statement in the `mysql` client:

```
sys> mysql -uroot -poracle -e"USE world; \
 > EXPLAIN SELECT Name FROM City \
 > WHERE CountryCode = 'usa' ORDER BY Population \
 > DESC LIMIT 5\G"
```

- What is the select type for this query? \_\_\_\_\_
- What is the type of access strategy that MySQL plans to use for this query?

\_\_\_\_\_

- What are the possible keys that MySQL plans to use for this query?

\_\_\_\_\_

- What items are located in the `Extra` field output for `EXPLAIN`?

\_\_\_\_\_

- Would this query produce good performance or poor performance?

3. What action could you perform to remove the full table scan?

\_\_\_\_\_

4. Perform the action you believe can remove the full table scan and reissue the statement from step 1.

- What is the select type for this query? \_\_\_\_\_
- What is the type of access strategy that MySQL plans to use for this query?

\_\_\_\_\_

- What are the possible keys that MySQL plans to use for this query?

\_\_\_\_\_

- What items are located in the `Extra` field output for EXPLAIN?

\_\_\_\_\_

- Would this query produce good performance or poor performance?

5. What action could you perform to remove the need for the query to perform a `filesort`?

6. Perform the action you believe can remove the `filesort` and reissue the statement from step 1.

- What is the select type for this query? \_\_\_\_\_
- What is the type of access strategy that MySQL plans to use for this query?

\_\_\_\_\_

- What are the possible keys that MySQL plans to use for this query?

\_\_\_\_\_

- What items are located in the `Extra` field output for EXPLAIN?

\_\_\_\_\_

- Would this query produce good performance or poor performance?

7. What action could you perform to prevent this query from having to access the table at all?

8. Perform the action you believe can prevent this query from accessing the `City` table at all and reissue the statement from step 1.

- What is the select type for this query? \_\_\_\_\_
- What is the type of access strategy that MySQL plans to use for this query?

- What are the possible keys that MySQL plans to use for this query?

---

- What items are located in the `Extra` field output for `EXPLAIN`?

---

- Would this query produce good performance or poor performance?

---

9. Remove the modifications made to the `City` table during this task.

## Solutions 10-2: Improve Query Executions

---

### Tasks

1. Open a terminal window, start the mysql client, and select the world database.

**Note:** You can use the existing terminal window from the previous practice. However, at this point, you should have only one terminal window open to minimize confusion as more terminal windows are opened up.

2. Issue the following statement in the mysql client:

```
sys> mysql -uroot -poracle -e" USE world; \
> EXPLAIN SELECT Name FROM City \
> WHERE CountryCode = 'usa' ORDER BY Population \
> DESC LIMIT 5\G"
***** 1. row *****
 id: 1
 select_type: SIMPLE
 table: City
 type: ALL
possible_keys: NULL
 key: NULL
 key_len: NULL
 ref: NULL
 rows: 4079
 Extra: Using where; Using filesort
```

- What is the select type for this query? SIMPLE
- What is the type of access strategy that MySQL plans to use for this query? ALL
- What are the possible keys that MySQL plans to use for this query? None
- What items are located in the Extra field output for EXPLAIN? Using where;  
Using filesort
- Would this query produce good performance or poor performance?

This query would require a full scan of the data (ALL access strategy), has no keys that it can utilize, and would require a quicksort on the Population field (Using filesort in the Extra field). Due to these three negative performance issues, this query would produce poor performance.

**Note:** Using filesort states that MySQL performs quick sorts on chunks of data that can fit into its memory (size controlled by the sort\_buffer\_size system variable) and then performs a merge sort method to combine the chunks. If there is more than one chunk of data required to perform the quick sort, MySQL uses a temporary file to store the chunks.

3. What action could you perform to remove the full table scan?

Adding an index to the CountryCode column in the City table.

4. Perform the action you believe can remove the full table scan and reissue the statement from step 1.

```
sys> mysql -uroot -poracle -e" USE world;\
> ALTER TABLE City ADD INDEX (CountryCode)"
sys> mysql -uroot -poracle -e" USE world;\
> EXPLAIN SELECT Name FROM City \
> WHERE CountryCode = 'usa' ORDER BY Population \
> DESC LIMIT 5\G"
***** 1. row *****
 id: 1
 select_type: SIMPLE
 table: City
 type: ref
possible_keys: CountryCode
 key: CountryCode
 key_len: 3
 ref: const
 rows: 267
 Extra: Using where; Using filesort
```

- What is the select type for this query? SIMPLE
  - What is the type of access strategy that MySQL plans to use for this query? ref
  - What are the possible keys that MySQL plans to use for this query? CountryCode
  - What items are located in the Extra field output for EXPLAIN? Using where; Using filesort
  - Would this query produce good performance or poor performance?  
The query no longer requires a full scan of the data (ref access strategy), there is a key that can now be used, but would still require a quick sort on the Population field (Using filesort in the Extra field). This query has been improved greatly, but there is still some negative aspects of the query that could prevent the best performance possible.
5. What action could you perform to remove the need for the query to perform a filesort?
- Adding a composite index that includes the CountryCode and Population columns in the City table.

6. Perform the action you believe can remove the `filesort` and reissue the statement from step 1.

```
sys> mysql -uroot -poracle -e" USE world;\
> ALTER TABLE City \
> ADD INDEX Code_Pop (CountryCode, Population)"
sys> mysql -uroot -poracle -e" USE world;\
> EXPLAIN SELECT Name FROM City \
> WHERE CountryCode = 'usa' ORDER BY Population \
> DESC LIMIT 5\G"
***** 1. row *****
 id: 1
 select_type: SIMPLE
 table: City
 type: ref
possible_keys: CountryCode, Code_Pop
 key: Code_Pop
 key_len: 3
 ref: const
 rows: 267
 Extra: Using where
```

- What is the select type for this query? SIMPLE
  - What is the type of access strategy that MySQL plans to use for this query? ref
  - What are the possible keys that MySQL plans to use for this query?  
CountryCode, Code\_Pop
  - What items are located in the Extra field output for EXPLAIN? Using where;
    - Would this query produce good performance or poor performance?  
The query no longer requires a full scan of the data (ref access strategy), there is a key that can now be used, and no longer requires a quick sort on the Population field (Using filesort removed from the Extra field). There are no longer any negative performance issues that would stop this query from performing well.
7. What action could you perform to prevent this query from having to access the table at all?  
Adding a composite index that includes the CountryCode, Population and Name columns in the City table.

8. Perform the action you believe can prevent this query from accessing the City table at all and reissue the statement from step 1.

```
sys> mysql -uroot -poracle -e" USE world;\
> ALTER TABLE City \
> ADD INDEX Code_Pop_Name (CountryCode, Population, Name)"
sys> mysql -uroot -poracle -e" USE world;\
> EXPLAIN SELECT Name FROM City \
> WHERE CountryCode = 'usa' ORDER BY Population \
> DESC LIMIT 5\G"
***** 1. row *****
 id: 1
 select_type: SIMPLE
 table: City
 type: ref
possible_keys: CountryCode,Code_Pop,Code_Pop_Name
 key: Code_Pop_Name
key_len: 3
 ref: const
 rows: 188
 Extra: Using where; Using index
1 row in set (0.00 sec)
```

- What is the select type for this query? SIMPLE
- What is the type of access strategy that MySQL plans to use for this query? ref
- What are the possible keys that MySQL plans to use for this query?  
CountryCode, Code\_Pop, Code\_Pop\_Name
- What items are located in the Extra field output for EXPLAIN? Using where;  
Using index
- Would this query produce good performance or poor performance?  
This query would now only use the Code\_Pop\_Name index and would not need  
to access the City table at all (Using index from the Extra field). This query  
would now perform at the best performance possible; however, there could be  
negative performance issues when updates are made to the table due to the  
index needing to be updated also.

9. Remove the modifications made to the City table during this practice.

```
sys> mysql -uroot -poracle -e" USE world;\
> DROP INDEX CountryCode ON City"
sys> mysql -uroot -poracle -e" USE world;\
> DROP INDEX Code_Pop ON City"
sys> mysql -uroot -poracle -e" USE world;\
> DROP INDEX Code_Pop_Name ON City"
```

## Practice 10-3: Locate and Correct Problematic Queries

### Overview

In this practice, you locate and correct queries that are executing longer than determined to be acceptable. To accomplish this objective, you do the following:

- Review the current system variables associated with the slow query log.
- Set the system variables to control the types of queries to be captured by the slow query log.
- Use the `mysqlslap` application to execute a pre-defined script that contains multiple SQL statements.
- Review the statements captured by the slow query log and perform the steps necessary to improve the statement execution times.

### Assumptions

- The MySQL server is installed and running.
- The `mysqlslap` application is installed.
- The `isfdb` database is installed.
- The `isfdb_selects.sql` file is accessible and located in the `/stage/script` directory.

### Duration

This practice should take 60 minutes to complete.

### Tasks

1. Open a terminal window and review the current setting of system variables that start with the words “log”.
  - What is the current setting for the `log_output` system variable? \_\_\_\_\_

**Note:** You can use the existing terminal window from the previous practice. However, at this point, you should have only one terminal window open to minimize confusion as more terminal windows are opened up.
2. Review the current setting of system variables that start with the words “long”.
  - What is the current setting for the `long_query_time` system variable? \_\_\_\_\_
3. Review the current setting of system variables that start with the words “slow”.
  - What is the current setting for the `slow_query_log` system variable? \_\_\_\_\_
  - What is the current setting for the `slow_query_log_file` system variable? \_\_\_\_\_
4. Update the `slow_query_log_file` system variable to point to the `/tmp/mysql_slow.log` file.
5. Update the `long_query_time` system variable to 50 milliseconds.
6. Turn on the `slow_query_log` system variable.

**Note:** `log_slow_queries` has been deprecated by `slow_query_log` as of MySQL 5.1.29.

7. In a separate terminal window, execute the following command to obtain a dynamic real-time view of the `slow_query_log` file:

```
sys> tail -f /tmp/mysql_slow.log
```

- The output you see may be similar to the output below:

```
/usr/sbin/mysqld, Version: 5.5.8-log (MySQL Community Server (GPL)). started
with:
Tcp port: 3306 Unix socket: /var/lib/mysql/mysql.sock
```

8. In the first terminal window (step 6), use the `mysqlslap` application to create a load on the MySQL server to run the `isfdb_selects.sql` once by issuing the following command:

```
sys> mysqlslap \
> --user=root \
> --password=oracle \
> --delimiter=";" \
> --query="/stage/scripts/isfdb_selects.sql" \
> --create-schema="isfdb" \
> --iterations=1
```

9. In the second terminal window, view the output of the `slow_query_log` file.

- Which queries were captured by the `slow_query_log` file?

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

10. In the second terminal window (step 8), terminate the `tail -f /tmp/mysql_slow.log` execution then shut down the terminal window.

- Pressing the [CTRL] and [c] keys together terminates the execution of the `tail` command.

11. Start the `mysql` client and select the `isfdb` database.

12. Run the first SELECT statement issued from the `slow_query_log` file against the MySQL server.

- How long did the first SELECT statement take to execute? \_\_\_\_\_

13. Run EXPLAIN command for the first SELECT statement issued from the `slow_query_log` file against the MySQL server.

14. From reviewing the first SELECT statement issued from the `slow_query_log` file itself and reviewing the EXPLAIN output, what is the most obvious way to improve the performance of this query?

---

---

---

15. Perform the steps necessary to improve the performance of this query based on your evaluation in step 14 and execute your solutions against the server.

**Note:** If necessary, review the modeling map of the `isfdb` database in appendix A to determine the best solutions for the remainder of this practice.

- How long did your solution take to complete? \_\_\_\_\_
- Was there a noticeable improvement?
  - If your answer is yes, congratulations!
  - If your answer is no, attempt another solution. (It is most likely that your best solution runs under 2 seconds.)

16. Run the second SELECT statement issued from the `slow_query_log` file against the MySQL server.

- How long did the second SELECT statement take to execute? \_\_\_\_\_

**Note:** The actual execution time may not seem excessive, but it is more than the 50 milliseconds that was set in the `long_query_time` system variable.

17. Run EXPLAIN command for the second SELECT statement issued from the `slow_query_log` file against the MySQL server.

18. From reviewing the second SELECT statement issued from the `slow_query_log` file itself and reviewing the EXPLAIN output, what is the most obvious way to improve the performance of this query?

---

---

---

19. Perform the steps necessary to improve the performance of this query based on your evaluation in step 19 and execute your solutions against the server.

- How long did your solution take to complete? \_\_\_\_\_
- Was there a noticeable improvement? \_\_\_\_\_
  - If your answer is yes, congratulations!
  - If your answer is no, attempt another solution. (It is most likely that your best solution runs 30 milliseconds.)

20. Run the third SELECT statement issued from the slow\_query\_log file against the MySQL server.

- How long did the third SELECT statement take to execute? \_\_\_\_\_

21. Run EXPLAIN command for the third SELECT statement issued from the slow\_query\_log file against the MySQL server.

22. From reviewing the third SELECT statement issued from the slow\_query\_log file itself and reviewing the EXPLAIN output, what is the most obvious way to improve the performance of this query?

---

---

23. Perform the steps necessary to improve the performance of this query based on your evaluation in step 12 and execute your solutions against the server.

- How long did your solution take to complete? \_\_\_\_\_
- Was there a noticeable improvement? \_\_\_\_\_
  - If your answer is yes, congratulations!
  - If your answer is no, attempt another solution. (It is most likely that your best solution runs 0 . 8 seconds.)

## Solutions 10-3: Locate and Correct Problematic Queries

### Tasks

1. Open a terminal window and review the current setting of system variables that start with the words “log”.

```
sys> mysql -uroot -poracle -e"SHOW VARIABLES LIKE 'log%'"
```

| Variable_name                   | Value                        |
|---------------------------------|------------------------------|
| log                             | OFF                          |
| log_bin                         | ON                           |
| log_bin_trust_function_creators | OFF                          |
| log_error                       | /var/lib/mysql/EDTDR14P2.err |
| log_output                      | FILE                         |
| log_queries_not_using_indexes   | OFF                          |
| log_slave_updates               | OFF                          |
| log_slow_queries                | OFF                          |
| log_warnings                    | 1                            |

- What is the current setting for the `log_output` system variable? FILE

**Note:** You can use the existing terminal window from the previous practice. However, at this point, you should have only one terminal window open to minimize confusion as more terminal windows are opened up.

2. Review the current setting of system variables that start with the words “long”.

```
sys> mysql -uroot -poracle -e"SHOW VARIABLES LIKE 'long%'"
```

| Variable_name   | Value     |
|-----------------|-----------|
| long_query_time | 10.000000 |

- What is the current setting for the `long_query_time` system variable? 10.00

3. Review the current setting of system variables that start with the words “slow”.

```
sys> mysql -uroot -poracle -e"SHOW VARIABLES LIKE 'slow%'"
```

| Variable_name       | Value                             |
|---------------------|-----------------------------------|
| slow_launch_time    | 2                                 |
| slow_query_log      | OFF                               |
| slow_query_log_file | /var/lib/mysql/EDTDR14P2-slow.log |

- What is the current setting for the `slow_query_log` system variable? OFF
- What is the current setting for the `slow_query_log_file` system variable?

/var/lib/mysql/EDTDR14P2-slow.log

4. Update the `slow_query_log_file` system variable to point to the `/tmp/mysql_slow.log` file.

```
sys> mysql -uroot -poracle -e"SET GLOBAL \
> slow_query_log_file = '/tmp/mysql_slow.log'"
```

5. Update the `long_query_time` system variable to 50 milliseconds.

```
sys> mysql -uroot -poracle -e"SET GLOBAL \
> long_query_time = 0.05"
```

6. Turn on the `slow_query_log` system variable.

```
sys> mysql -uroot -poracle -e"SET GLOBAL slow_query_log = ON"
```

**Note:** `log slow queries` has been deprecated by `slow query log` as of MySQL 5.1.29.

7. In a separate terminal window, execute the following command to obtain a dynamic real-time view of the `slow_query_log` file:

```
sys> tail -f /tmp/mysql_slow.log
```

- The output you see may be similar to the output below:

```
/usr/sbin/mysqld, Version: 5.5.8-log (MySQL Community Server (GPL)). started
with:
Tcp port: 3306 Unix socket: /var/lib/mysql/mysql.sock
```

8. In the first terminal window (step 6), use the `mysqlslap` application to create a load on the MySQL server to run the `isfdb_selects.sql` one iteration by issuing the following command:

```
sys> mysqlslap \
> --user=root \
> --password=oracle \
> --delimiter=";" \
> --query="/stage/scripts/isfdb_selects.sql" \
> --create-schema="isfdb" \
> --iterations=1
```

9. In the second terminal window, view the output of the `slow_query_log` file.

- Which queries were captured by the `slow_query_log` file?

```
SELECT DISTINCT (SELECT author canonical FROM authors WHERE
author_id=4) AS author, pub_title FROM pubs WHERE pub_id IN
(SELECT pub_id FROM pub_authors WHERE author_id=4) ORDER BY
pub_title;
```

```
SELECT "A's" AS LastName, COUNT(*) FROM authors WHERE
author_lastname LIKE 'A%';
```

```
SELECT author canonical, COUNT(award_id) AS Number_Awards FROM
authors, title_awards, canonical_author WHERE
canonical_author.author_id = authors.author_id AND
title_awards.title_id = canonical_author.title_id GROUP BY
author canonical ORDER BY author_lastname;
```

10. In the second terminal window (step 8), terminate the `tail -f /tmp/mysql_slow.log` execution then shut down the terminal window.

- Pressing the [CTRL] and [c] keys together terminates the execution of the `tail` command.

11. Start the mysql client and select the isfdb database.

```
sys> mysql -uroot -poracle

mysql> USE isfdb
```

12. Run the first SELECT statement issued from the slow\_query\_log file against the MySQL server.

```
mysql> SELECT DISTINCT
-> (SELECT author_canonical FROM authors
-> WHERE author_id=4)
-> AS author, pub_title FROM pubs WHERE pub_id IN
-> (SELECT pub_id FROM pub_authors WHERE author_id=4)
-> ORDER BY pub_title;
...
189 rows in set (5.78 sec)
```

- How long did the first SELECT statement take to execute? 5.78 seconds

13. Run EXPLAIN command for the first SELECT statement issued from the slow\_query\_log file against the MySQL server.

```
mysql> EXPLAIN SELECT DISTINCT
-> (SELECT author_canonical FROM authors
-> WHERE author_id=4)
-> AS author, pub_title FROM pubs WHERE pub_id IN
-> (SELECT pub_id FROM pub_authors WHERE author_id=4)
-> ORDER BY pub_title\G

 1. row *****
 id: 1
 select_type: PRIMARY
 table: pubs
 type: ALL
possible_keys: NULL
 key: NULL
 key_len: NULL
 ref: NULL
 rows: 176458
 Extra: Using where; Using temporary; Using filesort

 2. row *****
 id: 3
 select_type: DEPENDENT SUBQUERY
 table: pub_authors
 type: index_subquery
possible_keys: pub_id,author_id
 key: pub_id
 key_len: 5
 ref: func
 rows: 1
 Extra: Using where
```

```
***** 3. row *****
 id: 2
 select_type: SUBQUERY
 table: authors
 type: const
possible_keys: PRIMARY
 key: PRIMARY
 key_len: 4
 ref: const
 rows: 1
 Extra:
3 rows in set (0.00 sec)
```

14. From reviewing the first SELECT statement issued from the slow\_query\_log file itself and reviewing the EXPLAIN output, what is the most obvious way to improve the performance of this query?

Rewriting the query to remove the subqueries.

15. Perform the steps necessary to improve the performance of this query based on your evaluation in step 14 and execute your solutions against the server.

**Note:** If necessary, review the modeling map of the isfdb database in appendix A to determine the best solutions for the remainder of this practice.

```
mysql> SELECT DISTINCT author_canonical,pub_title
 -> FROM authors,pub_authors,pubs WHERE
 -> pub_authors.author_id=authors.author_id AND
 -> pub_authors.pub_id=pubs.pub_id AND authors.author_id=4
 -> ORDER BY pub_title;
...
189 rows in set (1.81 sec)
```

- How long did your solution take to complete? 1.81 seconds
- Was there a noticeable improvement? Yes
  - If your answer is yes, congratulations!
  - If your answer is no, attempt another solution. (It is most likely that your best solution runs under 2 seconds.)

16. Run the second SELECT statement issued from the slow\_query\_log file against the MySQL server.

```
mysql> SELECT "A's" AS LastName, COUNT(*)
 -> FROM authors WHERE author_lastname LIKE 'A%';
+-----+-----+
| LastName | COUNT(*) |
+-----+-----+
| A's | 2416 |
+-----+-----+
1 row in set (0.08 sec)
```

- How long did the second SELECT statement take to execute? 0.08 seconds

**Note:** The actual execution time may not seem excessive, but it is more than the 50 milliseconds that was set in the long\_query\_time system variable.

17. Run EXPLAIN command for the second SELECT statement issued from the slow\_query\_log file against the MySQL server.

```
mysql> EXPLAIN SELECT "A's" AS LastName, COUNT(*)
 -> FROM authors WHERE author_lastname LIKE 'A%\G

 id: 1
 select_type: SIMPLE
 table: authors
 type: ALL
possible_keys: NULL
 key: NULL
 key_len: NULL
 ref: NULL
 rows: 68700
 Extra: Using where
1 row in set (0.00 sec)
```

18. From reviewing the second SELECT statement issued from the slow\_query\_log file itself and reviewing the EXPLAIN output, what is the most obvious way to improve the performance of this query?

By adding an additional index to the authors table based on the author\_lastname.

19. Perform the steps necessary to improve the performance of this query based on your evaluation in step 18 and execute your solutions against the server.

```
mysql> ALTER TABLE authors ADD INDEX (author_lastname(10));

mysql> SELECT "A's" AS LastName, COUNT(*)
 -> FROM authors WHERE author_lastname LIKE 'A%';
+-----+-----+
| LastName | COUNT(*) |
+-----+-----+
| A's | 2416 |
+-----+-----+
1 row in set (0.03 sec)
```

- How long did your solution take to complete? 0.03 seconds
- Was there a noticeable improvement? Yes
  - If your answer is yes, congratulations!
  - If your answer is no, attempt another solution. (It is most likely that your best solution runs 30 milliseconds.)

**Note:** BLOB and TEXT columns can be indexed, but a prefix length must be given. author\_lastname is a MEDIUMTEXT column which requires you to identify a prefix length.

20. Run the third SELECT statement issued from the slow\_query\_log file against the MySQL server.

```
mysql> SELECT author_canonical,
-> COUNT(award_id) AS Number_Awards FROM authors,
-> title_awards, canonical_author
-> WHERE canonical_author.author_id = authors.author_id
-> AND title_awards.title_id = canonical_author.title_id
-> GROUP BY author_canonical ORDER BY author_lastname;
...
3054 rows in set (1.25 sec)
```

- How long did the third SELECT statement take to execute? 1.25 seconds

21. Run EXPLAIN command for the third SELECT statement issued from the slow\_query\_log file against the MySQL server.

```
mysql> EXPLAIN SELECT author_canonical,
-> COUNT(award_id) AS Number_Awards FROM authors,
-> title_awards, canonical_author
-> WHERE canonical_author.author_id = authors.author_id
-> AND title_awards.title_id = canonical_author.title_id
-> GROUP BY author_canonical ORDER BY author_lastname\G

 1. row ****
 id: 1
 select_type: SIMPLE
 table: title_awards
 type: ALL
possible_keys: title_id
 key: NULL
 key_len: NULL
 ref: NULL
 rows: 17879
 Extra: Using temporary; Using filesort

 2. row ****
 id: 1
 select_type: SIMPLE
 table: canonical_author
 type: ref
possible_keys: titles,authors
 key: titles
 key_len: 5
 ref: isfdb.title_awards.title_id
 rows: 1
 Extra: Using where

 3. row ****
 id: 1
 select_type: SIMPLE
 table: authors
 type: eq_ref
possible_keys: PRIMARY
 key: PRIMARY
 key_len: 4
 ref: isfdb.canonical_author.author_id
 rows: 1
 Extra:
3 rows in set (0.00 sec)
```

22. From reviewing the third SELECT statement issued from the slow\_query\_log file itself and reviewing the EXPLAIN output, what is the most obvious way to improve the performance of this query?

Create a temporary table that stores the author\_id and the award count using the canonical\_author and title\_awards tables. Then join the temporary table to the authors table to connect the awards to the author's name.

23. Perform the steps necessary to improve the performance of this query based on your evaluation in step 12 and execute your solutions against the server.

```
mysql> CREATE TEMPORARY TABLE award_count
-> (author_id INT PRIMARY KEY, award_count INT)
-> SELECT author_id, COUNT(award_id) AS award_count
-> FROM canonical_author,title_awards
-> WHERE title_awards.title_id =
-> canonical_author.title_id
-> GROUP BY author_id;
Query OK, 3054 rows affected (0.62 sec)
Records: 3054 Duplicates: 0 Warnings: 0
mysql> SELECT author_canonical, award_count FROM authors,
-> award_count WHERE authors.author_id =
-> award_count.author_id ORDER BY author_lastname;
3054 rows in set (0.11 sec)
```

- How long did your solution take to complete? 0.73 seconds (combined)
- Was there a noticeable improvement? Yes
  - If your answer is yes, congratulations!
  - If your answer is no, attempt another solution. (It is most likely that your best solution runs 0.8 seconds.)

# **Practices for Lesson 11: Performance Tuning Extras**

## **Chapter 11**

## **Overview of Practices for Lesson 11**

---

### **Practices for Lesson 11**

There are no practices for this lesson.

## **Practices for Lesson 12: Conclusion**

**Chapter 12**

## Overview of Practices for Lesson 12

---

### Practices for Lesson 12

There are no practices for this lesson.