



Integrated Cloud Applications & Platform Services

MySQL Performance Tuning

Student Guide

D61820GC40

Edition 4.0 | February 2018 | D102561

Learn more from Oracle University at education.oracle.com



Author

Mark Lewin

**Technical Contributors
and Reviewers**

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

Editors

Arijit Ghosh
Aju Kumar
Aishwarya Menon
Vijayalakshmi Narasimhan

Graphic Editors

Kavya Bellur
Seema Bopaiah
Prakash Dharmalingam

Publishers

Pavithran Adka
Asief Baig
Srividya Rameshkumar
Jobi Varghese

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

Disclaimer

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

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

Restricted Rights Notice

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

U.S. GOVERNMENT RIGHTS

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

Trademark Notice

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

Contents

1 Introduction

- Objectives 1-2
- Course Goals 1-3
- Course Lesson Map 1-4
- Introductions 1-6
- Classroom Environment 1-7
- A Modern Database for the Digital Age 1-8
- High Scalability 1-9
- MySQL Enterprise Edition 1-10
- Oracle Premier Support for MySQL 1-11
- Why MySQL Enterprise Edition? 1-12
- MySQL and Oracle Integration 1-13
- MySQL Cloud Service 1-14
- MySQL Cloud Service: Product Overview 1-15
- Use Cases 1-17
- Administering MySQL Cloud Service Instances 1-18
- MySQL Websites 1-19
- Community Resources 1-20
- Oracle University: MySQL Training 1-21
- MySQL Certification 1-22
- MySQL-Supported Operating Systems 1-23
- Summary 1-24
- Practice 1-25

2 Performance Tuning Concepts

- Topics 2-2
- Objectives 2-3
- Topics 2-4
- Improving Performance 2-5
- Areas to Tune 2-6
- Topics 2-7
- Performance Tuning Terminology 2-8
- Response Time 2-9
- Measuring Response Times 2-10
- Throughput 2-11

Scalability	2-12
Queuing Theory	2-13
Quiz	2-14
Topics	2-15
Benchmarking	2-16
Benchmarking Best Practices	2-17
More Benchmarking Best Practices	2-18
Benchmarking Bad Practices	2-19
More Benchmarking Bad Practices	2-20
Quiz	2-21
Topics	2-22
Before You Start: Establish a Performance Baseline	2-23
Key Steps in Troubleshooting Performance Issues	2-24
Establishing the Nature of the Problem	2-25
Starting to Troubleshoot	2-26
Identifying Possible Causes of Performance Issues	2-27
Application Profiling	2-28
Localizing Database Problems	2-29
Topics	2-30
Tuning Steps	2-31
General Tuning Procedure	2-32
Topics	2-33
Deploying MySQL	2-34
RAM Requirements	2-35
Maintaining MySQL	2-36
Summary	2-37
Practice	2-38

3 Performance Tuning Tools

Topics	3-2
Objectives	3-3
Topics	3-4
MySQL Monitoring Tools	3-5
SHOW [SESSION GLOBAL] STATUS	3-6
Filtering SHOW STATUS Output	3-7
Status Variables: Handler Operations	3-8
Status Variables: Temporary Tables and Files	3-10
SHOW STATUS: Examples	3-11
SHOW ENGINE INNODB STATUS	3-13
SHOW PROCESSLIST	3-14
Quiz	3-15

mysqladmin 3-16
MySQL Utilities 3-17
Information Schema 3-18
Performance Schema 3-19
MySQL Workbench 3-20
MySQL Enterprise Monitor: Overview 3-21
MySQL Enterprise Monitor: Architecture 3-22
MySQL Enterprise Monitor: Global Overview 3-23
MySQL Enterprise Monitor: Query Analyzer 3-24
MySQL Enterprise Monitor: InnoDB Performance 3-25
Topics 3-26
Oracle Enterprise Manager for MySQL 3-27
Oracle SQL Developer 3-28
Quiz 3-29
Topics 3-30
Community Monitoring Tools 3-31
Topics 3-33
Linux Tools 3-34
iostat 3-35
vmstat 3-36
sar 3-38
top 3-39
Topics 3-40
Benchmarking Tools 3-41
MySQL BENCHMARK() Function 3-42
Stress Tools 3-43
mysqlslap Stress Tool 3-44
sysbench 3-45
Using SysBench 3-46
Sample SysBench Output - 1 3-47
Sample SysBench Output - 2 3-48
Summary 3-49
Practices 3-50

4 Performance Schema

Topics 4-2
Objectives 4-3
Topics 4-4
Why Use Performance Schema? 4-5
Performance Schema 4-6
How Performance Schema Works 4-7

Instrument Names	4-8
Instrument Prefixes	4-9
Instrument Suffixes	4-10
Instrument Names: Example	4-11
Topics	4-12
Structure of the Performance Schema	4-13
Table Groups	4-14
Topics	4-16
Configuring the Performance Schema	4-17
Setup Tables	4-18
Configuring Instruments	4-19
Configuring Instruments: Examples	4-20
Event Timing	4-21
Configuring Consumers	4-22
Consumer Table Hierarchy	4-23
Configuring Consumers	4-24
Configuring Objects	4-25
Default setup_objects Configuration	4-26
Configuring Actors	4-27
Configuring Thread Monitoring	4-28
Configuring Instruments and Consumers at Startup	4-29
Performance Schema Overhead	4-30
How Performance Schema Uses Memory	4-31
Configuring Performance Schema System Variables	4-32
Quiz	4-33
Topics	4-34
Instance Tables	4-35
cond_instances Table	4-36
file_instances Table	4-37
Read/Write Locks and Mutexes	4-38
mutex_instances Table	4-39
Mutex Life Cycle in the Performance Schema	4-40
rwlock_instances Table	4-41
socket_instances Table	4-42
Identifying Connections in the socket_instances Table	4-43
Connection Tables	4-44
Retrieving Detailed Connection Information	4-45
Quiz	4-46
Topics	4-47
Examining Raw Event Data: Example 1	4-48
Examining Raw Event Data: Example 2	4-49

Examining Raw Event Data: Example 3	4-50
Event Hierarchy	4-51
Nested Events	4-52
Querying Nested Events	4-54
Summary Tables	4-55
Prepared Statements	4-56
Retrieving Statement Summary Information: Example	4-57
Retrieving Wait Summary Information: Example	4-58
Retrieving Memory Summary Information: Example	4-59
Topics	4-61
sys Schema	4-62
Setting sys Schema Privileges	4-63
sys Views	4-64
Using sys for Statement Analysis: Example	4-67
Using sys to Identify Queries with Full Table Scans: Example	4-68
Using sys to Identify the Slowest Queries: Example	4-69
Using sys to Identify Unused Indexes: Example	4-70
Using sys as a Non-Blocking SHOW PROCESSLIST: Example	4-71
sys Stored Routines	4-72
Configuring Performance Schema by Using sys: Example	4-74
Using sys Functions to Format Output: Example	4-76
Quiz	4-77
Topics	4-78
MySQL Workbench Performance Dashboard	4-79
MySQL Workbench Performance Reports	4-80
MySQL Workbench Performance Schema Setup	4-81
Summary	4-82
Practices	4-83

5 General Server Tuning

Topics	5-2
Objectives	5-3
Topics	5-4
Major Components of the MySQL Server	5-5
Quiz	5-6
MySQL Memory Usage	5-7
Tuning the MySQL Server	5-8
CPU and I/O Saturation	5-9
Global Buffers	5-10
MySQL Thread Handling	5-11
Per-Thread Buffers	5-12

Topics	5-16
Estimating Maximum MySQL Memory Usage	5-17
Scenario: Total MySQL Memory Usage	5-18
Displaying Memory Usage by Using sys	5-19
Quiz	5-20
Topics	5-21
Simultaneous Connections in MySQL	5-22
Connection Status Variables	5-23
Monitoring Idle Connections with Performance Schema	5-24
Scenario: Users Unable to Connect	5-25
Diagnosing Network Problems	5-26
Topics	5-27
Reusing Threads	5-28
Quiz	5-30
Other Thread Status Variables	5-31
Scenario: Calculating Thread Cache Effectiveness	5-32
Scenario: Setting <code>thread_cache_size</code>	5-33
The MySQL Enterprise Thread Pool	5-34
Summary	5-36
Practices	5-37

6 Tuning Tables, Files, and Logs

Topics	6-2
Objectives	6-3
Topics	6-4
Reusing Tables	6-5
How MySQL Caches Tables	6-6
Setting <code>table_open_cache</code>	6-7
Sizing the Table Cache	6-8
Setting <code>table_definition_cache</code>	6-9
Setting <code>table_open_cache</code> : Scenario 1	6-10
Setting <code>table_open_cache</code> : Scenario 2	6-11
Setting <code>table_open_cache</code> : Scenario 3	6-12
Quiz	6-13
Topics	6-14
Tables and Files	6-15
Managing the Number of Open Files	6-16
Setting File Descriptors in the Operating System	6-17
Quiz	6-18
Topics	6-19
Binary Logs	6-20

ACID and Database Transactions	6-21
Transactions and the Binary Log	6-22
Sizing the Binary Log Caches	6-23
Monitoring Binary Logs	6-24
Binary Log Group Commit Settings	6-25
Improving Binary Log Performance	6-26
Scenario: Binary Log File Cache Effectiveness	6-27
Quiz	6-28
Summary	6-29
Practices	6-30

7 Tuning InnoDB

Topics	7-2
Objectives	7-3
Topics	7-4
InnoDB Storage Engine	7-5
Topics	7-6
Operating System Buffers	7-7
How InnoDB Reads Data	7-8
How InnoDB Writes Data	7-9
Topics	7-10
Sizing the InnoDB Buffer Pool	7-11
Dumping and Restoring the Buffer Pool	7-12
Sizing the InnoDB Transaction Log Files	7-13
Other InnoDB Transaction Log Settings	7-15
InnoDB Tablespace Settings	7-16
Thread Concurrency	7-18
InnoDB Buffer and Log File Flushing	7-19
Adaptive Flushing	7-21
Purge Behavior	7-22
Undo Logs	7-24
Recommended InnoDB Settings for OLTP and Benchmarking	7-25
Recommended InnoDB Settings for Replication	7-27
Linux Filesystem Recommendations	7-28
Topics	7-29
InnoDB Support in the Information Schema	7-30
Information Schema InnoDB Metrics	7-32
Example: Enabling InnoDB Monitoring	7-33
Example: Displaying InnoDB Metrics	7-34
Example: Resetting InnoDB Counters and Disabling Monitoring	7-35
InnoDB Support in the Performance Schema	7-36

Monitoring InnoDB Performance in MySQL Workbench	7-37
SHOW ENGINE INNODB STATUS	7-38
SHOW ENGINE INNODB STATUS: Semaphores and Transactions	7-39
SHOW ENGINE INNODB STATUS: File I/O	7-41
SHOW ENGINE INNODB STATUS: Insert Buffer and Adaptive Hash Index	7-42
SHOW ENGINE INNODB STATUS: Log	7-43
SHOW ENGINE INNODB STATUS: Buffer Pool and Memory	7-44
SHOW ENGINE INNODB STATUS: Row Operations	7-45
SHOW ENGINE INNODB STATUS: Latest Foreign Key Error	7-46
SHOW ENGINE INNODB STATUS: Latest Detected Deadlock	7-47
Quiz	7-48
Summary	7-49
Practices	7-50

8 Optimizing Your Schema

Topics	8-2
Objectives	8-3
Topics	8-4
Schema Design Considerations	8-5
Schema Design Tasks	8-6
Normalization and Performance	8-7
Denormalizing Specific Data for Performance	8-8
Quiz	8-10
Data Types	8-11
Topics	8-12
Indexes	8-13
Indexing	8-15
Optimizing Indexes	8-16
Index Types	8-17
B+TREE Index	8-18
Quiz	8-20
Topics	8-21
InnoDB Table Compression	8-22
Tuning Compression for InnoDB Tables	8-23
Topics	8-24
Partitioning	8-25
Partitioning Types	8-26
RANGE Partitioning: Example	8-27
LIST Partitioning: Example	8-28
HASH Partitioning: Example	8-29
KEY Partitioning: Example	8-30

Partitioning and Performance	8-31
Partitioning Limitations	8-32
Retrieving Partition Information	8-33
Quiz	8-34
Summary	8-35
Practices	8-36

9 Monitoring Queries

Topics	9-2
Objectives	9-3
Topics	9-4
Query Monitoring	9-5
Identifying Queries That Require Optimization	9-6
Identifying Frequent Queries	9-7
Topics	9-8
General Query Log	9-9
Slow Query Log	9-10
Using mysqldumpslow to View Slow Query Log Entries	9-12
mysqldumpslow Output: Example	9-13
Topics	9-14
General Statement Status Variables	9-15
SELECT Statement Status Variables	9-17
Quiz	9-19
Topics	9-20
Finding Slow Queries with Performance Schema	9-21
Example: Querying events_statements_current	9-22
Example: Querying events_statements_summary_by_digest	9-23
Identifying Slow Queries with the sys Schema	9-24
Investigating Aggregated Statement Metrics with the sys.statement_analysis View	9-25
Topics	9-26
MySQL Enterprise Monitor Query Analyzer	9-27
Query Analyzer Tab	9-28
Query Analyzer Detailed Query View	9-29
MySQL Workbench Query Statistics	9-30
MySQL Workbench Visual EXPLAIN	9-31
Summary	9-32
Practices	9-33

10 Query Optimization

- Topics 10-2
- Objectives 10-3
- Topics 10-4
- MySQL Query Processing 10-5
- Optimizer Main Stages 10-6
- Logical Transformations 10-7
- Example: Logical Transformation 10-8
- Cost-Based Optimization 10-10
- Cost Model Inputs 10-11
- Access Method 10-12
- Join Order 10-13
- Subquery Optimizations 10-15
- Plan Refinement 10-17
- Index Condition Pushdown Optimization 10-18
- Quiz 10-19
- Topics 10-20
- Understanding the Query Plan 10-21
- Example: EXPLAIN Output 10-22
- EXPLAIN Output 10-23
- EXPLAIN Output: select_type Column 10-25
- EXPLAIN Output: type Column 10-26
- EXPLAIN Output: key Column and Index Hints 10-28
- EXPLAIN Output: Extra Column 10-29
- EXPLAIN Output: Extra column 10-30
- Structured EXPLAIN 10-31
- Example: Standard EXPLAIN 10-32
- Example: EXPLAIN FORMAT=JSON 10-33
- Visual EXPLAIN in MySQL Workbench 10-34
- Quiz 10-35
- Optimizer Trace 10-40
- Topics 10-41
- Improving Query Performance 10-42
- Data Retrieval: Best Practices 10-43
- Filtering Data with WHERE 10-44
- Indexing for Query Performance 10-45
- Query Using WHERE Clause 10-46
- Query Using Covering Index 10-47
- Query Using Index Condition Pushdown 10-48
- Costs of Indexing 10-49
- Indexing: Best Practices 10-50

Leftmost Prefixes	10-51
Improving the Performance of SELECT Statements	10-52
Optimizing Table Joins	10-53
Optimizing Sorting Operations	10-54
Order of Sort Operations	10-55
Optimizing Bulk DML Statements	10-56
Creating Summary Tables	10-57
Quiz	10-58
Summary	10-59
Practices	10-60

11 Optimizing Locking Operations

Topics	11-2
Objectives	11-3
Topics	11-4
Locks in MySQL	11-5
Implicit Locks	11-6
Explicit Locks	11-7
Avoiding Locks with InnoDB Multiversion Concurrency Control	11-8
Topics	11-9
InnoDB Lock Modes	11-10
Lock Type Compatibility Matrix	11-11
Topics	11-12
Metadata Locks	11-13
Metadata Lock Example	11-14
Topics	11-15
Identifying Locks in the Process List	11-16
Example: Table Lock Information in SHOW PROCESSLIST	11-17
Displaying InnoDB Lock Information with SHOW ENGINE INNODB STATUS	11-18
Topics	11-19
Viewing Metadata Lock Information in the Performance Schema	11-20
Interpreting metadata_locks.LOCK_STATUS	11-21
Example: Querying the metadata_locks Table	11-22
Viewing Table Lock Information in the Performance Schema	11-23
Example: Querying the table_handles Table	11-24
Viewing Table Lock Information in sys.innodb_lock_waits	11-25
Quiz	11-26
Summary	11-27
Practices	11-28

12 Tuning Replication

- Topics 12-2
- Objectives 12-3
- Topics 12-4
- MySQL Replication 12-5
- Replication Use Cases 12-6
- Replication Logs 12-7
- Replication Log Events 12-8
- Role of the Master 12-9
- Role of the Slave 12-10
- Topics 12-14
- What Is Replication Lag? 12-15
- Common Causes of Replication Lag 12-16
- Quiz 12-17
- Topics 12-18
- Diagnosing Replication Lag 12-19
- Binary Log File Name and Position of the Master 12-20
- Binary Log File Name and Position of the Slave 12-21
- Comparing Binary Log File Name and Position 12-22
- Timing Replication Lag 12-23
- Comparing GTID Sets 12-24
- Executed GTIDs 12-25
- Retrieved GTIDs 12-26
- Purged GTIDs 12-27
- Example: I/O Thread Lag 12-28
- Example: SQL Thread Lag 12-29
- Example: SQL Thread Lagging Behind I/O Thread 12-30
- Performance Schema Replication Table Hierarchy 12-31
- Performance Schema Replication Tables 12-32
- Replication Support in MySQL Enterprise Monitor 12-33
- Quiz 12-34
- Topics 12-35
- Resolving I/O Thread Lag 12-36
- Resolving SQL Thread Lag 12-37
- Summary 12-38
- Practices 12-39

13 Conclusion

- Course Goals 13-2
- Oracle University: MySQL Training 13-3
- MySQL Websites 13-4
- Your Evaluation 13-5
- Thank You 13-6
- Q&A Session 13-7

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

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

Introduction

1



MySQL™

ORACLE®

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

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

Objectives



After completing this lesson, you should be able to:

- Describe the course goals
- List MySQL products and services
- State the benefits of using MySQL Cloud Service
- Determine support for your operating system
- Access MySQL information and services from Oracle websites and community resources
- Find information about MySQL courses and certification options

ORACLE®

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

Course Goals

After completing this course, you should be able to:

- Understand performance tuning concepts
- List factors that affect performance
- Use a range of performance tuning tools
- Configure and use the Performance Schema
- Tune the MySQL server instance
- Design a schema for optimal performance
- Understand how MySQL optimizes queries
- Identify and fix slow queries
- Diagnose and resolve common performance issues
- Identify and fix replication lag



ORACLE®

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

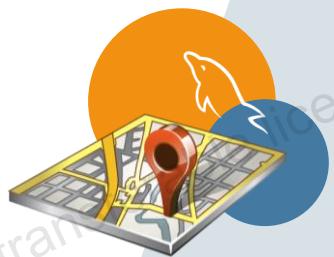
Course Lesson Map

Day 1

- Introduction
- Performance Tuning Concepts
- Performance Tuning Tools

Day 2

- Performance Schema
- General Server Tuning
- Tuning Tables, Files, and Logs



ORACLE®

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

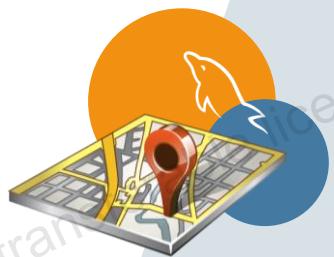
Course Lesson Map

Day 3

- Tuning InnoDB
- Optimizing Your Schema
- Monitoring Queries

Day 4

- Optimizing Queries
- Optimizing Locking Operations
- Tuning Replication



ORACLE®

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

Introductions

- Name
- Company affiliation
- Title, function, and job responsibilities
- Experience related to topics covered in this course
- Reason for enrolling in this course
- Expectations for this course



ORACLE®

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

Classroom Environment

- Logistics
 - Restrooms
 - Break rooms and designated smoking areas
 - Cafeterias and restaurants in the area
- Emergency evacuation procedures
- Instructor contact information
- Mobile phone usage
- Online course attendance confirmation form

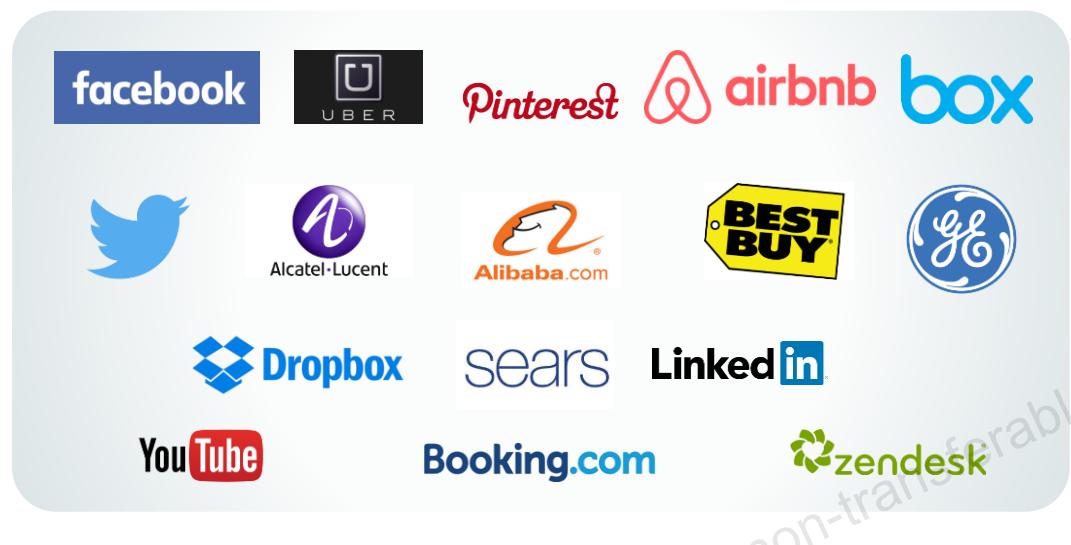


ORACLE®

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

A Modern Database for the Digital Age

Digital Disruptors and Large Enterprises Rely on MySQL to Innovate

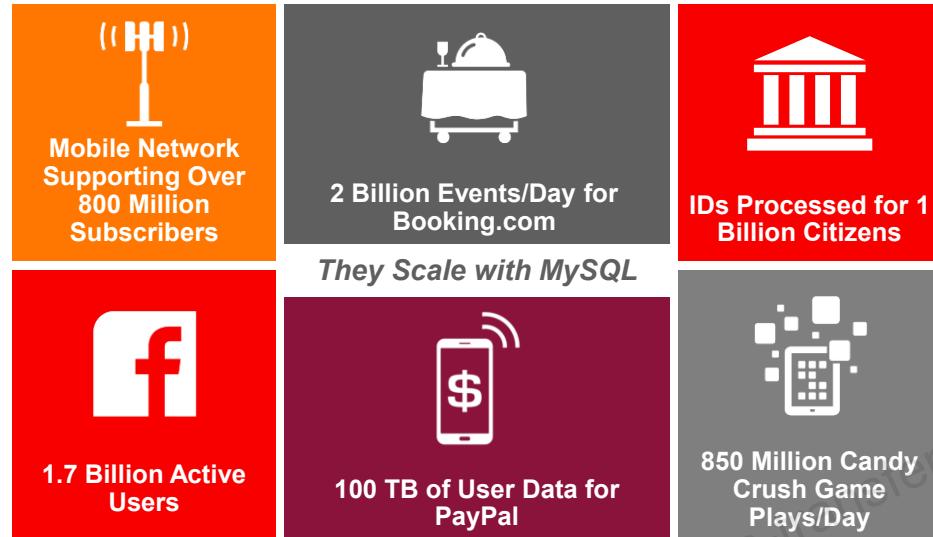


Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The slide lists several digital disruptors and companies that have redefined customer expectations. They rely on MySQL, and so do large enterprises driving digital transformation initiatives.

They choose MySQL because it is a modern database designed for web-based applications, which allows them to innovate quickly. But those are not the only reasons. As the world's most popular open source database with over 20 years of existence, MySQL is a proven, battle-tested, and mature technology.

High Scalability



ORACLE

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

- MySQL Cluster powers a mobile network in Asia counting over 800 million subscribers.
- A government, also in Asia, has processed over one billion IDs for its citizens.

MySQL Enterprise Edition

Advanced Features

- **Scalability**
 - MySQL Enterprise Thread Pool
 - Group Replication
 - InnoDB Cluster
- **High Availability**
 - MySQL Enterprise HA
- **Security**
 - Network Access Control
 - MySQL Enterprise Authentication
 - MySQL Enterprise Audit
 - MySQL Enterprise Encryption/Transparent Data Encryption (TDE)
 - MySQL Enterprise Firewall



Management Tools

- **Monitoring**
 - MySQL Enterprise Monitor
- **Backup**
 - MySQL Enterprise Backup
- **Development**
 - MySQL Connectors
- **Administration**
 - MySQL Workbench
 - MySQL Utilities
- **Migration**



Support

- **Technical and Consultative Support**
 - Oracle Premier Support for MySQL
- **Oracle Certifications**
 - Oracle Certified MySQL Database Administrator
 - Oracle Certified MySQL Developer



ORACLE®

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

Scalability

- MySQL Enterprise Thread Pool allows you to scale the performance of your application in the face of increasing user, query, and data loads.
- The MySQL Group Replication feature is a multi-master, update-anywhere replication plugin for MySQL.
- InnoDB Cluster is an integrated, native, full stack high availability solution for MySQL.

MySQL Enterprise High Availability

MySQL Enterprise High Availability enables you to meet the availability requirements of even the most demanding, mission-critical applications. MySQL Group Replication provides native high availability with built-in group membership management, data consistency guarantees, conflict detection and handling, node failure detection, and database failover-related operations, all without the need for manual intervention or custom tooling.

MySQL Security

- **Network Access Control:** Restrict connections to your MySQL instances.
- **MySQL Enterprise Authentication:** Authenticate MySQL users against your existing directory services and security rules.
- **MySQL Enterprise Audit:** Generate a complete audit trail to track MySQL access and usage.
- **MySQL Enterprise Encryption:** Protect sensitive data stored in MySQL databases, in backups, or during transfer.
- **MySQL Transparent Data Encryption (TDE):** Protect data at rest and securely manage your encryption keys.
- **MySQL Enterprise Firewall:** Ensure real-time protection against database-specific attacks.

Oracle Premier Support for MySQL

- Largest MySQL engineering and support organization
- Backed by the MySQL developers
- World-class support, in 29 languages
- Hot fixes and maintenance releases
- 24 x 7 x 365
- Unlimited incidents
- Consultative support
- Global scale and reach

Get immediate help for any MySQL issue, plus expert advice.



"The MySQL support service has been essential in helping us with troubleshooting and providing recommendations for the production cluster. Thanks."

– Carlos Morales (playfulplay.com)

ORACLE®

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

Why MySQL Enterprise Edition?

In addition to all the features you love...



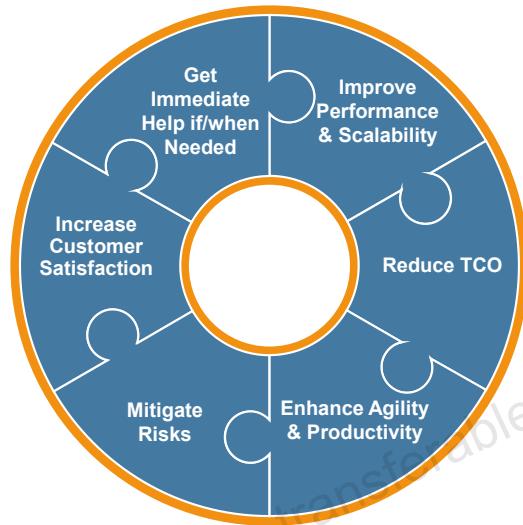
- Insure your deployment



- Get the best results



- Delight customers



ORACLE®

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

MySQL and Oracle Integration

MySQL integrates into the Oracle Environment

Oracle Linux
Oracle Solaris
Oracle OpenStack
Oracle Fusion Middleware
Oracle Enterprise Manager
Oracle Secure Backup
Oracle Clusterware
Oracle Audit Vault
and Database Firewall



ORACLE®

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

In addition to Oracle Enterprise Manager, MySQL is now integrated with nearly all relevant Oracle products.

Oracle wants to make it very easy for existing Oracle customers to integrate and manage MySQL in their current environment.

MySQL Cloud Service

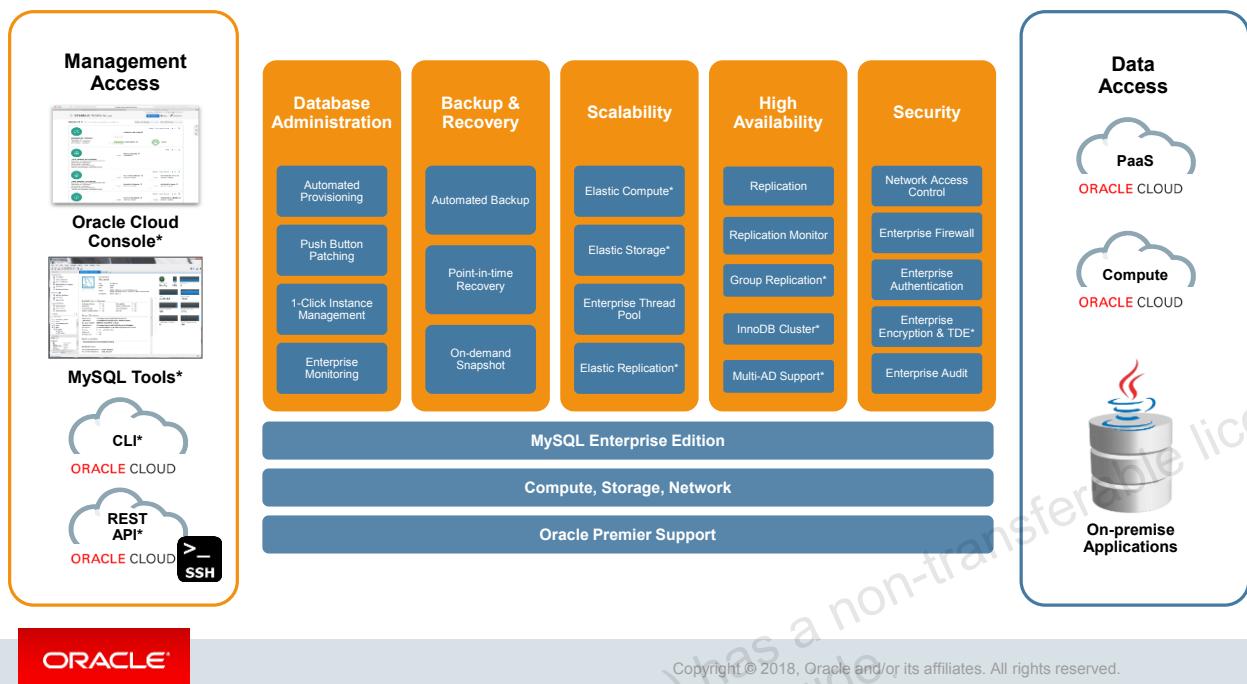
- Host your MySQL databases in the cloud
 - Rapid provisioning of MySQL server instances
 - Easy to test, apply, and roll back patches
- Scale quickly, cheaply, and efficiently
 - Elastic compute: Increase or decrease resources as required
 - Elastic storage: Expand storage capacity when your data grows
 - Elastic replication: Distribute workloads dynamically
- Improve security
 - Protect against attacks and misuse with advanced tools
 - Ensure regulatory compliance
- Reduce total cost of ownership
 - Cost of buying and maintaining on-premise database servers



ORACLE

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

MySQL Cloud Service: Product Overview



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

Management Access

- **Oracle Cloud Console:** Manage all your Oracle cloud instances via a web GUI.
- **Oracle Enterprise Manager:** Dashboard-style management of your entire Oracle environment.
- **MySQL Enterprise Monitor**
- **MySQL Workbench**
- **Host Access via SSH:** Gain administrative access to the underlying virtual machine.
- **REST API:** Programmatically manage your MySQL instances via a simple API.
- **CLI:** Manage your instances via the command line.

Scalability

Cloud resources are “elastic” and can grow or shrink automatically based on usage.

- **Elastic Replication:** Add or remove replicated MySQL instances to distribute workloads dynamically based on demands.
- **Elastic Compute:** Scale computing resources up or down as required.
- **Elastic Storage:** Increase the block storage for your MySQL instance when the amount of data grows

High Availability

- **Replication:** Enables you to set up flexible topologies for high availability
- **Replication Monitor:** Gives you visibility into the performance, availability, and health of all MySQL Masters and Slaves
- **Group Replication:** The MySQL Group Replication feature is a multi-master update anywhere replication plugin for MySQL with built-in conflict detection and resolution, automatic distributed recovery, and group membership.
- **InnoDB Cluster:** MySQL InnoDB Cluster tightly integrates across MySQL server, the new Group Replication plugin and MySQL Router providing a complete and easy-to-use solution for High Availability based on the proven and mature InnoDB transactional storage engine, all managed through a scriptable API in the MySQL Shell.
- **Multi-AD (Availability Domain) Support:** Enables you to replicate MySQL instances across different domains

Integration into the Oracle Cloud Environment

Oracle MySQL Cloud Service is integrated with Oracle Cloud Application Development solutions, such as Oracle Java Cloud Service and Oracle Application Container Cloud Service. This enables developers to quickly spin up an environment for rapid development and deployment. It is also integrated with Oracle Cloud Infrastructure, so that organizations can, for example, choose to back up data to Oracle Storage Cloud.

Use Cases

- Migrated on-premise databases
- “Born in the cloud” applications
- Hybrid database applications
 - Partly on-premise, partly in the cloud
 - Geographic dispersion
 - For example, a front end for data replicated from individual on-premises data centers
- Throwaway short-lived development and testing instances
- Testing new patches
 - Test, apply, and roll back patches easily.



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

Administering MySQL Cloud Service Instances

You work with MySQL in the cloud as you would an on-premise instance, using the same methods you will learn in this course.

Additionally, MySQL Cloud Services offers DBAs:

- Automated provisioning
 - Quickly create a MySQL Cloud Service instance that is preconfigured and optimized for performance
- Easy instance management
 - Using Oracle Cloud Console GUI
- “Push-button” Patching
 - Easily test, apply, and roll back patches
- Real-time monitoring
 - With MySQL Enterprise Monitor



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

MySQL Websites

- <http://www.mysql.com> includes:
 - Product information
 - Services (Training, Certification, Consulting, and Support)
 - White papers, webinars, and other resources
 - MySQL Enterprise Edition downloads (trial versions)
- <http://dev.mysql.com> includes:
 - Developer Zone (forums, articles, Planet MySQL, and more)
 - Documentation
 - Downloads
- <https://github.com/mysql>
 - Source code for MySQL Server and other MySQL products



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

Download MySQL Community Edition general availability (GA) and development releases from the <http://dev.mysql.com> website. This site also hosts the online documentation, which is an extremely valuable resource for both beginners and expert users of MySQL, and includes several example databases. Download trial versions of MySQL Enterprise Edition software, view detailed product information, and find out more about Oracle MySQL services at <http://www.mysql.com>.

Community Resources

- Mailing lists
- Forums
- Developer articles
- MySQL Newsletter (published monthly)
- Planet MySQL blogs
- Social media channels
 - Facebook, Twitter, and Google+
- Physical and virtual events, including:
 - Developer days
 - MySQL Tech Tours
 - Webinars
- Bug tracking
- Github repositories



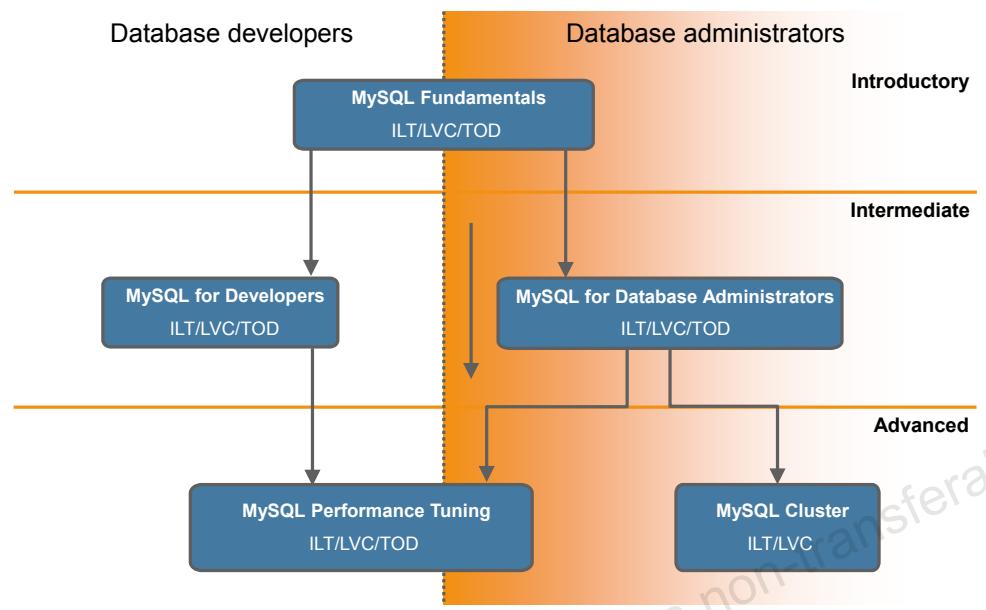
ORACLE

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

MySQL has a very large and active community. Engage with your peers, report bugs, share information, and discover what is happening in the world of MySQL by using the following resources:

- Mailing lists (<http://lists.mysql.com>)
- Forums (<http://forums.mysql.com>)
- Developer articles (<http://dev.mysql.com/tech-resources/articles>)
- MySQL Newsletter (<http://www.mysql.com/news-and-events/newsletter/>)
- Planet MySQL blogs (<http://planet.mysql.com>)
- Social media channels:
 - Facebook: <http://facebook.com/mysql>
 - Twitter: https://twitter.com/mysql_community and <http://twitter.com/MySQL>
 - Google+: <http://plus.google.com/+mysql>
- Physical and virtual events (<http://www.mysql.com/news-and-events>)
- Bug tracking (<http://bugs.mysql.com>)
- Github repositories

Oracle University: MySQL Training



ORACLE

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

Training Formats

- **Instructor-Led Training (ILT):** Delivered in a classroom with instructor and students present at the same location and time
- **Live Virtual Class (LVC):** Delivered by using video and audio through a web-based delivery system (WebEx) in which geographically distributed instructor and students participate, interact, and collaborate in a virtual class environment
- **Training On Demand (TOD):** On-demand training that takes traditional classroom training (complete with all classroom content including lectures, white boarding, and practice videos) and makes it available in a video-based, online format so that you can start your customized training at your convenience

For full details about MySQL training options, go to <http://education.oracle.com/mysql>.

MySQL Certification

The Oracle Certification Program validates your expertise in the following areas:

- Oracle Certified Professional: MySQL 5.6 Database Administrator
- Oracle Certified Professional: MySQL 5.6 Developer

Take the examination at a local Pearson VUE test center:

- At a time suitable to you
- In over 175 countries



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

For full details of MySQL and other Oracle Certification options, visit:
<http://education.oracle.com/certification>.

MySQL-Supported Operating Systems

MySQL:

- Provides control and flexibility for users
- Supports multiple commodity platforms, including:
 - Windows (x86, x86_64)
 - Linux (x86, x86_64)
 - Oracle Solaris (SPARC_64, x86_64, x86)
 - Mac OS X (x86, x86_64)
- Can be compiled to run on other platforms



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

For a full list of supported operating systems, see the website at:
<http://www.mysql.com/support/supportedplatforms/database.html>

Summary



In this lesson, you should have learned how to:

- Describe the course goals
- List MySQL products and services
- State the benefits of using MySQL Cloud Service
- Determine support for your operating system
- Access MySQL information and services from Oracle websites and community resources
- Find information about MySQL courses and certification options

ORACLE®

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

Practice

- 1-1: Course Environment Overview



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

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

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

2

Performance Tuning Concepts

ORACLE®



MySQL™

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

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

Topics

- Overview
- Terminology
- Benchmarking
- Troubleshooting performance issues
- Tuning steps
- Deploying and maintaining MySQL



ORACLE®

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

The slide lists the topics for this lesson.

Objectives



After completing this lesson, you should be able to:

- Describe general strategies for improving performance
- Identify useful metrics for evaluating performance
- Explain performance tuning terminology
- Design suitable benchmarks
- List the key steps to perform when troubleshooting performance issues
- Consider MySQL guidelines for deployment and maintenance

ORACLE®

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

Topics

- Overview
- Terminology
- Benchmarking
- Troubleshooting performance issues
- Tuning steps
- Deploying and maintaining MySQL



ORACLE®

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

This section describes what should be your goals when tuning performance and which areas of the system you can tune to achieve those goals.

Improving Performance

Focus on reducing response times:

- Optimize query performance.
- Reduce load on existing systems:
 - By increasing their capacity to do work
- Scale your systems to accommodate more requests:
 - **Scaling up:** Upgrading to better, faster, servers
 - **Scaling out:** Acquiring more servers



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

There is a common misconception that performance tuning is all about reducing CPU or memory utilization. This is not what you must focus on. CPU and memory are resources, and they are designed to be used. Instead, concentrate on improving the user experience by reducing response times. Reduce response times by optimizing your queries and by enabling your system to serve more requests.

Areas to Tune

- The MySQL server instance is the primary area that a DBA can tune:
 - Memory/threads/connections
 - Schema
 - Instance configuration
- Other areas:
 - Application tuning
 - SQL statement performance
 - Managing changes or increases in the size of the data
 - Operating system (OS) tuning
 - Input/output (I/O)
 - Swapping
 - Network issues
 - User limits (such as open files)



ORACLE®

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

The database administrator's main responsibility is the MySQL server, and many administrators are responsible for multiple server instances. Therefore, this course focuses primarily on tuning queries, configuration settings, and other MySQL server settings. However, note that the database is not the only factor that affects performance. The application itself, and the infrastructure that supports it, might cause significant bottlenecks which have nothing to do with the database. Therefore, you must involve other personnel in the tuning effort.

Topics

- Overview
- Terminology
- Benchmarking
- Troubleshooting performance issues
- Tuning steps
- Deploying and maintaining MySQL



ORACLE®

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

This section defines some of the common terminology used when discussing MySQL performance.

Performance Tuning Terminology

- Response time
 - Waits
 - Service time
- Throughput
- Scalability
- Queuing theory
- Benchmarking
 - Performance baseline



ORACLE®

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

Like any discipline, performance tuning has its own terminology, which is important to understand. The next few slides define the key terms.

Response Time

Response time is how long a user must wait to receive the results of his or her query.

Response time is the sum of:

- **Wait (or queue) time:** The amount of time during which a requested task is inactive and waiting to be processed. Common causes of waits are:
 - Waiting for I/O operations to complete
 - Waiting for a lock to be released
- **Service time:** The actual time taken to process the query. Factors affecting service time include:
 - Hardware/networking
 - Concurrency
 - Query efficiency



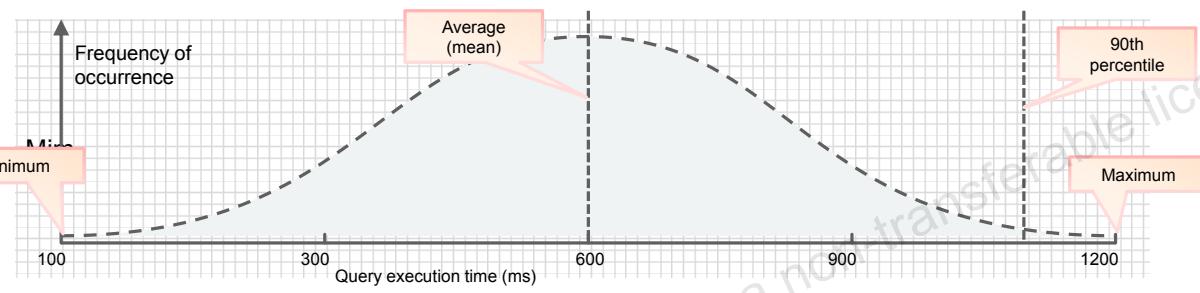
Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The most valuable metric for evaluating the performance of a MySQL server is response time. Because the job of the MySQL server is to evaluate queries, the response time is the time elapsed between when the user submits the query and the server returns the query results.

The response time consists of the time the user's request spends in the queue waiting to be processed (the wait, or queue, time) and the time it takes to actually process the query (the service time). Factors affecting the response time are not limited to MySQL and can include latency added by different parts of the system, such as hardware capability, the speed of DNS resolution, general networking problems, and other issues. MySQL-related factors include the amount of concurrency in the system, the efficiency of queries, and resource locking, but these are not the only causes.

Measuring Response Times

- Average/minimum/maximum values
 - Milliseconds, seconds, or minutes
 - Maximum values are rarely useful.
- 90th percentile:
 - If the 90th percentile response time for a query is 1.1 seconds, the query finishes in less than 1.1 seconds 90% of the time.



ORACLE®

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

The graph in the slide shows the frequency of queries completing in a given time and the values of each of the metrics described.

Average/Minimum/Maximum: You can base the metric for response time on the average, minimum and maximum response times (milliseconds/seconds/minutes) for a transaction to process. The maximum value is rarely useful, as it can vary significantly between executions.

90th Percentile: A more reliable metric for response time and latency is based on 90th percentile response time (milliseconds/seconds/minutes) for each group of transactions.

Throughput

Throughput is the number of tasks the system can perform in a given period of time. It is:

- An important measurement for interactive multiuser systems
- Usually measured in transactions per second
- Managed carefully to prevent *starvation*
 - The user is “starving” for data and the server cannot respond.
 - The user experiences long wait times.



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

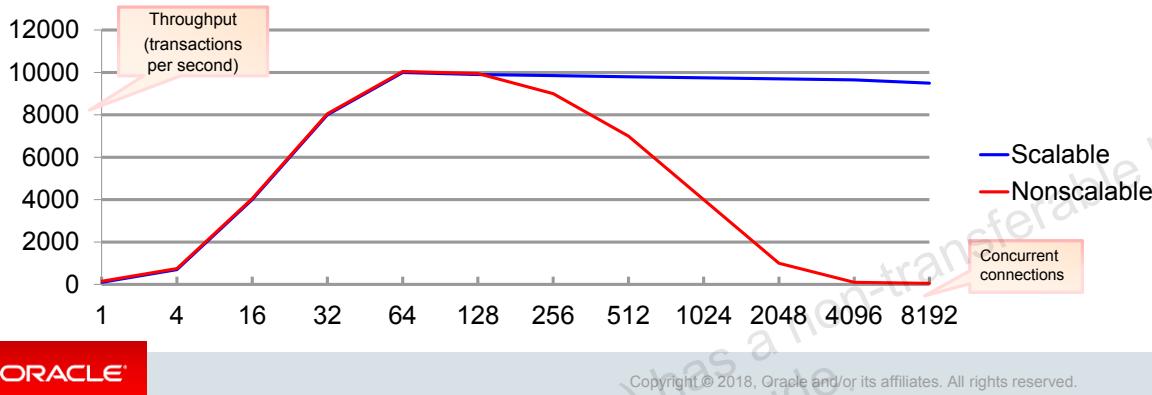
Throughput is a metric that you use to evaluate performance, but is not in itself an indicator of performance. For example, the throughput of a system with a large number of hardware threads might be very high compared to a typical system, but if the tasks being submitted on each of those threads take a long time to service due to a slow single-threaded processor, the performance of the system as a whole is poor.

Note that throughput tells you nothing about response time. A server might be able to process ten queries concurrently in ten seconds, but you do not know how long each of those queries took to execute.

Scalability

The *scalability* of a system is the relationship between resource utilization and throughput.

- As utilization increases, throughput degrades.
- In a scalable system, throughput grows as demand increases, and response times remain stable.



ORACLE

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

The scalability of a system is its ability to increase throughput as demand grows, while keeping response times within acceptable values.

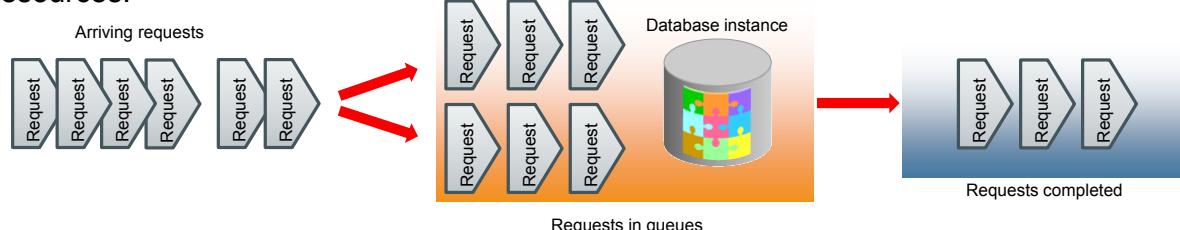
In the example in the slide, the graph for the nonscalable system peaks at 10,000 transactions per second with 64 concurrent connections to the database. Throughput then degrades rapidly and response times increase accordingly.

The scalable system plateaus at the same point, but degrades much more slowly. However, as the number of connections to the database increases beyond the values shown, throughput decreases and response times continue to rise. This is typical in cases of *scaling up*: increasing the capacity of an existing server.

In an ideal situation, the throughput graph neither drops nor becomes flat. Throughput continues to grow and response times remain stable. However, this type of scalability is usually achieved only by *scaling out*, that is, adding extra servers to cope with the increasing load.

Queuing Theory

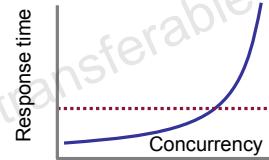
Queuing theory is a model for evaluating the efficiency of a system that consumes multiple resources.



- Applying this model to databases:
 - Use response times and throughput metrics to evaluate and improve performance.

"Hockey stick" phenomenon:

In systems where the queue is growing faster than the system is able to handle the transactions (saturation), the end user experiences an exponential delay.



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

Queuing theory states that when you increase the load on a system, there will always be a bottleneck. That bottleneck will reach high utilization as it nears its capacity, and large queues will form in front of it. The key to successful performance tuning is:

- Identifying where the queues are in a system
- Determining what the bottleneck is
- Deciding what to do about the above two items

When you look for a solution for the "Hockey Stick" phenomenon, consider:

- The delay directly related to the queue
- The service time associated with the processing of the transactions

If you improve one, you reduce the user response time. Improving both is the optimal solution.

Quiz



Which of the following can provide metrics suitable for measuring response time and latency?

- a. Average/minimum/maximum
- b. Hockey stick
- c. Queuing theory
- d. 90th percentile



ORACLE®

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

Answer: a, d

Topics

- Overview
- Terminology
- **Benchmarking**
- Troubleshooting performance issues
- Tuning steps
- Deploying and maintaining MySQL



ORACLE®

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

This section describes how to establish performance baselines by benchmarking your system.

Benchmarking

A benchmark is a test designed to compare the qualities or performance of different systems. Use benchmarks to:

- Quantify application performance and develop baselines
 - The baseline is a basic standard of values that serves as a comparison for changes on a system.
- Measure the effect of changes:
 - To determine if a change has a positive effect on performance, run a benchmark before and after the change.
- Evaluate system scaling:
 - Measure the effect on performance of adding systems to an existing architecture.
- Plan deployment:
 - Determine the performance of different deployment options.



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

Benchmarking enables you to test how a system responds when it is given work to do, and removes the guesswork from your performance tuning efforts. However, note that although benchmarking simulates a system, it is not the system itself. The workloads you use when benchmarking are very different from real-life deployments, which can be extremely variable. If you remember this fact, benchmarking is extremely useful, enabling you to measure current performance and then determine the performance impact of any changes. Such changes could be the addition of new hardware or opening up your application to more users.

Benchmarking Best Practices

- Replicate the real-world application as closely as possible.
 - A benchmark is a measure of a snapshot of your total system.
 - Consider using a log of transactions recorded from your live system in the simulated environment.
- Use a representative data population.
 - Ensure that the data population you study is representative of the total data population of your system.
- Use real database size.
 - Use a database size that is equivalent to the data in your system.
- Use real input values.
 - Use data that is equivalent to the real data in your system.



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

If you accept that a benchmark is only a simulation, then the closer that simulation resembles your real-life application, the more useful the results will be. Consider your data and how your users interact with it when designing a benchmark.

Do not benchmark scenarios that are unlikely to occur in the real world. The fact that one storage engine creates 10,000 tables faster than another might be interesting; however, it is unhelpful, because it is very unlikely that any application would ever need to do this during its normal day-to-day operation.

More Benchmarking Best Practices

- Use a similar number of connections.
 - Execute benchmarks with the same number of concurrent connections that your real system must handle.
- Use a similar “think time.”
 - **Think time:** The average time a user spends looking at a webpage before determining the next action
 - The time required to load the page is not considered in this step.
- Consider the effects of caching.
 - Clear caches where necessary to prevent repetitive data from producing inaccurate results.
 - Ensure that each benchmark starts from the same state.
- Use similar server settings, hardware, OS, network, and so on.



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

Benchmarking Bad Practices

- Testing against different amounts of data
 - Do not test with 1 GB of data if the production server holds 100 GB of data.
- Using a uniform distribution that is not natural in the real world.
 - Use data that contains a similar representation of the transactions that your production server faces.
- Testing in a single-user scenario
 - If your production server has multiple connections made to it, a benchmark is not accurate if it is set up as a single-user scenario.
- Minimizing (or not using) think times



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

This slide lists the approaches to avoid when you are creating benchmarks. The point about using a uniform distribution of data is an important one. For example, if you run an online bookshop, there are certainly many more orders for the “Harry Potter” series than the “Zulu Dictionary” and your environment must simulate this.

More Benchmarking Bad Practices

- Benchmarking on a single host
 - Connect to the MySQL server from numerous hosts.
- Constantly using cached query results
 - Simulate queries that are unique to avoid the query cache affecting results.
- Ignoring server warmup
 - Let your queries run on the benchmark server until the working set is cached, before you collect results.
- Using default MySQL server settings without understanding them
 - Make adjustments to the MySQL default server settings if necessary if they are more appropriate for your system.



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

The default MySQL server settings are good for many environments and reflect typical hardware at the time of release. However, you must understand what these settings mean so that you can tune them for your own system if you need to. This course discusses the most important settings and outlines the server defaults.

Note: The MySQL query cache is not covered in this course. It was deprecated in MySQL 5.7.20 and Oracle does not recommend using it.

Quiz



Which of the following is not a typical benchmark error?

- a. Connecting to the MySQL server from multiple hosts
- b. Minimizing (or not using) think times
- c. Ignoring server warmup
- d. Executing the same query repeatedly



ORACLE®

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

Answer: a

Topics

- Goal
- Terminology
- Benchmarking
- Troubleshooting performance issues
- Tuning steps
- Deploying and maintaining MySQL



ORACLE®

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

This section describes the factors you must consider when troubleshooting performance issues.

Before You Start: Establish a Performance Baseline

Establish a baseline while the system is running normally.

- Define what is normal:
 - When you encounter a problem, run a new benchmark test and compare its values against the baseline.
- Record operating system metrics:
 - OS memory and CPU usage
 - top, iostat, vmstat
- Record MySQL status:
 - SHOW PROCESSLIST to see running processes
 - mysqladmin extended-status to see status variables
 - Use -i<seconds> --relative to record value deltas.
- Record application use-case response times:
 - Log in, search, create, read, update, delete



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

To establish a baseline, record how the system performs normally. When you record this baseline behavior, you have something to compare against the changed behavior that results from any problem that later arises.

The baseline consists of key metrics at different levels:

- The application's response time for all important use cases
- MySQL's settings, typical process load, and status (static and dynamic)
- Operating system load (CPU, memory, processes)
- I/O performance (disk subsystem, network throughput)

Use monitoring tools such as MySQL Enterprise Monitor to record and track the performance of the system over time. This often enables you to identify developing problems before they become significant. For example, an unexpected increase in the number of rows that the application accesses results in an extra load on the server but not affect the users' experience of the application. However, it might warn of a serious problem that, if left unresolved, results in loss of service.

Re-establish the baseline at regular intervals. A gradual increase in data size can degrade performance slowly until it reaches some tipping point, and a recent baseline can prove more useful during troubleshooting than one generated when the workload was different.

Key Steps in Troubleshooting Performance Issues

- Prepare to troubleshoot before the problem occurs.
 - Measure the system's baseline performance.
 - Define what is a normal problem-free state.
- Diagnose the problem when it occurs.
 - Measure the system's performance.
 - Compare the baseline with these new measurements.
 - Identify the cause of the changes.
- Resolve the problem by changing part of the system.
 - Database schema or configuration
 - Application code
 - Operating system configuration
 - Network or other environmental configuration



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

To troubleshoot problems with any system, you must be able to identify the unwanted changes in the behavior of that system and diagnose the cause of those changes. You can identify those changes only by comparing the system's changed performance against a previously established baseline. This means that you need to prepare for troubleshooting a problem long before that problem occurs.

Establishing the Nature of the Problem

To identify the problem, answer the following questions:

- Has the application, database, or server configuration changed recently?
- Has the problem resolved itself since it first occurred?
 - Was there a sudden growth of application activity due to a batch operation or a spike in web traffic?
 - Were system resources taken up by operations that are external to the database?
 - Network traffic causing router problems
 - Filesystem backups causing I/O problems
- Does the problem occur regularly at the same time?
- What are the precise circumstances and symptoms of the problem?



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

A frequent first sign of a performance problem is a user-visible application error. To find the cause of the error, you must track through the components from the application to the database to determine where the problem lies.

Similarly, if you experience large increases in data volume or traffic, the change in usage patterns can negatively affect the performance of the database.

Performance problems, such as long wait times, might appear to resolve themselves after a period of time. Such problems might be caused by sudden increases in activity from batch operations, highly publicised campaigns, or pages that go viral and encounter a much larger amount of traffic than usual.

Problems that appear to manifest at the database level can also be caused by environmental factors in the operating system, server hardware, or network. If a network router or switch crashes or becomes overloaded due to a large amount of traffic, this interrupts communication between the application and the database. On servers that use a large amount of RAM to perform disk caching, you might notice occasional drops in I/O performance when the server flushes the cache to disk. Similarly, a large amount of hard disk traffic (such as that caused by a filesystem backup) can interrupt I/O operations on the database server.

Starting to Troubleshoot

- Compare application, MySQL, and OS settings and other metrics with the baseline.
 - Re-execute the tests used to create the baseline.
- Localize the problem at a functional level.
 - Identify what is manifesting the problem.
 - Specific application use cases
 - Specific clients
- Create a clear problem statement.
 - Error messages
 - Specific behavioral changes
 - Intermittent or continuous symptoms
 - Reproducible symptoms



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

When a problem occurs, you must establish the behavior of the system at that time. The process to do this is similar to that used when you create a baseline.

In some situations you might need to troubleshoot a problem when you do not have a baseline. In such situations, you still need to measure the performance of the system when the problem occurs. With some experience, you can recognize metrics that are abnormal even without comparing them with baseline values. For example, you might know that the query cache should be enabled only in very rare cases. Therefore, if you see that it is enabled, you have a point at which you can start investigating further. Similarly, if you know that the InnoDB buffer pool hit rate is low, you know that InnoDB is not performing at optimal capacity and you can continue your investigations by examining the balance of queries and the quantity of hot data.

You must localize the problem at a functional level. For example, if logging in is much slower than before, but the rest of the application is working correctly, that problem is likely to have a very different root cause than the search function being slow. Similarly, if every application function slows down every afternoon at 2:30 PM, that problem is likely to have yet another cause.

Identifying Possible Causes of Performance Issues

Causes of performance issues:

- Application design issues
 - Reading large text or XML files
 - Calls to remote servers
- Concurrent activity on the hardware
 - Network throttling
 - Contention on shared storage or virtualized servers
 - Degraded hardware performance
 - Example: RAID disk failure, CPU temperature throttling
- Database design and configuration
 - Lack of appropriate indexes
 - Inefficient schema design
 - Inappropriate values for tunable settings



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

Many performance problems are rooted in the application code rather than the database server. Examples include:

- Reading large data files
- Calling remote web services
- Inefficient sorting or searching algorithms against large data sets

Problems caused by concurrent activity on the hardware are often intermittent although they might occur at similar times daily or weekly. Such problems might be difficult to diagnose because their causes are often independent of the application that manifests the problem. Degraded hardware performance can result in overall system issues but might be difficult to detect without looking in system logs. For example, a degraded RAID set continues to perform I/O and serve requests with no application-visible symptoms other than lower I/O performance.

If the problem has the database at its root, you can often find a solution without changing any application code by improving the indexes on tables used in application queries. If you change the schema to improve performance (for example, by changing column data types or by normalizing or denormalizing some tables) you might need to change the application code as a result.

In some rare cases, you might also need to make changes to system variables. However, it is usually much easier to achieve orders-of-magnitude improvements in performance by optimizing other elements of the database configuration, such as table indexes and data types. Focus on changes to system variables only when you have performed these other optimizations.

Application Profiling

- Records the time at which specific events occur
 - Function entry/exit
 - Calls to external systems such as databases
- Shows how long each part of an operation takes
- Is available as a built-in part of many development environments and interpreters, or as plugins
 - Alternatively, you can instrument your code to profile significant points.
- Shows if it is useful to troubleshoot the performance of the database
 - Remember *Amdahl's Law*.



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

Application profiling enables you to find bottlenecks in the entire system by examining the code's execution and seeing how long each step takes.

When you use application profiling, you can see the duration of key parts of each function or use case. This enables you to see if the application is experiencing most of its delay on calls to the database, in setting up connections, or if the delay is due to some other application operation such as internal memory allocation, a sort function, or calls to some other external process.

You can also see if the delays are distributed between multiple operations, and this can inform your decision about if and when to apply your troubleshooting efforts. For example, you might find that a database query costs 10% of a particularly complex use case, with other functions making up the remaining 90%. Amdahl's law states that no matter how much you optimize the database query, the maximum performance improvement is 10%.

Localizing Database Problems

Start your investigations by focusing on the most common problems and their typical causes.

- A small number of queries:
 - Query plans
 - Locks caused by other queries on the same table
- Many (or all) queries:
 - General server activity, contention, and mutexes
- Many queries on a single table:
 - Schema design, table indexes, and statement structure
- Intermittent or consistent performance problems:
 - Consider the concurrent load when problems occur
 - Examine transactions that occur at that time



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

When a problem affects many queries that interact with different tables, look at the configuration of the database server, its internal activity, or its interaction with its environment. There might be contention on a low-level MySQL resource, or there could be a problem with the volume of I/O generated by MySQL.

When a problem affects a few queries, look at the indexes in the table, the design of those statements, or the design of your schema.

There might be multiple causes to a general server performance degradation. For example, if multiple queries on different tables are slower than usual, you might first consider that it is a general server problem. However, the problem could come from two or more problems with poorly performing queries. Taking the time to identify the precise cause of each problem enables you to avoid making changes based on incorrect assumptions.

When you have several application functions that interact with one large table, a problem with that table's schema might have a disproportionate effect on the overall performance of the application.

An intermittent problem that you have localized to the database (and that is not, for example, a hardware or network contention problem) might be caused by a change in the application's usage patterns or load, and might manifest as locks or some other form of data or server contention.

Topics

- Goal
- Terminology
- Benchmarking
- Troubleshooting performance issues
- **Tuning steps**
- Deploying and maintaining MySQL



ORACLE®

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

This section describes the steps you should perform when tuning your system.

Tuning Steps

- Check your system.
 - Check OS and general machine health.
- Tune the areas that have the most potential for gain:
 - Longest waits
 - Longest service times
 - Most frequently executed queries
- Tune to a goal.
 - Stop tuning when the goal is met.
 - Tuning goals should be specific, measurable, and achievable.



ORACLE®

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

When deciding which areas to tune, consider “Amdahl’s Law” which, loosely paraphrased, states that a task that takes 5% of your system resources can only be optimized by a maximum of 5%. Monitor your applications by using the techniques described in this course and identify where the “big wins” are. Concentrate your tuning efforts on those.

Do not assume that only the slowest queries require optimization. You can improve the overall performance of the system by focusing on the most frequently executed queries, even if such queries already execute relatively quickly. For example:

- You optimize a query that takes two seconds so that it now takes 800 ms, but this query executes on average once per minute.
- You optimize a query that takes 20 ms so that it now takes 15 ms, but this query executes several thousand times per minute.

The second optimization will result in the best performance improvement overall.

General Tuning Procedure

1. Define the problem and state the goal.
2. Collect current performance statistics.
3. Consider some common performance errors.
4. Build a trial solution.
5. Implement and measure the change.
6. Evaluate the solution.
 - If the solution meets the goal, define the new baseline.
 - If the solution does not meet your goal, consider other common performance errors and then start over.



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

Be methodical when tuning. Start with a clear goal, proceed to a hypothesis, and then test that hypothesis by using a good simulation of the environment. Repeat the process with different hypotheses until your goal is met. This approach ensures that you truly understand the problem and why your fix solves it.

Topics

- Goal
- Terminology
- Benchmarking
- Troubleshooting performance issues
- Tuning steps
- Deploying and maintaining MySQL



ORACLE®

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

This section describes the best practices for deploying and maintaining MySQL.

Deploying MySQL

- Use automation whenever possible.
- Use different servers for different systems.
 - Database and web servers have different configuration, quality, and scaling requirements.
- Deploy MySQL on an internal network.
 - A MySQL server must not be directly accessible from the Internet.
 - MySQL servers that are accessible via a LAN are acceptable if the LAN is separated from the Internet by a firewall (preferably a hardware firewall).
 - Use SSH tunneling to give access to a MySQL server remotely.



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

RAM Requirements

- The InnoDB buffer pool
 - Must be large enough to hold the active data
- Connections to the MySQL database
- MySQL caches
 - Query, table, connections, stored programs, and more
- Extra RAM needed for:
 - Filesystem cache
 - Monitoring
 - RAM disk/tmpfs (Linux only)
- Error-correcting code (ECC) RAM preferred



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

One of the biggest bottlenecks to server performance is not having enough RAM to support your active data and connections. This slide outlines the areas in a typical MySQL application that require sufficient RAM to perform well. Error-correcting code (ECC) memory is optimal, as it protects against common forms of data corruption.

Note: The InnoDB buffer pool is covered in the lesson titled “Tuning InnoDB.” Caching strategies are discussed throughout the course.

Maintaining MySQL

- Design and implement a backup schedule.
 - Check whether your backups work.
- Use binary logging to enable point-in-time recovery.
- Enable the slow query log to identify slow queries.
- Monitor the entire infrastructure (systems and network)
 - Performance problems are not always related to the MySQL server.
- Upgrade regularly
 - Security improvements
 - Bug fixes
 - Performance enhancements
- Perform regular health checks
 - MySQL support can help.



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

Note

- For information about upgrading MySQL, see the *MySQL Reference Manual* at:
<http://dev.mysql.com/doc/mysql/en/installing.html>
- Consider a “MySQL Health Check” from an experienced MySQL support professional:
<http://www.mysql.com/news-and-events/health-check>

Summary



In this lesson, you should have learned how to:

- Describe general strategies for improving performance
- Identify useful metrics for evaluating performance
- Explain performance tuning terminology
- Design suitable benchmarks
- List the key steps to perform when troubleshooting performance issues
- Consider MySQL guidelines for deployment and maintenance

ORACLE®

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

Practice

- 2-1: Quiz – Performance Tuning Concepts



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

3

Performance Tuning Tools

ORACLE®



MySQL™

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

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

Topics

- MySQL monitoring tools
- Oracle DB monitoring tools
- Community monitoring tools
- Linux tools
- Benchmarking tools



ORACLE®

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

Objectives



After completing this lesson, you should be able to:

- Monitor server status and performance by using MySQL tools and databases
- Describe how to use Oracle Database tools to manage and monitor MySQL
- Monitor system performance by using Linux commands
- Perform benchmarks using MySQL and third-party tools

ORACLE®

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

Topics

- MySQL monitoring tools
 - Oracle DB monitoring tools
 - Community monitoring tools
 - Linux tools
 - Benchmarking tools



ORACLE®

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

MySQL Monitoring Tools

- mysql client commands:
 - SHOW [SESSION | GLOBAL] STATUS
 - SHOW ENGINE INNODB STATUS
 - SHOW PROCESSLIST
- The mysqladmin client program
- MySQL system databases:
 - Information Schema
 - Performance Schema
- MySQL GUI tools:
 - MySQL Workbench
 - MySQL Enterprise Monitor



ORACLE®

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

Oracle provides a range of MySQL monitoring tools. The MySQL Enterprise Monitor tool monitors MySQL instances continuously, offers extensive configuration options, and alerts the DBA if there are any issues.

The mysql client and mysqladmin program enable a DBA to display specified system status metrics at a particular moment in time. To monitor system activity over a period, you can use these tools in scripts.

In many situations, it is more useful to investigate performance issues from a number of different angles. The Performance Schema and, to a lesser extent, the Information Schema, enable DBAs to execute queries to retrieve performance metrics from many different parts of the system simultaneously. MySQL Workbench, in addition to being a very useful tool for writing and optimizing queries, assists in the configuration of the Performance Schema and can generate reports on its status.

Note: Performance Schema is covered separately in the lesson titled “Performance Schema.”

SHOW [SESSION | GLOBAL] STATUS

- SHOW [SESSION | GLOBAL] STATUS:
 - Lists server status variables in key/value pairs
 - Filter using LIKE and WHERE
 - Retrieves the values of the status variables from the Performance Schema
- SHOW STATUS or SHOW SESSION STATUS displays only status values for the current connection.
- SHOW GLOBAL STATUS displays status values for the instance as a whole.
- Values are either counters or metrics.
 - Counters increment when the server completes an action.
 - For example, `Select_scan` increments for every full table scan.
 - Counters are reset when their maximum value is reached.
 - Metrics can increase and decrease.
 - For example, `Threads_connected` shows the number of open connections.
- Reset session status variables with FLUSH STATUS.



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

Many server status variables store two values: one for the current session and another for all sessions. Others store only global status values. By default, SHOW STATUS shows only session status variables. Use SHOW GLOBAL STATUS to see the status values for the MySQL server instance as a whole.

Counters are stored as unsigned integers, occupying 8 bytes on 64-bit build of MySQL, but only 4 bytes on 32-bit builds. When the counter reaches the maximum value, it resets to zero, which can be confusing when you are monitoring a server that has been running continuously for some time.

Filtering SHOW STATUS Output

- Handler operations:
 - SHOW GLOBAL STATUS LIKE 'Handler_%';
- Temporary tables and files:
 - SHOW GLOBAL STATUS LIKE 'Created_tmp%';
- Command counters:
 - SHOW GLOBAL STATUS LIKE 'Com_%';
- Thread counters:
 - SHOW GLOBAL STATUS LIKE 'Threads_%';
- Other metrics include:
 - Uptime: The server's uptime, in seconds
 - Binlog_*: Binary log status
 - Connections: The number of connection attempts
 - Questions: The number of queries sent to the server



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

There are many server status variables, and this slide shows only a few of them. Related variables share a common prefix, which makes them easier to query. For example, Com_select, Com_insert, and Com_stmt_prepare report how many times each of these commands executed. If you combine status values with the Uptime statistic, you can calculate values such as the average queries per day or the connections per hour.

The SHOW STATUS command retrieves the values of these variable from the Performance Schema, which you can query directly.

Note: Performance Schema is discussed in the lesson titled “Performance Schema.”

Status Variables: Handler Operations

These status variables count the operations performed by the storage engine, providing information about query efficiency and other details related to storage engine usage. They include:

- Handler_read_first:
 - It shows the number of times the first entry was read from an index.
 - If this value is high, it suggests that the server is doing a lot of full index scans.
- Handler_read_key:
 - It shows the number of requests to read a row based on a key.
 - If this value is high, it is a good indication that the tables are properly indexed.
- Handler_read_rnd_next:
 - It shows the number of requests to read the next row in the data file.
 - This value is high if the queries require a lot of table scans.
 - Generally, this suggests that the tables are not properly indexed or that the queries are not taking advantage of the existing indexes.



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

Status Variables: Handler Operations

- Handler_read_rnd: The number of requests to read a row based on a fixed position. This value is high if you are doing a lot of queries that require sorting of the result. You probably have a lot of queries that require MySQL to scan entire tables or you have joins that do not use keys properly.
- Handler_read_next: The number of requests to read the next row in key order. This value is incremented if you are querying an index column with a range constraint or if you are performing an index scan.
- Handler_read_prev: The number of requests to read the previous row in key order. This read method is mainly used to optimize ORDER BY ... DESC.



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

Status Variables: Temporary Tables and Files

These status variables show you how many times MySQL has created temporary tables and files. These are expensive operations and can help you diagnose poorly-performing queries. They include:

- `Created_tmp_disk_tables`:
 - If a temporary table becomes too large, MySQL automatically writes it to disk.
 - The on-disk version of the table is considerably slower than the one in memory.
 - If `Created_tmp_disk_tables` is large, you might want to increase the `tmp_table_size` or `max_heap_table_size` value; or, better still, optimize your query.
- `Created_tmp_tables`:
 - It shows the number of in-memory temporary tables created by the server while executing statements.
 - Note that each invocation of `SHOW STATUS` uses an in-memory temporary table.
- `Created_tmp_files`:
 - It shows the number of temporary files created by the server.



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

These, and many other useful status variables are covered in later lessons.

Note: For a full list of MySQL server status variables, see:
<http://dev.mysql.com/doc/mysql/en/server-status-variables.html>

SHOW STATUS: Examples

```
SELECT REPEAT('a',1) UNION SELECT REPEAT('b',10);
```

```
mysql> SHOW STATUS LIKE 'Created_tmp%';
+-----+-----+
| Variable_name      | Value |
+-----+-----+
...
| Created_tmp_tables | 2      |
+-----+-----+
3 rows in set (#.## sec)
```

```
SELECT Name, Population FROM Country WHERE code IN
(SELECT countrycode FROM CountryLanguage WHERE Percentage > 50);
```

```
mysql> SHOW GLOBAL STATUS LIKE 'Handler_read%';
+-----+-----+
| Variable_name      | Value |
+-----+-----+
| Handler_read_first | 5      |
...
| Handler_read_rnd   | 0      |
| Handler_read_rnd_next | 1256 |
+-----+-----+
7 rows in set (#.## sec)
```

ORACLE

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

The status variables provide only a count of the operations. They do not tell you how long those operations took to perform. Even so, they can provide some insight into performance problems.

The examples in the slide show the result of executing queries and then running `SHOW STATUS` on selected status variables.

The first example uses a `UNION` statement. `UNION` statements create temporary tables and, therefore, increment the `Created_tmp_tables` status variable. Queries that create temporary tables, especially tables that are too large to be held in memory and must be created on disk (`Created_tmp_disk_tables`), have a negative effect on performance.

The second example executes a query that requires many table scans (`Handler_read_rnd_next = 1256`). You can optimize this query by making better use of indexes.

The `SHOW STATUS` command creates its own temporary table, which increments the global `Created_tmp_tables` status variable, as shown in the following example:

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

```
mysql> SHOW GLOBAL STATUS LIKE 'created_tmp_tables';
+-----+-----+
| Variable_name      | Value |
+-----+-----+
| Created_tmp_tables | 515   |
+-----+-----+
1 row in set (#.# sec)
```

```
mysql> SHOW STATUS LIKE 'created_tmp_tables';
+-----+-----+
| Variable_name      | Value |
+-----+-----+
| Created_tmp_tables | 0     |
+-----+-----+
1 row in set (#.# sec)
```

```
mysql> SHOW GLOBAL STATUS LIKE 'created_tmp_tables';
+-----+-----+
| Variable_name      | Value |
+-----+-----+
| Created_tmp_tables | 517   |
+-----+-----+
1 row in set (#.# sec)
```

Note: If you want to check the status values affected by the statement you have just executed, then you should use session status instead of global status, because global status is affected by statements executed in other sessions. The `SHOW STATUS` command shows session status. Add the `GLOBAL` modifier to retrieve global status variable values.

SHOW ENGINE INNODB STATUS

The `SHOW ENGINE INNODB STATUS` command reports activity in InnoDB. Its output contains a lot of detailed information about the storage engine, including:

- Background thread activity
- Semaphores
- Foreign key errors
- Deadlocks
- Transactions
- File I/O
- Insert buffer and adaptive hash index
- Log file usage
- Buffer pool and memory
- Row operations



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

The `SHOW ENGINE INNODB STATUS` command includes a lot of information and can be very useful when debugging problems related to InnoDB. To make sense of the results, you must have a good understanding of InnoDB.

Statistics are calculated using data from either the time of the last execution of `SHOW ENGINE INNODB STATUS` or the last system reset. The length of time used for calculations is displayed in the header information.

Note: This course covers key InnoDB metrics in the lesson titled “Tuning InnoDB.”

SHOW PROCESSLIST

- The `SHOW PROCESSLIST` command:
 - Is used to display information about currently running queries, including:
 - How long the query has been running
 - The user who executed the query
 - The host name or IP address where the query originated
 - The current state of the query
 - Helps you to determine if:
 - A query is taking too long to complete
 - You have too many open queries
 - A transaction is waiting for a lock on a table, or holds a lock on the table
- The output of `SHOW PROCESSLIST` shows only the first 100 characters of the query text by default.
 - Use `SHOW FULL PROCESSLIST` to display the full query.



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

`SHOW PROCESSLIST` is similar to the Linux `ps` command, or Windows Task Manager. It can help to diagnose problem query activity elsewhere on your system that is affecting the query you are interested in.

Example 1

A table hangs on a single row `UPDATE`, ultimately timing out. `SHOW PROCESSLIST` shows that another transaction holds a lock on the table, preventing the `UPDATE`. This information did not appear in `SHOW ENGINE INNODB STATUS`, because the table lock is handled outside of the InnoDB storage engine (although `SHOW ENGINE INNODB STATUS` still reports sleeping transactions).

Example 2

MySQL hangs on a single row `INSERT`. `SHOW ENGINE INNODB STATUS` shows nothing unusual. `SHOW PROCESSLIST` shows a very poorly performing `SELECT` query that does not lock any tables, but slows everything else down, including `INSERT`.

Note: Executing `SHOW PROCESSLIST` locks mutexes and can stall application connections. To avoid such locks, query the `performance_schema.processlist` table, which is covered in the lesson titled “Performance Schema.”

Quiz



Which of the following is a potential problem when using the SHOW PROCESSLIST command?

- a. It cannot tell you which user executed a query.
- b. It can affect the server's ability to accept connections.
- c. It cannot show you the full text of long queries.
- d. It locks tables, affecting query performance.



ORACLE

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

Answer: b, c

Note: c is also true in this situation, because you can only see the full query text when executing SHOW FULL PROCESSLIST. If you do not use the FULL keyword, only the first 100 characters of each statement are shown in the Info field.

mysqladmin

You can use the `mysqladmin` command-line program with the following options to monitor the MySQL server:

- Display a short status message:

```
$ mysqladmin status
Uptime: 240236 Threads: 5 Questions: 1683400 Slow queries: 3
Opens: 3440 Flush tables: 1 Open tables: 140
Queries per second avg: 7.007
```

- Display server status variables and their values:

```
$ mysqladmin extended-status
- Equivalent to SHOW GLOBAL STATUS
- Use the flush-status option to clear status values.
```

- List active server threads:

```
$ mysqladmin processlist --verbose
- Equivalent to SHOW FULL PROCESSLIST
```

ORACLE

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

You can combine the `mysqladmin` client program and shell options to generate useful outputs for monitoring.

For example, the following command displays the difference between the current and previous values of all server status variables every 100 seconds:

```
$ mysqladmin extended --relative -i100
```

MySQL Utilities

- The MySQL Utilities collection of python scripts helps to maintain and administer MySQL servers:
 - Encapsulate lower-level commands
 - Cover common DBA use cases
- MySQL Utilities are available as stand-alone programs, or from within MySQL Workbench.
- The following utilities are useful for performance tuning:
 - `mysqlauditgrep`: See who connected when, and which queries did they execute
 - `mysqldiskusage`: Display disk usage for databases, log files, and InnoDB tablespaces
 - `mysqlindexcheck`: Identify duplicate and redundant indexes



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

`mysqlauditgrep`

The `mysqlauditgrep` utility enables you to query the audit log that is generated when the MySQL Enterprise Audit plugin is enabled. Using `mysqlauditgrep`, you can search and filter the audit log records by: users (`--users`), date and time ranges (`--start-date` and `--end-date`), SQL query types (`--query-type`), logged event and record types (`--event-type`), status (`--status`), and matching patterns (`--pattern`). You can combine and use any of these search options to create useful reports.

`mysqldiskusage`

The `mysqldiskusage` utility displays the amount of disk space used by databases. You can also display disk usage for the binary log, show query log, error log, general query log, relay log, and InnoDB tablespaces by specifying the `--all` option. The default is to show only database disk usage.

`mysqlindexcheck`

The `mysqlindexcheck` utility reads index details for one or more tables in a database and reports duplicate or redundant indexes. You can also identify the best (`--best`) or worst (`--worst`) primary key indexes for a table.

Note: For detailed usage instructions, see: <http://dev.mysql.com/doc/mysql-utilities/1.6/en/utils-manuals.html>.

Information Schema

- Is an online data dictionary implemented as the INFORMATION_SCHEMA database
- Holds information in virtual tables
- Can be queried using SQL
 - Use prepared statements, stored procedures, and so on.
 - Control how you use or display the information.
- Contains detailed information about InnoDB:
 - Includes buffer pool, transactions, and locking details
 - Supports enabling/disabling each metric individually



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

With simple queries of the Information Schema, you can check the overall health of the system. With more detailed queries, you can diagnose issues such as performance bottlenecks and resource shortages.

Use the INNODB_SYS_* tables in the INFORMATION_SCHEMA database to access information about schema objects that InnoDB manages.

Use the Information Schema InnoDB tables to manage:

- **InnoDB table compression:** Requires balancing I/O reduction, CPU usage, buffer pool management, and how much compression is possible for your data
- **Transactions and locks:** Features that balance high performance for a single operation against the ability to run multiple operations concurrently. A lock is the low-level mechanism that controls access to a resource, such as a row or a table.

Note: Although the Information Schema can help diagnose poor performance, gathering too many InnoDB metrics can degrade overall system performance. This course covers InnoDB metrics in the lesson titled “Tuning InnoDB.”

Performance Schema

- Comprises a set of in-memory tables that MySQL uses to track performance metrics
 - Implemented as the PERFORMANCE_SCHEMA storage engine
 - Operates on tables in the performance_schema database
- Helps provide insight into database activity. For example:
 - Which queries are running
 - I/O wait statistics
 - Historical performance data
- Is complemented by the sys schema
 - Consists of views and stored routines that make it easier to query the Performance Schema

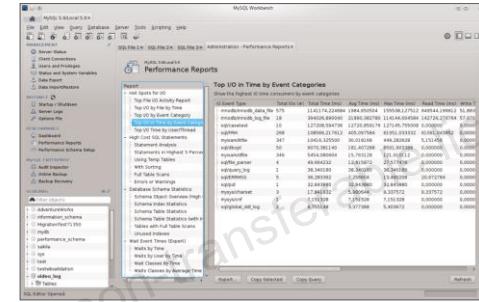


Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Note: Detailed information about the Performance Schema and sys is provided in the lesson titled “Performance Schema,” and examples of Performance Schema queries are provided throughout the course.

MySQL Workbench

- Is a visual tool for DBAs, developers, and database architects
- Comprises data modeling, SQL development, and comprehensive administration tools for server configuration, user administration, backup, and more
- Provides a graphical front-end for configuring and fine-tuning the Performance Schema
- Includes *Performance Reports*, which help troubleshoot performance issues:
 - I/O hotspots
 - High-cost SQL statements
 - Wait statistics
 - InnoDB storage engine metrics
- Displays query execution steps visually, to help identify poorly performing queries



ORACLE®

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

Note: For more information about MySQL Workbench, see:
<https://www.mysql.com/products/workbench/>

MySQL Enterprise Monitor: Overview

- Provides real-time visibility into the performance and availability of all your MySQL databases
- Enables real-time, continuous performance monitoring
 - Generates alerts when performance differs significantly from configured values
 - Best practice advisors recommend changes that improve performance and security.
- Manages multiple MySQL servers using a visual interface
 - Remotely monitors cloud servers without installing agents
- Enables visual query analysis
 - Monitors query performance in real time
 - Identifies and fixes problem queries
- Monitors key InnoDB metrics that affect performance



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

Visual Query Analysis

MySQL Enterprise Monitor data is gathered directly from `performance_schema` without requiring any additional software or configuration, so that you can monitor real-time query performance, check execution statistics, and identify poorly-performing SQL. Correlated graphs enable you to compare execution parameters, such as the server load, thread statistics, or RAM usage against the queries that were executing at that time. Highlight a time period on a graph to find the most expensive queries and identify potential causes for the wider performance issue.

InnoDB Monitoring

Monitor key InnoDB metrics that impact MySQL performance. Receive alerts on inefficient index usage, locking issues, and InnoDB buffer pool usage and get hints and tips on how to improve your InnoDB configuration based on the current performance and the analyzed trends.

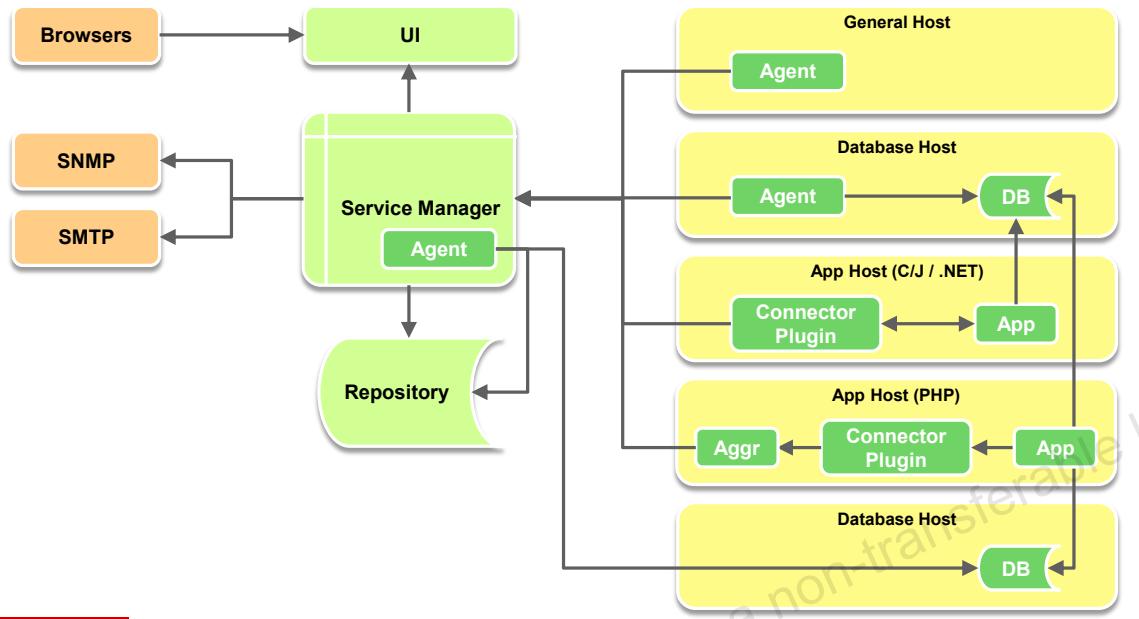
Operating System Monitoring

Visually monitor operating system-level performance metrics, including load average, CPU usage, RAM usage, swap usage, filesystem usage, and disk I/O, in real time.

MySQL Enterprise Monitor also enables you to monitor the performance of:

- MySQL Cluster metrics
- Replication, such as slave lag
- Backups, including lock duration

MySQL Enterprise Monitor: Architecture



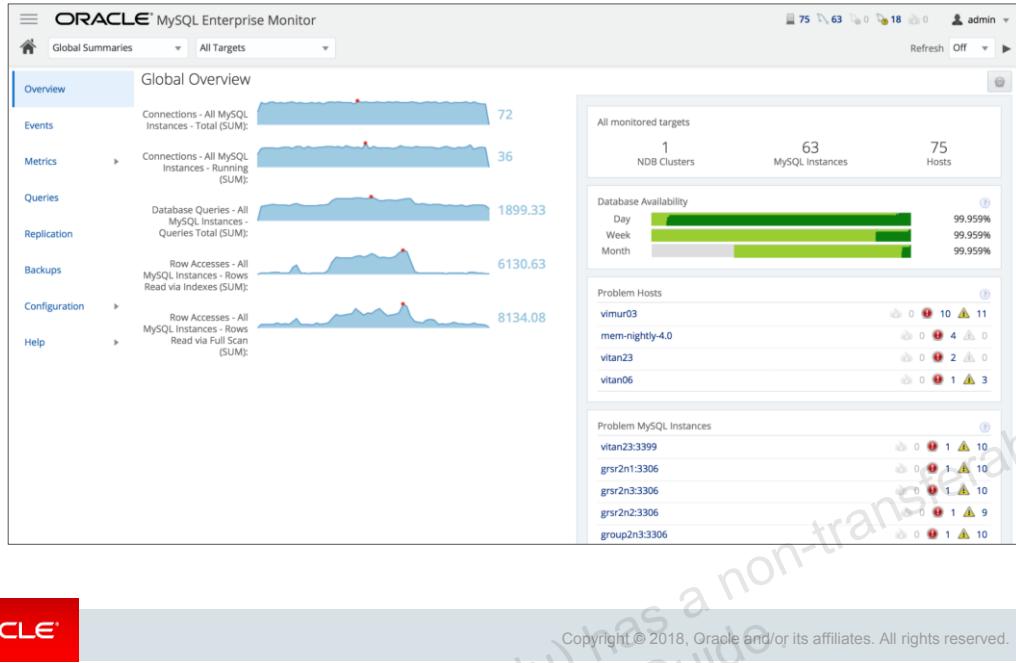
ORACLE®

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

MySQL Enterprise Monitor can monitor one or more MySQL instances, whether they are deployed on a single machine or across multiple hosts. MySQL Enterprise Monitor consists of the following core components:

- **MySQL Enterprise Monitor Agents:** Collects variables, status, and health information from MySQL instances and hosts, and send this information to the MySQL Enterprise Monitor Service Manager. You typically run one agent process on each host.
- **MySQL Enterprise Monitor Service Manager:** Collates the information sent by the agents. The Service Manager checks to see if the status variables it receives are within acceptable limits. If not, it can trigger alarms and send notifications to the DBA.
- **MySQL Enterprise Monitor User Interface:** You interact with MySQL Enterprise Monitor using a browser-based interface. The application enables you to configure monitoring behavior and view the information collected by agents. It can produce graphs to help you understand the data and identify the timeframes and trends related to potential issues.
- **MySQL Enterprise Monitor Advisors:** Warn about potential problems and provide troubleshooting advice. All advisors are enabled by default. You configure the advisors to trigger events that you are interested in.

MySQL Enterprise Monitor: Global Overview



ORACLE®

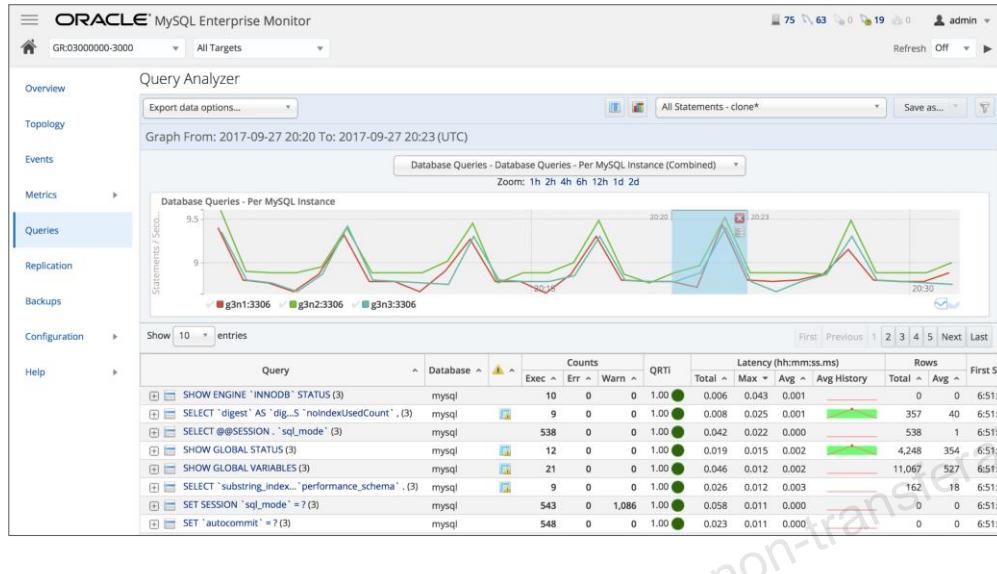
Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The screenshot in the slide shows the MySQL Enterprise Monitor Global Overview. The Global Overview provides a web-based interface for you to examine various aspects of your system, including:

- Database performance
- Availability
- Critical events

Visually examine a single server, a custom group, or all servers. Real-time and historical graphs allow you to drill down into detailed server statistics.

MySQL Enterprise Monitor: Query Analyzer



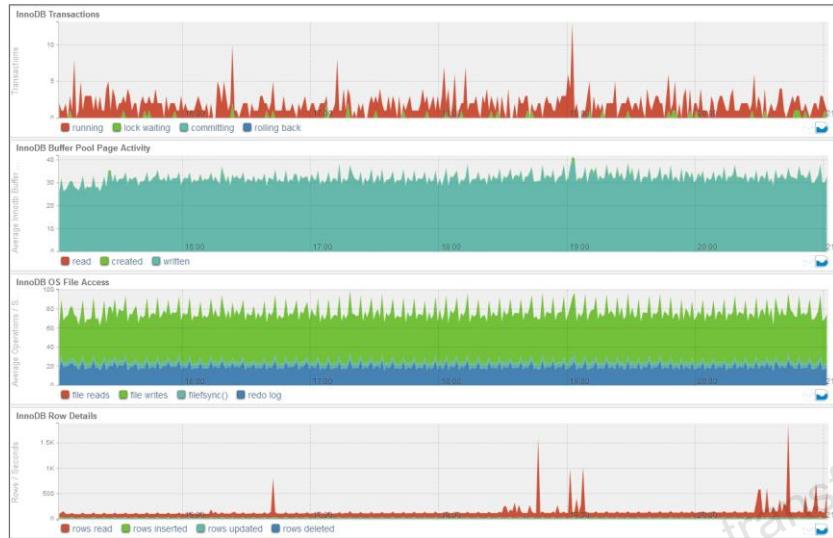
ORACLE®

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

The slide shows a screenshot of Query Analyzer. The Query Analyzer presents detailed information about database queries, including row counts and performance times. User applications can provide the Query Analyzer with statistical information about queries in three different ways:

- **Using the MySQL Proxy functionality built into MySQL Enterprise Monitor Agent:** The queries from the client application are directed through the proxy to the MySQL server and all results come back to the client via the proxy. The proxy maintains statistical information and supplies this data to the Service Manager for analysis.
- **Using a MySQL Connector with a corresponding MySQL Enterprise Monitor Plugin:** The connectors handle the collection of statistical information and send it to the Service Manager. This method does not require the MySQL Proxy.
- **Using a MySQL Connector with the Aggregator Service:** The MySQL Connector sends the raw statistical data to the MySQL Enterprise Monitor Agent, which aggregates the information in a format that enables the Server Manager to perform the analysis.

MySQL Enterprise Monitor: InnoDB Performance



ORACLE®

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

The screenshot in the slide shows a graphical view of some of the InnoDB monitoring options that are available in MySQL Enterprise Monitor. MySQL Enterprise Monitor InnoDB monitoring options include:

- **InnoDB Performance Monitor:** Displays the key metrics for InnoDB engine performance
- **InnoDB Configuration:** Provides hints and tips on how to improve your InnoDB configuration based on the current performance and trend analysis
- **InnoDB Locking:** Keeps you informed about existing and potential InnoDB locking issues
- **InnoDB Buffer Pool:** Aggregates statistics about your InnoDB buffer pool usage, and the related configuration options
- **Index Tuning:** Alerts you when your index usage needs improvement, and provides advice on how to improve it

Topics

- MySQL monitoring tools
- Oracle DB monitoring tools
- Community monitoring tools
- Linux tools
- Benchmarking tools



ORACLE®

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

Oracle Enterprise Manager for MySQL

Provides a range of useful MySQL performance monitoring functions, including:

- Real-time performance metrics and Key Performance Indicators (KPIs)
- Insight into InnoDB storage engine activity
- A range of predefined graphs and reports



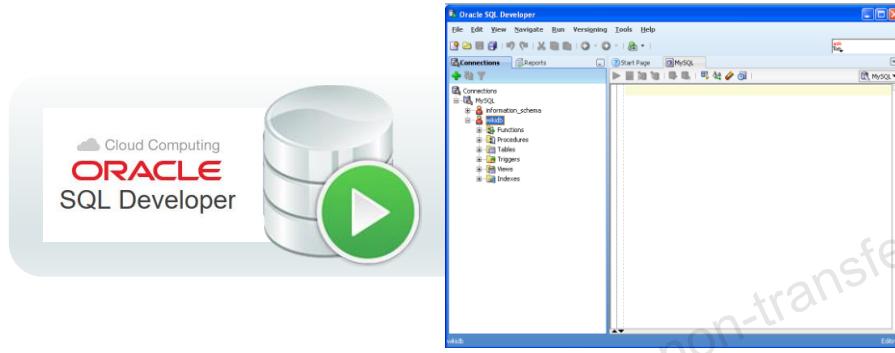
Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Oracle Enterprise Manager might be a familiar tool for DBAs who have previously worked with Oracle Database. In addition to its performance-monitoring capabilities, Oracle Enterprise Manager for MySQL also includes many features that assist in general monitoring and maintenance of MySQL servers, including:

- Availability and uptime statistics for all of your MySQL instances
- Integration with MySQL Enterprise Firewall and MySQL Enterprise Audit for security
- Monitoring of slave and master servers in a MySQL replication topology
- Enterprise Configuration Management (ECM)
- Remote monitoring of MySQL instances

Oracle SQL Developer

- Is an integrated development environment (IDE)
- Simplifies development against and management of on-premise and cloud deployments of Oracle Database
- Can also be used to connect to MySQL databases using the MySQL Connector/J JDBC driver



ORACLE®

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

Quiz



Which Oracle commercial products continuously monitor your MySQL servers and tell you about potential problems before they affect your system?

- a. The mysqladmin command-line client
- b. MySQL Utilities
- c. MySQL Enterprise Monitor
- d. Oracle Enterprise Manager for MySQL



ORACLE®

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

Answer: c, d

Topics

- MySQL monitoring tools
- Oracle DB monitoring tools
- Community monitoring tools
- Linux tools
- Benchmarking tools



ORACLE®

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

Community Monitoring Tools

Some of the more popular tools include:

- **dim_STAT**: Created by Dmitri Kravtchuk
- **Mytop**: Created by Jeremy Zawodny
- **Percona Toolkit**: Created by the Percona MySQL support team



ORACLE®

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

dim_STAT

A powerful solution for system and application monitoring and performance analysis on UNIX-based systems. The main features of **dim_STAT** include:

- Web-based user interface
- All collected data stored in a database
- Multiple data views
- Interactive (Java) or static graphs (PNG)
- Real-time and multi-host monitoring
- Statistics integration add-in
- Reporting capability with automated features

mytop

The `mytop` tool is a Perl script that you can use to monitor MySQL databases. It is similar to the Linux `top` command that monitors system processes. However, `mytop` monitors MySQL threads and the database's overall performance, allowing system administrators or developers to see how applications interact with a database.

Percona Toolkit

Percona Toolkit is a collection of command-line tools developed by Percona, which perform a variety of MySQL server and system tasks. The toolkit includes tools for:

- Verifying data consistency in replicated databases
- Archiving rows efficiently
- Locating duplicate indexes
- Summarizing MySQL server information
- Analyzing queries from logs
- Collecting diagnostic information when problems occur

Note: More information about all these tools is available online in the following locations:

- **dim_STAT:** <http://dimitrik.free.fr>
- **mytop:** <http://jeremy.zawodny.com/mysql/mytop/>
- **Percona Toolkit:** <http://www.percona.com/software/percona-toolkit>

Topics

- MySQL monitoring tools
- Oracle DB monitoring tools
- Community monitoring tools
- **Linux tools**
- Benchmarking tools



ORACLE®

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

Linux Tools

- **iostat:** Provides valuable information about your server, including:
 - Amount of I/O each device processes
 - Type of process
 - Request type
 - Queue size
 - Response time
- **vmstat:** Shows a high-level view of system load
- **sar:** Collects, reports, and saves system activity data for historical comparison
- **top:** Displays system information in real time, including:
 - Summary information
 - Current tasks being managed by the kernel



ORACLE®

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

iostat

- Display activity since system startup:

```
$ iostat
Linux 4.1.12-32.2.1.el7uek.x86_64 (MyMachine) 06/07/2017
_x86_64_ (1 CPU)

avg-cpu: %user  %nice %system %iowait  %steal   %idle
          0.90    0.00   0.34    0.41    0.00   98.36

Device:      tps  Blk_read/s  Blk_wrtn/s  Blk_read  Blk_wrtn
sda           3.63      35.16       34.22   1330316   1294730
dm-0          1.84      29.57       22.35   1119066   845832
dm-1          2.16      5.38        11.86   203552    448840
```

- Display extended information: `iostat -x`
- Continuous monitoring: `iostat -x [interval] [count]`



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

The example in the slide shows the default output of `iostat`, which displays CPU utilization and I/O statistics since the system started. View more detailed information about both CPU and I/O activity by using the `-x` option.

To view current activity, specify an interval and, optionally, a count.

- Current activity is displayed at each interval, measured in seconds.
- Output continues until interrupted or when the specified count is reached.

The output of `iostat -x` includes the columns `r/s` (reads per second), `w/s` (writes per second), and `svctm` (service time, in milliseconds). You can establish the number of I/O requests that the server processes per second by using the following equation based on Little's Law (a theorem in queuing theory), which relates the average number of concurrent requests to the arrival rate and duration of those requests:

$$\text{Average I/O requests per second} = (r/s + w/s) * (svctm/1000)$$

Note: For more information about `iostat` options, see the `iostat` man page.

vmstat

- Show summary information, including active and inactive memory:

```
$ vmstat -a
procs -----memory----- --swap-- -----io----- --system-- -----cpu-----
r b swpd free inact active si so bi bo in cs us sy id wa st
0 0 179292 89888 391096 376840 3 6 17 17 51 97 1 0 98 0 0
```

- List event counters and memory statistics:

```
$ vmstat -s
1020876 total memory
919084 used memory
364644 active memory
391380 inactive memory
101792 free memory
37180 buffer memory
98956 swap cache
2064380 total swap
179256 used swap
1885124 free swap ...
```



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

The first example in the slide uses `vmstat` with the `-a` option to show active and inactive memory in a tabular format.

The “procs” field has two columns:

- r: The number of processes waiting for run time
- b: The number of processes in uninterruptible sleep (blocked processes)

The “memory” field has four columns:

- swpd: The amount of swap space (virtual memory) used.
- free: The amount of idle memory (free RAM)
- inact: The amount of inactive memory
- active: The amount of active memory

The “swap” field has two columns:

- si: The amount of memory swapped in from disk
- so: The amount of memory swapped out to disk

Extensive swapping is bad for MySQL performance, so the values for `si` and `so` should usually be close to, or at, zero. Ensure that there is enough system RAM to prevent swapping.

The “io” field has two columns:

- bi: Blocks received from a block device (blocks in)
- bo: Blocks sent to a block device (blocks out)

The “system” field has two columns:

- in: The number of system interrupts per second, including the clock
- cs: The number of process context switches per second

The “cpu” field has five columns:

- us: Time spent running non-kernel code (user time, including nice time)
- sy: Time spent running kernel code (system time)
- id: Time spent idle
- wa: Time spent waiting for I/O
- st: “Steal time”. This is the percentage of time a virtual CPU waits for a real CPU while the hypervisor is servicing another virtual processor.

Like iostat, you can observe current activity with vmstat by specifying an interval and count. For example, the following output delays 10 seconds before the next update and repeats three times:

```
vmstat -a -n 10 3
```

The -n option prevents the header information being displayed in every update.

The -s option shown in the second slide example displays a table of various event counters and memory statistics. This display does not repeat.

sar

- Display today's CPU activity so far:

```
$ sar
Linux 2.6.39-200.24.1.el6uek.x86_64 (MyServer) 06/07/2017
_x86_64_ (1 CPU)
09:30:01 AM   CPU   %user   %nice   %system   %iowait   %steal   %idle
09:40:01 AM   all    0.94    0.00     0.25     0.14     0.00    98.67
09:50:01 AM   all    0.88    0.00     0.22     0.19     0.00    98.71
```

- Report swap statistics every second for 3 seconds, and log the results:

```
$ sar -S 1 3 -o ~/sar/sar.log
Linux 2.6.39-200.24.1.el6uek.x86_64 (MyServer) 06/07/2017
_x86_64_ (1 CPU)
02:34:24 PM kbswpfree kbswpused %swpused kbswpcad %swpcad
02:34:25 PM 1883596 180784 8.76 36896 20.41
02:34:26 PM 1883596 180784 8.76 36896 20.41
02:34:27 PM 1883596 180784 8.76 36896 20.41 ...
```



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

The `sar` command writes the contents of selected cumulative activity counters in the operating system to standard output. If you do not supply an interval parameter, `sar` displays the average statistics since the system was started. If you specify an interval, but no count parameter, `sar` generates reports continuously.

You can save the collected data in the file specified by `-o [filename]` flag, which makes `sar` useful for analyzing historical data. If you do not supply a file name, `sar` uses the standard system activity daily data file, the `/var/log/sysstat/sa[dd]` file, where the `dd` parameter indicates the current day. By default, `sar` stores all the available data, but the command supports many options that allow you to specify which information you are interested in. See the `sar` man page for full details.

top

- List all running processes in order of CPU usage, updated every five seconds by default:

```
$ top
top - 15:58:43 up 13:07,  3 users,  load average: 0.00, 0.01, 0.05
Tasks: 153 total,   1 running, 151 sleeping,   1 stopped,   0 zombie
Cpu(s): 3.3%us, 1.3%sy, 0.0%ni, 94.0%id, 1.3%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 1020876k total, 927764k used, 93112k free, 23748k buffers
Swap: 2064380k total, 191188k used, 1873192k free, 117168k cached

      PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+   COMMAND
 3327 student    20   0  896m 151m 18m S  2.3 15.2  10:22.34 firefox
 2696 student    20   0  287m 12m 8484 S  2.0  1.3  0:15.68 gnome-terminal
 2045 root      20   0  136m 26m 5944 S  1.7  2.7  0:28.72 Xorg
 1594 root      20   0  691m 2160 1520 S  0.3  0.2  0:22.95 mysql-monitor-a
 1853 mysql     20   0 1335m 352m 2880 S  0.3 35.4  0:44.84 mysqld
 5216 root      20   0      0      0      0 S  0.3  0.0  0:00.17 kworker/
```



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

The `top` command helps you monitor general system performance and the proportion of system resources each process uses. If system resources become too low, it can result in problems with MySQL and the applications that depend on it. System resources can be taken up by individual users, or by other services hosted by your system. Knowing which processes are consuming a lot of resources (RAM and CPU cycles) can help you decide if you need to upgrade your system, or if some services need to be moved to another machine.

By default, the `top` command displays the processes in order of CPU usage. Press M (uppercase) while `top` is running to display processes sorted by memory usage. Press O (uppercase) to see a full list of available sort fields.

For full details and the options available, see the `top` man page.

Topics

- MySQL monitoring tools
- Oracle DB monitoring tools
- Community monitoring tools
- Linux tools
- Benchmarking tools



ORACLE®

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

Benchmarking Tools

- MySQL benchmarking tools:
 - BENCHMARK ():
 - Is a built-in MySQL function
 - mysqlslap:
 - Emulates client load for a MySQL Server
 - Reports the timing of each stage
- Open source community benchmarking tools:
 - sysbench
 - Is a modular, cross-platform, and multithreaded benchmark tool
 - Can be used to evaluate important operating system parameters for a system running a database under an intensive load



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

MySQL BENCHMARK() Function

- The MySQL BENCHMARK() function:
 - Executes a given expression a specified number of times
 - Always returns zero
- Execute BENCHMARK() from the mysql command-line client program:
 - mysql prints a line showing approximately how long the statement took to execute.
- Example:

```
mysql> SELECT BENCHMARK(1000000000,2+2);  
+-----+  
| BENCHMARK(1000000000,2+2) |  
+-----+  
| 0 |  
+-----+  
1 row in set (18.25 sec)
```

ORACLE

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

The built-in MySQL functions are typically highly optimized, but there are some exceptions. The BENCHMARK() function helps you determine if a particular function is problematic for your queries.

The example in the slide shows that MySQL took just over 18 seconds to execute the expression “2+2” a billion times on a particular server. The same operation on a different server might execute in more or less time.

Stress Tools

- Stress tools simulate a heavy load on a database.
 - Used for benchmarking
 - Do not simulate “typical” database usage
- Popular stress tools include:
 - mysqlslap
 - MySQL command-line tool
 - Installed by default in standard binary MySQL distributions
 - Community tools:
 - db_STRESS
 - Apache JMeter
 - Benerator
 - stress_driver



ORACLE

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

A stress tool is an application, which simulates a heavy load on a database that you can use to benchmark performance. It is designed to work the database as hard as possible and is not typical of normal usage.

Note: Details of the community tools referred to in the slide:

- **db_STRESS:** A tool that enables you to put a load on your database system that produces a high-level metric (TPS: transactions per second). See the website at: http://dimitrik.free.fr/db_STRESS.html.
- **Apache JMeter:** A tool that you can use to simulate a heavy load on your server, network, or object to test its strength or to analyze overall performance under different load types. See the website at: <http://jmeter.apache.org>.
- **Benerator:** A performance test data-generation tool that you can use to completely synthesize test data or import and anonymize your production data. See the website at: <http://databene.org/databene-benerator>.
- **stress_driver:** A general-purpose stress-test tool using the Perl scripting language. See the website at: <http://stress-driver.sourceforge.net>.

mysqlslap Stress Tool

- Example 1: Automatically generate and execute SQL statements from 100 concurrent connections:

```
$ mysqlslap --user=student --auto-generate-sql --concurrency=100
Benchmark
Average number of seconds to run all queries: 0.886 seconds
Minimum number of seconds to run all queries: 0.886 seconds
Maximum number of seconds to run all queries: 0.886 seconds
Number of clients running queries: 100
Average number of queries per client: 0
```

- Example 2: Execute a specified query five times, from 200 concurrent connections:

```
$ mysqlslap --user=student --create-schema=world_innodb \
> --query="SELECT City.Name, City.District FROM City, Country \
> WHERE City.CountryCode = Country.Code AND Country.Code='IND';" \
> --concurrency=200 --iterations=5
Benchmark
Average number of seconds to run all queries: 0.374 seconds
Minimum number of seconds to run all queries: 0.242 seconds
Maximum number of seconds to run all queries: 0.847 seconds
Number of clients running queries: 200
Average number of queries per client: 1
```



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

The `--auto-generate-sql` switch performs the following actions:

- Creates a table
- Executes an `INSERT` on the table with dummy data
- Executes a `SELECT` query to retrieve the dummy data
- Drops the table

Use the `-v` option to display the individual steps. Add extra v's (for example, `-vv`) to increase the verbosity of the output.

You can create larger tables with the `--number-char-cols` and `--number-int-cols` options. Use the `--number-of-queries` option to force each "client" to execute the specified number of queries.

To save your benchmark results, pipe them to an output file, optionally specifying an output format. For example, the following `mysqlslap` command uses an automatically generated table with 11 columns (`--number-char-cols + --number-int-cols`). It executes 1,000 queries (`--number-of-queries`) from each of 100 client connections (`--concurrency`) and saves the report in CSV (`--csv`) format:

```
$ mysqlslap --csv=/tmp/output.csv --user=student --auto-generate-sql
--concurrency=100 --number-of-queries=1000 --number-char-cols=4
--number-int-cols=7
```

sysbench

sysbench is a modular, cross-platform, and multithreaded benchmark tool. It evaluates OS parameters that are important for a system running a database under a heavy load, including:

- File I/O performance
- Scheduler performance
- Memory allocation and transfer speed
- POSIX threads implementation performance
- Database server performance (OLTP benchmark)



ORACLE®

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

The sysbench tool also enables you to emulate custom loads using the Lua scripting language. POSIX is a set of formal descriptions that provide a standard for the design of operating systems, especially those that are compatible with UNIX.

Note: For more information and to download sysbench, see the website at <https://github.com/akopytov/sysbench>.

Using SysBench

Benchmarking MySQL:

1. Create an `oltp` test table in the test database with 1,000,000 rows:

```
# sysbench --test=oltp --oltp-table-size=1000000 --mysql-db=test  
--mysql-user=root --mysql-password=oracle prepare
```

2. Execute the benchmark:

```
# sysbench --test=oltp --oltp-table-size=1000000 --mysql-db=test  
--mysql-user=root --mysql-password=oracle --max-time=60  
--oltp-read-only=on --max-requests=0 --num-threads=8 run
```

3. Remove the `oltp` test table:

```
# sysbench --test=oltp --mysql-db=test --mysql-user=root  
--mysql-password=oracle cleanup
```



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

You can also use SysBench to benchmark your CPU and file I/O.

Benchmarking the CPU:

```
$ sysbench --test=cpu --cpu-max-prime=20000 run
```

Benchmarking file I/O:

First, create a test file that is much bigger than system RAM (otherwise, the system uses RAM for caching, which affects the results). For example:

```
$ sysbench --test=fileio --file-total-size=150G prepare
```

Then, execute the benchmark:

```
$ sysbench --test=fileio --file-total-size=150G --file-test-mode=rndrw --  
init-rng=on --max-time=300 --max-requests=0 run
```

Note: For a full explanation of these and other SysBench options, see the SysBench documentation at: <https://github.com/akopytov/sysbench#usage>.

Sample SysBench Output - 1

```
# sysbench --test=oltp --oltp-table-size=1000000 --mysql-db=test  
--mysql-user=root --mysql-password=oracle --max-time=60  
--oltp-read-only=on --max-requests=0 --num-threads=8 run  
sysbench 0.4.12: multi-threaded system evaluation benchmark  
  
No DB drivers specified, using mysql  
Running the test with following options:  
Number of threads: 8  
  
Doing OLTP test.  
Running mixed OLTP test  
Doing read-only test  
Using Special distribution (12 iterations, 1 pct of values are returned  
in 75 pct cases)  
Using "BEGIN" for starting transactions  
Using auto_inc on the id column  
Threads started!  
Time limit exceeded, exiting...  
(last message repeated 7 times)  
Done.
```



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

The benchmark in the slide example executes read-only queries (`--oltp-read-only=on`) against a table containing 1,000,000 rows, using eight concurrent threads. It works the database as hard as possible for one minute (`--max-time=60`) and then displays the results, as shown in the slide.

Sample SysBench Output - 2

```
OLTP test statistics:
  queries performed:
    read:                      2253860
    write:                     0
    other:                     321980
    total:                     2575840
  transactions:               160990 (2683.06 per sec.)
  deadlocks:                  0      (0.00 per sec.)
  read/write requests:        2253860 (37562.81 per sec.)
  other operations:           321980 (5366.12 per sec.)

  Test execution summary:
    total time:                 60.0024s
    total number of events:     160990
    total time taken by event execution: 479.3419
    per-request statistics:
      min:                      0.81ms
      avg:                      2.98ms
      max:                      3283.40ms
      approx. 95 percentile:    4.62ms
    Threads fairness:
      events (avg/stddev):    20123.7500/63.52
      execution time (avg/stddev): 59.9177/0.00
```



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

The most important metric to observe in these results is transactions per second:

transactions: 160990 (2683.06 per sec.)

Summary



In this lesson, you should have learned how to:

- Monitor server status and performance by using MySQL tools and databases
- Describe how to use Oracle Database tools to manage and monitor MySQL
- Monitor system performance by using Linux commands
- Perform benchmarks using MySQL and third-party tools

ORACLE®

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

Practices

- 3-1: Using MySQL Monitoring Tools
- 3-2: Using MySQL Enterprise Monitor
- 3-3: Using Benchmark Tools
- 3-4: Using Linux System Monitoring Tools



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

4

Performance Schema



MySQL™

ORACLE®

Topics

- What is Performance Schema?
- Schema overview
- Configuration
- The Instance and Connection Tables
- Querying event data
- The `sys` schema
- MySQL Workbench performance tools



ORACLE®

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

Objectives



After completing this lesson, you should be able to:

- Describe the general structure of the Performance Schema
- Configure the Performance Schema to collect desired performance statistics
- Query the Performance Schema to retrieve statistics of interest
- Describe and use the `sys` schema
- Configure and use the Performance Schema in MySQL Workbench

ORACLE®

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

Topics

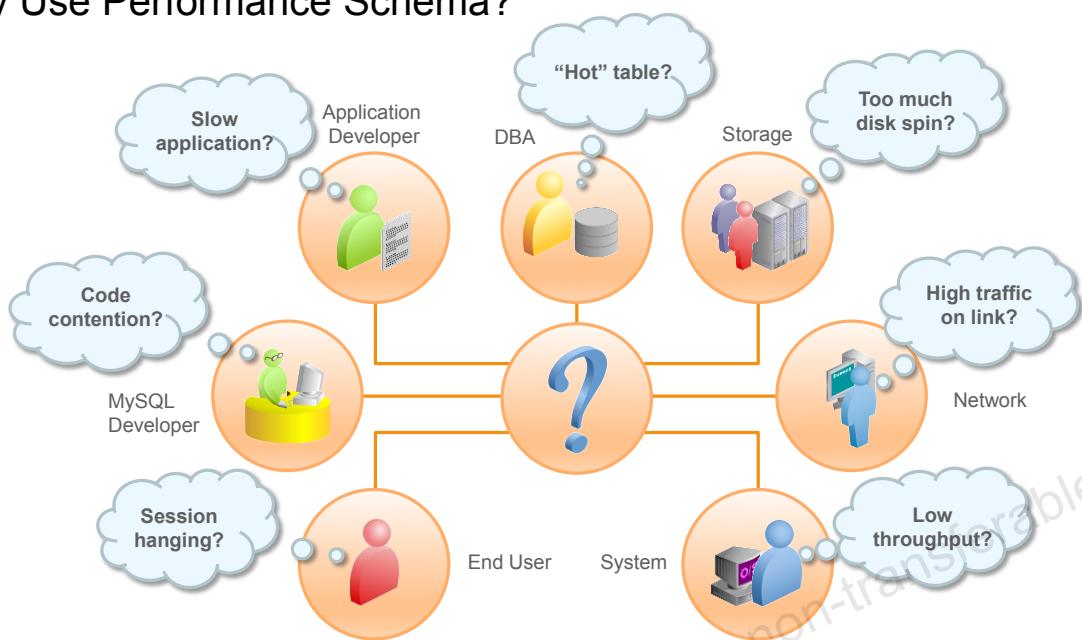
- What is Performance Schema?
- Schema overview
- Configuration
- The Instance and Connection Tables
- Querying event data
- The sys schema
- MySQL Workbench performance tools



ORACLE®

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

Why Use Performance Schema?



ORACLE

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

The Performance Schema enables DBAs to investigate performance issues at a low level by providing an insight into the internal operations of the MySQL server. With access to this information, DBAs can answer questions such as:

- Which user or file incurs the most I/O?
- What are the longest-running queries?
- Which queries sort the entire table?
- Which queries are using temporary tables?

Performance Schema

- Is a database, consisting of many in-memory temporary tables and views
- Relies on its own dedicated storage engine: Performance Schema
- Stores current and historical information about internal server operations, the values of server variables, replication status, and more
- Enables DBAs to access this information by querying the schema
- Assists in the diagnosis of performance issues at a low level
 - You must understand the schema to use it effectively.
 - Consider using the MySQL `sys` schema helper tool, or MySQL Workbench.
- Is enabled by default



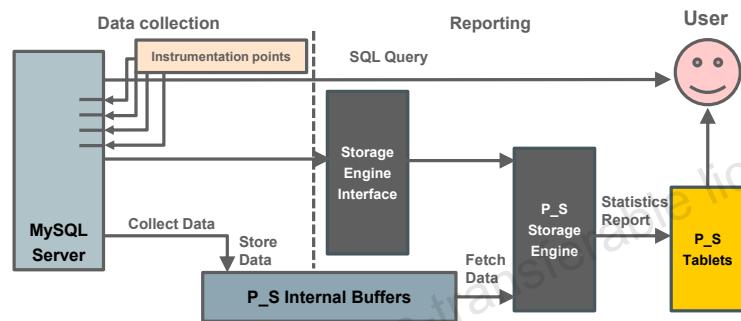
Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The Performance Schema is a system database, which enables the MySQL server to gather performance metrics and the DBA to query them by using SQL. The data is stored in temporary tables and views, with no persistent disk storage. The schema is complex and you can use a helper tool, such as the MySQL `sys` schema or MySQL Workbench Performance Reports (both are covered later in this lesson), to make it easier to query the Performance Schema.

It is important to recognize that the Performance Schema is a diagnostic tool, providing insight into the operation of your server. It can highlight problems, but will not resolve them for you.

How Performance Schema Works

- The Performance Schema collects information about server events using “instrumentation points” (*instruments*) in the server’s source code.
 - An “event” is any server operation that takes time to execute.
- To monitor events, the appropriate instruments must be enabled in the Performance Schema.
- When enabled, the instrumented event data is stored in Performance Schema event tables that can be queried by using SQL.



ORACLE

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

In the context of the Performance Schema, an “event” is any operation that the server performs which takes time to complete and has been instrumented so that timing information can be collected. An instrument is a point within the server source code that captures information about when and how that code executes.

Performance Schema events are distinct from events written to the server’s binary log (which describe data modifications) and Event Scheduler events (which are a type of stored program).

These events include function calls, a wait for the operating system, a stage of a SQL statement execution such as parsing or sorting, or an entire statement or group of statements. Event collection provides access to information about synchronization between processes, file and table I/O, table locks, and so forth for the server and for several storage engines.

The collection and storage of instrumented event data is managed by the Performance Schema storage engine. The user can query the `performance_schema` database to retrieve this information.

Instrument Names

- Describe the type of event data that the instrument collects
 - Naming is hierarchical, like a file path, where each component is separated by a “/”.
- Consist of:
 - A prefix, denoting the Performance Schema instrumentation category, such as `wait` or `stage`
 - A suffix, denoting the instrumented component (such as `io/file/innodb`) within the category and a specific instrument (such as `innodb_data_file`)
- Progress left to right from more general to more specific components
 - The interpretation of a given component in a name depends on the components to the left of it.
 - For example, InnoDB appears in both of the following instrument names, but the context is different:
 - `wait/io/file/innodb/innodb_log_file`: Refers to a wait for file I/O on the InnoDB log file
 - `wait/synch/cond/innodb/commit_cond`: Relates to a synchronization instrument



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

Instrument Prefixes

The top-level instrument component denotes the type of event:

- **idle**: An instrumented idle event. This instrument has no further components and is generated when a socket is waiting for a request from the client.
- **memory**: An instrumented memory event, of the form `memory/component/instrument_name`
- **stage**: An instrumented stage event that indicates the processing stage of a statement, as reported by `SHOW PROCESSLIST`, such as `stage/sql/Copying to tmp table`
- **statement**: An instrumented statement event. Often-used components include **commands** (`statement/com`), **stored programs** (`statement/sp`), and **SQL** (`statement/sql`).
- **transaction**: An instrumented transaction event. This instrument has no further components.
- **wait**: An instrumented wait event, such as `wait/io`, `wait/lock`, and `wait/synch`



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

Instrument Suffixes

Describe the code area being instrumented and may include:

- The name of the major component, for example:
 - A server module, such as innodb, mysys, or sql
 - The name of a plugin, such as performance_schema
- The name of a code variable, in the form of:
 - The variable name, for example, COND_thread_cache for a global variable
 - Class::Module, for example, BINLOG::LOCK_index for the member variable LOCK_index, of the stated class BINLOG



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

Instrument Names: Example

```
wait/io/file/myisam/log
wait/io/file/mysys/charset
wait/lock/table/sql/handler
wait/synch/cond/sql/BINLOG::update_cond
wait/synch/mutex/mysys/BITMAP_mutex
wait/synch/mutex/sql/LOCK_delete
wait/synch/rwlock/sql/Query_cache_query::lock
stage/sql/closing tables
stage/sql/Sorting result
statement/com/Execute
statement/com/Query
statement/sql/create_table
statement/sql/lock_tables
```



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

Topics

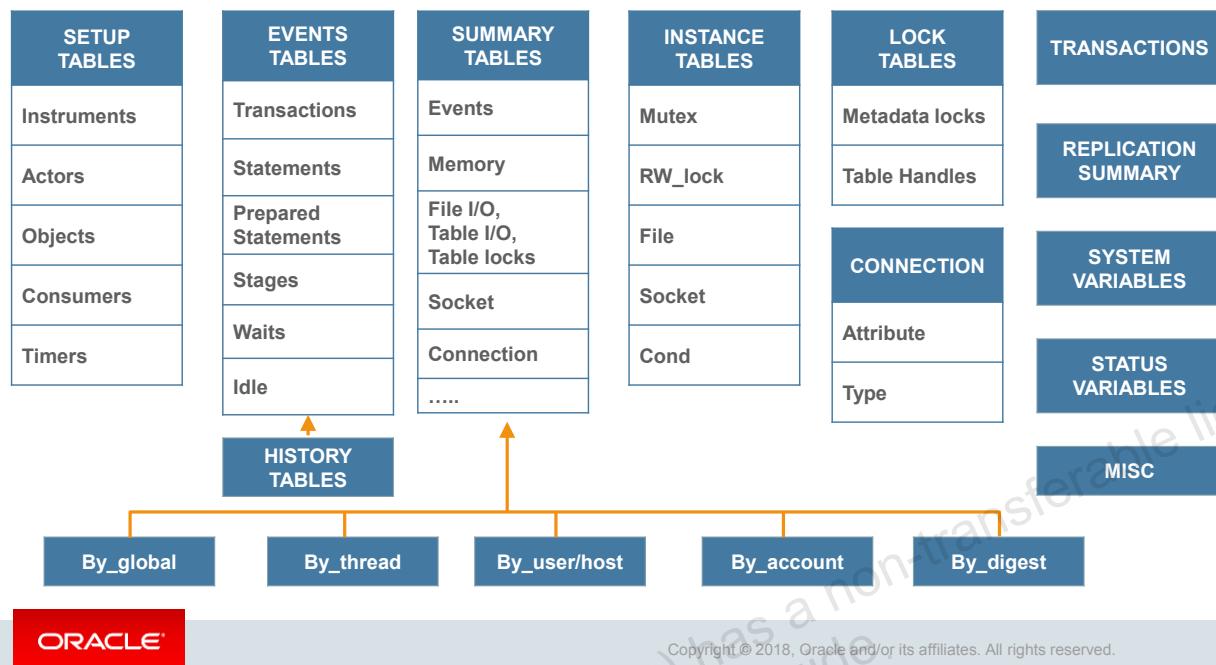
- What is Performance Schema?
- Schema overview
- Configuration
- The Instance and Connection Tables
- Querying event data
- The sys schema
- MySQL Workbench performance tools



ORACLE®

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

Structure of the Performance Schema



The Performance Schema contains 87 tables that contain a lot of information about many aspects of the MySQL server. The diagram in the slide shows mainly broad categories of tables rather than individual table names.

Table Groups

The tables in the Performance Schema can be grouped into the following main categories:

- **Current events tables (`events_<type>_current`)**: Store the most recently monitored events. For example:
 - `events_waits_current`: The most recent wait event for each thread
 - `events_stages_current`: The most recent stage event for each thread
 - `events_statements_current`: The most recent statement event for each thread
 - `events_transactions_current`: The most recent transaction event for each thread
- **Historical events tables (`events_<type>_history*`)**: Similar in structure to the current events tables, but contain more rows in which to store historical event data. For example:
 - `events_waits_history`: The 10 most recent wait events per thread
 - `events_waits_history_long`: The 10,000 most recent wait events for the MySQL instance



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

Query the current events tables to see what the server is doing right now. Query the historical events tables to see what the server has been doing recently.

Note

- The `*_current` and `*_history` tables contain information about threads that exist at the time the table is queried. The `*_history_long` tables might include events for threads that no longer exist.
- Queries that execute as prepared statements do not appear in any of the `events_statements_*` tables, including the summary tables discussed later in this lesson.

Table Groups

- **Instance Tables (*<type>_instances*)**: Store information about the types of objects that are instrumented (such as files, locks, mutexes, and so on). An instrumented object, when used by the server, produces an event. These tables provide event names and explanatory notes or status information.
- **Summary Tables (*events_<type>_summary_**)**: Contain aggregated event information, including details of events that have been discarded from the history tables. Tables in this category include event data grouped either globally, by thread, by user/host, by account, and by digest.
- **Setup Tables (*setup_**)**: Enables you to make entries to configure which user events to monitor, what events to monitor, how to monitor those events, and where to save the data
- **Miscellaneous**: All the other tables that do not fit into the other groups, such as tables containing information about threads and event timing units



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

Topics

- What is Performance Schema?
- Schema overview
- Configuration
- The Instance and Connection Tables
- Querying event data
- The sys schema
- MySQL Workbench performance tools

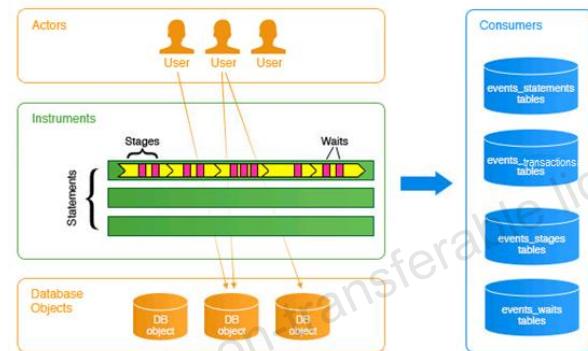


ORACLE®

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

Configuring the Performance Schema

- Use the Setup Tables to configure:
 - **Instruments:** What types of event data to monitor
 - **Consumers:** Which instrumented event data to collect
 - **Actors:** Which client connections to monitor
 - **Objects:** Which database objects to monitor
- Collect only what you need to minimize overhead.



ORACLE®

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

Although the Performance Schema includes sensible defaults, you should understand what they are and, if necessary, adjust the configuration to capture only the event data that you are interested in. The Performance Schema has been designed to incur minimal overhead, but if you enable every instrument and consumer for every user and database object on a busy server, performance might degrade.

The process of configuring the Performance Schema in this way is known as “filtering.”

Setup Tables

```
mysql> SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES  
      WHERE TABLE_SCHEMA = 'performance_schema'  
      AND TABLE_NAME LIKE 'setup_%';  
+-----+  
| TABLE_NAME |  
+-----+  
| setup_actors |  
| setup_consumers |  
| setup_instruments |  
| setup_objects |  
| setup_timers |  
+-----+  
5 rows in set (#.# sec)
```



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

Note: The `setup_timers` table is deprecated in MySQL version 5.7.21 and will be removed in a later version, so it is not covered in this course.

Configuring Instruments

The `setup_instruments` table lists the instruments that can collect event data.

- If the `ENABLED` column value is `YES`, then the instrument collects event data.
- If the instrument is enabled and the `TIMING` column value is `YES`, then the instrument also collects timing information. All instruments can be timed.

```
mysql> SELECT * from setup_instruments;
+-----+-----+
| NAME | ENABLED | TIMED |
+-----+-----+
...
| stage/sql/Waiting for commit lock | NO    | NO   |
| stage/sql/User lock             | NO    | NO   |
| stage/sql/Waiting for locking service lock | NO    | NO   |
| stage/innodb/alter table (end) | YES   | YES  |
| stage/innodb/alter table (flush) | YES   | YES  |
| stage/innodb/alter table (insert) | YES   | YES  |
...
+-----+-----+
1014 rows in set (#.## sec)
```



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

Enabling and disabling instruments is the most basic way to filter the event data that Performance Schema collects. If an instrument is disabled, it will not collect any event data and, therefore, the other setup tables will have no further effect on events of that type.

As a general rule, the lower level the instrument is, the more overhead is involved in collecting its data. For example, monitoring transactions incurs a relatively low overhead, monitoring mutexes incurs a high overhead.

Configuring Instruments: Examples

- Turn off timing for all CSV-related instruments:

```
UPDATE setup_instruments SET TIMED = 'NO'  
WHERE NAME LIKE '%CSV%';
```

- Disable all CSV-related instruments:

```
UPDATE setup_instruments SET ENABLED = 'NO'  
WHERE NAME LIKE '%CSV%';
```

- Enable only I/O-related instruments:

```
UPDATE setup_instruments  
SET ENABLED = IF(NAME LIKE '%/io/%', 'YES', 'NO');
```



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

Event Timing

The `performance_timers` table lists the available timers and their characteristics.

```
mysql> SELECT * FROM performance_schema.performance_timers;
+-----+-----+-----+-----+
| TIMER_NAME | TIMER_FREQUENCY | TIMER_RESOLUTION | TIMER_OVERHEAD |
+-----+-----+-----+-----+
| CYCLE      | 3391522388    | 1               | 18             |
| NANOSECOND  | 10000000000   | 1               | 36             |
| MICROSECOND | 1000000       | 1               | 36             |
| MILLISECOND | 1038          | 1               | 33             |
| TICK        | 102            | 1               | 580            |
+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```



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

The `performance_timers` table contains the following information:

- `TIMER_NAME`: The type of timer. For example, the `CYCLE` timer is based on the central processing unit's (CPU's) cycle counter.
- `TIMER_FREQUENCY`: The number of timer units per second. For `TICK`, the frequency may vary by platform (for example, some use 100 ticks/second, others 1000 ticks/second). The other timers are based on fixed fractions of seconds.
- `TIMER_RESOLUTION`: The increment value. If a timer has a resolution of 10, its value increases by 10 each time.
- `TIMER_OVERHEAD`: The minimum number of cycles of overhead to time an event once with the given timer. The actual overhead incurred is twice the displayed value because the timer is invoked both at the beginning and end of the event.

If the `TIMER_FREQUENCY`, `TIMER_RESOLUTION`, and `TIMER_OVERHEAD` columns contain `NULL` values, then that timer is not available on your platform.

Configuring Consumers

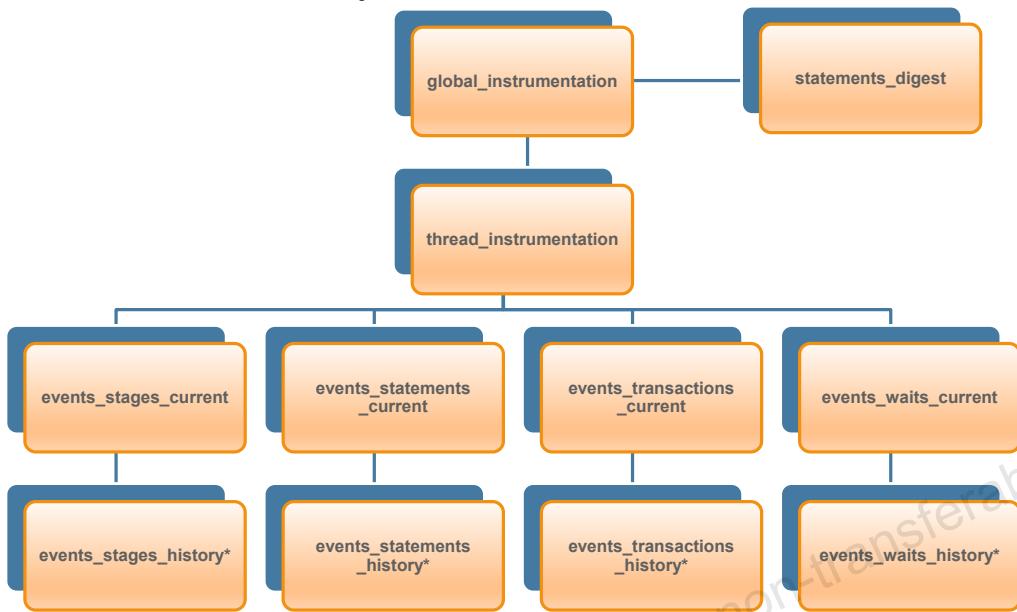
- The `setup_instruments` table specifies which instruments are enabled or disabled.
- The `setup_consumers` table specifies whether the event tables collect or ignore instrumented events.
- If the `ENABLED` column value is YES, the event tables collect instrumented events of the type referred to in the `NAME` column.
- The sample output shows the default configuration of `setup_consumers`.

```
mysql> SELECT * FROM setup_consumers;
+-----+-----+
| NAME          | ENABLED |
+-----+-----+
| events_stages_current | NO      |
| events_stages_history | NO      |
| events_stages_history_long | NO     |
| events_statements_current | YES    |
| events_statements_history | NO      |
| events_statements_history_long | NO     |
| events_transactions_current | YES    |
| events_transactions_history | NO      |
| events_transactions_history_long | NO     |
| events_waits_current | NO      |
| events_waits_history | NO      |
| events_waits_history_long | NO     |
| global_instrumentation | YES    |
| thread_instrumentation | YES    |
| statements_digest | YES    |
+-----+-----+
15 rows in set (0.00 sec)
```



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

Consumer Table Hierarchy



ORACLE

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

The entries in the `setup_consumers` table form a logical hierarchy. For a consumer to collect events, both the consumer and the consumers above it in the hierarchy must be enabled. The slide shows this hierarchy in a graphical format. The slide titled “Configuring Consumers” describes the effect of this hierarchy.

Note: The `sys` schema stored procedure `ps_setup_enable_consumer(consumer)` considers this hierarchy when enabling the specified consumer. The `sys` schema is described in the slide titled “`sys` schema” later in this lesson:

```
mysql> CALL ps_setup_enable_consumer('statement');
+-----+
| summary |
+-----+
| Enabled 4 consumers |
+-----+
```

Configuring Consumers

The Performance Schema determines how much event data to collect from the enabled instruments by processing the `setup_consumers` table as follows:

- **global_instrumentation**: The highest-level consumer. If global instrumentation is disabled, the Performance Schema ignores other consumer configuration settings.
- **thread_instrumentation**: If this and global instrumentation are enabled, the Performance Schema maintains thread-specific information and checks the configuration of `events_xxx_current` and other consumers.
- **events_xxx_current**: If an event consumer is enabled, the configuration of its associated `events_xxx_current_history` and `events_xxx_current_history_long` consumers are also consulted.
- **statements_digest**: If this consumer is enabled, the Performance Schema normalizes SQL statements in “digest” form that enables grouping of similar statements for analysis.



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

Configuring Objects

Modify the contents of the `setup_objects` table to configure the database objects for which events are collected.

- The `OBJECT_TYPE` column indicates the type of object to which a row applies.
- `TABLE` filtering affects table I/O events (wait/io/table/sql/handler instrument) and table lock events (wait/lock/table/sql/handler instrument).
- The `OBJECT_SCHEMA` and `OBJECT_NAME` columns must contain a literal schema or object name, or '%' to match any name.
- The `ENABLED` column indicates whether matching objects are monitored.
- The `TIMED` column indicates whether to collect timing information.



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

Default setup_objects Configuration

```
mysql> SELECT * FROM setup_objects;
+-----+-----+-----+-----+-----+
| OBJECT_TYPE | OBJECT_SCHEMA | OBJECT_NAME | ENABLED | TIMED |
+-----+-----+-----+-----+-----+
| EVENT | mysql | % | NO | NO |
| EVENT | performance_schema | % | NO | NO |
| EVENT | information_schema | % | NO | NO |
| EVENT | % | % | YES | YES |
| FUNCTION | mysql | % | NO | NO |
| FUNCTION | performance_schema | % | NO | NO |
| FUNCTION | information_schema | % | NO | NO |
| FUNCTION | % | % | YES | YES |
| PROCEDURE | mysql | % | NO | NO |
| PROCEDURE | performance_schema | % | NO | NO |
| PROCEDURE | information_schema | % | NO | NO |
| PROCEDURE | % | % | YES | YES |
| TABLE | mysql | % | NO | NO |
| TABLE | performance_schema | % | NO | NO |
| TABLE | information_schema | % | NO | NO |
| TABLE | % | % | YES | YES |
| TRIGGER | mysql | % | NO | NO |
| TRIGGER | performance_schema | % | NO | NO |
| TRIGGER | information_schema | % | NO | NO |
| TRIGGER | % | % | YES | YES |
+-----+-----+-----+-----+-----+
20 rows in set (0.00 sec)
```



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

In the default configuration of the `setup_objects` table shown in the slide, you can see that the Performance Schema collects events for all database objects, except those in `mysql`, `performance_schema`, and `INFORMATION_SCHEMA`.

Configuring Actors

The `setup_actors` table specifies which users' foreground threads are logged and monitored. Its configuration affects the contents of the `threads` table.

- For each new foreground thread, `setup_actors` is consulted:
 - If the host and user match a row in `setup_actors`, the `ENABLED` and `HISTORY` column values are used to set the `INSTRUMENTED` and `HISTORY` columns, respectively, of the `threads` table.
 - The `HOST` and `USER` columns should contain a literal host or username, or '%' to match any name.
 - If there is no matching row in `setup_actors`, the `threads` table's `INSTRUMENTED` and `HISTORY` column values are set to `NO`, and no monitoring occurs.
- Background threads have no associated user.
 - `INSTRUMENTED` and `HISTORY` are set to `YES` by default in the `threads` table and the `setup_actors` table is not used.
- By default, foreground threads are monitored for all user and host combinations.



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

The default configuration of `setup_actors` is as follows:

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

A "%" value in any column matches any name.

Configuring Thread Monitoring

- To monitor threads, the value of the `thread_instrumentation consumer` in the `setup_consumers` table must be YES.
 - The corresponding instrument for the event data you are interested in must also be enabled in `setup_instruments`.
- To enable historical data, ensure that:
 - The user and host are configured in `setup_actors` with `HISTORY` set to YES so that the `threads.HISTORY` column value is also set to YES.
 - The relevant history-related consumers are enabled in `setup_consumers`.
 - For example, to log wait event data to `events_waits_history` and `events_waits_history_long`, you must enable the corresponding `events_waits_history` and `events_waits_history_long` consumers.
- To determine the actual thread connection ID, query the `threads` table.
 - The number stored in `THREAD_ID` columns of the Performance Schema event tables is an internal reference to the thread, and not the actual thread identifier.



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

Configuring Instruments and Consumers at Startup

You can configure instruments and consumers at server startup by setting options.

- Configure instruments by using the following option format:

```
--performance-schema-instrument='instrument_name=value'
```

- Use wildcards (%) to match instrument names
- Valid values are:
 - OFF, FALSE, or 0: Disable the instrument
 - ON, TRUE, or 1: Enable and time the instrument
 - COUNTED: Enable the instrument for counts instead of timings

- Configuring consumers by using the following option format:

```
--performance-schema-instrument='instrument_name=value'
```

- Name pattern matching is not supported
- Valid values are:
 - OFF, FALSE, or 0: Do not collect event data for the consumer
 - ON, TRUE, or 1: Collect event data for the consumer



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

Examples

- Configure all synchronization instruments as enabled and counted:
`--performance-schema-instrument='wait/synch/cond/%=COUNTED'`
- Disable all instruments:
`--performance-schema-instrument='%=OFF'`
- Enable the `events_waits_history` consumer:
`--performance-schema-consumer-events-waits-history=ON`

Performance Schema Overhead

- The Performance Schema is enabled by default and incurs minimal overhead.
- Reduce the overhead on production servers by disabling instruments and consumers that you do not need to monitor:
 - Disable the `global_instrumentation` consumer to disable all monitoring.
 - Disable the Performance Schema entirely by setting the global `performance_schema` variable to OFF.
- Issue a `SHOW ENGINE` statement to display Performance Schema memory usage:

```
mysql> SHOW ENGINE performance_schema STATUS;
+-----+-----+-----+
| Type | Name | Status |
+-----+-----+-----+
...
| performance_schema | performance_schema.memory | 154723512 |
+-----+-----+-----+
229 rows in set (0.00 sec)
```



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

Instruments named with the prefix `memory/performance_schema/` expose how much memory is allocated for internal buffers in the Performance Schema. The `memory/performance_schema/` instruments are built in, always enabled, and cannot be disabled at startup or runtime. Built-in memory instruments are displayed only in the `memory_summary_global_by_event_name` table.

Note: Disabling the Performance Schema by setting `performance_schema=OFF` requires a system restart.

How Performance Schema Uses Memory

The Performance Schema:

- Reserves memory for internal buffers
 - Each buffer is instrumented, so you can track memory usage for individual buffers:

```
SELECT * FROM memory_summary_global_by_event_name
WHERE EVENT_NAME LIKE 'memory/performance_schema/%';
```
- Allocates memory automatically to:
 - Reduce the amount of configuration required
 - Minimize the memory overhead
 - Scale consumption of memory according to server load
- Uses the following memory allocation model:
 - Might allocate memory at server startup
 - Might allocate additional memory during server operation
 - Never frees memory during server operation, although memory may be recycled
 - Frees all memory at server shutdown



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

Configuring Performance Schema System Variables

To change the default values of Performance Schema system variables, specify the new values at server startup.

- Example: Change the sizes of the history tables for wait events in my.cnf:

```
[mysqld]
performance_schema=ON
performance_schema_events_waits_history_size=20
performance_schema_events_waits_history_long_size=15000
```

If you do not set values explicitly, many Performance Schema system variables are autosized either:

- At server startup
 - SHOW GLOBAL VARIABLES shows the autosized value
- Dynamically during server operation.
 - SHOW GLOBAL VARIABLES shows -1



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

Performance Schema system variables that are autosized are based on the values of the following server system variables:

- max_connections
- open_files_limit
- table_definition_cache
- table_open_cache

Quiz



Which of the following statements is true about the Performance Schema?

- a. It uses the InnoDB storage engine.
- b. It is disabled by default.
- c. If you disable the `thread_instrumentation` consumer, you cannot monitor statement event statistics.
- d. You can monitor only a specific instrument if it is enabled in the `setup_objects` table.



ORACLE®

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

Answer: c

Topics

- What is Performance Schema?
- Schema overview
- Configuration
- **The Instance and Connection Tables**
- Querying event data
- The sys schema
- MySQL Workbench performance tools



ORACLE®

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

Instance Tables

- Specify which types of object are instrumented
- Provide event names and explanatory notes or status information
- Are related to the `setup_instruments` table
 - The `NAME` or `EVENT_NAME` column in the instance table refers to the name of the monitoring instrument in `setup_instruments`
- Comprise the following tables:
 - `cond_instances`: Condition synchronization object instances
 - `file_instances`: File instances
 - `mutex_instances`: Mutex synchronization object instances
 - `rwlock_instances`: Lock synchronization object instances
 - `socket_instances`: Active connection instances



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

Note: The `prepared_statements_instances` table has more in common with the summary tables and is discussed in that context later in this lesson in the slide titled “Prepared Statements.”

cond_instances Table

- Stores details of all *conditions* encountered while the server executes
 - A *condition* is a synchronization mechanism in the code.
 - It is used to signal that a specific event has occurred, so that threads waiting for the condition can resume operation.
 - The condition name tells you which event a thread is waiting for.
- Contains the following columns:
 - NAME: The name of the instrument associated with the condition
 - OBJECT_INSTANCE_BEGIN: The memory address of the instrumented condition

```
mysql> SELECT * FROM cond_instances;
+-----+-----+
| NAME           | OBJECT_INSTANCE_BEGIN |
+-----+-----+
| wait/synch/cond/sql/COND_manager          |            33215584 |
| wait/synch/cond/sql/COND_server_started   |            33135808 |
...
+-----+-----+
40 rows in set (#.## sec)
```



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

Although the condition name provides some clue as to which particular condition is causing a thread to wait, there is no easy way of telling which other thread, or threads, caused the condition to occur.

The `cond_instances` table is one of the few tables in the Performance Schema that does not permit you to execute `TRUNCATE TABLE` to clear its contents.

file_instances Table

- Lists all files encountered by the Performance Schema when you enable file I/O instrumentation
- Contains the following columns:
 - FILE_NAME: The name of the file
 - EVENT_NAME: The associated instrument
 - OPEN_COUNT: The number of open handles on the file.
- Lists all currently open files by executing the following query:

```
mysql> SELECT * FROM file_instances WHERE OPEN_COUNT > 0;
+-----+-----+-----+
| FILE_NAME          | EVENT_NAME          | OPEN_COUNT |
+-----+-----+-----+
| /var/lib/mysql/ibdata1 | wait/io/file/innodb/innodb_data_file | 3          |
| /var/lib/mysql/ib_logfile0 | wait/io/file/innodb/innodb_log_file | 2          |
| /var/lib/mysql/ib_logfile1 | wait/io/file/innodb/innodb_log_file | 2          |
| /var/lib/mysql/employees/departments.ibd | wait/io/file/innodb/innodb_data_file | 3          |
| /var/lib/mysql/employees/dept_emp.ibd    | wait/io/file/innodb/innodb_data_file | 3          |
...
```

ORACLE

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

The OPEN_COUNT column value is incremented by one when the file is opened. It is decremented by one when the file is closed.

Files that have never been opened will not appear in the file_instances table.

Read/Write Locks and Mutexes

- Enable multiple program threads to share the same resource (such as a file), but not simultaneously
- Help to avoid consistency problems that can arise in high concurrency environments
- Differ as follows:
 - A **mutex** is a *mutual exclusion* object:
 - A lock on a resource by the thread that prevents other threads from accessing it until the lock is released
 - A **read/write lock** is a special type of mutex that enables more than one thread to access a resource simultaneously, to improve concurrency. Access is either:
 - **Shared**: Multiple threads can have a read lock on the resource simultaneously
 - **Exclusive**: Only one thread can have a write lock at any one time
 - **Shared-Exclusive** : A thread can have a write lock and allow inconsistent reads by other threads



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

mutex_instances Table

- Lists all internal mutexes recorded by the Performance Schema while the server executes
 - Used by MySQL developers to troubleshoot internal deadlocks in the code
- Contains the following columns:
 - NAME: The name of the associated instrument
 - OBJECT_INSTANCE_BEGIN: The address in memory of the instrumented mutex
 - LOCKED_BY_THREAD_ID: Contains THREAD_ID of the locking thread if a thread has a lock on a mutex. If there is no lock, the column contains NULL. For example:

```
mysql> SELECT * FROM mutex_instances WHERE
    -> LOCKED_BY_THREAD_ID IS NOT NULL\G
***** 1. row *****
      NAME: wait/synch/mutex/sql/MDL_wait::LOCK_wait_status
OBJECT_INSTANCE_BEGIN: 37104744
LOCKED_BY_THREAD_ID: 18
1 row in set (#.## sec)
```



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

Mutex instruments are resource-intensive. They are not normally useful in typical production systems and therefore all `wait/synch/mutex/%` instruments are disabled by default.

The example in the slide queries the `mutex_instances` table to display all mutexes currently locked by a thread (that is, those instances where the `LOCKED_BY_THREAD_ID` column value is not `NULL`).

Mutex Life Cycle in the Performance Schema

- When a thread attempts to lock a mutex, the `events_waits_current` table shows a row for that thread, indicating that it is waiting on a mutex (`EVENT_NAME` column), and specifying which mutex (`OBJECT_INSTANCE_BEGIN`).
- When a thread succeeds in locking a mutex:
 - The `events_waits_current` table shows that the wait on the mutex is complete (in the `TIMER_END` and `TIMER_WAIT` columns)
 - The completed wait event is added to the wait event history tables (if configured)
 - The `mutex_instances` table shows that the mutex is now owned by the thread (`THREAD_ID` column)
- When a thread unlocks a mutex, `mutex_instances` shows that the mutex now has no owner (the `THREAD_ID` column is `NULL`).
- When a mutex object is destroyed, the corresponding row is removed from the `mutex_instances` table.



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

You can use the following tables to detect bottlenecks or deadlocks between threads that involve mutexes:

- `events_waits_current`: To determine which mutex a thread is waiting for
- `mutex_instances`: To determine which thread currently owns the mutex

rwlock_instances Table

- Lists all read/write locks encountered by the Performance Schema while the server executes:
 - The type of access granted depends on the number of threads requesting a lock, and the nature of the lock requested.
 - Access is granted in shared, exclusive, or shared-exclusive mode, or not at all if other threads must complete first.
- Contains the following columns:
 - NAME: The name of the associated instrument
 - OBJECT_INSTANCE_BEGIN: The memory address of the instrumented lock
 - WRITE_LOCKED_BY_THREAD_ID: If a thread has locked the resource in exclusive (write) mode, this column contains the ID of the thread; otherwise, it contains NULL.
 - READ_LOCKED_BY_COUNT: Increments by one when a thread has an rwlock in shared (read) mode. This cannot identify the thread or threads that holds the read lock, but can help to diagnose read contention on a lock.



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

socket_instances Table

Contains details of active connections to the MySQL server. The server has a *listening socket* for each supported network protocol.

- The listening sockets are monitored by the following instruments:
 - TCP/IP: wait/io/socket/sql/server_tcpip_socket
 - UNIX socket file: wait/io/socket/sql/server_unix_socket
- When a listening socket detects a connection, the server transfers the connection to a new socket managed by a separate thread.
 - The instrument for the new connection thread is wait/io/socket/sql/client_connection.
- When the connection terminates, the row associated with it in `socket_instances` is deleted.



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

Identifying Connections in the socket_instances Table

```
mysql> SELECT * FROM socket_instances;
+-----+-----+-----+-----+-----+
| EVENT_NAME | OBJECT_ | THREAD_ID | SOCKET_ID | IP           | PORT | STATE |
+-----+-----+-----+-----+-----+
| INSTANCE_
| BEGIN
+-----+-----+-----+-----+-----+
| ....server_tcpip_socket | 65202432 |      1 |     17 | ::          | 3306 | ACTIVE |
| ....server_unix_socket | 65202752 |      1 |     18 |             | 0    | ACTIVE |
| ....client_connection  | 65244992 |   6458 |     41 | ::ffff:127.0.0.1 | 58640 | IDLE  |
| ....client_connection  | 65245312 |   6459 |     79 | ::ffff:127.0.0.1 | 58641 | IDLE  |
| ....client_connection  | 65245952 |   6461 |     83 | ::ffff:127.0.0.1 | 58643 | IDLE  |
| ....client_connection  | 65251392 |   6478 |     82 |             | 0    | IDLE  |
| ....client_connection  | 65251712 |   6479 |     89 |             | 0    | ACTIVE |
+-----+-----+-----+-----+-----+
7 rows in set (#.## sec)
```



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

The values of the **IP** and **PORT** columns for each row identifies the connection. This combination value is used in the **OBJECT_NAME** column of the **events_waits_xxx** tables, to identify the connection from which socket events derive:

- For the UNIX domain listener socket (`server_unix_socket`), the port is 0, and the IP value is empty.
- For client connections via the UNIX domain listener (`client_connection`), the port is 0, and the IP value is empty.
- For the TCP/IP server listener socket (`server_tcpip_socket`), the port is always the MySQL server port (by default 3306), and the IP value is always 0.0.0.0.
- For client connections via the TCP/IP listener (`client_connection`), the port is whatever the server assigns to the client, but never 0. The IP value is the IP value of the originating host (127.0.0.1 or ::1 for the local host)

Connection Tables

The users, hosts, and accounts tables store connection metrics.

- The users and hosts tables each include only user and host data, respectively.
- The accounts table includes both username and host data.
 - Example: Connections per account (username and host)

```
mysql> SELECT * FROM accounts;
+-----+-----+-----+
| USER | HOST      | CURRENT_CONNECTIONS | TOTAL_CONNECTIONS |
+-----+-----+-----+
| root | localhost |                 1 |                  15 |
| NULL | NULL      |                 18 |                  20 |
+-----+-----+-----+
2 rows in set (0.00 sec)
```



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

Retrieving Detailed Connection Information

Attribute tables provide more detailed information about individual connections:

- `session_connect_attrs`: Connection details for all sessions
- `session_account_connect_attrs`: Connection details for current session only

```
mysql> SELECT * FROM performance_schema.session_connect_attrs WHERE PROCESSLIST_ID = 5;
+-----+-----+-----+
| PROCESSLIST_ID | ATTR_NAME      | ATTR_VALUE        | ORDINAL_POSITION |
+-----+-----+-----+
|      5 | _runtime_version | 1.7.0_85          |          0         |
|      5 | _client_version | 5.1.36           |          1         |
|      5 | _client_name    | MySQL Connector Java |          2         |
|      5 | _client_license | commercial       |          3         |
|      5 | _runtime_vendor | Oracle Corporation |          4         |
+-----+-----+-----+
5 rows in set (0.00 sec)
```



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

Related to the connection tables are two connection attribute tables, `session_connect_attrs` and `session_account_connect_attrs`, which provide further information about individual connections, including the version of the client and details of the connector. The `ORDINAL_POSITION` column displays the order in which the attribute was added to the table for the given `PROCESSLIST_ID`.

Note: You can retrieve the `PROCESSLIST_ID` to identify a connection by querying the `sys.processlist` or `sys.session` views as described in the slide titled “Using `sys` as a Non-Blocking SHOW PROCESSLIST” later in this lesson.

Quiz



When a thread successfully locks a mutex, the Performance Schema writes to which of the following tables? (Select all that apply.)

- a. The `mutex_instances` table
- b. The `events_stages_current` table
- c. The `events_waits_current` table
- d. The `events_wait_history` table



ORACLE®

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

Answer: a, b, d.

Topics

- What is Performance Schema?
- Schema overview
- Configuration
- The Instance and Connection Tables
- **Querying event data**
- The sys schema
- MySQL Workbench performance tools



ORACLE®

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

Examining Raw Event Data: Example 1

Which SELECT queries did not use indexes?

```
mysql> SELECT SQL_TEXT FROM events_statements_history  
      -> WHERE NO_INDEX_USED != 0 AND EVENT_NAME='statement/sql/select';  
+-----+  
| SQL_TEXT |  
+-----+  
| ... |  
| select * from employees.employees where year(birth_date) < 1969 order by birth_date desc |  
| ... |  
+-----+
```



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

Examining Raw Event Data: Example 2

What were the five longest waits for a particular thread?

```
mysql> SELECT EVENT_ID, EVENT_NAME, TIMER_WAIT  
      FROM events_waits_history  
     WHERE THREAD_ID=33  
    ORDER BY TIMER_WAIT DESC LIMIT 5;
```

EVENT_ID	EVENT_NAME	TIMER_WAIT
8893	idle	1759049975000000
8876	idle	2302640520000000
8898	idle	1038258900000000
8903	idle	818539900000000
8872	wait/io/file/myisam/kfile	26902845

5 rows in set (#.# sec)

Timing information is
in picoseconds
(10^{-12})



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

You can use the `sys` schema formatting functions to present the timing information in a more human-readable format, as described in the slide titled “Using `sys` Functions to Format Output: Example” later in this lesson.

Examining Raw Event Data: Example 3

Which stage of statement processing is taking the longest for a particular thread?

```
mysql> SELECT EVENT_NAME, MAX(TIMER_WAIT)
      FROM events_stages_history
     WHERE THREAD_ID=37;
+-----+-----+
| EVENT_NAME | MAX(TIMER_WAIT) |
+-----+-----+
| stage/sql/end | 23192727000 |
+-----+-----+
1 row in set (#.## sec)
```

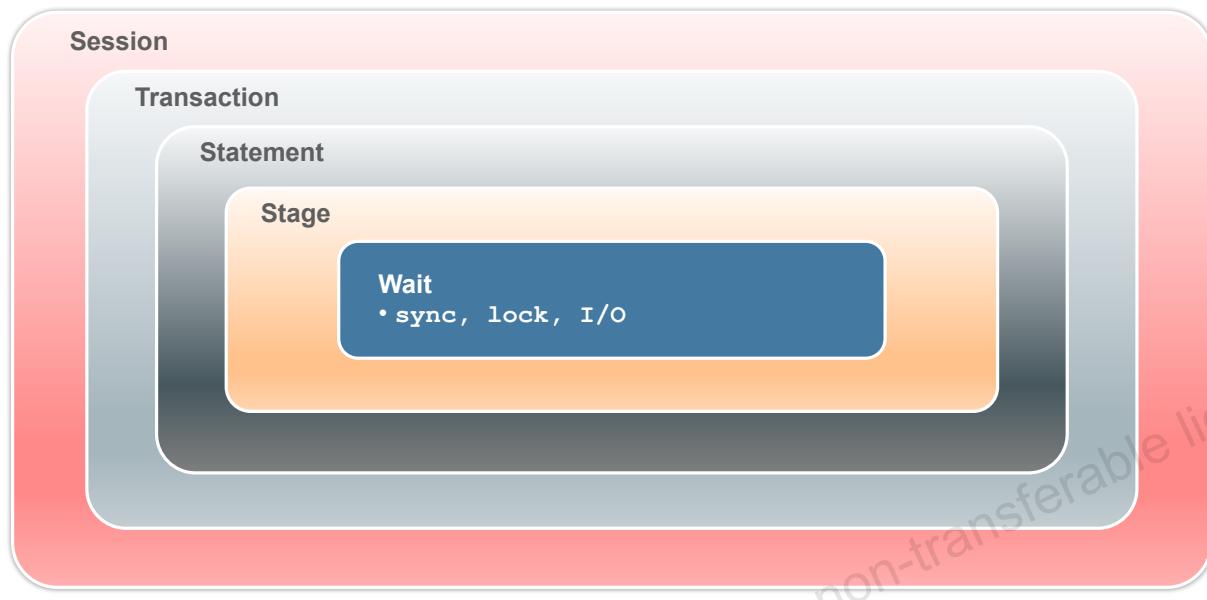
What timing units are being used for this event type?

```
mysql> SELECT * FROM setup_timers WHERE NAME='stage' \G
***** 1. row *****
  NAME: stage
  TIMER_NAME: NANOSECOND
1 row in set (#.## sec)
```

ORACLE

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

Event Hierarchy



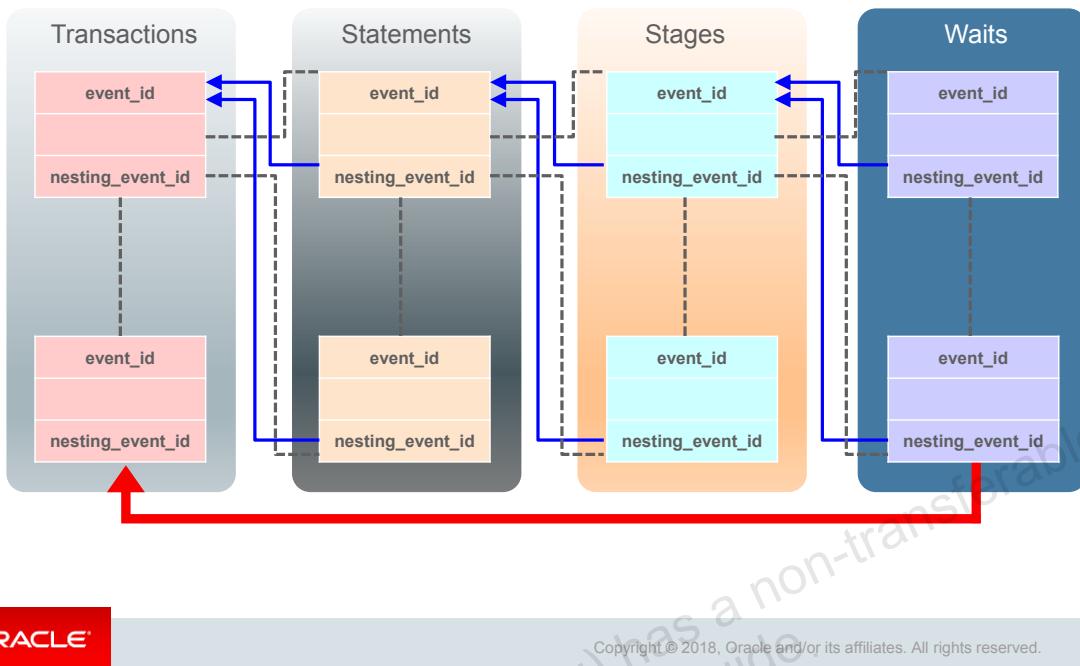
ORACLE®

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

Events form a hierarchy:

- A session might contain multiple transactions.
- A transaction might consist of multiple statements.
- Statements undergo multiple stages during their processing.
- Each processing stage can result in multiple wait events.

Nested Events



ORACLE®

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

In the Performance Schema, the non-summary event tables each have a column called `event_id` that uniquely identifies each event in the table. Each event table also has a `NESTING_EVENT_ID` column, which links to its parent event, and a `NESTING_EVENT_TYPE` (of WAIT, STAGE, STATEMENT, TRANSACTION, or SESSION) that defines the class of the event's parent. If both `NESTING_EVENT_ID` and `NESTING_EVENT_TYPE` are NULL for an event, then this event has no parent event but might be the parent of other nested events.

For example, when you execute a query within a transaction, transaction and statement events are raised. Executing the query results in several stage events, such as parsing the statement, opening a table, or performing a `filesort` operation. Each of these stages results in wait events for resources. All of these lower-level events can be related to their parent events via `NESTING_EVENT_ID`, right back to the generating statement, and the transaction it belongs to.

Nested Events (continued)

A lower-level event can be the parent of a higher-level event. For example, a `START TRANSACTION` statement creates a new transaction that in turn might become the parent of one or more statements:

```
mysql> (
    SELECT EVENT_ID, 'Transaction' AS EventType, '' AS SQL_TEXT,
           NESTING_EVENT_ID, NESTING_EVENT_TYPE
      FROM events_transactions_history
     WHERE THREAD_ID = 33
  ) UNION (
    SELECT EVENT_ID, 'Statement' AS EventType, SQL_TEXT,
           NESTING_EVENT_ID, NESTING_EVENT_TYPE
      FROM events_statements_history
     WHERE THREAD_ID = 33
)
   ORDER BY EVENT_ID;
```

EVENT_ID	EventType	SQL_TEXT	NESTING_EVENT_ID	NESTING_EVENT_TYPE
1	Statement	START TRANSACTION	NULL	NULL
2	Transaction		1	STATEMENT
3	Statement	UPDATE world.City	2	TRANSACTION
		SET Population = Population + 1		
		WHERE ID = 130		
4	Statement	COMMIT	2	TRANSACTION

4 rows in set (#.## sec)

Querying Nested Events

```
mysql> SELECT ... FROM events_stages_history_long WHERE THREAD_ID=3462536;
+-----+
| EVENT_ID | EVENT_NAME
+-----+
| 1413    | stage/sql/init
| 1417    | stage/sql/checking permissions
| ...
+-----+
```

NESTING_EVENT_ID	NESTING_EVENT_TYPE
1412	STATEMENT
1412	STATEMENT

```
mysql> SELECT ... FROM events_statements_history_long WHERE EVENT_ID=1412;
```

```
+-----+
| THREAD_ID | EVENT_ID | EVENT_NAME           | SQL_TEXT
+-----+
| 3462536  |     1412 | statement/sql/select | select * from...
| ...
+-----+
```

ORACLE

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

Summary Tables

- Aggregate raw event data over time, for wait, stage, statement, transaction, object, file I/O, table I/O and lock waits, socket, and memory events, and status variables
- Present the raw event data in summary form, making it easier to query. For example, the statement summary tables include:
 - `events_statements_summary_by_account_by_event_name`
 - `events_statements_summary_by_digest`
 - `events_statements_summary_by_host_by_event_name`
 - `events_statements_summary_by_program` (stored procedures and functions, triggers, and events)
 - `events_statements_summary_by_thread_by_event_name`
 - `events_statements_summary_by_user_by_event_name`
 - `events_statements_summary_global_by_event_name`
 - `prepared_statements_instances` (prepared statement instances and statistics)



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

Prepared Statements

Prepared statement instrumentation enables inspection of prepared statements in the server and writes information to the `prepared_statement_instances` table:

- Statement preparation: Adds a new row to the table
- Prepared statement execution: Updates the corresponding row in the table
- Prepared statement deallocation: Deletes the associated table row

The `prepared_statement_instances` table includes the following columns

- `STATEMENT_ID`, `STATEMENT_NAME`, and `OWNER_*`: Identifying information
- `SQL_TEXT`: The prepared statement text, with ‘?’ placeholders for values
- `COUNT_EXECUTE`, `SUM_AVG_TIMER_EXECUTE`, `MIN_MAX_TIMER_EXECUTE`: Aggregated statistics for executions of the prepared statement
- `TIMER_PREPARE`: The time it took to prepare the statement
- `COUNT_REPREPARE`: The number of times the statement was reprepared internally



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

The `prepared_statement_instances` table is technically an instance table, but it contains more much more information than the other instance tables. For that reason it is usually considered as a summary table, but it is atypical. Although the table does aggregate data per prepared statement instance, when `DEALLOCATE PREPARE` is executed or the session terminates then the instance disappears, so it does not retain its data like the other summary tables.

`OWNER_OBJECT_TYPE`, `OWNER_OBJECT_SCHEMA`, `OWNER_OBJECT_NAME`: For a prepared statement created by a client session, these columns are `NULL`. For a prepared statement created by a stored program, these columns point to the stored program. A typical user error is forgetting to deallocate prepared statements. These columns can be used to find stored programs that leak prepared statements:

```
SELECT OWNER_OBJECT_TYPE, OWNER_OBJECT_SCHEMA, OWNER_OBJECT_NAME,  
      STATEMENT_NAME, SQL_TEXT  
   FROM performance_schema.prepared_statements_instances  
 WHERE OWNER_OBJECT_TYPE IS NOT NULL;
```

`COUNT_REPREPARE`: The number of times the statement was reprepared internally. There is no timing information for repreparation because this step is counted as part of statement execution, not as a separate operation.

Note: The instance tables are covered in this lesson in the slide titled “Instance Tables.”

Retrieving Statement Summary Information: Example

```
mysql> SELECT * FROM events_statements_summary_by_digest\G
***** 1. row *****
SCHEMA_NAME: world_innodb
DIGEST: 497c7...
DIGEST_TEXT: SELECT NAME
FROM `Country` WHERE CODE IN (
SELECT CODE FROM `CountryLanguage`
WHERE `population` > ? )
COUNT_STAR: 1
SUM_TIMER_WAIT: 84878009000
MIN_TIMER_WAIT: 84878009000
AVG_TIMER_WAIT: 84878009000
MAX_TIMER_WAIT: 84878009000
SUM_LOCK_TIME: 1054000000
SUM_ERRORS: 0
SUM_WARNINGS: 0
SUM_ROWS_AFFECTED: 0
SUM_ROWS_SENT: 163
SUM_ROWS_EXAMINED: 239
SUM_CREATED_TMP_DISK_TABLES: 0
SUM_CREATED_TMP_TABLES: 0
SUM_SELECT_FULL_JOIN: 1
SUM_SELECT_FULL_RANGE_JOIN: 0
SUM_SELECT_RANGE: 0
SUM_SELECT_RANGE_CHECK: 0
SUM_SELECT_SCAN: 1
SUM_SORT_MERGE_PASSES: 0
SUM_SORT_RANGE: 0
SUM_SORT_ROWS: 0
SUM_SORT_SCAN: 0
SUM_NO_INDEX_USED: 1
SUM_NO_GOOD_INDEX_USED: 0
FIRST_SEEN: 2017-06-07
12:17:54
LAST_SEEN: 2017-22-07
12:17:54
...
```

Each row summarizes events by SCHEMA_NAME and DIGEST value

DIGEST_TEXT contains the corresponding normalized statement digest text

The remaining columns contain aggregated values from the raw event data

ORACLE

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

The `events_statements_summary_by_digest` shown in the slide has `SCHEMA_NAME` and `DIGEST` columns. Each row summarizes events per schema and digest value. The `DIGEST_TEXT` column contains the corresponding normalized statement digest text, but is neither a grouping nor a summary column. The `COUNT_STAR` field is equivalent to `COUNT(*)`.

The `events_statements_summary_by_digest` table is particularly helpful for analyzing statement performance. To use it, enable the `statements_digest` consumer in the `setup_consumers` table. The MySQL Enterprise Monitor tool uses `events_statements_summary_by_digest` in its Query Analyzer.

The `DIGEST` column contains a hash of the normalized version of a statement. To generate this digest, Performance Schema:

- Removes comments
- Adjusts whitespace so that only single space characters separate tokens
- Replaces literal values with a ? (question mark symbol) as a placeholder
- Hashes the resulting statement

The hash identifies statements that are structurally and effectively similar so that the Performance Schema can aggregate them. This enables you to analyze such statements in a group, even if there are small differences between statements.

Note: The maximum number of rows in the table is autosized at server startup. To set this maximum explicitly, set the `performance_schema_digests_size` system variable at server startup.

Retrieving Wait Summary Information: Example

```
mysql> SELECT * FROM events_waits_summary_global_by_event_name\G
...
***** 6. row *****
EVENT_NAME: wait/synch/mutex/sql/BINARY_LOG::LOCK_index
COUNT_STAR: 8
SUM_TIMER_WAIT: 2119302
MIN_TIMER_WAIT: 196092
AVG_TIMER_WAIT: 264912
MAX_TIMER_WAIT: 569421
...
***** 9. row *****
EVENT_NAME: wait/synch/mutex/sql/hash_filo::lock
COUNT_STAR: 69
SUM_TIMER_WAIT: 16848828
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 244185
MAX_TIMER_WAIT: 735345
...
```

The remaining columns contain aggregated values from the raw event data

Each row summarizes events by EVENT_NAME



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

In the `events_waits_summary_global_by_event_name` summary table, each row summarizes events for a given event name. An instrument might be used to create multiple instances of the instrumented object. For example, if there is an instrument for a mutex that is created for each connection, there are as many instances as there are connections. The summary row for the instrument summarizes over all these instances.

Retrieving Memory Summary Information: Example

```
mysql> SELECT * FROM memory_summary_global_by_event_name
      WHERE EVENT_NAME = 'memory/sql/TABLE'\G
*****
1. row ****
EVENT_NAME: memory/sql/TABLE
COUNT_ALLOC: 1381
COUNT_FREE: 924
SUM_NUMBER_OF_BYTES_ALLOC: 2059873
SUM_NUMBER_OF_BYTES_FREE: 1407432
LOW_COUNT_USED: 0
CURRENT_COUNT_USED: 457
HIGH_COUNT_USED: 461
LOW_NUMBER_OF_BYTES_USED: 0
CURRENT_NUMBER_OF_BYTES_USED: 652441
HIGH_NUMBER_OF_BYTES_USED: 669269...
```

An event raised by the
memory/sql/TABLE
instrument

Summary data for the
event



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

Each memory summary table has one or more grouping columns to indicate how the table aggregates events. Event names refer to names of event instruments in the `setup_instruments` table:

- The `memory_summary_global_by_event_name` table shown in the example in the slide has an `EVENT_NAME` column. Each row summarizes events for a given event name.
- `memory_summary_by_account_by_event_name` has `USER`, `HOST`, and `EVENT_NAME` columns. Each row summarizes events for a given account (user and host combination) and event name.
- `memory_summary_by_host_by_event_name` has `HOST` and `EVENT_NAME` columns. Each row summarizes events for a given host and event name.
- `memory_summary_by_thread_by_event_name` has `THREAD_ID` and `EVENT_NAME` columns. Each row summarizes events for a given thread and event name.
- `memory_summary_by_user_by_event_name` has `USER` and `EVENT_NAME` columns. Each row summarizes events for a given user and event name.

Each memory summary table has the following summary columns containing aggregated values:

- `COUNT_ALLOC`, `COUNT_FREE`: The aggregated numbers of calls to memory-allocation and memory-free functions.
- `SUM_NUMBER_OF_BYTES_ALLOC`, `SUM_NUMBER_OF_BYTES_FREE`: The aggregated sizes of allocated and freed memory blocks.

- CURRENT_COUNT_USED

The aggregated number of currently allocated blocks that have not been freed yet. This is a convenience column, equal to (COUNT_ALLOC – COUNT_FREE)

- CURRENT_NUMBER_OF_BYTES_USED

The aggregated size of currently allocated memory blocks that have not been freed yet.

This is a convenience column, equal to:

$$(\text{SUM_NUMBER_OF_BYTES_ALLOC} - \text{SUM_NUMBER_OF_BYTES_FREE})$$

- LOW_COUNT_USED, HIGH_COUNT_USED

The low and high water marks corresponding to the CURRENT_COUNT_USED column.

- LOW_NUMBER_OF_BYTES_USED, HIGH_NUMBER_OF_BYTES_USED

The low and high water marks corresponding to the CURRENT_NUMBER_OF_BYTES_USED column.

Topics

- What is Performance Schema?
- Schema overview
- Configuration
- The Instance and Connection Tables
- Querying event data
- **The sys schema**
- MySQL Workbench performance tools



ORACLE®

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

sys Schema

- Simplifies access to Performance Schema information
- Includes the following main objects:
 - **Views:** Summarize Performance Schema data so that it is easy to understand
 - Most views are in pairs: `view_name` (human-readable output) and `$x.view_name` (for use and further processing by tools)
 - **Stored procedures:** Perform operations such as the configuration of Performance Schema and generation of reports
 - **Stored functions:** Query Performance Schema configuration and format query output. These also include functions that are unrelated to the Performance Schema.
- Can be used only by an account with the required privileges on `sys` and `performance_schema`
- Works well with the default Performance Schema configuration settings.
 - You might need to enable additional instruments and consumers for certain `sys` features.



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

Setting sys Schema Privileges

User accounts need the following privileges to work with the sys schema:

- SELECT on all sys tables and views, and all tables in the Performance Schema that are accessed by sys objects
- EXECUTE on all sys stored routines and UPDATE on any Performance Schema tables they modify
- INSERT and UPDATE on the sys_config table, if you want to make persistent changes to the sys schema configuration.
 - You can make session-specific configuration changes without these privileges.
- UPDATE on Performance Schema tables that are modified by sys objects



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

Note: For most use cases, the default sys configuration settings are adequate.

sys Views

```
mysql> SELECT table_name AS 'sys View' FROM information_schema.tables  
      -> WHERE table_schema='sys' AND table_name NOT LIKE 'x$%';  
+-----+  
| sys View |  
+-----+  
| host_summary |  
| host_summary_by_file_io |  
| host_summary_by_file_io_type |  
| host_summary_by_stages |  
| host_summary_by_statement_latency |  
| host_summary_by_statement_type |  
| innodb_buffer_stats_by_schema |  
| innodb_buffer_stats_by_table |  
| innodb_lock_waits |  
+-----+  
| io_by_thread_by_latency |  
| io_global_by_file_by_bytes |  
| io_global_by_file_by_latency |  
| io_global_by_wait_by_bytes |  
| io_global_by_wait_by_latency |  
| latest_file_io |  
| ... |  
+-----+
```



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

The output in the slide shows a list of the views in the `sys` schema. Most of the views have a counterpart named `x$view_name` that contains unformatted data suitable for tooling. These are omitted from the output.

The views aggregate event data from the Performance Schema tables for common monitoring scenarios, making them easier to query and understand.

sys Views

```
...  
| memory_by_host_by_current_bytes      | | schema_table_statistics  
| memory_by_thread_by_current_bytes    | | schema_table_statistics_with_buffer  
| memory_by_user_by_current_bytes     | | schema_tables_with_full_table_scans  
| memory_global_by_current_bytes      | | schema_unused_indexes  
| memory_global_total                 | | session  
| metrics                            | | session_ssl_status  
| processlist                         | | statement_analysis  
| ps_check_lost_instrumentation      | | statements_with_errors_or_warnings  
| schema_auto_increment_columns       | | statements_with_full_table_scans  
| schema_index_statistics             | | statements_with_runtimes_in_95th_percentile  
| schema_object_overview              | | statements_with_sorting  
| schema_redundant_indexes           | | statements_with_temp_tables  
| schema_table_lock_waits             | | sys_config  
...  
...
```



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

Note: Although sys_config is shown in this output, it is actually a base table.

sys Views

```
...
| user_summary
| user_summary_by_file_io
| user_summary_by_file_io_type
| user_summary_by_stages
| user_summary_by_statement_latency
| user_summary_by_statement_type
| version
| wait_classes_global_by_avg_latency
| wait_classes_global_by_latency
| waits_by_host_by_latency
| waits_by_user_by_latency
| waits_global_by_latency
+-----+
53 rows in set (#.## sec)
```



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

Using sys for Statement Analysis: Example

```
mysql> SELECT * FROM statement_analysis LIMIT 1\G
***** 1. row ****
    query: SELECT * FROM `schema_object_o ... MA` , `information_schema` ...
          db: sys
      full_scan: *
    exec_count: 2
     err_count: 0
    warn_count: 0
total_latency: 16.75 s
max_latency: 16.57 s
avg_latency: 8.38 s
lock_latency: 16.69 s
  rows_sent: 84
rows_sent_avg: 42
rows_examined: 20012
rows_examined_avg: 10006
  rows_affected: 0
rows_affected_avg: 0
      tmp_tables: 378
tmp_disk_tables: 66
  rows_sorted: 168
sort_merge_passes: 0
        digest: 54f9bd520f0bbf15db0c2ed93386bec9
first_seen: 2014-03-07 13:13:41
last_seen: 2014-03-07 13:13:48
```



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

The statement analysis views provide some of the information stored in the events_statements_summary_by_digest table of performance_schema, but are much easier to use and extract information from.

Using sys to Identify Queries with Full Table Scans: Example

```
mysql> SELECT * FROM statements_with_full_table_scans LIMIT 1\G
***** 1. row ****
      query: SELECT * FROM `schema_tables_w ... ex_usage` .
`COUNT_READ` DESC
      db: sys
  exec_count: 1
total_latency: 88.20 ms
no_index_used_count: 1
no_good_index_used_count: 0
no_index_used_pct: 100
  rows_sent: 0
rows_examined: 1501
  rows_sent_avg: 0
rows_examined_avg: 1501
  first_seen: 2014-03-07 13:58:20
  last_seen: 2014-03-07 13:58:20
    digest: 64baecd5c1e1e1651a6b92e55442a288
```



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

The `statements_with_full_table_scans` view lists all normalized statements that have performed a full table scan. The rows are ordered by the descending percentage of times a full scan occurred, and then by the statement latency.

Using sys to Identify the Slowest Queries: Example

```
mysql> SELECT * FROM statements_with_runtimes_in_95th_percentile\G
***** 1. row ****
    query: SELECT * FROM `schema_object_o ... MA` , `information_schema` ...
      db: sys
  full_scan: *
  exec_count: 2
  err_count: 0
  warn_count: 0
total_latency: 16.75 s
max_latency: 16.57 s
avg_latency: 8.38 s
  rows_sent: 84
rows_sent_avg: 42
  rows_examined: 20012
rows_examined_avg: 10006
  first_seen: 2014-03-07 13:13:41
  last_seen: 2014-03-07 13:13:48
    digest: 54f9bd520f0bbf15db0c2ed93386bec9
...
...
```



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

The `statements_with_runtimes_in_95th_percentile` view lists the slowest queries.

Other useful statement analysis views include:

- **`statements_with_sorting`**: Lists all normalized statements that used sorting, ordered by `sort_merge_passes`, `sort_scans`, and `sort_rows`, all in descending order
- **`statements_with_temp_tables`**: Lists all normalized statements that use temporary tables. The statements that created the largest number of on-disk temporary tables appear first, followed by those that created the largest number of tables in memory.
- **`statements_with_errors_or_warnings`**: Lists all normalized statements that raised errors or warnings

Using sys to Identify Unused Indexes: Example

- Querying the sys.schema_unused_indexes view:

```
mysql> SELECT * from sys.schema_unused_indexes LIMIT 5;
+-----+-----+-----+
| object_schema | object_name | index_name |
+-----+-----+-----+
| employees     | departments | dept_name   |
| employees     | dept_emp    | dept_no     |
| employees     | dept_manager | dept_no    |
| perfct       | city        | CountryCode |
| perfct       | city_huge   | CountryCode |
+-----+-----+-----+
5 rows in set (0.00 sec)
```

- Equivalent performance_schema query:

```
SELECT object_schema, object_name, index_name
FROM performance_schema.table_io_waits_summary_by_index_usage
WHERE index_name IS NOT NULL AND count_star = 0 AND object_schema != 'mysql'
AND index_name != 'PRIMARY' ORDER BY object_schema, object_name;
```



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

The example in the slide demonstrates how much easier sys views are to query than the underlying performance_schema tables. Other useful schema statistics views include:

- **schema_table_statistics**: Useful table access metrics, ordered by the total wait time descending. The associated `schema_table_statistics_with_buffer` table contains InnoDB buffer pool usage statistics.
- **schema_index_statistics**: Listed in descending order of total wait time, where the top indexes are the most contended
- **schema_tables_with_full_table_scans**: Lists all tables where every record is accessed to complete a query, in descending order by the numbers of rows scanned

Note: Just because sys identifies an index as being “unused”, it does not automatically mean it is safe to drop. For example, it might be required for constraints or it may simply not have been used yet (because the data resets when MySQL restarts). For example an index may be required by a monthly report, so until that report has been created, the index will show up as unused. Always investigate further before dropping indexes.

Using sys as a Non-Blocking SHOW PROCESSLIST Example

```
mysql> SELECT * from sys.processlist WHERE conn_id IS NOT NULL
-> AND command != 'daemon' AND conn_id != connection_id()\G
***** 1 . row *****
    thd_id: 6458          last_statement: NULL
    conn_id: 6433          last_statement_latency: 25.58 us
      user: root@localhost  current_memory: 0 bytes
        db: sys              last_wait: idle
    command: Sleep          last_wait_latency: 10.00 m
      state: NULL           source: socket_connection.cc:63
      time: 411             trx_latency: NULL
  current_statement: NULL   trx_state: NULL
statement_latency: NULL    trx_autocommit: NULL
    progress: NULL         pid: 26877
  lock_latency: 0 ps       program_name: NULL
rows_examined: 0
rows_sent: 0
rows_affected: 0
tmp_tables: 0
tmp_disk_tables: 0
full_scan: NO
...
...
```



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

The sys.processlist view provides an alternative, non-blocking processlist view to replace SHOW PROCESSLIST. It provides more information than SHOW PROCESSLIST and performs less locking.

The sys.processlist view includes both background threads and user connections. If you want to see only user connections, query sys.session instead.

sys Stored Routines

```
mysql> SELECT SPECIFIC_NAME FROM INFORMATION_SCHEMA.ROUTINES  
      -> WHERE ROUTINE_SCHEMA='sys';  
  
+-----+-----+  
| SPECIFIC_NAME          | ps_is_account_enabled           |  
+-----+-----+  
| create_synonym_db      | ps_is_instrument_default_enabled |  
| diagnostics            | ps_is_instrument_default_timed   |  
| execute_prepared_stmt  | ps_is_thread_instrumented       |  
| extract_schema_from_file_name | ps_setup_disable_background_threads |  
| extract_table_from_file_name | ps_setup_disable_consumer         |  
| format_bytes            | ps_setup_disable_instrument      |  
| format_path              | ps_setup_disable_thread          |  
| format_statement         | ps_setup_enable_background_threads |  
| format_time              | ps_setup_enable_consumer          |  
| list_add                | ps_setup_enable_instrument       |  
| list_drop                | ps_setup_enable_thread           |  
+-----+-----+  
...  
...
```



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

The output in the slide shows a list of the stored routines in the `sys` schema. The routines in italics are stored functions, and the others are stored procedures.

Stored routines perform many useful operations such as configuring Performance Schema and formatting output that would otherwise involve writing complex SQL.

sys Stored Routines

```
...
| ps_setup_reload_saved          | | ps_thread_stack
| ps_setup_reset_to_default     | | ps_thread_trx_info
| ps_setup_save                  | | ps_trace_statement_digest
| ps_setup_show_disabled         | | ps_trace_thread
| ps_setup_show_disabled_consumers | | ps_truncate_all_tables
| ps_setup_show_disabled_instruments | | quote_identifier
| ps_setup_show_enabled          | | statement_performance_analyzer
| ps_setup_show_enabled_consumers | | sys_get_config
| ps_setup_show_enabled_instruments | | table_exists
| ps_statement_avg_latency_histogram | | version_major
| ps_thread_account              | | version_minor
| ps_thread_id                   | | version_patch
+-----+
48 rows in set (#.## sec)
```



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

Configuring Performance Schema by Using sys: Example

- You can use sys stored routines to configure the Performance Schema.
 - For example, to use the sys schema effectively, you must enable the wait, stage, and statement instruments and their associated history consumers (events_type_history and events_type_history_long):

```
CALL sys.ps_setup_enable_instrument('wait');
CALL sys.ps_setup_enable_instrument('stage');
CALL sys.ps_setup_enable_instrument('statement');
CALL sys.ps_setup_enable_consumer('current');
CALL sys.ps_setup_enable_consumer('history_long');
```

- You can check whether a particular object is enabled by calling a sys stored function:

```
mysql> SELECT ps_is_consumer_enabled('thread_instrumentation');
+-----+
| ps_is_consumer_enabled('thread_instrumentation') |
+-----+
| YES
+-----+
1 row in set (#.## sec)
```



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

The example in the slide demonstrates using the ps_setup_enable* stored procedures to enable instruments and consumers and the ps_is_*_enabled functions to check whether a given instrument or consumer is enabled.

You can use the ps_setup_disable* routines to disable Performance Schema configuration options and the ps_show_enabled* or ps_show_disabled* routines to list enabled and disabled configuration options, respectively.

Note: The ps_is_consumer_enabled() function considers the entire consumer hierarchy, so it may return NO for a consumer even if you have just enabled it, as shown in the following example:

```
mysql> SELECT *, sys.ps_is_consumer_enabled(NAME) AS 'ENABLED (sys)'
   -> FROM setup_consumers
   -> WHERE NAME LIKE 'events_transactions_%';
+-----+-----+-----+
| NAME           | ENABLED | ENABLED (sys) |
+-----+-----+-----+
| events_transactions_current | NO      | NO          |
| events_transactions_history | NO      | NO          |
| events_transactions_history_long | NO     | NO          |
+-----+-----+-----+
3 rows in set (#.## sec)
```

```
mysql> UPDATE setup_consumers SET ENABLED = 'YES'
      -> WHERE NAME = 'events_transactions_history';
Query OK, 1 row affected (#.## sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT *, sys.ps_is_consumer_enabled(NAME) AS 'ENABLED (sys)'
      -> FROM setup_consumers WHERE NAME LIKE 'events_transactions_%';
+-----+-----+
| NAME           | ENABLED | ENABLED (sys) |
+-----+-----+
| events_transactions_current | NO      | NO          |
| events_transactions_history   | YES     | NO          |
| events_transactions_history_long | NO      | NO          |
+-----+-----+
3 rows in set (0.00 sec)

mysql> UPDATE setup_consumers SET ENABLED = 'YES'
      -> WHERE NAME = 'events_transactions_current';
Query OK, 1 row affected (#.## sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT *, sys.ps_is_consumer_enabled(NAME) AS 'ENABLED (sys)'
      -> FROM setup_consumers WHERE NAME LIKE 'events_transactions_%';
+-----+-----+
| NAME           | ENABLED | ENABLED (sys) |
+-----+-----+
| events_transactions_current | YES     | YES         |
| events_transactions_history   | YES     | YES         |
| events_transactions_history_long | NO      | NO          |
+-----+-----+
3 rows in set (#.## sec)
```

Using sys Functions to Format Output: Example

- `format_time()` / `format_bytes()`: Makes raw timing output easier to read
 - Example (picoseconds to HH:MM:SS):

```
mysql> SELECT format_time(342342342342345) AS TidyTime;
+-----+
| TidyTime |
+-----+
| 00:05:42 |
+-----+
1 row in set (#.## sec)
```

- `format_statement()` / `format_path()`: Compresses data for display at the command line
- `extract_[schema | table]_from_file_name()`: Extracts the schema name from a raw file path



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

Quiz

Q

Which of the following statements about the `sys` schema are false? (Select all that apply.)

- a. It requires `SELECT` and `UPDATE` privileges on certain Performance Schema tables.
- b. It contains a selection of temporary tables that aggregate Performance Schema statistics.
- c. It provides views of some data that are unformatted for tooling purposes.
- d. You can configure it by using MySQL Workbench.



ORACLE

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

Answer: b, d

Topics

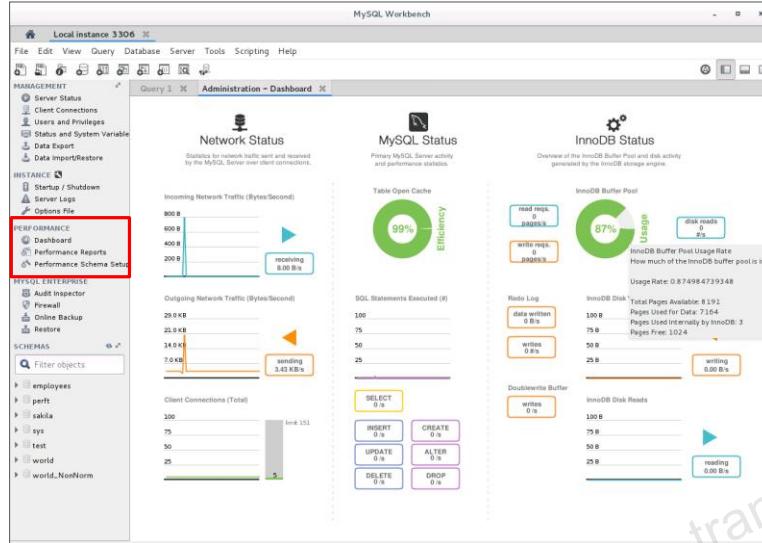
- What is Performance Schema?
- Schema overview
- Configuration
- The Instance and Connection Tables
- Querying event data
- The sys schema
- MySQL Workbench performance tools



ORACLE®

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

MySQL Workbench Performance Dashboard



ORACLE®

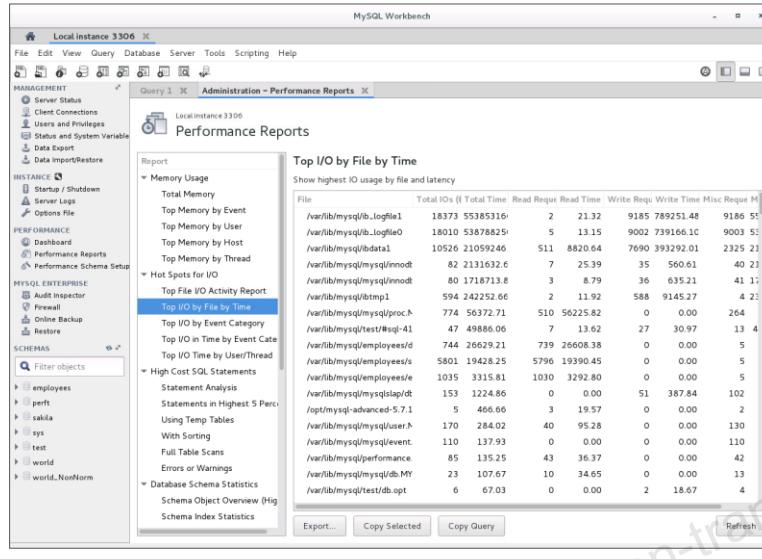
Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

MySQL Workbench offers a graphical user interface for working with Performance Schema.

The Dashboard shown in the slide displays the following information:

- **Network Status:** Shows the statistics for network traffic sent and received by the MySQL server over client connections. Data points include the Incoming Network Traffic, Outgoing Network Traffic, and Client Connections.
- **MySQL Status:** Highlights the primary MySQL server activity and performance statistics. Data points include the Table Open Cache efficiency, SQL Statements Executed, and counts (per second) for SELECT, INSERT, UPDATE, DELETE, CREATE, ALTER, and DROP statements.
- **InnoDB Status:** Provides an overview of the InnoDB Buffer Pool and disk activity that is generated by the InnoDB storage engine. Data points are separated into usage, writes, and reads statistics.

MySQL Workbench Performance Reports

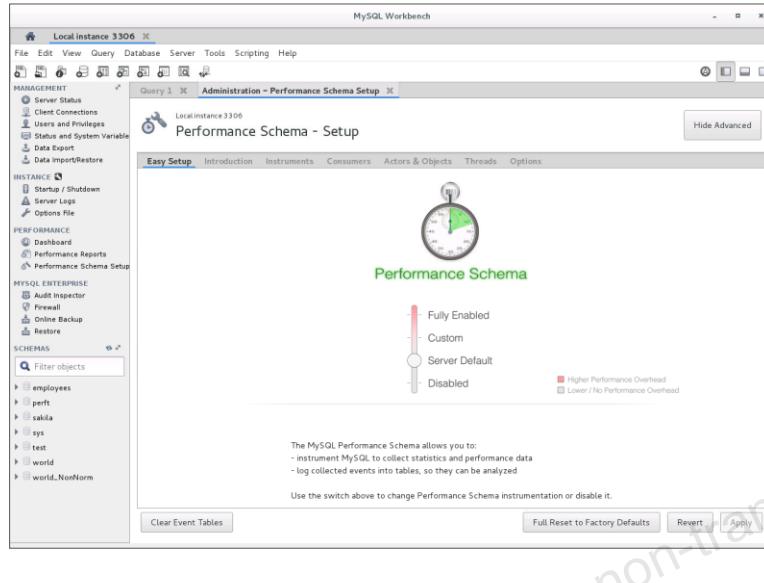


ORACLE®

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

MySQL Workbench provides a comprehensive selection of reports. The report shown in the slide is “Top I/O by File by Time,” which displays the highest I/O usage by file and latency. The performance reports depend on the `sys` schema. The `sys` schema is bundled with MySQL Server 5.7 and later, and MySQL Workbench uses that version. However, for MySQL Server 5.6, Workbench installs its own bundled version of the `sys` schema.

MySQL Workbench Performance Schema Setup



ORACLE®

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

The “Performance Schema – Setup” page provides a GUI for configuring and fine-tuning Performance Schema instrumentation. Initially, this loads an Easy Setup page that is adequate for most users. Slide the Performance Schema Full Enabled slider to YES to enable all available Performance Schema instruments.

Note: Enabling all instruments might incur a significant overhead. Test before enabling all instruments on a production system.

Summary



In this lesson, you should have learned how to:

- Describe the general structure of the Performance Schema
- Configure the Performance Schema to collect desired performance statistics
- Query the Performance Schema to retrieve statistics of interest
- Describe and use the `sys` schema
- Configure and use the Performance Schema in MySQL Workbench

ORACLE®

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

Practices

- 4-1: Examining Performance Schema Configuration
- 4-2: Querying the Performance Schema
- 4-3: Using `sys` to Work with the Performance Schema
- 4-4: Using MySQL Workbench for Performance Schema Configuration, Monitoring, and Reporting



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

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

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

5

General Server Tuning



ORACLE®

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

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

Topics

- How MySQL uses memory
- Calculating maximum MySQL memory usage
- Managing connections
- Tuning threads



ORACLE®

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

Objectives



After completing this lesson, you should be able to:

- Describe the components of the MySQL server architecture that directly affect performance
- Explain how the MySQL server uses memory
- List the system and status variables that relate to:
 - Memory usage
 - Connections to the MySQL server
 - Thread behavior
- Describe the benefits of using the MySQL Enterprise Thread Pool plugin

ORACLE®

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

Topics

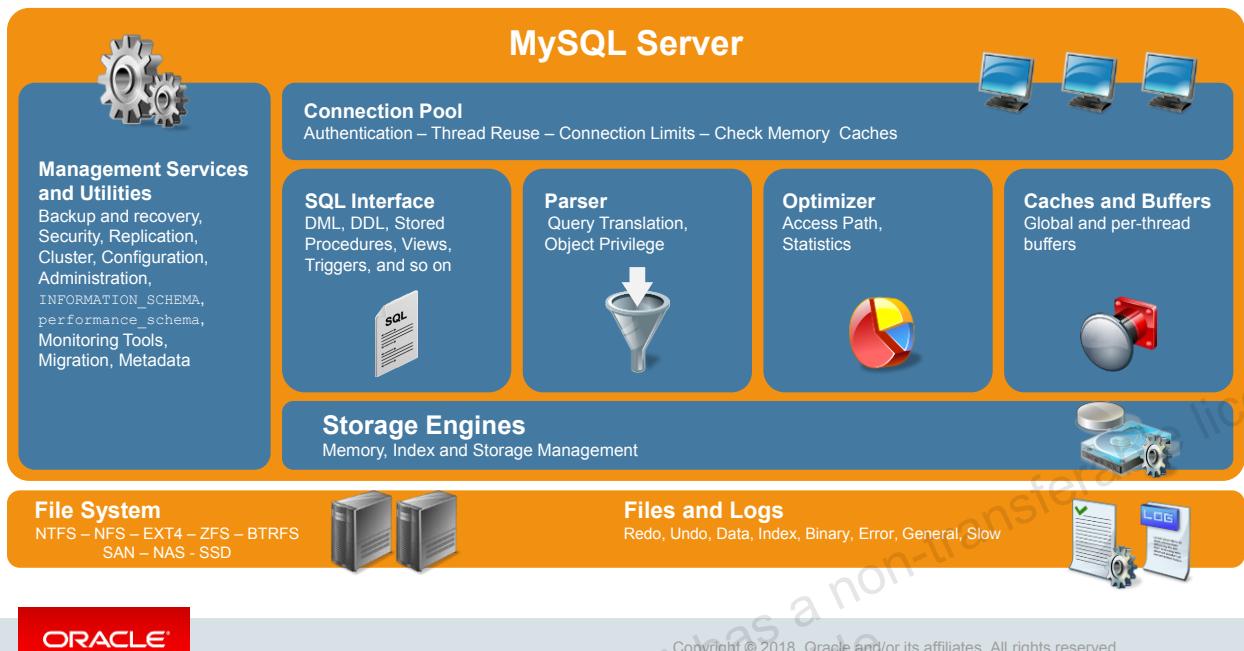
- How MySQL uses memory
 - Calculating maximum MySQL memory usage
 - Managing connections
 - Tuning threads



ORACLE®

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

Major Components of the MySQL Server



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

The slide shows the following MySQL server elements:

- Connection pool
- SQL interface
- Parser
- Optimizer
- Caches and buffers
- Storage engines
- File system
- Files and logs
- Management services and utilities

Quiz



Which of the following is *not* a function of the MySQL server connection pool?

- a. Authentication
- b. Connection limits
- c. Query translations
- d. Thread reuse



ORACLE®

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

Answer: c

MySQL Memory Usage

- MySQL needs memory for:
 - Managing its internal processes
 - You cannot change how MySQL allocates memory internally.
 - Maintaining client connections
 - Caching and buffering items in memory to avoid disk access
- Consider the following before re-assigning memory to the MySQL server:
 - The amount of physical RAM on the server host
 - The memory requirements of the operating system
 - The operating system architecture:
 - For example, 32-bit architectures limit maximum memory per process.
 - Other software running on the server host
 - Including periodic jobs



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

Although you cannot change how the MySQL server allocates memory to many of its internal processes, they are highly optimized and require minimal amounts of memory. Generally, you want to maximize the available memory for client connections and MySQL server caches and buffers to fit as much data as possible into memory and avoid disk I/O, which is considerably slower. However, you must be aware of the other processes running on the MySQL server host.

Tuning the MySQL Server

Change server behavior by changing the values of server system variables:

- System variables can be assigned values:
 - At server startup by using command-line options or a server configuration file
 - Dynamically at run time by using the `SET` command
 - Not all server variables can be changed dynamically.
 - Changes to session variables are lost when the session terminates.
- System variables have different scopes:
 - Global variables: Affect all connections.
 - Session variables: Affect only the current connection.
 - Many have dual scope.
 - You can set different values globally and per session.



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

The MySQL server is provided with default parameters that are suitable for many installations. In some cases, a parameter has a fixed default value. In other cases, the server calculates the default parameter value at startup using a formula based on other related parameters or server host configuration. For example, the setting for `thread_cache_size` is based on the value of `max_connections`.

Setting a global variable requires the `SUPER` privilege and affects the operation of all clients that connect from that time on. Setting the session variable affects only the current client.

You can set many variables dynamically at run time by using the `SET` command. For example, you can resize the InnoDB buffer pool without restarting the server by changing the value of `innodb_buffer_pool_size`.

Sometimes it makes sense to change a parameter for a specific operation within a session. You can revert to the default value after the operation has completed by using the `DEFAULT` keyword. For example:

```
SET @@session.sort_buffer_size = [new value];
/* Perform the query */
SET @@session.sort_buffer_size = DEFAULT;
```

CPU and I/O Saturation

- The main hardware factors affecting server performance are CPU and I/O saturation, when requests arrive quicker than they can be handled.
 - This is known as *saturation*.
- When most data fits in memory, or can be retrieved quickly from disk, the CPU can become saturated.
 - Consider faster, or more, CPUs.
 - Sequential disk access is faster than random disk access.
- When the server requires more data than can fit into memory, it leads to I/O saturation.



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

Sequential operations are faster than random access operations, whether these occur in memory or via disk I/O. Because of this, sequential operations do not benefit as much from caching, unless the entire dataset can fit into memory. However, if your system is doing lots of random access reads and writes, then you will benefit by adding more memory so that these operations can be buffered.

In an ideal environment, all your data would fit into memory, so that when the caches are “warmed up,” there is little or no disk I/O.

Global Buffers

MySQL allocates memory to the global buffers during the server startup process and the buffers hold onto this memory until the server process ends. Memory is allocated regardless of the number of connections.

Examples:

- `table_open_cache`: The number of open tables for all threads
- `innodb_buffer_pool_size`: The amount of memory available for caching InnoDB data and indexes
- `innodb_log_buffer_size`: The size of the buffer that InnoDB uses to write to the log files on disk



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

There are several global buffers in MySQL, and the slide shows only some of the most important ones that you might want to tune on a system using the default InnoDB storage engine.

If your schemas use the MyISAM storage engine, then the size of the key cache (specified by `key_buffer_size`) is also important. To minimize disk I/O, MyISAM caches the most frequently accessed tables in memory. The keys are stored in a special structure called the key cache. Data is cached in the OS file system cache.

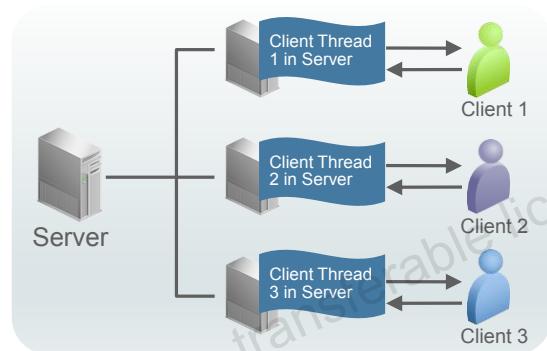
Start with the default values for these buffers and only change them as part of a properly benchmarked tuning strategy to optimize the MySQL server for your workload. Change one value at a time, increment in small amounts, and test by using realistic server loads, revising as necessary.

Note

- The `table_open_cache` variable is covered in the lesson titled "Tuning Tables, Files, and Logs."
- The following variables are covered in the lesson titled "Tuning InnoDB":
 - `innodb_buffer_pool_size`
 - `innodb_additional_mem_pool_size`
 - `innodb_log_buffer_size`

MySQL Thread Handling

- MySQL is a multithreaded server.
 - A single process, comprised of multiple threads
- Each connection has its own thread.
 - 1,000 connections require 1,000 threads.
- Additional helper threads include:
 - Connection manager threads
 - Signal threads (alarms)
 - InnoDB read and write threads
 - InnoDB purge threads
 - Event scheduler
- Thread caching allows unused threads to remain in memory for reuse.



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

You must consider thread support when choosing an operating system. A 32-bit OS limits the amount of memory a thread can use; a 64-bit OS allows a much higher limit.

Thread caching can make a big difference to server performance. During normal operation, threads that are no longer required are released from memory. Thread caching allows unused threads to remain in memory. These threads can then be re-used by a new connection without the overhead of starting a new thread.

Connection manager threads:

- **UNIX:** The TCP/IP connection request manager thread also handles Unix socket file connection requests.
- **Windows only:** MySQL creates a thread for named pipes and shared memory connections if the server supports them.

Per-Thread Buffers

- `sort_buffer_size`
 - The amount of memory available for buffering queries that use `ORDER BY` or `GROUP BY` options
- `read_rnd_buffer_size`
 - The amount of memory available for InnoDB table queries doing range scans on secondary indexes
- `join_buffer_size`
 - The minimum size of the buffer that is used for simple index scans, range index scans, and joins that cannot use indexes and perform full table scans as a result
- `binlog_cache_size / binlog_stmt_cache_size`
 - The size of the caches that store changes to the binary log during a transaction



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

Unlike global buffers, the per-thread buffers allocate memory only when a connection is made and often do not allocate the full amount at once, but only as much as the connection's workload requires.

Because MySQL uses a separate thread for each connection, these buffers are sometimes called per-connection buffers.

`sort_buffer_size`

When a query exceeds the available memory assigned in this buffer, it uses a disk-based sort (disk seek). The optimizer estimates how much space is required, but it can allocate up to the limit specified.

If you see many `Sort_merge_passes` in the output of `SHOW GLOBAL STATUS`, consider increasing the `sort_buffer_size` to speed up `ORDER BY` or `GROUP BY` operations that cannot be improved by optimizing the queries.

`read_rnd_buffer_size`

The `read_rnd_buffer_size` variable affects multi-range read (MRR) operations on InnoDB tables. Reading rows using a range scan on a secondary index can result in many random disk accesses to the base table when the table is large and not stored in InnoDB's cache.

Note: Sorting operations are covered in the lesson titled “Optimizing Queries.”

The multi-range read optimization reduces the number of random disk access operations for range scans by performing the following steps:

- It scans the index and collects the keys for the relevant rows.
- It then sorts the keys.
- Finally, it retrieves the rows from the base table in primary key order.

The `read_rnd_buffer_size` variable determines how much memory is available for the sorting operation. The default is 256KB and it should be kept at this setting for most servers. Consider changing the `read_rnd_buffer_size` variable only if lots of queries are triggering multi-range reads (the output of `EXPLAIN` shows Using MRR in the Extra column.)

join_buffer_size

The best way to achieve fast joins is to add indexes. When this is impossible, consider increasing the `join_buffer_size` from its default value of 256KB. All joins allocate at least this amount of memory, and the MySQL server allocates more memory if required. Except when the optimizer is using the batched key access (BKA) optimization, the join buffer only needs to hold a single matching row, so keep the value of `join_buffer_size` small and only increase it in sessions that are performing large joins. In a two table join, the buffer is allocated once. In a multi-table join, it is allocated once for every two tables involved in the join, so take care when increasing the `join_buffer_size` variable.

binlog_cache_size

A binary log cache is allocated to each connection to store changes to the binary log during a transaction. This cache is only allocated if the server has the binary log enabled (`--log-bin` option). If you often use large transactions, you can increase this cache size from its default of 32 KB to get better performance. Monitor the `Binlog_cache_disk_use` and `Binlog_cache_use` status variables to help you tune this setting. If the ratio of `Binlog_cache_disk_use` to `Binlog_cache_use` is large, consider increasing the value of `binlog_cache_size`.

binlog_stmt_cache_size

The MySQL server allocates separate binary log transaction and statement caches. The `binlog_stmt_cache_size` variable specifies the size of the cache for the binary log to hold nontransactional statements issued during a transaction. If you frequently use large nontransactional statements during transactions, increase the size of this cache to get better performance.

Note: Multi-range read (MRR) is covered in the lesson titled “Optimizing Slow Queries.”

Per-Thread Buffers

- `tmp_table_size`
 - Determines the size of internal memory tables that can be stored, in conjunction with `max_heap_table_size`
 - Associated status variables:
 - `Created_tmp_disk_tables`
 - `Created_tmp_tables`
- `thread_stack`
 - Determines the size of the stack assigned to each thread:
 - If the thread stack size is too small, it limits the complexity of the SQL operations the thread can handle.
 - Default value is 256KB for 64-bit systems.
- `net_buffer_length`
 - Determines the size of the connection and result buffers assigned to each thread



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

`tmp_table_size`

The `tmp_table_size` variable determines the size of the internal memory tables the MySQL server uses when it creates temporary tables to handle query execution. The actual limit is the lower value of the `tmp_table_size` and `max_heap_table_size` variables. The `max_heap_table_size` variable specifies the maximum size of MEMORY tables.

If the in-memory temporary table size exceeds the available memory assigned by the `tmp_table_size` buffer, MySQL creates a table on disk.

When setting this variable, the values of the following status variables can help:

- `Created_tmp_disk_tables` displays the number of on-disk temporary tables that were created to handle query executions.
- `Created_tmp_tables` displays the total number of temporary tables that were created to handle query executions.

You can compare the number of internal on-disk temporary tables created to the total number of internal temporary tables created by comparing the values of the `Created_tmp_disk_tables` and `Created_tmp_tables` variables. Note that each invocation of the `SHOW STATUS` statement uses an internal temporary table and increments the global `Created_tmp_tables` value.

The default value for `tmp_table_size` and `max_heap_table_size` is 16 MB.

`thread_stack`

The default value of `thread_stack` is 192 KB on 32-bit platforms and 256 KB on 64-bit platforms. Never reduce the size of the `thread_stack` system variable, as doing so can limit the following:

- The complexity of SQL statements that the server can handle. This can result in a thread stack overrun error.
- The recursion depth of stored procedures and other actions that consume memory.

Either of these situations might result in system crashes.

`net_buffer_length`

Each client thread is associated with a connection buffer and a result buffer. Both buffers begin at the size specified by the value of `net_buffer_length`, but are dynamically enlarged up to the value of `max_allowed_packet` in bytes, as required. The result buffer shrinks back to the value of `net_buffer_length` after each SQL statement has been executed.

You should normally use the default value of `net_buffer_length`. However, if your system has very little memory, you can set it to the expected length of statements sent by clients. If statements exceed this length, the connection buffer is automatically enlarged. The maximum value of `net_buffer_length` is 1MB.

Topics

- How MySQL uses memory
- Calculating maximum MySQL memory usage
- Managing connections
- Tuning threads



ORACLE®

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

Estimating Maximum MySQL Memory Usage

Maximum global buffer memory usage is the sum of the global buffers and Performance Schema memory usage:

```
query_cache_size +
(1.1 * innodb_buffer_pool_size)
+
innodb_additional_mem_pool_size
+ innodb_log_buffer_size
```

Maximum thread buffer memory usage is the sum of the per-thread buffers multiplied by the maximum number of connections:

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

Total Maximum Memory Usage =
Max global buffer memory + Max thread buffer memory



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

These formulae are greatly simplified. Additionally, the calculations assume that you are using the default InnoDB storage engine.

If the total maximum memory usage is greater than 85-90% of your system RAM, you will experience resource contention or bottlenecks. The worst-case scenario is that the system will start swapping RAM to disk, which severely degrades performance. In systems that support a large number of connections, you must carefully tune the various buffers and *_table_size variables so that you do not waste memory.

Not all the per-thread buffer memory is used at the same time. As a result, you can calculate a slightly more realistic thread memory usage by dividing the maximum thread memory usage by 4. For even greater accuracy, calculate the average query contribution:

Per Query = "buffer for reading rows" + "sorting" + "full joins" + "binlog cache" + "index preload" + internal tmp tables

```
= max(read_buffer_size, read_rnd_buffer_size)
+ max(sort_buffer_size/2, ("avg queries with scan" *
"avg scans with merge" * sort_buffer_size))
+ ("avg full joins" * join_buffer_size)
+ ("avg binlog cache use" * binlog_cache_size)
+ preload_buffer_size
+ ("avg tmp tables" * min(tmp_table_size, max_heap_table_size))
```

Per Thread = thread_stack + 2 * (net_buffer_length)

Total = "global" + (max_used_connections * ("per thread" + "per query"))

Scenario: Total MySQL Memory Usage

Estimate the maximum memory requirements for a server with the following per-thread buffer settings. Are these settings optimal for a server with 4 GB of memory?

```
join_buffer_size = 512K
read_rnd_buffer_size = 2M
sort_buffer_size = 2M
thread_stack = 512K
binlog_cache_size = 256K
max_connections = 300
max_heap_table_size = 24M
tmp_table_size = 32M
```



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

Estimate the maximum memory requirement:

$$300 \text{ threads} \times (512 \text{ KB} + 2 \text{ MB} + 2 \text{ MB} + 512 \text{ KB} + 256 \text{ KB} + 24 \text{ MB}) = 8.86 \text{ GB}$$

A more realistic estimate of total thread use is obtained by dividing the maximum usage value by 4:

$$8.86 \text{ GB} / 4 = 2.22 \text{ GB}$$

This does not take into account the global MySQL memory requirements and reinforces the necessity of tuning global per-thread variables carefully in systems with high concurrency. Use larger values for session per-thread variables as needed. For example, if a particularly complex and time-consuming query benefits from the extra memory allocation.

Displaying Memory Usage by Using sys

The following sys query aggregates currently allocated memory by code area:

```
mysql> SELECT SUBSTRING_INDEX(event_name,'/',2)
    AS code_area, sys.format_bytes(SUM(current_alloc)) AS current_alloc
  FROM sys.x$memory_global_by_current_bytes
 GROUP BY SUBSTRING_INDEX(event_name,'/',2) ORDER BY SUM(current_alloc) DESC;
+-----+-----+
| code_area | current_alloc |
+-----+-----+
| memory/innodb | 843.24 MiB |
| memory/performance_schema | 81.29 MiB |
| memory/mysys | 8.20 MiB |
| memory/sql | 2.47 MiB |
| memory/memory | 174.01 KiB |
| memory/myisam | 46.53 KiB |
| memory/blackhole | 512 bytes |
| memory/federated | 512 bytes |
| memory/csv | 512 bytes |
| memory/vio | 496 bytes |
+-----+-----+
10 rows in set (0.00 sec)
```



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

You can query the `memory_global_by_current_bytes` view for more specific memory event information. For example, the following query shows memory allocated to the InnoDB buffer pool:

```
mysql> SELECT * FROM sys.memory_global_by_current_bytes
      WHERE event_name LIKE 'memory/innodb/buf_buf_pool'\G
***** 1. row *****
event_name: memory/innodb/buf_buf_pool
current_count: 1
current_alloc: 131.06 MiB
current_avg_alloc: 131.06 MiB
high_count: 1
high_alloc: 131.06 MiB
high_avg_alloc: 131.06 MiB
...
...
```

Most memory instrumentation in the Performance Schema is enabled by default. You can enable all memory instruments by adding the following to your MySQL configuration file and restarting the server:

```
performance-schema-instrument='memory/=%=COUNTED'
```

Note: The sys schema in the example in the slide uses the `x$` version of the view to retrieve the raw `current_alloc` data in bytes and then convert it for display by using `sys.format_bytes()`.

Quiz



Which server setting determines the size of the internal memory tables that can be stored when the MySQL server creates temporary tables to handle query execution?

- a. join_buffer_size
- b. read_buffer
- c. thread_stack
- d. tmp_table_size



ORACLE®

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

Answer: d

Topics

- How MySQL uses memory
- Calculating maximum MySQL memory usage
- **Managing connections**
- Tuning threads



ORACLE®

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

Simultaneous Connections in MySQL

The number of simultaneous connections is determined by the `max_connections` system variable.

- Things to consider when setting this value:
 - How many threads the platform supports
 - RAM availability
 - Amount of RAM used for each connection
 - Workload required from each connection
 - Desired response time
- Checking OS use of file descriptors:
 - Linux/Solaris: Execute `cat /proc/sys/fs/file-max` to see the number of available descriptors and `cat /proc/sys/fs/file-nr` to see the number currently in use.
 - Windows: Use a third party tool, like Sysinternals Process Explorer.
- The `Max_used_connections` status variable helps to determine the optimal setting.



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

The default value for `max_connections` is 151. The server reserves one additional connection for users with the `SUPER` privilege. This is so that administrators can connect and diagnose problems even if the maximum number of connections is reached.

The `max_connections` System Variable and the Table Cache

The `table_open_cache` and `max_connections` variables are related. For more information on the `table_open_cache` system variable, see the lesson titled “Tuning Tables, Files, and Logs.”

The `Max_used_connections` Status Variable

This `Max_used_connections` status variable indicates the maximum number of simultaneous connections since the last `FLUSH STATUS`. The value provides a benchmark to help you decide the maximum number of connections that your server should support. It can also help in traffic analysis. The value of `max_connections` is auto-scaled down if it is set too high for the number of file descriptors available.

Note: You can download Sysinternals Process Explorer for Windows from <https://docs.microsoft.com/en-us/sysinternals/downloads/process-explorer>

Connection Status Variables

- Connections
 - The number of connection attempts
 - Consider enabling connection pooling in your application if this value is excessive.
- Max_used_connections
 - This status variable displays the maximum number of simultaneous connections.
 - If the value is equal (or close) to the max_connections setting, increase max_connections.
 - Possible sign of overload



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

When an application creates and destroys connections to fulfill each request, this adds overhead to each operation and negatively affects the performance of the server. You can reduce the overhead of repeated connection attempts by enabling connection pooling in your application so that it can reuse connections for subsequent requests.

Connection pooling can improve the performance of applications where the queries executed are few and simple. It is less effective if there are numerous connections executing complex queries. Consult the documentation for your chosen connector for information.

Monitor the `Connections` status variable to establish the number of client connection attempts. Monitor the `max_used_connections` server status variable over time to detect spikes in activity. If `max_used_connections` regularly reaches the `max_connections` limit, you can assume that some users are unable to connect to the database.

Note

- Connection pooling with Connector/J: <https://dev.mysql.com/doc/connector-j/en/connector-j- usagenotes-j2ee-concepts-connection-pooling.html>
- Connection pooling with Connector/Python: <https://dev.mysql.com/doc/connector-python/en/connector-python-connection-pooling.html>
- Persistent database connections with PHP:
<http://www.php.net/manual/en/features.persistent-connections.php>
- Support for persistent connections with the PHP mysqli library:
<http://www.php.net/manual/en/mysqli.persistconns.php>

Monitoring Idle Connections with Performance Schema

```
mysql> SELECT USER, HOST,
-> sys.format_time(SUM_TIMER_WAIT) total_idle,
-> sys.format_time(AVG_TIMER_WAIT) average_idle,
-> sys.format_time(MAX_TIMER_WAIT) max_idle
-> FROM performance_schema.events_waits_summary_by_account_by_event_name
-> WHERE EVENT_NAME = 'idle' AND HOST IS NOT NULL;

+-----+-----+-----+-----+
| USER | HOST      | total_idle | average_idle | max_idle |
+-----+-----+-----+-----+
| root | localhost | 3.03h     | 00:02:06.93 | 1.44h    |
| mark | localhost | 2.04h     | 00:01:07.33 | 1.01h    |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```



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

Idle connections can cause problems for applications that use persistent connections (increasing the risk of timeouts) and on the server (because resources remain allocated to idle connections.)

Any application with persistent connections will have periods where connections are idle, but too much idle time can point to application problems. For example, it might mean that the connection pool is too large and if average or maximum idle times approach or exceed the values of the global `wait_timeout` or `interactive_timeout` system variables it might suggest that application connections are not being maintained properly.

The Performance Schema query in the slide shows total, average, and maximum idle times by user account. If you want to display idle time by host, use the following query:

```
SELECT HOST, sys.format_time(SUM_TIMER_WAIT) total_idle,
sys.format_time(AVG_TIMER_WAIT) average_idle,
sys.format_time(MAX_TIMER_WAIT) max_idle
FROM events_waits_summary_by_host_by_event_name
WHERE EVENT_NAME = 'idle' AND HOST IS NOT NULL;
```

Scenario: Users Unable to Connect

During busy periods, several database users have reported the following error message:

Unable to connect to the database: Too many connections

What steps do you take to resolve this issue?



ORACLE®

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

There are two ways to solve the issue outlined on the slide. The first approach is to increase the connection limit (`max_connections`). The second is to try and reduce MySQL server usage during these periods.

Many applications that access the database do not close their connection to it. If you set the connections limit to a very high value, these can build up and affect server performance. A better approach is to investigate what is causing these spikes in connections. You might want to check if any particular users or processes are slowing the system. For example, you could issue the following command to show all running processes for the `appuser` user.

```
# mysqladmin -uappuser -p processlist
```

Diagnosing Network Problems

- **Aborted_clients**
 - The number of connections terminated because the client died without closing the connection
- **Aborted_connects**
 - The number of failed attempts to connect to your server
- **Bytes_received**
 - The number of bytes received from all clients
- **Bytes_sent**
 - The number of bytes sent to all clients



ORACLE®

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

Many connection problems are related to the network. The status variables shown in the slide help you diagnose network issues.

Aborted_clients

A large number here could be a sign of a network problem. Verify that the `max_allowed_packet` setting is a reasonable size for the data types in your database. Increasing the value of the `wait_timeout` system variable might also help minimize this problem.

Aborted_connects

A non-zero value for this status variable could be a sign of a network problem, perhaps caused by one of the following:

- Attacks on the server (bad password, port scans, and so on)
- Problems with the client application (For example, the client does not close the connection to the server before exiting, or the connection ends abruptly during a data transfer.)

Bytes_received and Bytes_sent

Are the numbers here within the values you expect? Can the network handle the amount of traffic?

Topics

- How MySQL uses memory
- Calculating maximum MySQL memory usage
- Managing connections
- Tuning threads



ORACLE®

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

Reusing Threads

- The `thread_cache_size` determines the number of threads that the server can store for reuse.
- Things to consider when setting this value:
 - Review `Connections` and `Threads_created`.
 - Calculate thread cache hit rate:

$$\text{Thread cache hit rate \%} = 100 - ((\text{Threads_created} / \text{Connections}) * 100)$$
- Evaluate these settings against the time that the server has been up and running, or since the status variables were flushed.
 - `Connections/Uptime`: Connections per second
 - `Threads_created/Uptime`: Threads created per second
 - `Uptime_since_flush_status`: Time elapsed since last FLUSH STATUS



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

Every time an application creates or destroys a connection, it adds a performance overhead to the server's operation. To avoid this overhead, use persistent connections in the application, or provide a connection pool so that the application's operations can share persistent connections instead of destroying and re-creating client connections.

If you cannot use persistent connections, you can adjust the `thread_cache_size` system variable to configure how many threads the server caches for reuse. Choose a value that covers typical variations in system activity:

- For servers that vary from 50 to 150 active connections, set the `thread_cache_size` value to 100.
- Setting this system variable to accommodate rare bursts of connections at one time (such as 500 connections for a system that typically sees 50 to 150 active connections) is a waste of memory.
- Increase the `back_log` system variable from the default (autosized to $50 + (\text{max_connections} / 5)$) to 200 or more in cases where connections are being requested faster than the OS and MySQL can process them.

By examining the difference between the `Connections` and `Threads_created` status variables, you can assess the efficiency of the thread cache.

- The `Connections` status variable lists the number of connection attempts (successful or not) to the MySQL server.
- The `Threads_created` status variable lists the number of threads created to handle connections.

Uptime status variable

- The number of seconds the server has been up and running

Uptime_since_flush_status status variable

- The number of seconds the server has been running since the last `FLUSH STATUS` statement was issued

Quiz



Which status variable can help you determine the optimal `max_connections` server setting for your system?

- a. `key_buffer_size`
- b. `Max_used_connections`
- c. `query_cache_size`
- d. `table_open_cache`



ORACLE®

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

Answer: b

Other Thread Status Variables

- **Threads_cached**
 - This status variable displays the number of threads currently in the cache.
- **Threads_connected**
 - This status variable displays the current number of connections.
- **Threads_running**
 - This status variable displays the number of queries currently executing.
 - It provides a valuable picture of the load on your server.
 - The following command displays the number of threads running at a one second interval:

```
mysqladmin -il extended | grep Threads_running
```



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

Threads_cached

- Obtain the value of the `threads_cached` status variable by using:
`SHOW STATUS LIKE 'Threads_cached'.`

Threads_running

- This variable encompasses a large number of activities, such as queries waiting for I/O, queries using locks, queries waiting to acquire a mutex, and so on.
- If there is no activity on the server, one thread will be shown as being active if you issue a `SHOW STATUS` command to view the value of this status variable. In the slide example, `SHOW STATUS` is executed by the `mysqladmin` command. Subtract one from the number to get the actual count.

Scenario: Calculating Thread Cache Effectiveness

Given the following values, what is the thread cache effectiveness?

```
thread_cache_size = 30
Threads_created = 500000
Connections = 2000000
Uptime = 100000
```

Is the `thread_cache_size` system variable set appropriately for this server?



ORACLE

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

Thread Cache: Scenario

- Using the calculation $100 - [(\text{Threads_created} / \text{Connections}) * 100]$ tells you the thread cache hit rate for your server. For this particular scenario, the thread cache hit rate is 75%.
- Using the calculation `Connections/Uptime` can tell you the number of connections made per second. For this particular scenario, an average of 20 connections are made per second.
- Using the calculation `Threads_created/Uptime` can tell you the number of threads needed to be created per second. For this particular scenario, an average of five threads are created per second.

Conclusion: The `thread_cache_size` setting is too small for this server, if this represents a typical workload. Use the default value for `thread_cache_size` (autosized to $8 + \text{max_connections} / 100$) initially, recalculate the thread cache hit rate and tune it as necessary. This server might also benefit from connecting pooling (see the slide titled “Connection Status Variables”)

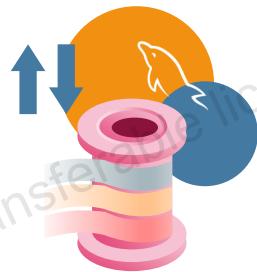
Scenario: Setting `thread_cache_size`

Your MySQL server host is running at 90-95% of CPU capacity.

Issuing the following statement gives you a picture of the potential problem:

```
mysql> SHOW STATUS LIKE 'Threads%';  
+-----+-----+  
| Variable_name      | Value   |  
+-----+-----+  
| Threads_cached     | 0       |  
| Threads_connected  | 54      |  
| Threads_created    | 345678  |  
| Threads_running    | 41      |  
+-----+-----+  
4 rows in set (#.# sec)
```

What is your next step?



ORACLE®

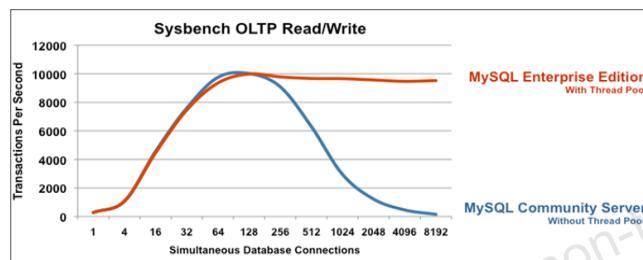
Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Many threads are being created, but none are available in the thread cache for reuse. In this situation, you might want to consider other status variables such as `Uptime`, `Uptime_since_flush_status`, and `Max_used_connections`. If those values are reasonable, increasing the value of the `thread_cache_size` setting to a larger number has the potential to improve performance.

The MySQL Enterprise Thread Pool

The MySQL Enterprise Thread Pool is a server plugin included in commercial distributions of the MySQL server.

- Allows several connections for one execution thread
- Stabilizes transactions per second in high concurrency systems
- Decreases context switches
- Useful if `Threads_running > number of hardware threads`



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

Commercial editions of MySQL include the MySQL Thread Pool server plugin. The default thread-handling model in MySQL Server executes statements using one thread per client connection. As more clients connect to the server and execute statements, overall performance degrades. The MySQL Thread Pool plugin provides an alternative thread-handling model designed to reduce overhead and improve performance.

The thread pool plug-in increases server performance by efficiently managing statement execution threads for large numbers of client connections. The thread pool separates connections and threads, so there is no fixed relationship between connections and the threads that execute statements received from those connections. As new connections are established, the plugin assigns them into a configurable number of groups. The thread pool tries to ensure a maximum of one thread executing in each group at any time, but sometimes permits more threads to execute temporarily for best performance.

The MySQL Thread Pool Plugin and Denial of Service (DoS) Attacks

The MySQL Thread Pool plugin also helps prevent denial of service (DoS) attacks. In extreme cases, a DoS attack can render the MySQL server completely unresponsive, without any way for the DBA to shut it down in a controlled fashion. This might result in a “hard” shutdown and corrupted table data.

The thread pool mitigates such attacks by limiting the amount of time each thread can execute before control passes to another thread. This ensures that the available resources are distributed equally among requests. The MySQL Thread Pool plugin also provides methods for the DBA to bypass any sessions that are affected by a DoS attack.

Tuning the MySQL Thread Pool

- `thread_pool_size`: The number of threads in the thread pool. It controls how many threads can execute simultaneously. The default value is 16.
- `thread_pool_stall_limit`: The timeout period before a thread is considered blocked and the thread pool can begin executing another statement. The default value is 60ms.
- The maximum number of threads in the system is the sum of `max_connections` and `thread_pool_size`.

You can monitor thread pool operation by querying the following `INFORMATION_SCHEMA` tables:

- `TP_THREAD_STATE`: Information about thread pool thread states
- `TP_THREAD_GROUP_STATE`: Information about thread pool thread group states
- `TP_THREAD_GROUP_STATS`: Thread group statistics

For example, the following query helps you tune `thread_pool_stall_limit` for your server workload by determining the fraction of executed statements that stalled. This number should be as low as possible:

```
SELECT SUM(STALLED_QUERIES_EXECUTED) / SUM(QUERIES_EXECUTED)
FROM INFORMATION_SCHEMA.TP_THREAD_GROUP_STATS;
```

Summary



In this lesson, you should have learned how to:

- Describe the components of the MySQL server architecture that directly affect performance
- Explain how the MySQL server uses memory
- List the system and status variables that relate to:
 - Memory usage
 - Connections to the MySQL server
 - Thread behavior
- Describe the benefits of using the MySQL Enterprise Thread Pool plugin

ORACLE®

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

Practices

- 5-1: Calculating Total Thread Memory Usage
- 5-2: Managing the Number of Client Connections
- 5-3: Evaluating the Effect of Multiple Simultaneous Connections
- 5-4: Investigating the Effects of Thread Caching



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

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

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

6

Tuning Tables, Files, and Logs



ORACLE®

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

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

Topics

- Sizing the table cache
- Managing tables and files
- Tuning the binary logs



ORACLE®

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

Objectives



After completing this lesson, you should be able to:

- Tune the table cache
- Describe the effects of the table cache on database files
- Tune the binary log caches

ORACLE®

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

Topics

- Sizing the table cache
- Managing tables and files
- Tuning the binary logs



ORACLE®

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

Reusing Tables

- When you cache tables, you:
 - Improve performance by keeping a pool of open tables so that each thread requiring access to the table does not have to open it independently
 - Use more memory to maintain the cache
- The `table_open_cache` system variable determines how many open tables the server caches for all threads.
 - If the `table_open_cache` value is too low:
 - The MySQL server has to constantly send disk I/O requests to open the required tables.
 - If the `table_open_cache` value is too high:
 - Your server might run out of file descriptors, resulting in instability.
 - The cache is expensive to maintain.



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

The MySQL server is multithreaded. If multiple connections access the same table at the same time, each thread opens the table independently. This minimizes the problems that can occur when multiple client sessions have different states on the same table, but is resource-intensive.

You should normally set `table_open_cache` large enough so that the server keeps most of the tables it is using open continuously. Each query that is running uses one entry for each table it is using, so caching tables is not really helpful when many concurrent connections open the same table.

If your table cache is large but not all open tables fit in it, the cache becomes increasingly expensive to maintain. Evicting a table from the cache to make space for a new table is a slow process for large caches.

How MySQL Caches Tables

- Every time a connection opens a table, it stores the table descriptor in the cache.
 - The `table_open_cache` system variable limits the number of open tables for all threads
 - MySQL might open extra tables temporarily to execute queries.
- MySQL closes an unused table and removes it from the cache when:
 - The cache is full and a thread tries to open a table that is not in the cache
 - The cache contains more tables than `table_open_cache` permits, and a table in the cache is not being used
 - Tables are flushed
- The table that is used first is discarded first.



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

Consider the value of `max_connections` when configuring `table_open_cache`. For example, with 300 concurrent connections, specify a table cache size of at least $300 * N$, where N is the maximum number of tables per join in any of your queries. Allow extra for temporary tables and files.

If the cache is full and all its tables are being used:

- MySQL extends the cache temporarily
- When a table is no longer being used, it is removed and the table cache reverts to its previous size.

You can improve scalability by partitioning the table cache into two or more smaller cache instances. This provides better performance for operations that use the cache because it reduces contention between sessions. Partition the table cache by changing the value of the `table_open_cache_instances` to a number greater than its default of one. The size of each cache instance is the value of `total_open_cache` divided by `table_open_cache_instances`. This benefits DML statements, but DDL statements still lock the entire cache.

Note: The status variable `Flush_commands` tells you how often you execute a flush.

Setting table_open_cache

- The `table_open_cache` variable is a global setting.
 - You can change its value without restarting the server.
- The `innodb_open_files` setting is autosized to match `table_open_cache`.
 - The minimum value of `innodb_open_files` is 300.
- The following status variables help you to determine the proper `table_open_cache` setting:
 - `Open_tables`: Displays the number of tables currently open
 - `Opened_tables`: Displays the number of tables that have been opened
 - `Table_open_cache_hits`: Calculates the table cache hit rate



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

By default, the MySQL server uses multiple InnoDB tablespaces. If `innodb_file_per_table` is set to 1, MySQL creates and stores table data and associated indexes in a single data file for each InnoDB table. In this situation, the value of `table_open_cache` affects the `innodb_open_files` setting. The value of `innodb_open_files` is the higher of `table_open_cache` and 300, and specifies the number of .ibd files that MySQL can keep open simultaneously.

The following status variables help you determine the table cache hit rate:

```
mysql> SHOW GLOBAL STATUS LIKE 'Table_open%';
+-----+-----+
| Variable_name      | Value |
+-----+-----+
| Table_open_cache_hits | 255   |
| Table_open_cache_misses | 91    |
| Table_open_cache_overflows | 0     |
+-----+-----+
3 rows in set (0.00 sec)
```

Note however, that the value of `Opened_tables` increments by two each time you create a temporary table. Therefore, if you are using temporary tables, a large value in `Opened_tables` does not necessarily indicate that your table cache value is too small, and enlarging the table cache does not stop `Opened_tables` from growing.

Note: For more information about InnoDB settings, see the lesson titled “Tuning InnoDB.”

Sizing the Table Cache

The default `table_open_cache` value of 2000 is sufficient for most servers. If the default `table_open_cache` setting is not ideal for your workload, consider sizing the table cache to one of the following:

- The number of tables opened during a ten second period on a production server that has been running for at least an hour. To calculate this, perform the following steps:
 1. Check the value of `Opened_tables` on a production server that has been running for more than one hour.
 2. Repeat after ten seconds.
 3. The increase in `Opened_tables` is a reasonable guide to the amount by which you should increase the value of the `table_open_cache` setting.
 - Alternatively, increase by 100 and repeat the checks until the number of tables opened per second is minimal, and preferably zero per minute on average.
- $2 \times \text{max_connections}$
- $2 \times \text{the number of tables}$



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

Setting `table_definition_cache`

- The table definition cache is part of the table cache and stores only the table definitions.
 - The definition stores the table metadata, such as column names, data types, and so on.
 - Unlike opened tables, table definitions are shared among all connections.
- Ideally, set `table_definition_cache` to a size that is large enough to hold all your table definitions.
- The table definition cache imposes the following limits on InnoDB tables:
 - The number of tables stored in the InnoDB data dictionary cache
 - The number of InnoDB per-file tablespaces that can be open simultaneously



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

The minimum value of the `table_definition_cache` setting is 400. The default value is based on the following formula, and has an upper limit of 2,000:

$$400 + (\text{table_open_cache} / 2).$$

Note: This course covers the effects of the `table_definition_cache` setting on InnoDB tables in the lesson titled “Tuning InnoDB.”

Setting table_open_cache: Scenario 1

Given the values in the following example, what can you conclude about the configured value of table_open_cache?

```
table_open_cache = 600    Open_tables = 112
max_connections = 151    Opened_tables = 1712
                        Uptime = 14325067
                        Uptime_since_flush_status = 5045067
```

1. How long has the server been up and running?
2. How long has the server been up and running since the status variables were reset?
3. How often did the server have to open a table?
4. What is significant about the number of current open tables in comparison to the table cache?
5. Is the table_open_cache setting too high, too low, or just right?



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

1. The Uptime status variable tells you that the server has been running for approximately 166 days (in seconds).
2. The Uptime_since_flush_status status variable tells you that the last FLUSH STATUS was approximately 59 days ago.
3. The value of Opened_tables tells you that MySQL had to open a table (not in the cache) only about once every hour.
4. The number of tables that are currently open (Open_tables) is approximately one fifth of the table cache setting.
5. The table_open_cache setting is probably too high and is potentially wasting system resources.

Setting table_open_cache: Scenario 2

Given the values in the following example, what can you conclude about the configured value of table_open_cache?

```
table_open_cache = 128    Open_tables = 128
max_connections = 151    Opened_tables = 678
                        Uptime = 2459870
                        Uptime_since_flush_status = 1349870
```

1. How long has the server been up and running?
2. How long has the server been up and running since the status variables were reset?
3. How often did the server have to open a table?
4. What is significant about the number of current open tables in comparison to the table cache?
5. Is the table_open_cache setting too high, too low, or just right?



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

1. The Uptime status variable tells you that the server has been running for approximately 29 days (in seconds).
2. The Uptime_since_flush_status status variable tells you that the last FLUSH STATUS was approximately 16 days ago.
3. The Opened_tables status variable tells you that MySQL has opened only two tables (not in the cache) approximately once every hour.
4. The number of tables that are currently open (Open_tables) is equal to the number of tables in the cache.
5. The table_open_cache system variable is set correctly for this server.

Setting table_open_cache: Scenario 3

Given the values in the following example, what can you conclude about the table_open_cache setting?

```
table_open_cache = 64      Open_tables = 130
max_connections = 151     Opened_tables = 1560160
                          Uptime = 250576
                          Uptime_since_flush_status = 44576
```

1. How long has the server been up and running?
2. How long has the server been up and running since the status variables were reset?
3. How often did the server have to open a table?
4. What is significant about the number of current open tables in comparison to the table cache?
5. Is the table_open_cache setting too high, too low, or just right?



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

1. The Uptime variable tells you that the server has been running for approximately three days.
2. The Uptime_since_flush_status variable tells you that last FLUSH STATUS was approximately half a day ago.
3. The value of the Opened_tables status variable tells you that MySQL had to open approximately 35 tables per second.
4. The number of currently open tables (Open_tables) is greater than the table_open_cache system variable.
5. The current setting of table_open_cache is too low for this system. Increasing table_open_cache on this system should improve performance.

Another factor to consider is how fast the value of the Opened_tables status variable is increasing. If the value of Opened_tables is increasing rapidly, the table_open_cache setting is too low; increasing table_open_cache can improve server performance by reducing the rate at which tables are opened. If the value of Opened_tables is not increasing rapidly, keep the current value of table_open_cache and reevaluate it later.

Quiz



The `table_open_cache` system variable can be temporarily extended when the cache is full and a new table must be opened.

- a. True
- b. False



ORACLE®

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

Answer: a

Topics

- Sizing the table cache
- Managing tables and files
- Tuning the binary logs



ORACLE®

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

Tables and Files

- If you change the `table_open_cache` and `max_connections` system variables, it affects the number of files the operating system keeps open.
- Different operating systems have limits on the per-process number of open file descriptors.
- Many operating systems enable you to increase the limit on open file descriptors.



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

Note: The “Setting File Descriptors in the Operating System” slide later in this lesson describes how to increase the number of available file descriptors.

Managing the Number of Open Files

- The `open_files_limit` system variable determines the maximum number of file descriptors the server can use.
 - When the MySQL server requires more than the number of file descriptors that this setting allows, the server raises a “too many open files” error.
 - The actual value the MySQL server uses depends on the operating system and might be different from the value specified.
- The `Open_files` status variable:
 - Displays the number of files that the MySQL server has open currently
 - Does not include files opened by storage engines (or other non-server processes such as sockets or pipes)
- The `Opened_files` status variable displays the number of regular files that have been opened by the server.
 - Not all files that the server opens increment this variable; for example, sockets and pipes, and the files that storage engines open using their own internal functions.



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

The actual value of the `open_files_limit` variable at runtime is dependent on the OS and might be different from the value you specify at server startup. If MySQL cannot change the number of open files, the value is zero. The effective limit is determined by the MySQL server at runtime using the following calculations:

1. $10 + \text{max_connections} + (\text{table_open_cache} * 2)$
2. The value of `max_connections` * 5
3. The `open_files_limit` value specified at startup, 5000 if there is no value provided.

The server attempts to obtain the number of file descriptors using the maximum of the three calculated values. If it cannot obtain that many descriptors, it writes a warning to the error log and obtains as many file descriptors as the OS permits.

Setting File Descriptors in the Operating System

- Verify that the operating system maximum number of file handles is set properly.
 - A reasonable number is 256 for each 4 MB of RAM.
- Set the Linux number of file descriptors:
 - View the current `file-max` limits.
 - Update the `file-max` limits.
 - Reload the changes.
- Set the Windows Server number of file descriptors:
 - Ensure that the Application Server role is installed.
 - Edit the `CONFIG.NT` file in `%SystemRoot%\System32`.
 - Change the `Files` entry.
 - If it does not exist, add `Files=40` and increment gradually in blocks of 20.



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

Before you change the value of `open_files_limit`, ensure that you set the maximum number of file handles properly in the OS.

Using the formula in the slide, if you have a 2 GB system, set the number of file descriptors to 128,000 (2 GB/4 MB * 256).

To make this change in Oracle Linux:

1. View current `file-max` limits.

```
sysctl fs.file-max
```

2. Update `file-max` limits.

```
sysctl -w file-max=128000
```

3. Reload `/etc/sysctl.conf` to avoid restarting the server.

```
sysctl -p
```

Quiz



Which variable determines the maximum number of file descriptors that the server can use?

- a. open_files_limit
- b. Open_files
- c. Opened_files
- d. file-max



ORACLE®

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

Answer: a

Topics

- Sizing the table cache
- Managing tables and files
- Tuning the binary logs



ORACLE®

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

Binary Logs

The binary log records events that make changes to database structure and table data. It is used for:

- Replication
 - The binary log records changes on the master that must be reflected on the slave.
- Point-in-time recovery
 - You use the binary log to apply changes since the last backup.



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

Enabling the binary log reduces performance slightly. However, the benefits of the binary log in enabling you to set up replication and for restore operations generally outweigh this minor performance decrement.

ACID and Database Transactions

- A transaction is a set of data change events that supports the ACID properties:
 - **Atomic:** If any transaction event fails, the whole transaction fails.
 - **Consistent:** The transaction changes data from one consistent state to another.
 - **Isolated:** No transaction can see the partially completed state of another transaction.
 - **Durable:** When the server tells the client that the transaction commits, the transaction must be stored persistently.
- Transaction support is a feature of the storage engine.
 - InnoDB supports transactions.
 - MyISAM does not support transactions.



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

Example: When you transfer funds from account *A* to account *B*, the transaction includes at least two operations:

1. Subtract a value from account *A*
2. Add a value to account *B*

The transaction fulfils the ACID properties in the following ways:

- If operation 2 fails, then the transaction rolls back operation 1 so that the transaction is atomic.
- After the transaction completes, the sum of the values in account *A* and account *B* is consistent with the sum of their values before the transaction started.
- The transaction is isolated so that no other transaction can see the state of account *A* after operation 1 completes but before operation 2 completes. This is because:
 - The sum total of accounts *A* and *B* at that time is not consistent with the sum before or after the transaction completes
 - Operation 2 might not succeed, so operation 1 might roll back.
- When the transaction commits and clients can see the modified state of accounts *A* and *B*, the result must be durable so that no other operation (such as a server crash) can roll back the transaction's operations.

Transactions and the Binary Log

- The transaction log is *crash safe*:
 - The binary log only records complete events or transactions.
 - If the server crashes during a transaction, the partial transaction is not logged.
- Threads allocate buffers to store details of data modification statements before they are written to the binary log.
 - Transactional cache for operations that modify data in a transactional storage engine
 - Statement cache for nontransactional operations
- If a transaction includes nontransactional operations, those operations cannot be rolled back.



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

The binary log records both transactional and nontransactional operations. If a transaction includes data operations on tables that do not use a transactional storage engine and that transaction fails, then those data operations cannot be rolled back. The binary log records all of the operations in such a transaction, and includes a ROLLBACK event so that any server that subsequently applies the binary log executes the nontransactional statement in the same way that the original transaction executed it.

Sizing the Binary Log Caches

- The following variables control the size of the binary log caches:
 - `binlog_cache_size`
 - Defines the size of the log cache that holds the SQL statements for the binary log during a transaction.
 - `binlog_stmt_cache_size`
 - Defines the size of the log cache that holds nontransactional statements issued during a transaction.
- If a transaction is too big for the cache, it is written to a temporary file.
 - This can affect performance.



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

The actual data stored in the binary log depends on the value of `binlog_format`.

- **STATEMENT:** The default setting. When `binlog_format` is STATEMENT, the binary log contains the actual SQL statements that cause the modification.
- **ROW:** The binary log stores events that describe the modifications, rather than the actual statements.
- **MIXED:** Uses statement logging by default, except for cases where only row-based replication is guaranteed to properly describe the modification.

`binlog_cache_size`

If your server enables a transactional storage engine and the binary log is enabled, each client that connects gets allocated a binary log cache. Increasing the size of the binary log cache can improve performance on systems that execute frequent, large, multistatement transactions. The default size is 32 KB.

`binlog_stmt_cache_size`

Consider increasing this cache size from the default 32 KB if your transactions frequently contain large nontransactional statements.

Note: The binary Logs and InnoDB log files are not related. InnoDB uses the redo logs for crash recovery and the undo logs to roll back uncommitted transactions. This course covers InnoDB settings in the lesson titled “Tuning InnoDB.”

Monitoring Binary Logs

- Use the following server status variables to help you tune the sizes of the binary log caches.
 - Binary log cache (transactional statements):
 - `Binlog_cache_use`
 - `Binlog_cache_disk_use`
 - Binary log statement cache (nontransactional statements):
 - `Binlog_stmt_cache_use`
 - `Binlog_stmt_cache_disk_use`
- Calculate the binary log cache effectiveness:

```
Binary log cache effectiveness percentage =  
(100 - (Cache Disk Use / Cache Use) * 100)
```



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

Binlog_cache_use / Binlog_stmt_cache_use

These status variables represent the number of statements that used the transactional and nontransactional binary log caches respectively.

Binlog_cache_disk_use / Binlog_stmt_cache_disk_use

These status variables store a count of statements that did not fit into their respective caches. Instead, the statements were stored in a temporary file on disk. Increasing the size of the binary log caches minimizes the need to write and read from disk, which can improve performance. Determine the binary log cache effectiveness by using the formula in the slide. If the ratio of cache disk use to overall cache use is large, increase the cache size to improve performance. The MySQL Enterprise Monitor “Binary Log Usage Exceeding Disk Cache Memory Limits” advisor alerts you if this is the case.

Binary Log Group Commit Settings

Transactions are committed to the binary log as a group to improve performance. The following system variables affect binary log group commit behavior:

- `binlog_max_flush_queue_time`: How long (in microseconds) to keep reading transactions from the flush queue before committing them to the binary log
- `binlog_order_commits`: Whether or not to commit transactions in the same order they are written to the binary log
- `sync_binlog`: How many commit groups the server writes to the binary log in memory before they are synchronized to disk



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

`binlog_max_flush_queue_time`

The default value for `binlog_max_flush_queue_time` is zero. This means that the server keeps reading new transactions until the flush queue is empty, and only then commits them to disk.

`binlog_order_commits`

By default, transactions are committed in the same order they are written to the binary log. If you disable this setting, transactions may be committed in parallel. In some cases, this will improve performance, but the binary log might not reflect the actual commit order of the transactions.

`sync_binlog`

The default value for `sync_binlog` is zero, which means that the MySQL server does not flush the binary log to disk but waits for the OS to do it. Although this is usually the fastest setting, be aware that there is a trade-off. The entire binary log is cached in memory and when it is rotated, the server must generate a lock and write everything to disk. This might cause the application to stall.

Setting `sync_binlog=1` is the safest choice (because you only lose a maximum of one commit group from the binary log).

Improving Binary Log Performance

- If all tables have primary keys:
 - Set `binlog_row_image=minimal`
- Configure binary log flushing with `sync_binlog`:
 - Set `sync_binlog` to 0 to rely on file system flushing
 - To explicitly flush after each commit, set `sync_binlog=1`
 - Set `sync_binlog` to a higher value to flush less frequently, for example if you are executing a large number of transactions



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

When using binary logging, and if all your tables have a primary key, set the `binlog_row_image` system variable to `minimal`. This saves binary log space and reduces the amount of binary log data written to disk, and it also saves write capacity if the binary logs are on SSD or flash drives.

If your tables do not have primary keys, setting `binlog_row_image=minimal` prevents the binary logs from being applied and breaks replication. Ensure that you create a primary key on all tables to avoid this problem and other problems that might occur when you do not have an explicit primary key. If you cannot set an explicit primary key on some table, use the default setting for the variable `binlog_row_image=full`.

Scenario: Binary Log File Cache Effectiveness

Given the values of the status variables in the following example, what can you say about the effectiveness of the binary log caches?

```
mysql> SHOW GLOBAL STATUS LIKE 'Binlog%';
+-----+-----+
| Variable_name          | Value   |
+-----+-----+
| Binlog_cache_disk_use | 25760   |
| Binlog_cache_use       | 237477  |
| Binlog_stmt_cache_disk_use | 207    |
| Binlog_stmt_cache_use  | 1103   |
+-----+-----+
4 rows in set (#.# sec)
```



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

Calculate the cache effectiveness:

- Binary log cache effectiveness is: $237477 / 25760 = 9.2$. This means that nearly 1 in 10 transactional statements are written to a temporary table before being flushed to the log.
- Binary statement log cache effectiveness is: $1103 / 207 = 5.3$. This means that approximately 1 in 5 nontransactional statements are written to a temporary table before being flushed to the log.

Both of these ratios are unacceptable. Consider increasing the values of the `binlog_cache_size` and `binlog_stmt_cache_size` system variables to avoid unnecessary disk I/O.

Quiz



Increasing the value of the `binlog_cache_size` and `binlog_stmt_cache_size` server settings minimizes the need to write and read from disk, which can improve performance.

- a. True
- b. False



ORACLE®

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

Answer: a

Summary



In this lesson, you should have learned how to:

- Tune the table cache
- Describe the effects of the table cache on database files
- Tune the binary log caches

ORACLE®

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

Practices

- 6-1: Sizing the Table Cache
- 6-2: Tuning Open Files Limit
- 6-3: Sizing the Binary Log Cache
- 6-4: Identifying I/O Hotspots with MySQL Workbench



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

Tuning InnoDB

7



MySQL™

ORACLE®

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

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

Topics

- What is InnoDB?
- How does InnoDB work?
- Tuning InnoDB buffer pool and log files
- Measuring InnoDB performance



ORACLE®

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

Objectives



After completing this lesson, you should be able to:

- List the key benefits of the InnoDB storage engine
- Describe how InnoDB uses log files and buffers
- Explain the `SHOW ENGINE INNODB STATUS` output
- Access key InnoDB metrics in the Information Schema
- Tune InnoDB settings for best performance

ORACLE®

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

Topics

- What is InnoDB?
- How does InnoDB work?
- Tuning InnoDB buffer pool and log files
- Measuring InnoDB performance



ORACLE®

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

InnoDB Storage Engine

- Is the default storage engine in MySQL
- Is ACID compliant
- Is fully transactional
- Enforces referential integrity through foreign key constraints
- Provides auto-recovery after a crash
- Supports multiversioning (MVCC) and row-level locking
- Provides its own (“native”) partitioning handler
- Arranges data on disk in primary key order to improve the performance of key lookups and join operations



ORACLE®

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

The MySQL pluggable storage engine architecture enables database professionals to select different storage engines to target specific application needs, without having to concern themselves about the implementation details. In the vast majority of situations, the default storage engine—InnoDB—is the best choice, and this course focuses on InnoDB.

InnoDB has proven to be a reliable data storage engine for modern, high-concurrency database systems. It is fully ACID compliant (atomic, consistent, isolated, and durable), and supports a wide range of isolation modes, from the least strict `READ-UNCOMMITTED` to the most strict `SERIALIZABLE`.

InnoDB multiversion concurrency control (MVCC) enables transactions that only read data to operate against a consistent snapshot of data based on a point in time, ignoring any changes that happen after that point. Each transaction that reads data records its own snapshot so that all read operations performed by that transaction are consistent. Other transactions can subsequently change the data without being blocked by the transaction holding the snapshot, so MVCC enables high performance for high-concurrency workloads.

Note: This course covers MVCC and locking in the lesson titled “Troubleshooting Performance Issues.”

The generic partitioning handler in the MySQL server is deprecated, and each storage engine is expected to provide its own partitioning handler. InnoDB and NDB are currently the only storage engines that do this.

Every InnoDB table has a special index called the clustered index, which stores the data rows and improves query performance. The clustered index is usually synonymous with the primary key.

Note: This course covers clustered indexes in the lesson titled “Query Optimization.”

Topics

- What is InnoDB?
- How does InnoDB work?
- Tuning InnoDB buffer pool and log files
- Measuring InnoDB performance

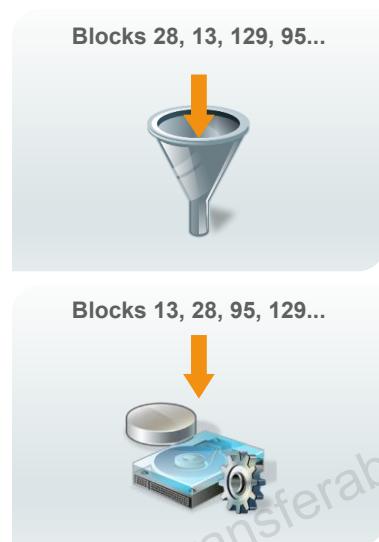


ORACLE®

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

Operating System Buffers

- A buffer acts like a funnel, preventing changes from being applied to disk all at the same time.
- Changes held in the buffer are reordered and merged to enable sequential I/O.
- If power is lost, the contents of the buffer disappear.
- InnoDB maintains its own buffers to ensure data consistency and durability.



ORACLE®

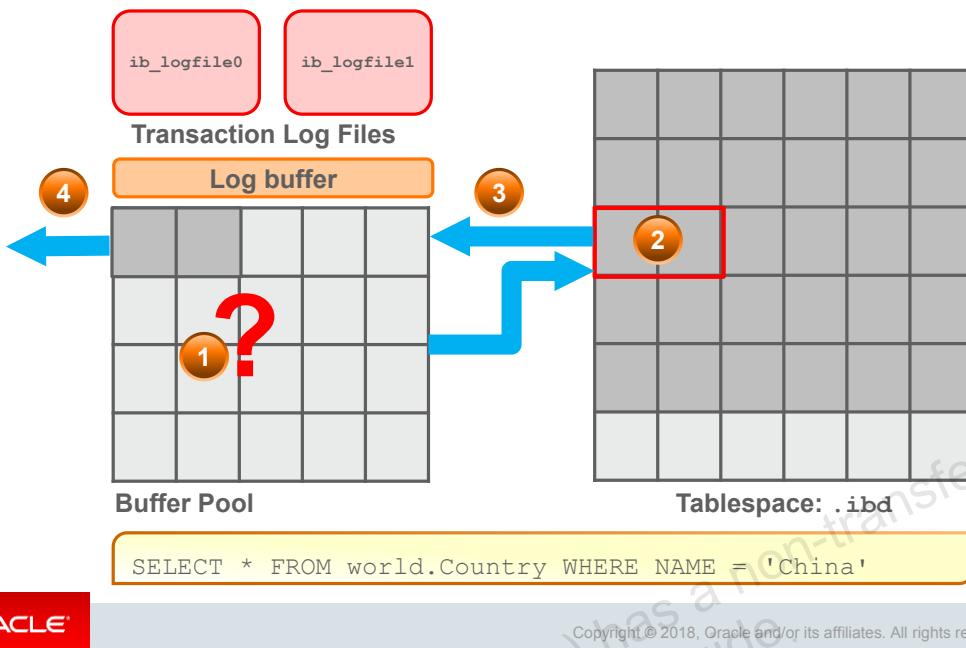
Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The operating system buffers changes to files before persisting them to disk. This allows it to reorder and merge data items in the buffer to apply them sequentially instead of randomly, which improves performance. If the machine fails, all the buffer changes are lost. InnoDB guarantees durability of data (the “D” in ACID) so InnoDB cannot rely entirely on the operating system and implements its own algorithms to manage the persistence of data.

If you use other storage engines, you must consider whether that storage engine ensures data consistency and durability. For example, the MyISAM storage engine relies on the operating system for caching and buffering and does not guarantee durability of the data. You can mitigate this by using a hardware RAID controller with its own battery-backed buffer.

Note: For more information on MyISAM, see <http://dev.mysql.com/doc/mysql/en/myisam-storage-engine.html>.

How InnoDB Reads Data



ORACLE

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

The diagram in the slide shows the basic operation of InnoDB at a very high level.

The tablespace is a file on disk where the table data and associated indexes are stored. By default, there is one `.ibd` tablespace file per table (`innodb_file_per_table=ON`).

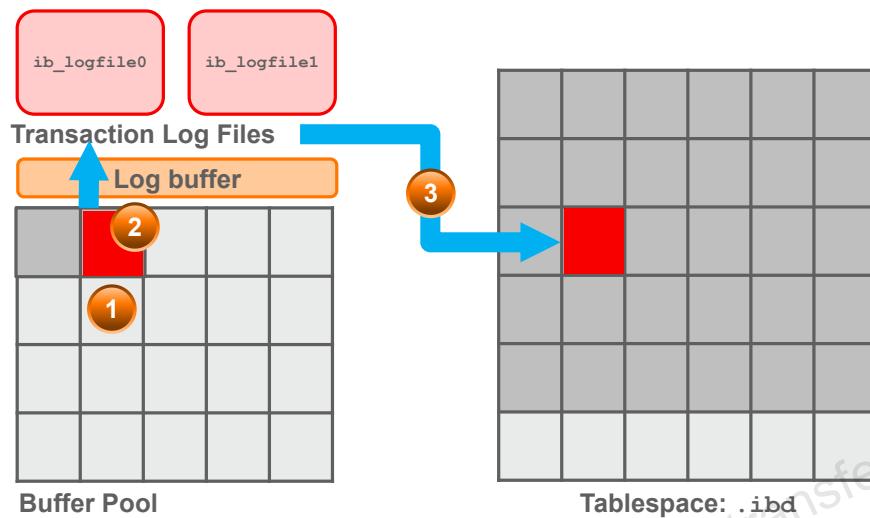
The buffer pool is a designated area of memory for caching what is in the tablespace.

The **transaction log files** record changes to the database and enable automatic crash recovery. In a default installation there are two transaction log files, called `ib_logfile0` and `ib_logfile1`.

When a user executes a read query, the following operations occur:

1. First, InnoDB checks to see if the data it needs is in the buffer pool.
2. If the required data is not in the buffer pool, InnoDB retrieves the data from the tablespace.
3. InnoDB puts the data in the buffer pool.
4. MySQL returns the results to the client.

How InnoDB Writes Data



```
UPDATE world.Country SET HeadOfState = 'Xi Jinping' WHERE NAME = 'China'
```

ORACLE

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

When an operation updates data, the following events occur:

1. The modification is made in-memory to the buffer pool. This results in a “dirty page” in the buffer pool.
2. The details of the modification are recorded in the log buffer, and the calling application is notified that the operation was successful. The log buffer files are written to the redo logs on disk when a commit occurs, or when the log buffer is full. At this point, the changes can be considered durable even though the tablespace is not yet updated.
3. The changes are eventually written to the tablespace, at an indeterminate time. When the changes from a particular dirty page are written to the tablespace, it is marked as clean, and the space it occupies in the buffer pool can be reclaimed if required. When the changes in the dirty pages are flushed to disk, this is known as a *checkpoint*. This step is performed as a background task, or during crash recovery.

This transaction log file I/O is sequential and results in much better performance than live updates to disk, but blocks other operations. The entries in the transaction log files are used to recover the state of the database in the event of a system crash, but if many modifications require syncing, this can be a lengthy operation.

By delaying the update to the tablespace on disk, MySQL can optimize the process by sorting and merging the updates.

Note: Other RDBMS sometimes refer to the transaction logs as “redo logs.”

Topics

- What is InnoDB?
- How does InnoDB work?
- Tuning InnoDB buffer pool and log files
- Measuring InnoDB performance



ORACLE®

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

Sizing the InnoDB Buffer Pool

- The `innodb_buffer_pool_size` variable is the most important InnoDB variable to tune.
 - Specifies the amount of memory available for caching InnoDB table data and indexes
 - Can be resized “online”, without a server restart
- A large buffer pool allows InnoDB to use memory instead of disk for most read operations.
 - Size the buffer pool to around 80% of the server’s RAM on InnoDB-only systems.
 - This is especially important for write-intensive workloads.
- For buffer pools larger than 1 GB, divide the buffer pool into separate instances.
 - Can improve concurrency and throughput
 - The number of instances is specified by the `innodb_buffer_pool_instances` server variable.
- The buffer size must be equal to, or a multiple of `innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances`.



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

Allocate as much of the server’s memory to the InnoDB buffer pool as you can without affecting the operating system, but scale back if necessary, as competition for physical memory might cause paging in the operating system. Write-intensive workloads also benefit from a large buffer pool, as data must be read before it is written.

When the InnoDB buffer pool is larger than 1 GB, the buffer pool is divided into the number of instances specified by the `innodb_buffer_pool_instances` server variable. This variable has no effect when the buffer pool is smaller than 1 GB.

Set the value of `innodb_buffer_pool_instances` so that each buffer pool instance is at least 1 GB in size. The default value is 8 (even on systems with buffer pools smaller than 1 GB) except on 32-bit Windows systems where it calculated based on the value of `innodb_buffer_pool_size`.

Partitioning the InnoDB buffer pool in this way can improve throughput substantially on machines with more than 16 cores, but it does not guarantee that each buffer pool instance is accessed equally.

Note that the buffer pool size is always equal to or a multiple of `innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances`. If you alter the buffer pool size to another value, the buffer pool size is automatically adjusted to a value that is not less than the specified buffer pool size.

Dumping and Restoring the Buffer Pool

- Large buffer pools take a long time to warm up.
 - Your applications must recreate the buffer pool by issuing queries, updates, and so on.
- Speed up server restarts by saving and then restoring the buffer pool.
 - MySQL has to only copy disk pages into memory.
 - It does not delay database startup.
- InnoDB startup options and system variables:
 - `innodb_buffer_pool_dump_at_shutdown`: Saves the buffer pool state at shutdown (enabled by default)
 - `innodb_buffer_pool_load_at_startup`: Restores the buffer pool at startup (enabled by default)
 - `innodb_buffer_pool_dump_pct`: Specifies the percentage of the most recently used pages for each buffer pool to read out and dump (25% by default)



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

- To save the buffer pool state while the server is running, issue the following statement:
`SET GLOBAL innodb_buffer_pool_dump_now=ON;`
- To restore the buffer pool state while the server is running, issue the following statement:
`SET GLOBAL innodb_buffer_pool_load_now=ON;`
- Display buffer pool dump and load status:

```
mysql> SHOW STATUS LIKE 'innodb_%status';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Innodb_buffer_pool_dump_status | not started |
| Innodb_buffer_pool_load_status | not started |
+-----+-----+
2 rows in set (0.00 sec)
```

Sizing the InnoDB Transaction Log Files

- The InnoDB transaction log files must be large enough to ensure that writes are fast and durable.
 - Larger log files reduce checkpoint flushes and disk I/O.
- If the transaction logs are too large, crash recovery can take a long time.
- The transaction logs should be large enough to store a maximum of one hour's worth of transactions.
 - Enough time for MySQL to optimize flushing
 - Reasonable crash recovery time
- The `innodb_log_file_size` server variable specifies the size (in bytes) of each log file in a log group.
 - `innodb_log_files_in_group` specifies the number of log files in a log group (default is 2).



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

The combined size of the transaction (redo) log is:

`(innodb_log_file_size * innodb_log_files_in_group)`.

This combined size cannot exceed the maximum value of slightly less than 512 GB. The default value of `innodb_log_file_size` is 48 MB, giving a default combined size of 96 MB, which is too small for most workloads.

To size your InnoDB transaction logs, calculate how many bytes are written to the transaction log per hour at peak times:

```
SELECT VARIABLE_VALUE INTO @baseline FROM
INFORMATION_SCHEMA.GLOBAL_STATUS
    WHERE VARIABLE_NAME = 'INNODB_OS_LOG_WRITTEN';
SELECT SLEEP(60 * 60);
SELECT VARIABLE_VALUE INTO @afteronehour FROM
INFORMATION_SCHEMA.GLOBAL_STATUS
    WHERE VARIABLE_NAME = 'INNODB_OS_LOG_WRITTEN';
SET @BytesWrittenToLog = @afteronehour - @baseline;
SELECT @BytesWrittenToLog / POWER(1024,2) AS MB_PER_HR;
```

For example, if the previous calculation gives the following result:

```
+-----+  
| MB_PER_HR |  
+-----+  
| 110.0827218 |  
+-----+
```

Then you can conclude that the server requires approximately 110 MB of log file space. Round it up to 128 MB and then divide by innodb_log_files_in_group (default is 2) to calculate a reasonable value for innodb_log_file_size:

```
innodb_log_file_size=64M
```

Another factor to take into account is that MySQL rejects a BLOB update that is large enough to write more than 10 percent of the total transaction log size. If you use BLOB data in your database, ensure that the transaction log is large enough so that no single update operation exceeds 10 percent of the log's combined size.

Other InnoDB Transaction Log Settings

- `innodb_flush_log_at_trx_commit` defines how often the log buffer is written out to the log file and when the flush to disk operation takes place.
 - 0 writes the log buffer to the log file and flushes the log file to disk every second (approximately).
 - 1 writes and flushes the log file at transaction commit.
 - 2 writes the log buffer to the log file after each commit and flushes the file to disk every second (approximately).
- `innodb_log_buffer_size` sets the size (in bytes) of the buffer that InnoDB uses to write to the log files on disk.



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

`innodb_flush_log_at_trx_commit`

- The default is 1 and this is required for InnoDB to be considered ACID compliant.
 - InnoDB's crash recovery works regardless of the setting.
 - This setting is slower than the others because of the I/O overhead of flushing to disk after every commit.
- Setting this variable to 0 or 2 can improve performance, but at the cost of losing approximately one second's worth of transactions in a crash.
 - You can change the InnoDB log flushing frequency from the default of one second by changing the value of the `innodb_flush_log_at_timeout` server variable. However, any `mysqld` process crash causes the loss of this number of second's worth of transactions.
- **Note:** For more details about this server variable setting, see the MySQL documentation at: http://dev.mysql.com/doc/refman/mysql/innodb-parameters.html#sysvar_innodb_flush_log_at_trx_commit.

InnoDB Tablespace Settings

- `innodb_file_per_table` defines whether InnoDB tables are created in per-table tablespaces.
 - Is set to `ON` (enabled) by default
 - Affects new tables that you create after changing the setting, but does not affect existing tables until you rebuild them
 - Can be overridden by using the `TABLESPACE` table property when creating or altering the table
- The advantages of enabling this setting include:
 - Support for compression, transportable tablespaces, and encryption at rest
 - Concurrent writes with `innodb_flush_method = O_DIRECT` even on `ext*` file systems
 - Ability to regain space per tablespace and use InnoDB symbolic links to migrate specific tables to another filesystem
 - Ability to monitor table size and date of last modification on the filesystem level



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

`innodb_file_per_table`

- If this server variable is enabled, InnoDB creates each new table by using its own `.ibd` file for storing data and indexes, rather than the shared tablespace.
- For solid state drive (SSD) systems, store the system tablespace on normal disk and store per-table tablespaces on SSD. This prevents the buffer and undo logs from wasting SSD write capacity with sequential writing.
- In general it is recommended to use one tablespace per table as it's the only option supporting all InnoDB features such as transportable tablespaces and encryption at rest.
- The disadvantages of enabling this setting include:
 - Increased filesystem fragmentation (even on Linux filesystems)
 - Slower table handling (creating and extending) because each table requires its own filesystem allocation.

innodb_autoextend_increment

- Default is 64 MB (not used for `innodb_file_per_table` tablespaces).
- A value that is too small results in many increments, each of which results in a period of reduced performance as the filesystem allocates that much space.
- A larger size helps the operating system allocate disk space in a nonfragmented way, but a value that is too large results in performance problems when the filesystem allocates a large part of the disk space.

Note: InnoDB table compression is covered in the lesson titled “Optimizing Your Schema.”

Thread Concurrency

- `innodb_thread_concurrency` defines the number of operating system threads that are allowed to run concurrently in InnoDB:
 - 0 is interpreted as infinite concurrency (no concurrency checking).
 - Most workloads run well without limiting the number of concurrent threads
- If a limit is used, adjust the value of `innodb_concurrency_tickets` from its default value of 5000 so that 99 percent of queries run with one allocation.
 - If `innodb_thread_concurrency=0`, there is no queue and therefore no need to give each query a “ticket”
- Symptoms of high thread concurrency include:
 - Very high CPU usage: This can be a sign of too many threads running (100+) and causing mutex contention in InnoDB. Limit thread concurrency, or use a thread pool.
 - Semaphore waits in the output of `SHOW ENGINE INNODB STATUS`.



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

`innodb_thread_concurrency`

- The default is 0, and the maximum is 1000.
- Scalability improvements in MySQL 5.5 and later reduce the need to limit the number of concurrently executing threads within InnoDB for servers with multi-core processors and a modern OS.

`innodb_concurrency_tickets`

- If a limit is set for the `innodb_thread_concurrency` server variable, it is important to set the `innodb_concurrency_tickets` server variable to reduce queue overhead.
- When a thread is permitted to enter InnoDB, it is given several “free tickets” equal to the value of `innodb_concurrency_tickets` and the thread can enter and leave InnoDB freely until it has used up its tickets. After all its tickets are used up, the thread again becomes subject to the concurrency check (and possible queuing) the next time it tries to enter InnoDB.

InnoDB Buffer and Log File Flushing

- `innodb_flush_method` specifies the operating system call used to flush both the data and log files.
- On Linux systems:
 - `fdatasync` uses `fsync()` and is the default setting.
 - `O_DSYNC` uses `O_SYNC` to open and flush the log files and `fsync()` to flush the data files (usually slow).
 - `O_DIRECT` uses `O_DIRECT` to open and flush the log files and `fsync()` to flush the data files.
- On Windows systems, the only available option is `async_unbuffered`.
- InnoDB uses doublewrite buffering by default.
 - To turn off doublewrite buffering, set `innodb_doublewrite=0`.



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

`innodb_flush_method`

- There have been problems with using `O_DSYNC` on many varieties of UNIX systems.
- `O_DIRECT` is available on some GNU/Linux versions, FreeBSD, and Oracle Solaris. It is usually a good choice when you have a hardware RAID controller with a battery-protected write-back cache.
 - `directio()` is used on Oracle Solaris systems.
- `fdatasync` (the default) is often the best choice for all other scenarios.
- Test all the options by using SysBench or another benchmarking tool.

Doublewrite Buffering

This flushing technique causes InnoDB to write pages to a special buffer called the doublewrite buffer before writing them to the data files. If the OS crashes during a page write, crash recovery can retrieve a good copy of the page from the doublewrite buffer. InnoDB optimizes this process, but consider disabling it if speed is more important than durability, or if you are using a filesystem that supports atomic writes, such as ZFS, BTRFS, or DIRECTFS. Similarly, if you use a journaling filesystem (for example if you mount ext4 with the `data=journal` option) consider disabling the doublewrite buffer.

`innodb_flush_neighbors`

By default, when InnoDB flushes a page from the buffer pool, it also flushes contiguous dirty pages within the same *extent* (`innodb_flush_neighbors=1`.) The extent is a group of pages that total one megabyte. The default page size is 16 KB, so an extent contains 64 pages. If you are storing data on an SSD, seek time is not relevant, so set `innodb_flush_neighbors=0`, which also reduces wear on the drive. SSDs often benefit from smaller page sizes. Set `innodb_page_size=4K`.

Adaptive Flushing

With adaptive flushing, InnoDB estimates the required rate of flushing, based on the speed of transaction log generation and the current rate of flushing.

- When InnoDB tries to flush everything from the buffer pool at the same time, this is known as “furious flushing”
 - Severely degrades performance
 - During furious flushing, SHOW ENGINE INNODB STATUS shows main thread state: flushing.
 - This status does not always indicate furious flushing.
- Adapting flushing:
 - Flushes small sets of modified pages from the buffer pool instead of all at the same time.
 - Used when `innodb_adaptive_flushing` is enabled (ON by default)



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

Furious flushing impacts performance because of the amount of disk I/O it causes. InnoDB's adaptive flushing algorithm tries to prevent this from occurring.

Furious flushing happens when the following condition is true:

```
(log file sequence number - last checkpoint) >  
((innodb_log_file_size * innodb_log_files_in_group) * 0.75 )
```

Disable adaptive flushing in systems with very high I/O loads, because it increases flushing when the server does more foreground I/O, and puts even more strain on the system. If you disable adaptive flushing, the `innodb_adaptive_flushing_lwm` setting controls the point at which it is automatically re-enabled. By default, this setting has a value of 10, which enables adaptive flushing when redo log usage reaches or exceeds 10 percent. Consider setting it to 60-70 percent, because otherwise the RAM allocated to the redo logs is being wasted.

Purge Behavior

- `innodb_purge_threads`: Defines the number of background threads devoted to the InnoDB purge operation
 - Running the purge operation in one or more background threads helps reduce internal contention within InnoDB, improving scalability.
- `innodb_max_purge_lag`: Controls how to delay INSERT, UPDATE, and DELETE options when purge operations are lagging
 - Purge operations might lag when you add or remove rows in small batches, resulting in many “dead” rows.
- `innodb_io_capacity`: Defines the maximum number of I/O operations per second that InnoDB background operations perform
- `innodb_io_capacity_max`: Is the limit up to which InnoDB can extend `innodb_io_capacity` in an emergency



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

Old page versions needed for read views are kept until they are no longer needed. Then, they are removed by the purge thread. The server can become I/O bound if these pages no longer fit in the buffer pool. Prevent this from happening by setting the `innodb_purge_threads`, `innodb_io_capacity`, and `innodb_io_capacity_max` server variables.

`innodb_purge_threads`

- The default is 4. The maximum is 32.
- If the value is set to 1, the purge operation is always performed by a background thread, never as part of the master thread.
- Monitor the history list length (in the Transactions section of the `SHOW ENGINE INNODB STATUS` report). The history contains records that can be purged and also records that cannot be purged because they are part of an ongoing transaction. You want to make sure that the records that can be purged are being purged.

`innodb_purge_lag`

If you insert and delete rows in smallish batches at about the same rate in the table, the purge thread can start to lag behind and the table can grow bigger and bigger because of all the “dead” rows, making everything disk-bound and very slow.

Do not raise the `innodb_max_purge_lag` server variable from its default of zero except perhaps for benchmarking, or extremely high loads with no long-running queries. When the list of transactions with index records marked for deletion by UPDATE or DELETE operations exceeds the length of `innodb_max_purge_lag`, InnoDB postpones further UPDATE and DELETE operations.

`innodb_io_capacity`

- Increasing this server variable allows purge operations to keep up with data modifications.
- If this variable's value is too high, InnoDB might prioritize background operations over foreground operations, which can severely impact performance.
- The default is 200. This is adequate for a few spinning disks and low-end SSDs. Use 100 for a single commodity spinning disk.
- On fast SSDs with a high write load, a value of 5000 might better represent the SSD's capacity and enable InnoDB to make better use of it. Benchmark this value carefully to ensure that background operations do not overwhelm foreground operations.

`innodb_io_capacity_max`

- The default value is 2,000 or twice the `innodb_io_capacity`, whichever is higher, and this is usually sufficient for most servers.
- For a few spinning disks and lower-end SSD the default is adequate.
 - For a single spinning disk, 200-400 is probably better. Reduce this for slow disks with low IOPS (I/O operations per second).
 - For high-end SSDs and bus-attached flash, consider setting a value of 15000.
- Use smaller values for systems with low write loads, and larger values with high write loads.
- Use the smallest value needed for flushing and purging to keep up.

Undo Logs

- Enabling the `innodb_undo_directory` option enables you to store undo logs outside the system tablespace.
- This improves performance:
 - Reduces the growth of the system tablespace
 - Enables parallel I/O if you place undo logs on separate fast storage
- Related settings are `innodb_undo_logs` and `innodb_undo_tablespaces`.



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

The undo logs contain a copy of a record before it was modified. You can improve performance moving these logs out of the system tablespace and into their own tablespace on an SSD. The `innodb_undo_directory` setting enables you to specify the new location of the undo logs.

The `innodb_undo_directory` and `innodb_undo_tablespaces` options cannot be changed on an existing MySQL server. You must select and set values for these options before you first launch InnoDB on a new MySQL server. To choose non-default values for these options on your existing database, you must create a new instance with your chosen option values and migrate your database to that new instance.

Recommended InnoDB Settings for OLTP and Benchmarking

- Include the following in your `my.cnf` or `my.ini` file:
 - `innodb_read_io_threads = 16`
 - `innodb_write_io_threads = 4`
 - `table_open_cache_instances = 16`
- Adjust the following where appropriate:
 - `innodb_log_file_size`
 - `innodb_flush_neighbors`
 - `innodb_page_size`
 - `innodb_io_capacity` and `innodb_io_capacity_max`
 - `innodb_checksum_algorithm`
 - `innodb_logged_compressed_pages`
 - `binlog_row_image`
 - `innodb_stats_persistent`



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

innodb_checksum_algorithm

By default, InnoDB writes a checksum along with data so that the data can be compared with the checksum during data validation and backups. InnoDB uses the `innodb` checksum algorithm by default, except in version 5.6.6 which defaulted to the `crc32` algorithm. When you enable the `crc32` algorithm, checksums of both types can coexist in the same database but over time, blocks gradually convert to `crc32` as they are modified.

When you use the `crc32` setting, InnoDB must identify the checksum algorithm that each block uses before validating that block. If you know that your system only uses `crc32` checksums, use the `strict_crc32` setting, which is faster because it does not identify the checksum algorithm of each block.

innodb_stats_persistent

By default, MySQL stores InnoDB index statistics created by `ANALYZE TABLE` on disk, only replacing them when `ANALYZE TABLE` is executed again or when 10% or more of the rows in the table have changed. If you disable `innodb_stats_persistent`, MySQL recalculates index statistics at other times, such as on server restart. This can lead to variations in the optimizer's query execution plans. If `innodb_stats_persistent` is disabled, you can enable it for specific tables by using the `CREATE TABLE` and `ALTER TABLE` statements with the `STATS_PERSISTENT` clause.

innodb_log_compressed_pages

Set to 0 if using compression. This avoids saving two copies of changes to the InnoDB log (one compressed, one not) and therefore reduces the number of log writes. This is particularly significant if the log files are on SSD or bus-attached flash, which is not recommended.

Note

- See the descriptions of the `innodb_log_file_size`, `innodb_flush_neighbors`, `innodb_page_size`, `innodb_io_capacity`, and `innodb_io_capacity_max` system variables earlier in this lesson for suggested values.
- For information about InnoDB table compression, see the lesson titled “Optimizing Your Schema.”
- Always benchmark before making changes to these settings permanent.

Recommended InnoDB Settings for Replication

- Single/Master server:
 - `innodb_flush_log_at_trx_commit = 1`
- Slave, server with no binlog, unsafe with binlog:
 - `innodb_flush_log_at_trx_commit = 0`
 - Can be set to safer setting if desired
 - `binlog off`
- Both master and slave:
 - Set `innodb_log_file_size` as large as possible.
 - Be aware that there is a minimum of two log files, and that this is the default setting.
 - `innodb_purge_threads = 1`



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

For the greatest possible durability and consistency in a replication setup using InnoDB with transactions, use `innodb_flush_log_at_trx_commit=1` and `sync_binlog=1` in your master server `my.cnf` file.

Linux Filesystem Recommendations

- Avoid using `ext2` or `ext3`.
- Mount `ext4` with the following options:
 - `rw, noatime, nodiratime, nobarrier, data=ordered`
- When using `XFS` or `ZFS` file systems:
 - Turn off `atime` (`noatime`) for the partition.
 - On `XFS`, also turn off `diratime` (`nodiratime`).
- Where possible, set `swappiness` to a value between 1 and 5, and set the deadline I/O scheduler.
- Consider disabling the InnoDB doublewrite buffer on any filesystem that supports atomic writes:
 - Such as `ZFS`, `BTRFS`, `DIRECTFS`, and `ext4` with `data=journal` option



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

The `ext2` and `ext3` filesystems lock inodes on writes, which can limit performance even with striped files. Also, `ext2` and `ext3` performance can be very slow with large numbers of files.

If you are using `ext4`, and do not have a battery-backed-up write caching disk controller, you can probably improve your write performance by as much as 50 percent by using the `ext4` option `data=journal` combined with the MySQL option `skip-innodb-doublewrite`. The `ext4` option provides similar protection against torn pages that the doublewrite buffer provides, but with less overhead. However, `ext4` is never completely safe, so take particular care when disabling doublewrite on `ext4` filesystems. A write caching controller is not likely to provide much of a performance benefit.

If you are using a block device with all hardware side caching disabled, or if you have a battery-backed write cache (BBU on RAID controller) then you can disable XFS or ZFS filesystem write barriers for the partition or mount point.

Topics

- What is InnoDB?
- How does InnoDB work?
- Tuning InnoDB buffer pool and log files
- Measuring InnoDB performance



ORACLE®

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

InnoDB Support in the Information Schema

The following Information Schema tables provide valuable information about InnoDB status:

- **INNODB_CMP** and **INNODB_CMP_RESET**
 - Contains status information about the operations related to compressed tables
- **INNODB_CMPPMEM** and **INNODB_CMPPMEM_RESET**
 - Contains status information about the compressed pages that reside in the buffer pool
- **INNODB_TRX**
 - Contains information about every transaction currently executing inside InnoDB
- **INNODB_LOCKS** and **INNODB_LOCK_WAITS**
 - Contains information about the locks that are blocking any transactions
 - These tables are deprecated in 5.7. Use the `sys` schema instead.



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

INNODB_CMP and **INNODB_CMP_RESET**

- These two tables have identical contents, but reading from the `INNODB_CMP_RESET` table resets the statistics on compression and uncompression operations.

INNODB_CMPPMEM and **INNODB_CMPPMEM_RESET**

- These two tables have identical contents, but reading from the `INNODB_CMPPMEM_RESET` table resets the statistics on page relocation operations. Page relocation occurs when the data exceeds the page size (because a row must be copied into a new page) and when two pages are merged (because their combined data can now be contained in one page).

INNODB_TRX

- Includes whether the transaction is waiting for a lock, when the transaction started, and which SQL statement the transaction is executing

INNODB_LOCKS

- Contains one row for each lock that is causing a wait in another transaction
- Includes the state of the transaction that holds the lock (`RUNNING`, `LOCK_WAIT`, `ROLLING BACK`, or `COMMITTING`)

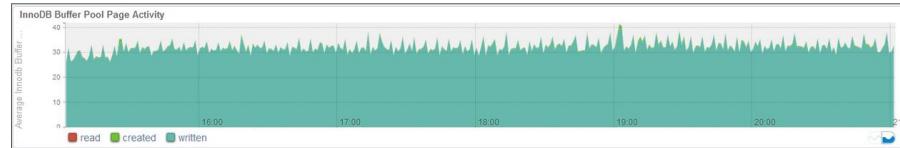
INNODB_LOCK_WAITS

- This table contains one or more rows for each blocked transaction, indicating the lock that it has requested and any locks that are blocking that request.
- The table is deprecated in 5.7. Use `sys.innodb_lock_waits` instead, as described in the slide titled “InnoDB Support in the Performance Schema.”

Note: For more information about tuning InnoDB table compression, see the lesson titled “Optimizing Your Schema.”

InnoDB Support in the Information Schema

- INNODB_BUFFER_POOL_STATS
 - Contains similar information to the buffer pool information shown in SHOW ENGINE INNODB STATUS
 - Is available in graphical format in MySQL Enterprise Monitor reports (not recommended for production systems)



- INNODB_BUFFER_PAGE
 - Contains information about each page in the buffer pool
 - Is available in graphical format in MySQL Enterprise Monitor reports
 - Can be very expensive to query when the buffer pool is large

ORACLE®

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

INNODB_BUFFER_POOL_STATS

- This table contains information the about total number of pages in buffer pool, the number of free pages, the number of pages that contain data, and dirty pages.

INNODB_BUFFER_PAGE

- This table contains an entry for each page in the buffer pool, including which pool it is part of, the number of threads using the page, and the table and index that the page belongs to. Be careful when you use this table on production systems. Querying this table can cause problems on servers that have large buffer pools and few buffer pool instances.

Information Schema InnoDB Metrics

- You can use the `INNODB_METRICS` table to monitor InnoDB metrics of interest.
 - `sys.metrics` provides even more information
- The `NAME` and `COUNT` columns give the monitor name and its value.
- Other columns provide extra information:
 - Status (running or stopped)
 - Maximum, minimum, and average values since monitoring began and when the counters were last reset
 - Type (whether value is incremental or a resource counter)
 - Timing information
- Monitors are grouped into modules.
 - For example, the `dml_inserts`, `dml_deletes`, and `dml_updates` monitors all belong to the `module_dml` module.
- Control monitoring by setting option variables.



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

The `INFORMATION_SCHEMA.INNODB_METRICS` table consolidates all InnoDB-related performance and resource-related counters and enables users to specify which counters they are interested in. Monitors are grouped into modules according to the InnoDB subsystem they belong to, which lets you enable, disable, or reset all monitors with a single command. Enable all metrics by setting `innodb_monitor_enable=all`.

<code>module_metadata (subsystem = metadata)</code>	<code>module_index (subsystem = index)</code>
<code>module_lock (subsystem = lock)</code>	<code>module_adaptive_hash (subsystem = adaptive_hash_index)</code>
<code>module_buffer (subsystem = buffer)</code>	<code>module_ibuf_system (subsystem = change_buffer)</code>
<code>module_buf_page (subsystem = buffer_page_io)</code>	<code>module_srv (subsystem = server)</code>
<code>module_os (subsystem = os)</code>	<code>module_ddl (subsystem = ddl)</code>
<code>module_trx (subsystem = transaction)</code>	<code>module_dml (subsystem = dml)</code>
<code>module_purge (subsystem = purge)</code>	<code>module_log (subsystem = recovery)</code>
<code>module_compress (subsystem = compression)</code>	<code>module_icp (subsystem = icp)</code>
<code>module_file (subsystem = file_system)</code>	

A better approach is to use `sys.metrics`, which combines `INNODB_METRICS` with `performance_schema.global_status` and memory information.

Example: Enabling InnoDB Monitoring

```
mysql> CREATE TABLE test.test_tbl (
    ->     id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
    ->     val INT NOT NULL
    -> );
Query OK, 0 rows affected (#.## sec)

mysql> SET GLOBAL innodb_monitor_enable = "dml_inserts"
Query OK, 0 rows affected (#.## sec)

mysql> INSERT INTO test VALUES(9);
Query OK, 1 row affected (#.## sec)
mysql> INSERT INTO test VALUES(10);
Query OK, 1 row affected (#.## sec)
mysql> INSERT INTO test VALUES(11);
Query OK, 1 row affected (#.## sec)
```



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

You start, stop, and reset counters by setting configuration options. The configuration options use either the name of the counter, the name of the module, a wildcard match for a module or counter name using the “%” character, or the special keyword `all`. The example in the slide uses the `innodb_monitor_enable` option to enable the `dml_inserts` counter.

Example: Displaying InnoDB Metrics

```
mysql> SELECT * FROM information_schema.innodb_metrics
-> WHERE NAME = "dml_inserts" \G
***** 1. row *****
      NAME: dml_inserts
    SUBSYSTEM: dml
        COUNT: 3
    MAX_COUNT: 3
    MIN_COUNT: NULL
    AVG_COUNT: 0.006060606060606061
  COUNT_RESET: 3
MAX_COUNT_RESET: 3
MIN_COUNT_RESET: NULL
AVG_COUNT_RESET: NULL
    TIME_ENABLED: 2017-08-07 09:23:55
TIME_DISABLED: NULL
    TIME_ELAPSED: 495
    TIME_RESET: NULL
      STATUS: enabled
        TYPE: status_counter
    COMMENT: Number of rows inserted
1 row in set (#.## sec)
```



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

Example: Resetting InnoDB Counters and Disabling Monitoring

```
mysql> SET GLOBAL innodb_monitor_reset = module_dml;
Query OK, 0 rows affected (#.## sec)
mysql> SELECT * FROM information_schema.INNODB_METRICS
-> WHERE NAME="dml_inserts"\G
***** 1. row *****
      NAME: dml_inserts
    SUBSYSTEM: dml
        COUNT: 3
    MAX_COUNT: 3
...
    COUNT_RESET: 0
MAX_COUNT_RESET: 0
MIN_COUNT_RESET: NULL
AVG_COUNT_RESET: 0
...
1 row in set (0.00 sec)

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



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

Setting the `innodb_monitor_reset` configuration option resets the `count_reset` and `max_count_reset` column values, but not the `count` and `max_count` values. You need to set `innodb_monitor_reset_all` to reset these values.

InnoDB Support in the Performance Schema

- The Performance Schema includes instrumentation for monitoring and troubleshooting InnoDB file I/O issues:
 - `/wait/io/file/innodb/innodb_data_file`: File I/O for central or single tablespace data files
 - `/wait/io/file/innodb/innodb_log_file`: File I/O for the redo logs
 - `/wait/io/file/innodb/innodb_temp_file`: File I/O against a sort merge file that is used when adding indexes
- The `mutex_instances` and `rwlock_instances` tables help you diagnose issues with *latches*.
 - Latches are the collective name for mutexes and RW-locks.
- The `sys schema innodb_lock_waits` view summarizes the InnoDB locks that transactions are waiting for. By default, rows are sorted in descending lock age.
 - Use this in preference to the Information Schema `INNODB_LOCK_WAITS` table, which is deprecated.



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

InnoDB File I/O

You can track how much time has been spent on these events by querying the `EVENTS_WAIT_SUMMARY_*` tables.

You can also track I/O use in terms of the amount of data involved by using the `FILE_SUMMARY_*` tables:

- `FILE_SUMMARY_BY_EVENT_NAME`: Summarizes file I/O events per instrumentation point
- `FILE_SUMMARY_BY_INSTANCE`: Summarizes file I/O events per file instance

Mutexes and RW-locks

Mutexes and RW-locks are collectively known as *latches*. A *mutex* is the low-level exclusive lock InnoDB uses to lock internal in-memory data structures. When a lock is acquired, all other processes are prevented from acquiring the same lock. An *RW-lock* is a shared lock on InnoDB internal in-memory data structures.

Monitoring InnoDB Performance in MySQL Workbench

InnoDB Buffer Stats by Schema							
Schema	Allocated	Data	Pages	Pages Hash	Pages Old	Rows Cached	
InnoDB System	2883584	2310033	176	6	5	2575	
mysql	425984	100543	26	0	19	1238	
sakila	327680	59412	20	0	20	1305	
jsontest	16384	197	1	0	0	1	
mark							
sys							

InnoDB Buffer Stats by Table							
Schema	Table	Allocated	Data	Pages	Pages Hash	Pages Old	Rows Cached
InnoDB System	SYS_TABLES	3276800	2900067	200	0	0	9858
mysql	innodb_index_stats	131072	64894	8	0	6	690
InnoDB System	SYS_COLUMNS	98304	55012	6	0	3	885
InnoDB System	SYS_INDEXES	81920	44693	5	0	1	617
mysql	innodb_table_stats	81920	35258	5	0	3	540
InnoDB System	SYS_DATAFILES	65536	29528	4	3	0	568
InnoDB System	SYS_FIELDS	65536	25516	4	0	1	648
InnoDB System	SYS_TABLESPACES	65536	28963	4	3	0	568



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

The images in the slide show sample outputs from the “InnoDB Buffer Stats by Schema” and “InnoDB Buffer Stats By Table” reports in MySQL Workbench. These reports are based on the sys schema’s innodb_buffer_stats_by_schema and innodb_buffer_stats_by_table views, respectively.

SHOW ENGINE INNODB STATUS

```
mysql> SHOW ENGINE INNODB STATUS\G
***** 1. row *****
Type: InnoDB
Name:
Status:
=====
2017-07-05 04:42:02 7f1988c28700 INNODB MONITOR OUTPUT
=====
Per second averages calculated from the last 4 seconds
-----
BACKGROUND THREAD
-----
srv_master_thread loops: 29 srv_active, 0 srv_shutdown, 97 srv_idle
srv_master_thread log flush and writes: 126
...
```



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

The SHOW ENGINE INNODB STATUS command provides detailed information about the state of the InnoDB storage engine. The command's output consists of a single row with multiple columns. The Status : column shows the time stamp, the InnoDB monitor name, and the number of seconds that per-second averages are based on. The status includes information about InnoDB, including the BACKGROUND THREAD: entry, which shows information about the work done by the main background thread, the `svr_master_thread`.

The following slides display the other columns in the output of SHOW ENGINE INNODB STATUS.

Note

- SHOW ENGINE INNODB STATUS can be misleading because the values displayed are collected at different times. The overhead required to obtain a global lock to provide consistent information throughout the report would itself produce a performance drag on the system. Comparing the results over a period of time provides more accurate and useful information to help you improve system performance. Do not execute it too frequently, because even though it does not acquire global locks, generating the status report consumes system resources.
- Although SHOW ENGINE INNODB STATUS provides useful “snapshot” information, it is difficult to parse. You cannot filter or reorder columns, or use it dynamically in prepared statements or stored procedures. The INFORMATION_SCHEMA and performance_schema databases store much of the same information and you can query their tables like any other database.

SHOW ENGINE INNODB STATUS: Semaphores and Transactions

```
...
-----
SEMAPHORES
-----
OS WAIT ARRAY INFO: reservation count 1048
OS WAIT ARRAY INFO: signal count 5813
Mutex spin waits 11158, rounds 9022, OS waits 183
RW-shared spins 2244, rounds 36680, OS waits 692
RW-excl spins 392, rounds 12305, OS waits 118
Spin rounds per wait: 0.81 mutex, 16.35 RW-shared, 31.39 RW-excl
-----
TRANSACTIONS
-----
Trx id counter 12610529
Purge done for trx's n:o < 8205659 undo n:o < 0 state: running but idle
History list length 496
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 12610453, not started
MySQL thread id 69, OS thread handle 0x7f19e80de700, query id 235692 localhost oracle freeing
items
SELECT ID, Name, Population FROM city_huge WHERE Name = 'Salerno'
...
...
```



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

Semaphores

The Semaphores section of the `SHOW ENGINE INNODB STATUS` output displays mutex information. A *semaphore* is a type of mutex.

This section displays:

- Mutex spin waits: The number of times a thread tried to get a mutex that was not available, so it waited in a spin wait
- rounds: The number of times threads looped in the spin wait cycle, checking the mutex
- OS waits: The number of times the thread gave up spin-waiting and went into the sleep state instead
- Spin rounds per wait: The number of spinlock rounds per OS wait for a mutex

If you have many threads waiting for semaphores, this may indicate lots of disk I/O, or contention problems inside InnoDB due to many concurrent queries, or problems in operating system thread scheduling. To improve such situations, limit the number of operating system threads available to InnoDB by configuring the `innodb_thread_concurrency` system variable. The default value of 0 does not limit operating system threads. The recommended starting value is equal to twice the number of CPUs plus the number of disks.

Transactions

- Purge done for trx's n:0 displays the number of purged transactions.
 - Uncommitted transactions can become stale and block the purge process.
- undo n:0 displays the undo log record number for the purge that is currently processing.
- History list length displays the number of transactions that have not been purged from undo space.
- Status of connection:
 - not started: No active InnoDB transaction is taking place with this connection.
 - ACTIVE: An active InnoDB transaction is taking place with this connection.
 - sleep: The transaction is delayed until a free slot is available in the InnoDB queue.
 - Adjusting the innodb_thread_sleep_delay setting may be necessary.
- Status of transaction can include “fetching rows,” “updating,” and other self-explanatory identifiers.

SHOW ENGINE INNODB STATUS: File I/O

```
...
-----
FILE I/O
-----
I/O thread 0 state: waiting for completed aio requests (insert buffer thread)
I/O thread 1 state: waiting for completed aio requests (log thread)
I/O thread 2 state: waiting for completed aio requests (read thread)
I/O thread 3 state: waiting for completed aio requests (read thread)
I/O thread 4 state: waiting for completed aio requests (read thread)
I/O thread 5 state: waiting for completed aio requests (read thread)
I/O thread 6 state: waiting for completed aio requests (write thread)
I/O thread 7 state: waiting for completed aio requests (write thread)
I/O thread 8 state: waiting for completed aio requests (write thread)
I/O thread 9 state: 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: 0; buffer pool: 0
4378 OS file reads, 34 OS file writes, 34 OS fsyncs
0.00 reads/s, 0 avg bytes/read, 1.00 writes/s, 1.00 fsyncs/s
...
```



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

The file I/O section provides information about the threads that InnoDB uses to perform various types of I/O. The first entries in the list are dedicated to general InnoDB processing. The section also displays information about pending I/O operations and statistics for I/O performance.

The number of read and write threads can be changed from the default value of four each by setting the `innodb_read_io_threads` and `innodb_write_io_threads` system variables.

SHOW ENGINE INNODB STATUS: Insert Buffer and Adaptive Hash Index

```
...
INSERT BUFFER AND ADAPTIVE HASH INDEX
-----
Ibuf: size 1, free list len 0, seg size 2, 0 merges
merged operations:
  insert 0, delete mark 0, delete 0
discarded operations:
  insert 0, delete mark 0, delete 0
Hash table size 276671, node heap has 603 buffer(s)
868983.25 hash searches/s, 48836.04 non-hash searches/s
...
```



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

The *insert buffer* and *adaptive hash index* are structures that InnoDB uses to optimize performance.

Insert Buffer

The insert buffer is a special structure maintained by InnoDB that reduces the amount of random disk I/O required to make changes to secondary indexes. If the secondary index page is already in the buffer pool, MySQL makes the change directly to the index page. If the index page is not in the buffer pool, MySQL stores the change in the insert buffer and periodically flushes the buffer changes to disk. Despite its name, the insert buffer is used not only when an index record is inserted, but also when it is marked for deletion, or deleted from a non-unique secondary index.

Adaptive Hash Index

The adaptive hash index is a feature of InnoDB that allows it to behave like an in-memory database for certain workflows if the InnoDB buffer pool is large enough. If an InnoDB table fits entirely within the buffer pool, and MySQL determines that queries will benefit from it, the server generates an adaptive hash index. The adaptive hash index is based on a prefix of the index key queries are using on that table to optimize lookups. MySQL does this automatically if the `innodb_adaptive_hash_index` option is enabled (it is `ON` by default). For some workloads, particularly write-intensive workloads, the adaptive hash index might become a bottleneck and dramatically reduce the server's overall performance. You can identify such bottlenecks in the output of `SHOW ENGINE INNODB STATUS` by the presence of semaphore waits in the file `btr0sea.cc`.

SHOW ENGINE INNODB STATUS: Log

```
...  
---  
LOG  
---  
Log sequence number 4373830059  
Log flushed up to 4373829789  
Pages flushed up to 4373820571  
Last checkpoint at 4373820571  
0 pending log writes, 0 pending chkp writes  
37 log i/o's done, 1.00 log i/o's/second  
...
```



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

The Log section displays information about the InnoDB transaction logs, including:

- Current log sequence number
- What proportion of the logs have been flushed to disk
- Position of the last checkpoint

SHOW ENGINE INNODB STATUS: Buffer Pool and Memory

```
...
-----
BUFFER POOL AND MEMORY
-----
Total memory allocated 137363456; in additional pool allocated 0
Dictionary memory allocated 59952
Buffer pool size     8191
Free buffers        3224
Database pages      4364
Old database pages  1630
Modified db pages   1
Pending reads       0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young    0, not young 0
0.00 youngs/s, 0.00 non-youngs/s
Pages read 4364, created 0, written 1
0.00 reads/s, 0.00 creates/s, 0.00 writes/s
Buffer pool hit rate 1000 / 1000, young-making rate 0 / 1000 not 0 / 1000
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s
LRU len: 4364, unzip_LRU len: 0
I/O sum[0]:cur[0], unzip sum[0]:cur[0] ...
```



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

- Includes the following information:
 - Total memory allocated by InnoDB
 - Amount of memory allocated in additional pool memory
 - Total number of pages in buffer pool
 - Number of free pages
 - Pages allocated by database pages
 - Dirty pages
- Provides valuable insight into how well your buffer pool is sized:
 - If there are numerous pages free, you may need to reduce your buffer pool size (`innodb_buffer_pool_size`).
- Calculates buffer pool hit rate:
 - 1000/1000 identifies a 100 percent hit rate.
 - Slightly lower hit rate values may be acceptable.

SHOW ENGINE INNODB STATUS: Row Operations

```
...
-----
ROW OPERATIONS
-----
0 queries inside InnoDB, 0 queries in queue
Main thread process no. 3004, id 7176, state: purging
Number of rows inserted 3738558, updated 127415, deleted 33707, read 755779
1586.13 inserts/s, 50.89 updates/s, 28.44 deletes/s, 107.88 reads/s
...
```



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

- InnoDB thread queue status:
 - Displays the number of threads that are inside InnoDB and active
 - Displays the number of threads that are currently waiting in the InnoDB queue
 - If `innodb_thread_concurrency = 0`, then InnoDB reports 0 queries inside InnoDB, 0 queries in queue.
- Read views open inside InnoDB displays the number of transactions that are using an MVCC snapshot for consistent reads.
- The state of the main InnoDB thread:
 - Controls the scheduling of several system operations
 - The values are self-explanatory.
- Number of row operations affected by INSERT, UPDATE, DELETE, and SELECT statements on InnoDB tables.
 - These values are counted from system startup.
 - Also displayed is the average number of row operations for each transaction.

SHOW ENGINE INNODB STATUS: Latest Foreign Key Error

```
-----  
LATEST FOREIGN KEY ERROR  
-----  
030709 13:00:59 Transaction:  
TRANSACTION 0 290328284, ACTIVE 0 sec, process no 3195  
inserting  
15 lock struct(s), heap size 2496, undo log entries 9  
MySQL thread id 25, query id 4668733 localhost heikki update  
insert into ibtest11a (D, B, C) values (5, 'khDk', 'khDk')  
Foreign key constraint fails for table test/ibtest11a:  
'  
CONSTRAINT `0_219242` FOREIGN KEY (`A`, `D`) REFERENCES `ibtest11b` (`A`,  
`D`) ON DELETE CASCADE ON UPDATE CASCADE  
Trying to add in child table, in index PRIMARY tuple:  
0: len 4; hex 80000101; asc ....;; 1: len 4; hex 80000005; asc ....;; 2:  
len 4; hex 6b68446b; asc khDk;; 3: len 6; hex 0000114e0edc; asc ...N...;; 4:  
len 7; hex 00000000c3e0a7; asc .....;; 5: len 4; hex 6b68446b; asc khDk;;  
But in parent table test/ibtest11b, in index PRIMARY,  
the closest match we can find is record:  
RECORD: info bits  
...  
...
```



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

The Latest Foreign Key section only appears in the output if a foreign key error has occurred. The contents include the statement that failed as well as information about the constraint that was violated and the referenced and referencing tables.

SHOW ENGINE INNODB STATUS: Latest Detected Deadlock

```
-----  
LATEST DETECTED DEADLOCK  
-----  
030709 12:59:58  
*** (1) TRANSACTION:  
TRANSACTION 0 290252780, ACTIVE 1 sec, process no 3185  
inserting  
LOCK WAIT 3 lock struct(s), heap size 320, undo log entries 146  
MySQL thread id 21, query id 4553379 localhost heikki update  
INSERT INTO alex1 VALUES(86, 86, 794,'aA35818','bb','c79166','d4766t',  
'e187358f','g84586','h794',date_format('2001-04-03 12:54:22','%Y-%m-%d  
%H:%i'),7  
*** (1) WAITING FOR THIS LOCK TO BE GRANTED:  
RECORD LOCKS space id 0 page no 48310 n bits 568 table test/alex1 index  
symbole trx id 0 290252780 lock mode S waiting  
Record lock, heap no 324 RECORD: info bits 0 0: len 7; hex 61613335383138;  
asc aa35818;; 1:  
*** (2) TRANSACTION:  
TRANSACTION 0 290251546, ACTIVE 2 sec, process no 3190  
inserting  
130 lock struct(s), heap size 11584, undo log entries 437  
MySQL thread id 23, query id 4554396 localhost heikki update ...
```



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

The Latest Detected Deadlock appears in the output only if a deadlock has occurred. It shows which transactions were involved, the statement each was attempting to execute, the locks they have and need, and which transaction InnoDB decided to roll back to break the deadlock.

Enable the `innodb_print_all_deadlocks` option to record information about each deadlock in the error log. This is useful if you cannot execute `SHOW ENGINE INNODB STATUS` after each deadlock, and because the output contains only information about deadlocks.

Quiz



Which section of the `SHOW ENGINE INNODB STATUS` report displays the state of the main InnoDB thread?

- a. Buffer pool and memory
- b. File I/O
- c. Log
- d. Row operations



ORACLE®

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

Answer: d

Summary



In this lesson, you should have learned how to:

- List the key benefits of the InnoDB storage engine
- Describe how InnoDB uses log files and buffers
- Explain the `SHOW ENGINE INNODB STATUS` output
- Access key InnoDB metrics in the Information Schema
- Tune InnoDB settings for best performance

ORACLE®

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

Practices

- 7-1: Investigating the effects of log files on transactions
- 7-2: Using SHOW_ENGINE_INNODB_STATUS
- 7-3: Monitoring InnoDB Metrics in the Information Schema
- 7-4: Evaluating InnoDB Buffer Pool Size



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

8

Optimizing Your Schema

ORACLE®



MySQL™

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

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

Topics

- Schema design considerations
- Indexes
- InnoDB table compression
- Partitioning tables



ORACLE®

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

Objectives



After completing this lesson, you should be able to:

- List important factors to consider when designing a schema
- Describe how normalization can affect performance
- Use appropriate data types to improve performance
- Use indexing to improve performance
- Describe the different types of indexes
- Identify InnoDB tables as suitable candidates for compression
- Use partitioning to improve query performance

ORACLE®

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

Topics

- Schema design considerations
 - Indexes
 - InnoDB table compression
 - Partitioning tables



ORACLE®

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

Schema Design Considerations

- Reports or output that the database will be responsible for:
 - Identify queries that must be executed.
- Type of data that must be collected to create the reports or output:
 - Determine data types and sizes.
 - Identify relationships between data items.
- Best way to store the data:
 - Not all data should be stored in the database.



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

When designing a schema, you must first understand what problem your application solves, and how users want to interact with it. When you know what the outputs are, you can consider the data requirements:

- Size and type of the data to store
- Relationships between data items
- Cardinality (or “uniqueness”) of the data items

Not all data should be stored in the database:

- Storing images and other binary files outside of the database can lead to better performance.
- It is often best to let your application handle XML or other data formats instead of storing them in the database, because such data might be opaque to MySQL and cannot be efficiently indexed or normalized.

Schema Design Tasks

- Normalize the data.
 - Eliminates redundant data
 - Provides flexible access to data
 - Preserves data integrity
- Choose the correct data types and size.
 - Improves performance
 - Protects data
- Create efficient indexes.
 - Improves query performance



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

Normalizing the data

There are several normalization levels that you can work through when determining the best way to store your data. However, over-normalizing is just as bad as under-normalizing data. Remember that the main goals of normalization are to get accurate data to the end user as quickly as possible and to make the maintenance of that data as easy as possible. If you are normalizing for the sake of normalizing, you risk over-normalizing your database.

Data types and size

Choosing the correct data type and size is a very important step in the schema design process. How data is stored internally affects how well the database performs. If your choice of data type allocates significantly more space to a data item than it requires, it can cause an exponential growth in file system and memory requirements as your database grows. Similarly, if you compare values that are not of the same data type (for example comparing an `int` to a `varchar`, or a `varchar` to a `datetime`), the operation is slower than if the values being compared are of the same type.

Creating efficient indexes

A good indexing strategy results in better query performance. Under-indexing a table can result in full table scans, while over-indexing a table can result in MySQL having to maintain multiple, possibly redundant, indexes, and the optimizer must consider those indexes every time it generates a query plan.

Normalization and Performance

- Normalized data:
 - Data is spread across many smaller tables.
 - It is easy to update (no duplication of data).
 - Most queries involve table joins, which are expensive.
 - The optimizer has fewer choices for data sorting and retrieval.
- Denormalized data:
 - Tables might be large.
 - Updates might be complex and costly due to duplication of data.



ORACLE

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

While there are significant benefits to normalizing data, there are performance implications and normalization is not the best approach for all applications.

Normalized databases perform well in write-intensive applications where the write load is greater than the read load, for example, an online transaction processing application. This is because normalized data is highly nonredundant so `UPDATE` operations only modify single values instead of multiple copies. Also, in a normalized database, each row tends to be smaller than in a non-normalized database, so `INSERT` and `DELETE` operations require less I/O than on a typical denormalized database.

Denormalized databases perform well in read-intensive applications, for example, in decision support systems. This is because denormalized databases contain redundant copies of data in tables that have larger row sizes so that queries can operate on fewer tables to get the same results.

Denormalizing Specific Data for Performance

- Denormalization involves duplicating selected columns from one table in another table.
- Denormalizing some data can result in performance gains.
 - Even if a full-table scan is required and the data does not fit in memory, this is faster than a join because it avoids random I/O.
- Using a single table might allow more efficient indexing.
- Denormalized tables can simulate “materialized views.”
- Consider the cost of updates before denormalizing data.



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

Most applications have a mixture of read and write loads, so you can improve performance by combining normalized and denormalized data in your database.

For example, consider an order processing application with `products` and `orders` tables. Usually, you would have only the `product_id` in the `orders` table and store all the information about the product in the `products` table. But if you have a query that searches for products by name and sorts the results by order date, it is inefficient because `product_name` and `order_date` are stored in different tables and must be retrieved by using a join:

```
SELECT product_name, order_date
  FROM orders INNER JOIN products ON product_id
 WHERE product_name LIKE '%widget%'
 ORDER BY order_date DESC
```

In this example, MySQL must scan the `order_date` index on the `orders` table and then compare the corresponding `product_name` in the `products` table to see if the name includes the word “widget.” The join does not allow sorting and filtering at the same time.

Improve the performance of this query by denormalizing the schema so that the `orders` table also includes the `product_name` column. Add an index on the `product_name` and `order_date` columns. You can then execute the query efficiently without a join:

```
SELECT product_name, order_date
FROM orders
WHERE product_name LIKE '%widget%'
ORDER BY order_date DESC
```

Materialized views

MySQL does not support materialized views. In other RDBMS, a materialized view is a view that is precomputed and stored on disk to improve query efficiency. A materialized view is not updated when the source data changes.

Creating a table that contains denormalized data at regular intervals can simulate a materialized view in MySQL:

- Create a normal view based on a `SELECT` query.
- Create a table based on the view with the required indexes.
- Run the following commands at regular intervals:

```
LOCK TABLES mView WRITE;
TRUNCATE mView;
INSERT INTO mView SELECT * FROM nView;
UNLOCK TABLES;
```

Quiz



What are some of the advantages of using denormalized data in a database application?
(Choose all that apply.)

- a. Redundancy is eliminated.
- b. Using denormalized data can allow more efficient indexing strategies.
- c. Denormalized tables can simulate materialized views.
- d. Data is easier to update.



ORACLE®

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

Answer: b, c

Data Types

- Use the correct data type for the data being stored.
- Assign an appropriate length for the data type.
 - Overallocating space for data items degrades performance.
- Use NOT NULL in columns that do not store NULL values.
- Use an appropriate length for character data types.
 - Use VARCHAR when the maximum column length is much larger than the average column length.
 - Use CHAR for very short strings, or if the strings are all a similar length.
 - Use a single-byte encoding such as latin1 instead of utf8 for best performance.
- Consider using BLOBS for long rows.
 - Use compression for BLOB and TEXT data types.



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

- Store true numeric values as numbers, not as strings. This is important for comparison. For example, "09" and "9" represent the same number but are different strings.
- Items such as postal codes and telephone numbers are not true numeric values and should be stored as strings.
- Specify NOT NULL on a column unless you intend to store NULL values. Indexing and value comparisons are more costly with NULL values.
- Store date values as dates and not strings.
- Use a single-byte encoding such as latin1 if you do not need extended characters. The utf8 character set is variable-length, so functions such as CHAR_LENGTH() must parse the entire string to get the number of characters. However, each single-byte encoding supports a fixed and limited number of characters, which can cause compatibility issues if your application must support multiple languages. Use utf8 for flexibility and internationalization, if not for performance.
- Consider using BLOBS for long rows. The BLOB column is only read if it is part of the SELECT statement. Otherwise, it is ignored. BLOB data is stored in a different set of pages from other row data, so that when you query a row that contains BLOB data but do not request the BLOB data, fewer page reads occur than if you had queried the entire row.
- Use compression if tables are large or contain lots of repetitive text or numeric data. This reduces the amount of disk I/O required to bring data into the buffer pool, or to perform full table scans.

Topics

- Schema design considerations
- **Indexes**
- InnoDB table compression
- Partitioning tables



ORACLE®

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

Indexes

- Indexes speed up data retrieval and sorting and grouping operations.
- Minimize the number of indexes on a column.
 - Ideally, use only one index for a column.
- Apply indexes only on columns with high selectivity.
- Define indexes on truly unique values as UNIQUE.
- Consider the order of columns in a composite index.
 - The optimizer uses the index to find values in the leftmost columns in the order that they appear in the index definition.
- Remove unused indexes.
 - Unused indexes must still be maintained during DML operations.
 - Identify by querying sys.schema_unused_indexes.



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

Indexes speed up reads and lookups either by index-only reads for SELECT statements or by the WHERE clause for SELECT/UPDATE/DELETE statements. Indexes also improve sorting and grouping performance by avoiding the need to create temporary tables. However, indexes increase the time for write operations (INSERT/UPDATE/DELETE statements) because MySQL must update all the indexes.

Indexes add value only when the columns they are based on have a high selectivity ("uniqueness" of values). Indexing a gender column where the only permissible values are "M" and "F" does not improve performance.

Composite Indexes

When you create an index on two or more columns, you can use only the first column of the index on its own, so the order of columns is important. Example:

KEY (a,b)

- a=4 uses the index.
- a=4 AND b=5 uses the index.
- b=5 does not use the index.

Indexing Unique Values

If a column contains unique values, declaring a unique index has performance advantages over a nonunique one. For example, SELECT COUNT(*) is faster for that table.

Calculating Index Selectivity

The selectivity of an index is defined as the percentage of rows in a table with the same value for the indexed column. An index selectivity of 33 percent or better is preferred.

Example 1:

- `SELECT COUNT(DISTINCT(given_name)) FROM electorate` results in 28,920. This is the **cardinality** of that column.
- `SELECT COUNT(given_name) FROM electorate` results in 68,700, the number of rows in the table that contain a value for the `given_name` column.

The selectivity of the index on the `given_name` column is the proportion of distinct values (cardinality) to the number of rows, which in this case is calculated as $28,920/68,700$, or 42%.

Example 2:

- `SELECT COUNT(gender) FROM electorate` results in 68,700.
- `SELECT COUNT(DISTINCT(gender)) FROM electorate` results in 2.

The selectivity of the index on the `gender` column is calculated as $2/68,700$, which is 0.003 percent.

You can see the estimated cardinality of an existing index by using `SHOW INDEXES`. Example:

```
mysql> SHOW INDEXES FROM City\G
***** 1. row *****
      Table: City
     Non_unique: 0
        Key_name: PRIMARY
      Seq_in_index: 1
     Column_name: ID
       Collation: A
   Cardinality: 3839
   ...
***** 2. row *****
      Table: City
     Non_unique: 1
        Key_name: CountryCode
      Seq_in_index: 1
     Column_name: CountryCode
       Collation: A
   Cardinality: 479
   ...
2 rows in set (0.00 sec)
```

Indexing

- Use covering indexes where possible.
 - Covering indexes return query results without reading the original table.
- Prefix an index if the data is selective enough by the first few characters.
- Use short keys to improve index performance.
 - Use integer data types where possible.
- Key values that are close to each other produce the best performance.
 - An indexed AUTO_INCREMENT field results in better performance than an indexed UUID() field.



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

Covering indexes

If a SELECT query references only those fields that are contained entirely within the index, MySQL does not have to access the original table to return the query results. Example:

```
... UNIQUE KEY 'rental_date' ('rental_date', 'inventory_id',
    'customer_id') ...
SELECT inventory_id, customer_id FROM rental
WHERE rental_date > '2005-05-25'
```

The query returns all the required data without having to read the full row.

Prefix indexes

If the first few characters of a long character column are sufficiently unique, specify a prefix length for the index on that column. This reduces the size of the index on disk and when it is stored in memory it results in better performance. Monitor the cardinality of a prefix index, because it might change over time. For example, if you have a 4 character prefix index on a CHAR(50) userid column and start adding xyz to the beginning of every user's ID, the optimizer might decide that there is no real benefit to using the index and perform a full table scan instead.

Note: This course covers the query optimizer in the lesson titled “Optimizing Queries.”

Optimizing Indexes

- Executing `OPTIMIZE TABLE` compacts and sorts indexes.
 - Performs `ALTER TABLE` and `ANALYZE TABLE` operations.
 - Rebuilds the table with current index statistics
 - Reduces fragmentation within the tablespaces and on disk
- Executing `ANALYZE TABLE` updates the key distribution statistics.
 - MySQL uses the key distribution statistics to assist with:
 - Ordering joined tables
 - Choosing the most efficient index for a query
 - Statistics are persisted by default.
 - (`innodb_stats_persistent=ON`)
 - InnoDB updates these statistics automatically after a significant quantity of table data changes.



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

OPTIMIZE TABLE

The benefits of executing `OPTIMIZE TABLE` depend on the data in the table and its storage engine. Some tables benefit more than others, and sometimes performance degrades over time until you optimize the table again. The operation can be slow for large tables, or if the indexes that are being rebuilt do not fit into the buffer pool. The first execution of `OPTIMIZE TABLE` after adding a lot of data is often slower than later executions. `OPTIMIZE TABLE` uses online DDL (`ALGORITHM=INPLACE`), locking the table only briefly and minimizing down time while the table is rebuilt. Tables with `FULLTEXT` indexes cannot be optimized online.

ANALYZE TABLE

By default, InnoDB persists the statistics generated by `ANALYZE TABLE` to disk to avoid them being re-created when the server restarts. This makes the execution plan for each query consistent. You can change this for each table with the `STATS_PERSISTENT` clause of the `CREATE TABLE` and `ALTER TABLE` commands.

InnoDB updates index statistics automatically by default, according to the settings of `innodb_stats_persistent` and `innodb_stats_auto_recalc`. You can also enable `innodb_stats_on_metadata` to cause InnoDB to update statistics when you run metadata statements such as `SHOW TABLE STATUS` and `SHOW INDEX`.

Index Types

- B+TREE is the default index type for InnoDB.
 - Good for equal and range queries, and sorting operations
- The adaptive hash index (AHI) is enabled by default.
 - InnoDB behaves like an in-memory database for certain workloads if the buffer pool is large enough.
- FULLTEXT improves the performance of natural language searches.
 - Each string is decomposed into words.
 - InnoDB batches individual word INSERTs.



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

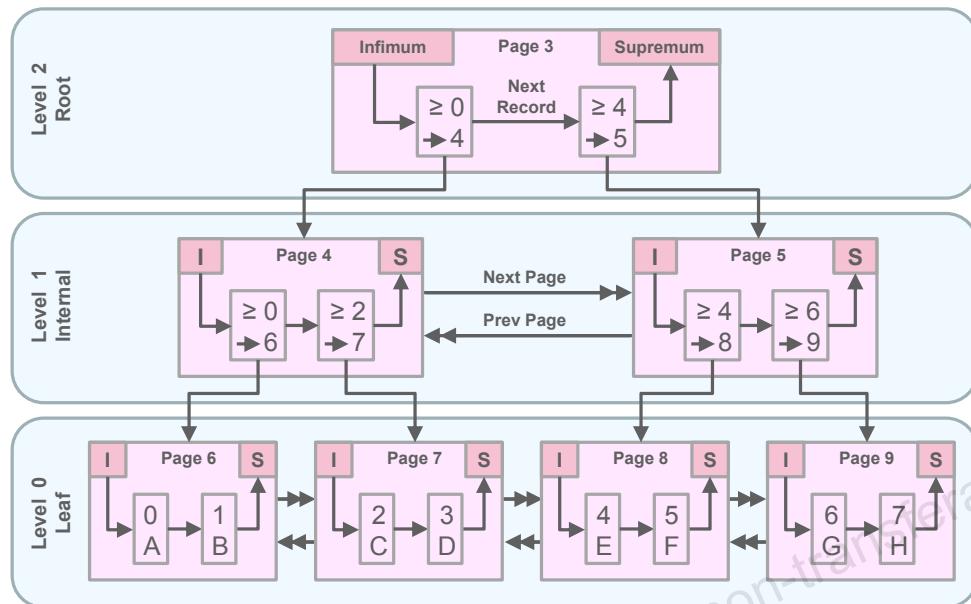
Adaptive Hash Index

The adaptive hash index is enabled by default. You can turn it off by using the `innodb_adaptive_hash_index` option. InnoDB monitors index searches and, if it determines that the workload and buffer pool size are suitable, it builds a hash index to improve performance. If a table fits almost entirely in main memory, a hash index speeds up queries by using the index value as a pointer to enable a direct lookup of any element. The hash index is always built based on an existing BTREE index on the table. Benchmark before disabling this feature.

FULLTEXT Index

A FULLTEXT index stores a list of words, and for each word, a list of documents that the word appears in. To support proximity search, InnoDB stores position information for each word as a byte offset. During an `INSERT` operation on a column with a FULLTEXT index, each string value is decomposed into individual words before insertion. InnoDB batches these inserts to improve performance. You can tune the size of the cache that InnoDB uses for this operation by setting `innodb_ft_cache_size_per_table`, or `innodb_ft_total_cache_size` to set a global limit for all tables.

B+TREE Index



ORACLE®

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

InnoDB stores table and index data in B+trees. A B+tree is an efficient structure for storing data on block devices such as hard disks. It is a variable-depth tree, with leaf pages containing column data at the lowest level, and non-leaf pages containing references to other pages. Each index contains a single root-level page that InnoDB identifies in the data dictionary for each table. Indexes might also contain internal non-leaf pages.

Pages within each level form a doubly-linked list so that each page can point to the previous and next pages at that level.

Each page contains keyed values. In leaf pages, the key identifies row data. On non-leaf pages, the key identifies other pages. Each page also contains *infimum* and *supremum* boundary pseudovalues, which refer to the gaps before and after the keys in that page respectively. This enables each page to contain a next-record linked list from before the first key to after the last key in that page, without explicitly defining the boundary values.

In the example in the slide, the leaf nodes contain row data that could be represented by the following statement:

```
INSERT INTO table VALUES (0, 'A'), (1, 'B'), (2, 'C'), (3, 'D'),
(4, 'E'), (5, 'F'), (6, 'G'), (7, 'H');
```

Each key, such as 0 or 1, has an associated row value, such as A or B. In the slide, pages each contain only two rows and each row contains only a single non-key column, but a real InnoDB page might contain hundreds of rows, each of which might contain multiple columns.

Non-leaf pages do not contain row data. Rather, they identify pages that are lower in the tree by specifying the first key in that page. For example, the root page in the slide shows that key 0 is the first key on page 4. Page 4 shows that key 2 is the first key on page 7, and page 7 contains row data for key 2 and key 3.

Example:

When you issue a primary-key-based query such as `SELECT * FROM table WHERE key = 5` against the InnoDB table represented in the slide, InnoDB finds the row for the key 5 as follows:

- The root page contains two page references, with first keys of 0 and 4 respectively.
- Key 5 (the key being searched) is greater than or equal to key 4 but less than the supremum, which is the pseudovalue that comes after any other key in the page.
- Key 4 is the first key on page 5.
- Page 5, a non-leaf page, refers to two pages (8 and 9) that have first keys of 4 and 6 respectively. Key 5 is less than 6, so if it exists it must be on page 8.
- Page 8, a leaf page, contains key 5, and the row data for that key is 'F'.

When you issue a query that results in a table or range scan, InnoDB traverses the linked list that connects each leaf page, and uses the next-record linked list within each page to read each row.

Quiz



Executing which of the following commands compacts and sorts indexes on the `city` table?

- a. `ANALYZE TABLE city;`
- b. `ORDER INDEX city;`
- c. `SHOW TABLES LIKE 'city';`
- d. `OPTIMIZE TABLE city;`



ORACLE

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

Answer: d

Topics

- Schema design considerations
- Indexes
- InnoDB table compression
- Partitioning tables



ORACLE®

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

InnoDB Table Compression

- Compressing table data:
 - Reduces the size of tables
 - Reduces disk I/O
 - Increases CPU utilization
- Compress an InnoDB table in a CREATE TABLE or ALTER TABLE statement, specifying:
 - ROW_FORMAT=COMPRESSED
 - (Optionally) a page size in the KEY_BLOCK_SIZE parameter
 - Smaller page sizes provide I/O benefits.
 - If pages are too small, overhead is incurred when pages are reorganized to accommodate multiple rows.
- Ensure that the buffer pool can accommodate both compressed and uncompressed pages.



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

Compression improves disk I/O but increases CPU utilization, because the data must be compressed when stored and uncompressed for retrieval. It is particularly effective in systems with read-intensive workloads and enough RAM to hold frequently used data in memory.

You can compress tables only when `innodb_file_per_table` is enabled (ON by default), and you should ensure that `innodb_file_format=Barracuda` (the default file format.)

The default size of a compressed table is half the value of the `innodb_page_size` system variable. Valid page sizes are 1 KB, 2 KB, 4 KB, 8 KB, or 16 KB, but 16 KB is the standard page size, so using this setting will not result in significant compression.

Smaller pages are quicker to read from and write to disk, especially when their data is stored on an SSD device. However, the smaller the page size, the less data it can contain, so InnoDB has to reorganize data more frequently and performance degrades.

The InnoDB buffer pool contains the compressed pages and a standard sized (16 KB) page for the uncompressed data. When changes are made to the uncompressed page within the buffer pool, these are written back to the uncompressed page. When space is required, InnoDB evicts the uncompressed pages from the pool and they must be recompressed when next accessed. For best performance, ensure that the buffer pool is large enough to contain both compressed and uncompressed pages.

Tuning Compression for InnoDB Tables

- Table compression works best on tables:
 - With a reasonable number of character string columns
 - That are read more often than they are written
- To check if compression is optimal for your workload:
 - Create compressed and uncompressed versions of tables and compare their sizes on disk
 - Determine how many compression operations complete successfully by querying INFORMATION_SCHEMA
 - For simple tests on a single compressed table, query INNODB_CMP. If using multiple compressed tables, turn on innodb_cmp_per_index_enabled and query INNODB_CMP_PER_INDEX.
 - The percentage of successful compression operations (COMPRESS_OPS_OK) compared to the total number of compression operations (COMPRESS_OPS) should be high.



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

Tables with a reasonable number of variable length character string columns such as VARCHARS or BLOBS increase compressibility.

When considering a table as a candidate for compression, compare the number of successful compression operations to unsuccessful ones. If the number of unsuccessful compression operations is routinely more than 1 or 2 percent of the total number of compression operations, the table is probably not suitable for compression, although this might be acceptable during a temporary operation such as a data load.

Topics

- Schema design considerations
- Indexes
- InnoDB table compression
- Partitioning tables



ORACLE®

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

Partitioning

- Divides a logical table into multiple physical subtables
 - You decide how the table is divided.
 - Partitioning types include RANGE, LIST, HASH, and KEY.
 - Partitioning is transparent.
 - SQL statements address tables, not their underlying partitions.
- Helps to optimize queries
 - Acts as a coarse indexing system with a very low overhead
 - Optimizer can “prune” tables to filter out irrelevant partitions
 - Can make “hot” rows easier to access
 - Makes queries on large tables more efficient
- Speeds up certain maintenance operations, for example:
 - OPTIMIZE and ANALYZE statements
 - Deleting unwanted rows



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

Partitioning also makes it possible to store more data in one table than can be held on a single disk or file system partition. Specify which partition holds each row of data by using the PARTITION BY clause when you create the table.

Partitioning provides several benefits that arise due to separating data rows into different files based on the partitioning condition.

- Some bottlenecks in specific workloads are avoided, such as per-index mutexes with InnoDB.
- The optimizer ignores partitions that do not contain result data when executing queries. This is known as “partition pruning.”
- You can host partition files on separate devices to increase parallel I/O throughput for multirow seeks, or to place more frequently-accessed data on faster storage.

Although partitioning is transparent, MySQL supports explicit partition selection for SELECT, DELETE, INSERT, REPLACE, UPDATE, LOAD DATA, and LOAD XML statements. For example, the following statement selects only those rows in partitions p0 and p1 that match the WHERE condition:

```
SELECT * FROM t PARTITION (p0, p1) WHERE c < 6
```

Partitioning Types

You can assign rows to partitions by using the following methods:

- RANGE: Column values exist within a certain range.
- LIST: Column values match one of a discrete set of values.
- HASH: Rows are selected based on a user-defined integer expression on column values.
- KEY: Like HASH, except that MySQL provides the hashing function
- Variants of the methods:
 - RANGE COLUMNS
 - LIST COLUMNS
 - LINEAR HASH
 - LINEAR KEY



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

The RANGE COLUMNS and LIST COLUMNS partitioning types enable the use of multiple columns in the partitioning key.

The LINEAR HASH and LINEAR KEY partitioning types use a different row distribution algorithm to that used with HASH and KEY partitioning.

RANGE Partitioning: Example

```
CREATE TABLE subscribers (
    sub_id INT NOT NULL,
    firstname VARCHAR(25) NOT NULL,
    lastname VARCHAR(25) NOT NULL,
    journalid INT NOT NULL,
    joined DATE NOT NULL
)
PARTITION BY RANGE( YEAR(joined) ) (
    PARTITION p0 VALUES LESS THAN (1970),
    PARTITION p1 VALUES LESS THAN (1980),
    PARTITION p2 VALUES LESS THAN (1990),
    PARTITION p3 VALUES LESS THAN (2000),
    PARTITION p4 VALUES LESS THAN MAXVALUE
);
```



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

With RANGE partitioning, rows with column or expression values falling within a specified range are assigned to a given partition. For example, suppose you have a table that stores subscriber data and you want to specify that subscribers that joined between 1970 and 1979 are stored in one partition, subscribers that joined between 1980 and 1989 are stored in another and so on. In other words, you partition the table based on contiguous ranges of values. RANGE partitioning is a good fit for such a partitioning scheme.

LIST Partitioning: Example

```
CREATE TABLE subscribers (
    sub_id INT NOT NULL,
    firstname VARCHAR(25) NOT NULL,
    lastname VARCHAR(25) NOT NULL,
    journalid INT NOT NULL,
    joined DATE NOT NULL
)
PARTITION BY LIST ( journalid ) (
    PARTITION p0 VALUES IN (5001, 5002),
    PARTITION p1 VALUES IN (6001, 6002),
    PARTITION p2 VALUES IN (7001, 7002),
    PARTITION p3 VALUES IN (8001, 8002),
    PARTITION p4 VALUES IN (9001, 9002)
);
```



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

With LIST partitioning, you specify lists of values for each partition so that rows with matching column values are stored in the corresponding partition. The example in the slide assumes that you want to partition your table based on the journals that your subscribers subscribe to. You can use LIST partitioning to achieve this.

Note

- RANGE and LIST partitioned tables can be subpartitioned into HASH or KEY subpartitions
- The COLUMNS partitioning methods are variants of RANGE and LIST, that support multiple columns in partitioning keys.

HASH Partitioning: Example

- HASH partitioning ensures an even distribution of data among partitions.
- You do not need to define the individual partitions, just the number of partitions.

```
CREATE TABLE subscribers (
    sub_id INT NOT NULL,
    firstname VARCHAR(25) NOT NULL,
    lastname VARCHAR(25) NOT NULL,
    journalid INT NOT NULL,
    joined DATE NOT NULL
)
PARTITION BY HASH ( YEAR (joined) )
PARTITIONS 4;
```



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

HASH is used primarily to ensure an even distribution of data among a predetermined number of partitions. This distribution is based on an expression that you supply when you create the table. Use the keyword PARTITIONS followed by the desired (integer) number of partitions.

KEY Partitioning: Example

- KEY is similar to HASH.
- The partition must be based on the table's PRIMARY KEY, or a UNIQUE KEY on the table.
- MySQL supplies the hashing algorithm.

```
CREATE TABLE subscribers (
    sub_id INT NOT NULL PRIMARY KEY,
    firstname VARCHAR(25) NOT NULL,
    lastname VARCHAR(25) NOT NULL,
    journalid INT NOT NULL,
    joined DATE NOT NULL
)
PARTITION BY KEY ( sub_id )
PARTITIONS 4;
```



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

Instead of using a partitioning expression that returns an integer or a null value, the expression following PARTITION BY KEY consists simply of a list of zero or more column names, with multiple names being separated by commas.

Partitioning and Performance

- Using a WHERE clause that allows pruning of partitions is usually faster than querying a nonpartitioned table.
- Altering and maintaining partitions is usually faster than altering the whole table.
- Creating useful indexes is just as important with partitioned tables.
 - MySQL creates separate indexes for each partition.
 - You need to consider partition pruning when designing indexes on partitioned tables.



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

Table Maintenance

The only case where altering partitions instead of altering the entire table is not faster is when the server needs to alter every partition. This is common for HASH/KEY partition management, which you can avoid by using LINEAR HASH/LIST. Dropping or truncating a partition is as fast as removing a single file, while deleting the same amount of data would take a long time and have different results depending on the storage engine.

Partitioning and Indexes

If you define an index that does not match the partitioning clause, the optimizer might not be able to prune the query. Avoid indexing nonpartitioned columns unless your queries include an expression that can help to prune partitions.

Note: Partition pruning is covered in the lesson titled “Optimizing Slow Queries.”

Partitioning Limitations

- Partitioning does not support the following:
 - Foreign keys
 - You must remove all referencing and referenced keys on the table before partitioning.
 - FULLTEXT indexes
 - Spatial columns
 - Temporary tables
 - Certain column data types
 - Certain functions
 - Subqueries as a partition key
- ALTER TABLE ... ORDER BY operations reorder only the rows in the partition, not the entire table.
- InnoDB tables support a maximum of 8,192 partitions.



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

Data type limitations

- Most partition types are based on INTEGER columns.
 - An expression that results in an integer is also acceptable.
 - Avoid using columns that contain NULL values.
- **For KEY partitioning:** MySQL's internal key-hashing functions produce a data type that is allowed in partitioning.
- **For RANGE and LIST partitioning:** The string, DATE, and DATETIME columns are acceptable.
- BLOB or TEXT data types cannot be used as columns for partitioning.

Functions

- MySQL partitioning is optimized for use with the YEAR(), TO_DAYS(), and TO_SECONDS() functions. You can, however, use other functions that return an integer or NULL.

Retrieving Partition Information

- Query the INFORMATION_SCHEMA.PARTITIONS table.
- Example: Display all database tables and their partitions.

```
mysql> SELECT TABLE_NAME,
-> GROUP_CONCAT(PARTITION_NAME)
-> FROM INFORMATION_SCHEMA.PARTITIONS
-> WHERE TABLE_SCHEMA='subscribers'
-> GROUP_BY TABLE_NAME;
+-----+-----+
| table_name | group_concat(partition_name) |
+-----+-----+
| subscribers_range | p0,p1,p2,p3,p4 |
| subscribers_list | p0,p1,p2,p3,p4 |
| subscribers_hash | p0,p1,p2,p3 |
| subscribers_key | p0,p1,p2,p3,
...

```



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

Note: To determine the effectiveness of queries on partitioned tables, use the EXPLAIN PARTITIONS statement. The EXPLAIN statement is covered in the lesson titled “Query Optimization.”

Quiz



Which of the following is not a limitation of MySQL partitioning?

- a. Foreign keys are not supported.
- b. Partitioning pruning is not supported.
- c. Spatial columns are not supported.
- d. Temporary tables cannot be partitioned.



ORACLE®

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

Answer: b

Summary



In this lesson, you should have learned how to:

- List important factors to consider when designing a schema
- Describe how normalization can affect performance
- Use appropriate data types to improve performance
- Use indexing to improve performance
- Describe the different types of indexes
- Identify InnoDB tables as suitable candidates for compression
- Use partitioning to improve query performance

ORACLE®

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

Practices

- 8-1: Comparing the Effects of Table Normalization on Query Performance
- 8-2: Choosing the Correct Data Type
- 8-3: Compressing Tables
- 8-4: Partitioning



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

9

Monitoring Queries

ORACLE®



MySQL™

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

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

Topics

- Identifying candidate queries for optimization
- Server logs
- Statement status variables
- Performance Schema and sys
- Graphical tools



ORACLE®

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

Objectives



After completing this lesson, you should be able to:

- List sources of information about queries running on the MySQL server
- Use the following tools and resources to identify slow queries
 - Server logs
 - Server status variables
 - Performance Schema and sys
 - The MySQL Enterprise Monitor and MySQL Workbench graphical tools
- Determine which queries will benefit most from optimization

ORACLE®

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

Topics

- Identifying candidate queries for optimization
 - Server logs
 - Statement status variables
 - Performance Schema and sys
 - Graphical tools



ORACLE®

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

Query Monitoring

Sources of information about the queries that the server executes include:

- Statement status variables
- General query log
- Slow query log
- Audit Log plugin
- MySQL Enterprise Manager's Query Analyzer
- MySQL Workbench Performance Reports
- `performance_schema.events_statements%tables`
- `sys schema views`



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

There are several ways to see the queries that the server executes. Some monitoring tools such as MySQL Enterprise Monitor record this information by reading Performance Schema or log files, and the Audit Log enterprise plugin records all client connections and operations.

Identifying Queries That Require Optimization

- Apply the *Pareto Principle* (the 80/20 rule) to your optimization strategy:
 - 20% of problem queries are the cause of 80% of all server slowdowns.
- Consider not only the performance of the query, but also how often the query executes.
- Prioritize slow queries that execute frequently.
- Deprioritize queries that are slow but execute only very infrequently. For example, queries that:
 - Create tables
 - Perform batch import or export operations
 - Generate complex reports



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

When you examine queries that are candidates for optimization, it is easy to assume that you need to optimize only the slowest queries. However, you might find that you improve the overall performance of the system by focusing on the most frequently executed queries, even if such queries are already executing relatively quickly.

Consider the following situations:

- You optimize a query that takes two seconds so that it now takes 800 ms, but this query executes once per minute on an average.
- You optimize a query that takes 20 ms so that it now takes 15 ms, but this query executes several thousand times per minute.

Making a frequent query faster (even by only a small amount) can free up server resources, reduce locking, and improve the response time and throughput of the server in general. Making an infrequent query faster is of less overall benefit to the server.

If you run reports frequently, or if you run complex reporting queries to generate results for on-demand dashboards, consider replicating your database to a dedicated reporting slave that has its schema optimized for serving such complex queries. Alternatively, consider implementing a data warehouse separate from your OLTP system.

Identifying Frequent Queries

- Parse the general query log.
- Monitor statement execution status variables.
- Query the `events_statements%` tables in Performance Schema.
- Regularly view currently executing statements to identify emerging patterns:
 - Execute `SHOW PROCESSLIST` or `sys.processlist`.
 - Monitor the general statement status variables.
 - Query the `INNODB_TRX` view in Information Schema.
- Use the slow query log with a low threshold to record most statements.
 - Use `mysqldumpslow` to summarize each type of statement and provide aggregate information.
 - Log slow queries to the `mysql.slow_log` table to query it with SQL.



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

Topics

- Identifying candidate queries for optimization
- Server logs
- Statement status variables
- Performance Schema and sys
- Graphical tools



ORACLE®

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

General Query Log

- Is enabled by setting the `general_log` server option
- Records every connection and statement that is executed against the server
 - Records the time of each connection and the process ID of all operations
 - Records all statements that are executed against all tables
 - Does not record update operations that use the `ROW` binary log format
 - The `ROW` format records update operations as changes to rows rather than as SQL statements.
 - If the binary log format is `MIXED`, only updates that the server can safely replicate as statements appear in the log.
 - Does not record the duration of statements
- Grows very quickly
 - Enable it only for short periods to gather a full record of all activity during those periods.



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

The general log contains entries such as the following:

```
1424 Query /* mem dbpool.default */ update
`mem_events`.events set lastUpdateTime=1397633565708 where id=7
1429 Query /* mem dbpool.default */ update
`mem_events`.events set lastUpdateTime=1397633565708 where id=4970
1424 Query /* mem dbpool.default */ commit
1429 Query /* mem dbpool.default */ commit
date and time 1434 Connect root@localhost on mysql
1434 Query SET NAMES latin1
1434 Query SET character_set_results = NULL
```

Slow Query Log

- Is enabled by using the `slow_query_log` server option
- Logs statements that take longer than a specified threshold
 - 10 seconds (by default)
 - Change this duration with the `long_query_time` option.
 - Specify the number of seconds with microsecond precision.
 - Example: 0.03 is 30 milliseconds
- Enables logging of statements that do not use indexes, even those below the `long_query_time` threshold
 - Enable `log_queries_not_using_indexes`.
- Is viewed with the `mysqldumpslow` command-line program



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

The `log_output` server option configures the destination of both the slow query log and the general query log. It can contain one or more of the values `FILE`, `TABLE`, or `NONE` (separated by commas). By default, the value is `FILE`.

- If the value is `NONE`, the slow query log does not write to either the file or the table, and ignores the presence of the values `FILE` and `TABLE`.
- If the value is `FILE` (and not `NONE`), the slow query log and general query log write to the files specified by the `slow_query_log_file` and `general_log_file` options, respectively. These options have the default values `hostname-slow.log` and `hostname.log` in the MySQL data directory.
- If the value is `TABLE` (and not `NONE`), the slow query log writes to the `slow_log` table and the general log writes to the `general_log` table, both in the `mysql` database.

By default, the slow query log does not record administrative statements. You can record such statements by enabling the `log_slow_admin_statements` server option.

Similarly, statements replicated from a replication master do not appear in the slow query log by default, even if they exceed the time specified by the `long_query_time` option. To record such statements, enable the `log_slow_slave_statements` server option.

There are two additional options that you can use to filter the slow query log output:

- The `min_examined_row_limit` option specifies the lowest number of rows that a statement must examine for the slow query log to record it.
- The `log_throttle_queries_not_using_indexes` option specifies the number of queries within a 60-second period that the slow query log records because they do not use indexes. After the slow query log records this number of those queries, it summarizes the number and aggregate time spent on the remaining statements in that time period. By default, this option has the value 0, indicating that it records all such queries.

The slow query log contains entries like the following:

```
# Time: date and time
# User@Host: root[root] @ localhost [127.0.0.1]  Id:  7694
# Query_time: 1.010099  Lock_time: 0.000023 Rows_sent: 0  Rows_examined: 1
SET timestamp=timestamp;
/* mem dbpool.default */ update `mem_inventory`.`MysqlServer` set
`timestamp`=timestamp where hid=x'08310BA6528CBF5783BBD95CE5B4561F';
```

Each entry contains the server date and time, along with information about the connection and query, and the time taken to run the query and hold locks.

Using mysql dump slow to View Slow Query Log Entries

- Summarizes the contents of the slow query log
- Enables you to provide a search term with the `-g` option
- Groups similar queries together
 - Changes numeric parameters to `N`
 - Changes string parameters to `'S'`
- Shows the number of similar queries, with timing information:
 - Average time taken
 - Total time taken



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

mysqldumpslow Output: Example

```
# mysqldumpslow -g 'update `mem__inventory`.`MysqlServer`' \
> /var/lib/mysql/hostname-slow.log

Reading mysql slow query log from /var/lib/mysql/hostname-slow.log
Count: 558  Time=0.94s (524s)  Lock=0.00s (0s)  Rows=0.0 (0),
root[root]@localhost
    /* mem dbpool.default */ update `mem__inventory`.`MysqlServer` set
`timestamp`=N where hid=x'S'

Count: 104  Time=0.93s (96s)  Lock=0.00s (0s)  Rows=0.0 (0),
root[root]@localhost
    /* mem dbpool.default */ update `mem__inventory`.`MysqlServer` set
`lastContact`=N, `hasLastContact`=N, `hasStartTime`=N, `timestamp`=N
where hid=x'S'
```



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

Topics

- Identifying candidate queries for optimization
- Server logs
- Statement status variables
- Performance Schema and sys
- Graphical tools



ORACLE®

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

General Statement Status Variables

The following status variables count the number of times the server executes statements:

- `Com_*`
 - Numerous status variables that display the number of times each statement (`DELETE`, `INSERT`, etc.) has been executed
- `Queries` and `Questions`
 - These status variables count the number of statements executed by the server.
 - `Queries` includes statements executed within stored programs. `Questions` does not include stored program statements.
- `Slow_queries`
 - This status variable displays the number of queries that took longer than the `long_query_time` setting.



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

`Com_*`

- Provides a count of how many times each type of statement was executed by the server
- There is one status variable for each type of statement. For example: `Com_select`, `Com_delete`, `Com_drop_table`.
- Use these values to monitor the effects of queries. For example: Did you expect there to be a value in the `Com_rollback` variable?
- The `Com_stmt_*` status variables count how many times each type of statement was requested within a prepared statement.

`Queries`

- Counts queries that were submitted to the server from the client, regardless of whether the server could execute them or not
- Includes statements executed within stored programs

`Questions`

- Similar to `Queries` status variable: counts all statements submitted from the client, including malformed queries
- Does not count statements executed within stored programs
- Provides a rough indication of server load

Slow_queries

- These queries are considered to be slow and are logged in the slow query log if it is enabled.
 - It is good practice to enable the slow query log.
- Monitor this setting to determine if the queries executed against your server are optimized.

SELECT Statement Status Variables

- `Select_full_join`
 - This status variable displays the number of joins that did not use indexes. Either the indexes do not exist, or they cannot be used.
- `Select_full_range_join`
 - This status variable displays the number of joins that used a range search on a referenced table.
- `Select_range`
 - This status variable displays the number of joins that used ranges on the first table.



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

`Select_full_join`

- For optimal server performance, this value should be 0. However, if your queries work with BLOB data or if the optimizer determines that it must perform a full table scan, this value increases even if you have covering indexes for all queries on the table.
- Locate these queries and provide indexes, or correct join statements that cannot use indexes.

`Select_full_range_join`

- These are rare. But if they appear, they might have a negative effect on performance.
- Locate these queries and determine if they can be optimized to improve performance.

`Select_range`

- If this status variable is a large number, research the cause and optimize the query concerned.

SELECT Statement Status Variables

- **Select_range_check**
 - This status variable displays the number of joins with keys that check for key usage after each row.
- **Select_scan**
 - This status variable displays the number of joins that performed a full table scan on the first table.
 - Use in conjunction with the slow query log.



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

Select_range_check

- These queries severely impact performance and should be optimized.
- Optimize these queries to improve performance.

Select_scan

- If this status variable is a large number, research the cause and optimize.
- Setting `log_queries_not_using_index` and setting `long_query_time` to 600 (to avoid logging faster queries) will write the majority of queries performing full table scans to the slow query log. This can be an excellent tool for locating queries that perform full table scans.

Quiz



Which status variable displays the number of statements executed by the server from clients, including those executed within stored programs?

- a. Com_rollback
- b. Queries
- c. Questions
- d. Slow_queries

ORACLE

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

Answer: b

Topics

- Identifying candidate queries for optimization
- Server logs
- Statement status variables
- **Performance Schema and sys**
- Graphical tools



ORACLE®

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

Finding Slow Queries with Performance Schema

- The following Performance Schema tables store performance metrics for queries:
 - `events_statements_current`: Information about currently executing statements
 - `events_statements_history` and `events_statements_history_long`: Information about previously executed statements
 - `events_statements_summary_by_digest`: Aggregated information about statements including their latency and lock time
- The `events_statements_current` and `events_statements_history` tables store the following information about queries:
 - Thread ID
 - Object schema and type
 - Timer information
 - Number of rows affected
 - Some status indicators
 - Including the number of sort merge passes and whether the query used an index



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

Example: Querying events_statements_current

```
mysql> SELECT sql_text, TIMER_WAIT, lock_time
-> FROM events_statements_current
-> ORDER BY timer_wait DESC LIMIT 1\G
***** 1. row *****
sql_text: UPDATE t1 SET intcol1 = 1137990260,charcol1 =
'pvYr3YntZ2DoGrwWfL91bW9Epw8iO6vDuR4xrkqOe3Dum1PPEQQpwRvmO3Kg2FtobqlQQRj4
Woq8hFZvhrokWBX8bRSFLc09Biut0L44jsvrZ2GKczwyAS6dvakGA1P'
TIMER_WAIT: 440485009000
lock_time: 55000000
1 row in set (#.## sec)
```



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

The events_statements_current table contains current statement events, one row per thread. It shows the current status of the thread's most recent monitored statement event. The sample query in the slide lists the query with the greatest timer_wait time. The timer_wait column records the duration of the event in picoseconds (trillionths of a second) and is calculated from the timer_start and timer_end column values. It also displays the time the query spent waiting for table locks (lock_time, also in picoseconds) and the text of the SQL statement that raised the event (sql_text.)

Example: Querying events_statements_summary_by_digest

```

mysql> SELECT * FROM
-> events_statements_summary_by_digest
-> ORDER BY avg_timer_wait DESC LIMIT 1\G
***** 1. row *****
SCHEMA_NAME: mysqlslap
DIGEST: 2d5a98ab60fb766c78424830ef918721
DIGEST_TEXT: UPDATE t1 SET
intcoll = ? , charcoll = ?
COUNT_STAR: 2529
SUM_TIMER_WAIT: 1607042751897000
MIN_TIMER_WAIT: 57515909000
AVG_TIMER_WAIT: 635445927000
MAX_TIMER_WAIT: 1512682649000
SUM_LOCK_TIME: 6088685000000
SUM_ERRORS: 0
SUM_WARNINGS: 0
SUM_ROWS_AFFECTED: 102366
SUM_ROWS_SENT: 0
SUM_ROWS_EXAMINED: 250371
SUM_CREATED_TMP_DISK_TABLES: 0
SUM_CREATED_TMP_TABLES: 0
SUM_SELECT_FULL_JOIN: 0
SUM_SELECT_FULL_RANGE_JOIN: 0
SUM_SELECT_RANGE: 0
SUM_SELECT_RANGE_CHECK: 0
SUM_SELECT_SCAN: 0
SUM_SORT_MERGE_PASSES: 0
SUM_SORT_RANGE: 0
SUM_SORT_ROWS: 0
SUM_SORT_SCAN: 0
SUM_NO_INDEX_USED: 0
SUM_NO_GOOD_INDEX_USED: 0
FIRST_SEEN: date and time
LAST_SEEN: date and time
1 row in set (#.## sec)

```



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

The aggregated summary tables are also useful when finding problem queries. In particular, the events_statements_summary_by_digest table aggregates the metrics for similarly normalized queries, that is, queries that use the same keywords and identifiers and differ only in their whitespace and literal values.

You can use this table to look at statements ordered by average timer wait, as in the example in the slide, or by other metrics such as the number of on-disk temporary tables this type of statement creates, the number of sort merge passes, or the number of queries of this type that do not use indexes.

Identifying Slow Queries with the sys Schema

- `statements_with_runtimes_in_95th_percentile`: Information about those statements with an average run time in the top 95th percentile, including the number of rows sent and examined, and the aggregated latency of the statements.
- `statements_with_full_table_scans`: Information about statements that read entire tables, including the rows sent and examined, and the aggregated latency of the statements.
- `statements_with_temp_tables`: Information about statements that require temporary tables, including the number in-memory and on-disk temporary tables, the average number of temp tables per query, and the percentage of temporary tables that were created on disk.
- `statements_with_sorting`: Information about statements that require sorting, including how many sorts used scans and how many used ranges, and the aggregated number of rows sorted



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

The `sys` database contains several views that summarize normalized statements. Some of those views show statements with specific characteristics, such as those that require sorting or those that create temporary tables when you execute them. The views listed in the slide share some common columns, including `query`, `db`, `exec_count`, `total_latency`, `first_seen`, and `last_seen`.

Investigating Aggregated Statement Metrics with the sys.statement_analysis View

```
mysql> SELECT * FROM sys.statement_analysis
-> LIMIT 1\G
***** 1. row *****
      query: UPDATE t1 SET intcol1 =
?, charcol1 = ?
      db: mysqlslap
      full_scan:
      exec_count: 93389
      err_count: 0
      warn_count: 0
      total_latency: 14.40h
      max_latency: 5.24 s
      avg_latency: 554.99 ms
      lock_latency: 00:01:18.94
      rows_sent: 0
      rows_sent_avg: 0
      rows_examined: 9245511
      rows_examined_avg: 99
      tmp_tables: 0
      tmp_disk_tables: 0
      rows_sorted: 0
      sort_merge_passes: 0
      digest:
      2d5a98ab60fb766c78424830ef918721
      first_seen: date 16:52:22
      last_seen: date 17:06:56
1 row in set (#.## sec)
```

ORACLE®

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

The statement_analysis table aggregates some of the information contained in the other views from all queries and puts it in a single table that contains all normalized statements, as shown in the example in the slide.

Topics

- Identifying candidate queries for optimization
- Server logs
- Statement status variables
- Performance Schema and sys
- Graphical tools



ORACLE®

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

MySQL Enterprise Monitor Query Analyzer

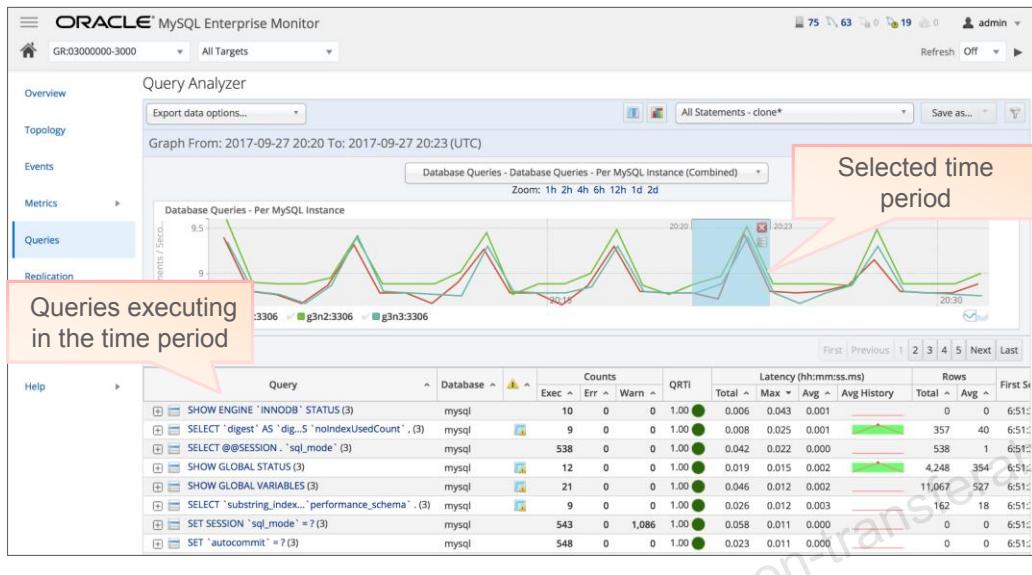
- Identify expensive queries that impact database performance
 - Query Analyzer table: Displays aggregated summary information for all queries
 - Query Response Time Index (QRTI): Highlights queries with unacceptable response times
- Visualize query activity to gain further insight into performance beyond query statistics
 - Drag and select a region on a graph to display the queries being executed during the selected time period
 - Combine timeframes with numerous filtering options such as query type, execution count, first appearance
- Drill down into detailed information for each query
 - Including execution plan, query text, and performance metrics
- Filter for specific query problems like full table scans and bad indexes
- Fix the root causes of poor performance directly in the SQL code



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

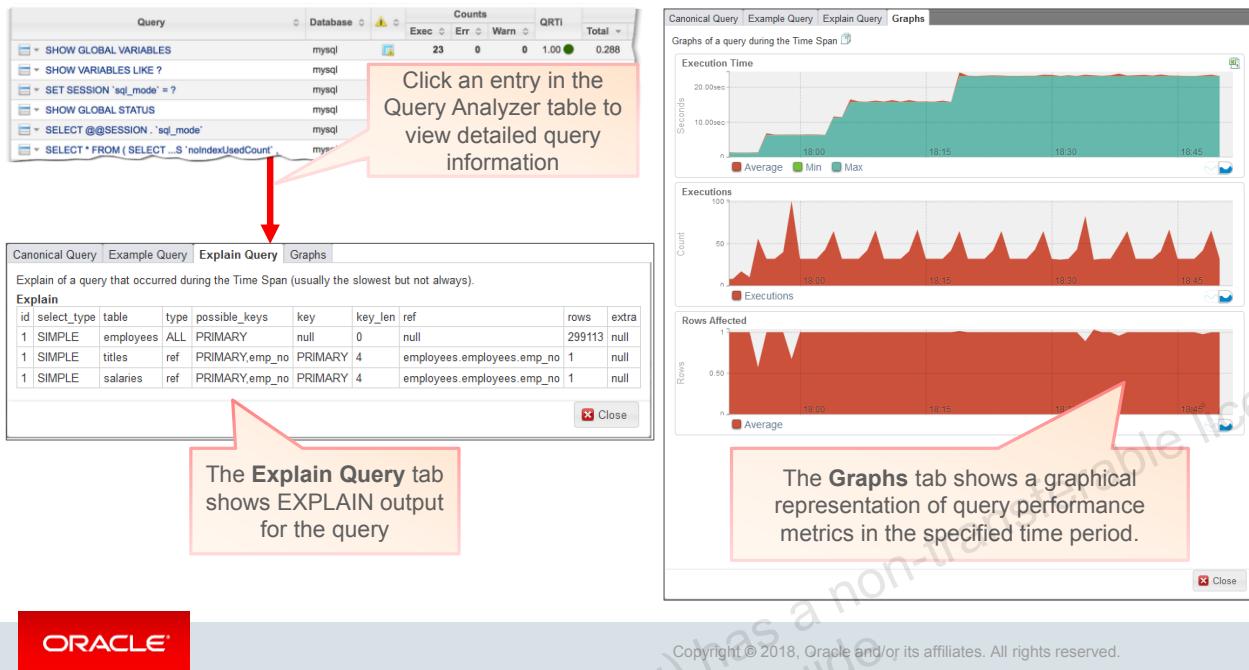
The Query Analyzer aggregates queries in their normalized form and displays summary information along with graphs of recent server activity. Use it to find queries that occur when a performance problem manifests, and examine the aggregated performance of those queries, a sample query of that form, and the query plan that you see by running EXPLAIN on those statements.

Query Analyzer Tab



The diagram in the slide shows the Query Analyzer tab in MySQL Enterprise Monitor. The user highlights the time period they are interested in and the queries that executed in the selected time period appear in the Query Analyzer table at the bottom of the page. Each row within the table provides the statistical information for one normalized query statement. This information includes the number of times the query executed in the selected time period, latency statistics for the query, the Query Response Time Index, the date and time the query first executed, and more.

Query Analyzer Detailed Query View

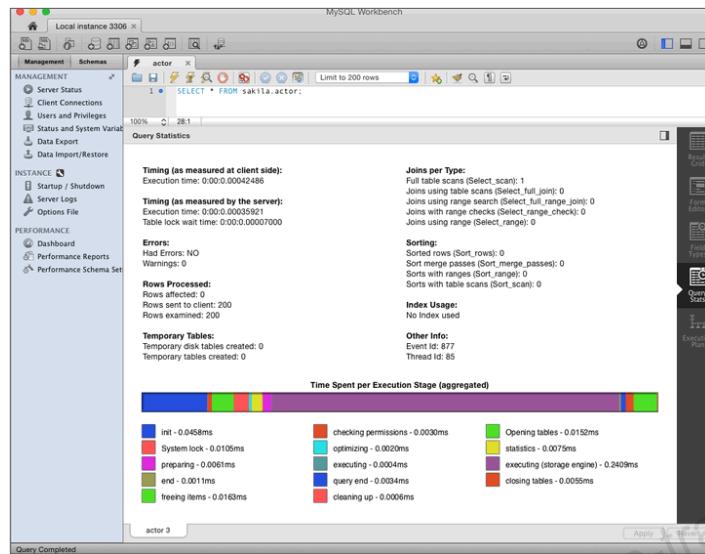


When you click a row in the Query Analyzer table, a tabbed dialog displays with the following tabs:

- **Canonical Query:** Provides different, de-normalized views of the query to better identify it. (Normalization removes the constants from individual queries so that queries that use the same structure are identified as the same basic query.) This tab also provides more detailed statistics for execution times and number of rows that the query accesses.
- **Example Query:** Provides detailed information about the most expensive query executed in the timeframe
- **Explain Query:** Displays the output of executing the query with the EXPLAIN prefix to understand the execution plan for the query.
- **Graphs:** The Graphs tab shows key graphs over the selected time period for the example query. The image in the slide shows graphs for execution time, number of executions, and the rows affected. Use the graphs to quickly identify deviations from the normal values.

Note: This course covers the output of EXPLAIN in more detail in the lesson titled “Optimizing Queries.”

MySQL Workbench Query Statistics



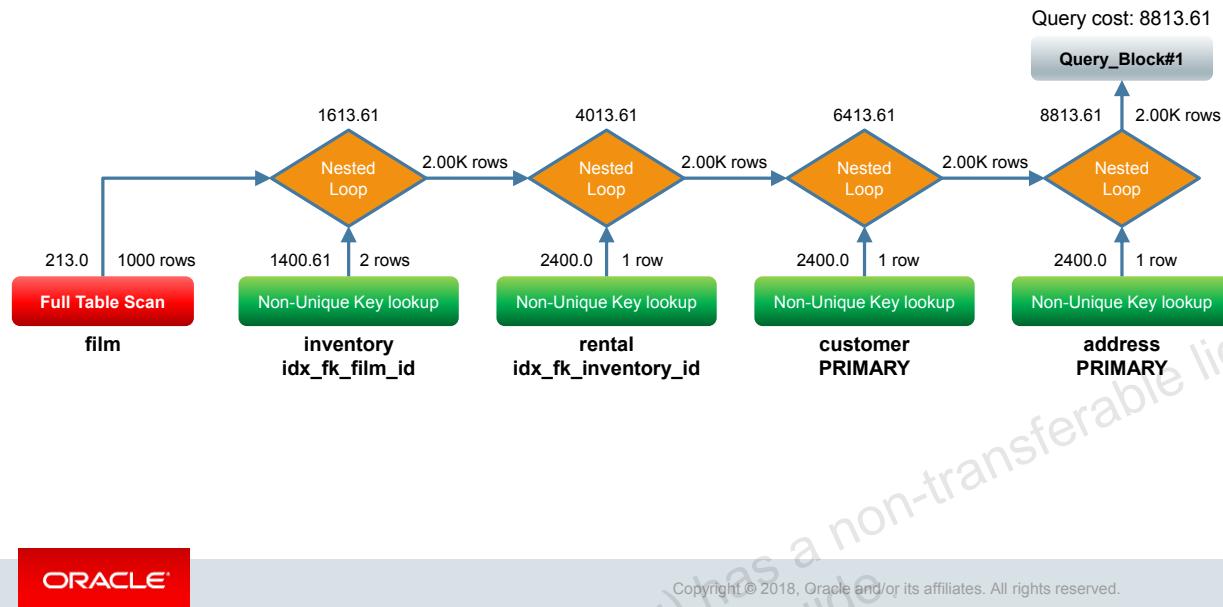
ORACLE®

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

You can use MySQL Workbench to display information about the performance of an individual query. To accomplish this, you must enable Performance Schema with statement instrumentation and turn on the Collect Performance Schema stats option in the Query menu.

The Query Statistics results tab displays when you execute a query in the SQL Editor window. It shows timing information and other details about the query such as whether it creates temporary tables, whether or not it uses an index, and the number of rows it processes. You can optionally include a graphical display of this data as shown in the slide.

MySQL Workbench Visual EXPLAIN



ORACLE®

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

The MySQL Workbench Visual EXPLAIN feature generates and displays a visual representation of the MySQL EXPLAIN statement. It can be very useful in diagnosing problems with a poorly-performing query, particularly in regard to index usage.

Note: This course covers the output of EXPLAIN in more detail in the lesson titled “Optimizing Queries.”

Summary



In this lesson, you should have learned how to:

- List sources of information about queries running on the MySQL server
- Use the following tools and resources to identify slow queries
 - Server logs
 - Server status variables
 - Performance Schema and sys
 - The MySQL Enterprise Monitor and MySQL Workbench graphical tools
- Determine which queries will benefit most from optimization

ORACLE®

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

Practices

- 9-1: Monitoring Statements
- 9-2: Using the Slow Query Log
- 9-3: Identifying Slow Queries with MySQL Enterprise Monitor Query Analyzer
- 9-4: Identifying Slow Queries with `sys` Views
- 9-5: Using MySQL Workbench Query Statistics



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

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

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

10

Query Optimization

ORACLE®



MySQL™

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

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

Topics

- How the optimizer works
- Understanding the query plan
- Optimizing queries



ORACLE®

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

Objectives



After completing this lesson, you should be able to:

- Explain the role and operation of the MySQL query optimizer
- Determine the execution plan for a query by using the EXPLAIN statement
- List good strategies for improving query execution times
- Choose appropriate indexes to maximize query performance
- Rewrite suboptimal queries to take full advantage of indexes
- Use MySQL GUI tools to investigate optimizer behavior

ORACLE®

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

Topics

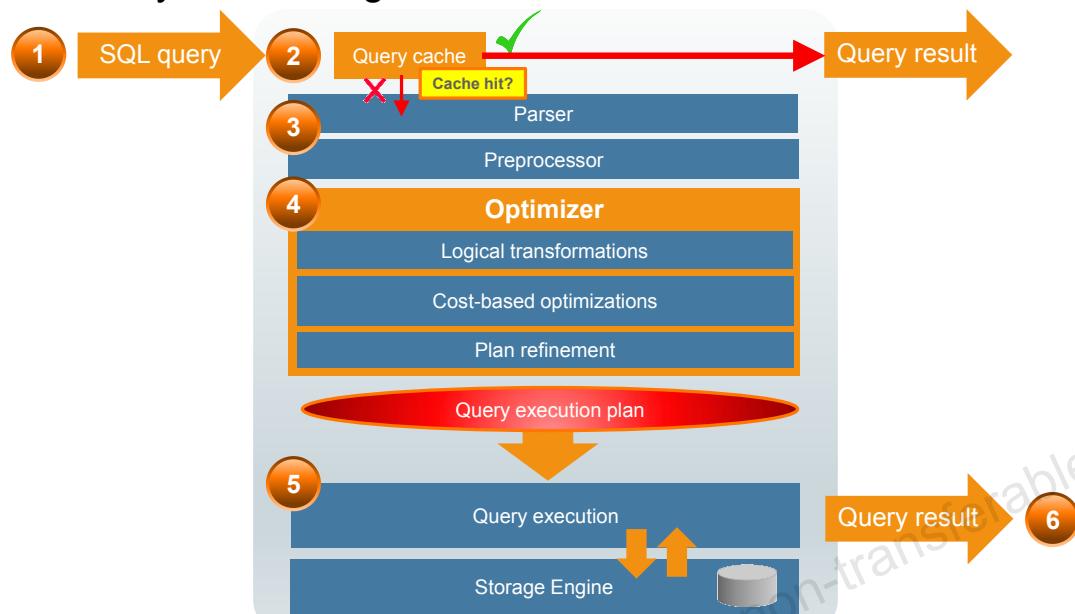
- How the optimizer works
- Understanding the query plan
- Optimizing queries



ORACLE®

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

MySQL Query Processing



1. The client sends the SQL statement to the server.
2. The server checks the query cache. Note that the query cache is disabled by default. If the cache is enabled and there is a cache hit, it returns the stored result to the client from the cache; otherwise, it passes the SQL statement to the next step.
3. The server parses and preprocesses the statement.
4. The query optimizer investigates the best way to implement the query and generates a query execution plan.
5. The query execution engine executes the plan by retrieving data from the storage engine.
6. The server sends the result to the client.

Note: Oracle does not recommend the use of the query cache.

Optimizer Main Stages

The optimizer's goal is to produce the most efficient query execution plan for a specific query. It achieves this by performing the following operations:

- Logical Transformations:
 - Simplify query conditions
 - Convert outer joins to inner joins (where appropriate)
- Cost-based optimizations:
 - Determine the best access method
 - Decide on the optimal order of tables within a join
 - Apply optimizations to subqueries
- Plan refinement:
 - Determines the best order of condition evaluation
 - Attempts to avoid sorting in ORDER BY queries
 - Index condition pushdown



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

The role of the optimizer is to produce an execution plan for the query that uses the least amount of resources. A process called *greedy optimization* enables the optimization engine to significantly reduce the amount of time it spends calculating optimal execution paths by a process of intelligent best-path reductions.

It is important to note that the optimizer is interested in the performance of only a specific query. It does not consider other queries executing concurrently when generating an execution plan for the query. It does not take into account available hardware, such as SSDs, which could affect query execution time.

Logical Transformations

- Apply transformation rules to simplify query conditions:
 - Negation elimination
 - Equality propagation
 - Evaluation of constant expressions
 - Removal of trivial conditions
- Convert outer joins to inner joins where possible and appropriate
 - Depends on the WHERE clause and table structure
 - Might provide performance benefits



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

- **Negation elimination:** The optimizer refactors Boolean operations to remove NOT where possible to avoid additional comparisons.
- **Equality and constant propagation:** Enables faster comparisons by removing redundant constant comparisons with identifiers where those identifiers are checked for equality with a constant elsewhere in the statement.
- **Constant expressions:** Constant expressions are evaluated before the statement is processed so that the optimizer does not have to perform the evaluation on each row.
- **Trivial conditions are removed:** The optimizer removes any conditions that are guaranteed to be true or false.

Example: Logical Transformation

- Original query:

```
SELECT * FROM t1, t2 WHERE  
t1.a = t2.a AND t2.a = 9 AND (NOT (t1.a > 10 OR t2.b > 3)  
OR (t1.b = t2.b + 7 AND t2.b = 5));
```

- Negation elimination:

```
SELECT * FROM t1, t2 WHERE  
t1.a = t2.a AND t2.a = 9 AND ((t1.a <= 10 AND t2.b <= 3 )  
OR (t1.b = t2.b + 7 AND t2.b = 5));
```

- Equality and constant propagation:

```
SELECT * FROM t1, t2 WHERE  
t1.a = 9 AND t2.a = 9 AND ((9 <= 10 AND t2.b <= 3 )  
OR (t1.b = 5 + 7 AND t2.b = 5));
```



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

The first statement in the slide is the original query. The items in red show the results of the optimizations in each step.

Example: Logical Transformation

- Evaluation of constant expressions:

```
SELECT * FROM t1, t2 WHERE  
t1.a = 9 AND t2.a = 9 AND ((9 <= 10 AND t2.b <= 3 )  
OR (t1.b = 12 AND t2.b = 5));
```

- Trivial condition removal:

```
SELECT * FROM t1, t2 WHERE  
t1.a = 9 AND t2.a = 9 AND (9 <= 10 AND t2.b <= 3 +  
OR (t1.b = 12 AND t2.b = 5));
```

- Final query:

```
SELECT * FROM t1, t2  
WHERE t1.a = 9 AND t2.a = 9 AND (t2.b <= 3 OR (t1.b = 12 AND t2.b = 5));
```



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

Cost-Based Optimization

- The optimizer uses a cost model to determine the most efficient query execution plan.
Cost-based optimization involves:
 - Assigning costs to query operations
 - Considering the costs of alternative plans
 - Choosing the plan with the lowest cost
- The cost “unit” is the cost of reading a random data page from disk. All other costs are relative to this cost unit.
- The cost-based optimizations are:
 - **Access method:** How the storage engine will retrieve rows from each table
 - **Join order:** The optimal order in which tables are joined
 - **Subquery strategy:** The most efficient way to execute subqueries



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

The cost model is derived from database statistics, such as the number of rows in a table, or the number of unique values in an indexed column.

Cost Model Inputs

The main inputs to the optimization cost model include:

- I/O Cost:
 - The number of index and data pages that must be read (from estimates provided by the storage engine)
 - The access method (random or sequential)
- Schema:
 - The length of records and keys
 - The uniqueness of indexes
 - Whether columns allow null values
- Statistics (maintained by the storage engine):
 - The number of rows in the tables
 - The average number of rows for each index entry
 - The number of rows within an index range



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

Note: The I/O cost of index and data pages does not include the contents of the InnoDB buffer pool, so the same query execution plan results in much better performance if the InnoDB buffer pool is warmed up.

Access Method

- The access method refers to the technique MySQL uses to fetch data from the storage engine.
- The optimizer chooses the most efficient method for each table involved in the query.
- The main access methods are:
 - **Table scan:** All rows are read in sequential order.
 - **Index scan:** All index entries are read in sequential order.
 - **Ref access (index lookup):** All rows are read with a given key value using an index.
 - **Range scan:** Reads all rows from a given range of values within an index
 - **Index merge:** Uses multiple indexes on the same table
 - **Loose index scan:** Optimizes index access for GROUP BY and DISTINCT queries



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

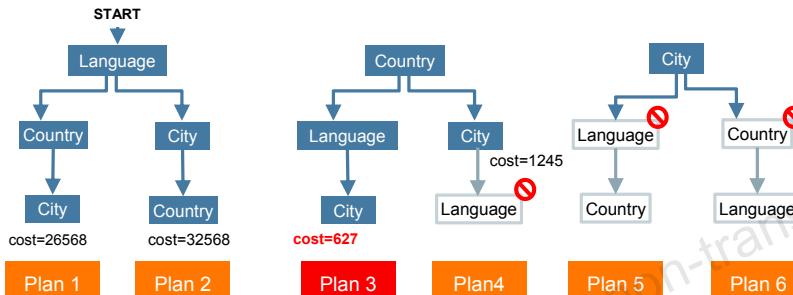
You can determine which access method MySQL uses for a particular query by checking the `type` output column of the `EXPLAIN` statement.

Note: See the slides titled “`EXPLAIN type`” in this lesson for a full list of access methods.

Join Order

- The optimizer evaluates possible join orders and selects the cheapest method.
- Example using the `world` database:

```
SELECT city.Name, Language FROM countrylanguage, country, city
WHERE city.CountryCode = country.Code
AND city.ID = country.Capital
AND city.Population >= 1000000
AND countrylanguage.Country = country.Code;
```



ORACLE

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

The `optimizer_search_depth` option controls the maximum depth of the plan search that the optimizer performs. If this value is larger than the number of tables joined in the query, the optimizer examines all possible plans. For queries with a large number of tables, this process might take a long time. The default value is 62. If you use a much lower value, such as 3 or 4, the optimizer is faster at choosing a plan for complex queries but the resulting plan might not be optimal.

The slide example considers a query that joins three tables in the `world` database: `country`, `countrylanguage`, and `city`. MySQL performs the following steps to find the best access method and join order for the query.

Plan 1

The optimizer considers the `countrylanguage` table as a candidate for the first table in the join order. There are no conditions with constants for this table, so MySQL must choose between table and index scans. The join optimizer then considers the cost of adding the `country` table to the join, based on the access methods available. Finally, the optimizer adds the `city` table to the join. The cost is calculated and the plan is stored.

Plan 2

The join optimizer then considers the effect of switching the join order of the `city` and `country` tables. Swapping the order of tables in the join might enable different access methods. Finally, it adds the `country` table and calculates the cost. Plan 2 is discarded, because the cost is higher than the previous plan.

Plan 3

The optimizer then considers using the effect of using the `country` table as the first table in the join sequence. It adds the `countrylanguage` and `city` tables and finds a plan with a lower cost than the preceding plans. This plan is now stored in preference to plan 1.

Plans 4, 5, and 6

MySQL tries different combinations but “pruning” allows MySQL to exit these calculations early. This pruning step is known as *greedy optimization*.

Now the optimization process is complete, the optimizer selects plan 3 as the plan with the lowest cost.

Condition Filter Effect

When the join optimizer calculates the order tables should be joined, *fanout* is of great importance. The fanout for a table `t_y` in a join plan consisting of `t_1, ..., t_x, t_y, ...` is the average number of rows from `t_y` that will match each partial row combination from `t_1, ..., t_x` and all predicates applicable to `t_y`. It is important to keep the fanout as low as possible because it directly influences how many row combinations will be joined with the next table in the join order.

Overriding the Optimizer

If you want to override the join order that the optimizers provide, use the `STRAIGHT_JOIN` to create the `JOIN`. `STRAIGHT_JOIN` is similar to `JOIN`, except that the left table is always read before the right table. This can be used for those rare cases for which the join optimizer processes the tables in a suboptimal order.

Subquery Optimizations

Subquery Type	Strategies
IN (SELECT ...)	Semi-join Materialization Rewrite IN as EXISTS
NOT IN (SELECT ...)	Materialization Rewrite NOT IN as EXISTS
FROM (SELECT ...)	Materialization Merge
<Comparison Operator> ALL/ANY (SELECT ...)	Rewrite as MAX() /MIN()
EXISTS/<other>	Execute subquery
outer_expr IN (SELECT inner_expr FROM ... WHERE subquery_where)	
EXISTS (SELECT 1 FROM ... WHERE subquery_where AND outer_expr = inner_expr)	



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

Semi-join

A “semi-join” between two tables returns rows from the first table where one or more matches are found in the second table. In a semi-join, rows in the first table are returned only once, unlike a conventional join. Semi-joins are written using the EXISTS or IN constructs.

Materialization

Materialization speeds up query execution by generating a subquery result as a temporary table, normally in memory. Materialization of subqueries in the FROM clause (derived tables) is postponed until their contents are needed during query execution, which improves performance. In the case of a query that joins the result of a query in the FROM clause to another table, if the optimizer processes the other table first and determines that it does not return rows, the join cannot be made and MySQL does not have to materialize the subquery.

Merging

Merging places the derived table into the outer query block instead of storing it as a temporary table. The difference between the merge and materialization optimizations can be seen in the following example:

Materialization:

```
SELECT *
FROM t1 JOIN (SELECT t2.f1 FROM t2) AS derived_t2 ON t1.f2=derived_t2.f1
WHERE t1.f1 > 0
```

Merge:

```
SELECT t1.* , t2.f1  
FROM t1 JOIN t2 ON t1.f2=t2.f1  
WHERE t1.f1 > 0;
```

IN or ANY as EXISTS()

MySQL can optimize subqueries that use the `IN` (or equivalent `ANY`) comparison operators. Consider the example in the slide, which instructs the subquery that the only rows to consider are those where the inner expression `inner_expr` is equal to `outer_expr`. This is done by pushing down an appropriate equality into the subquery's `WHERE` clause.

ALL or ANY as MIN() /MAX()

Subqueries with `ALL` or `ANY` can also be rewritten to use `MIN()` and `MAX()` to improve performance.
Example:

```
scalar > ANY (SELECT column...) becomes scalar > (SELECT (MIN(column) ...)
```

Plan Refinement

The main optimizations at the plan refinement stage are:

- Assigning query conditions to tables
 - Evaluate conditions as early as possible in the join order.
- ORDER BY optimization, to avoid sorting
 - Use a different index.
 - Read rows in descending order.
- Selecting a different access method
 - For example, using a range scan instead of a table scan or ref access
- Index Condition Pushdown



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

The optimizer determines the earliest possible stage to evaluate each query condition. Evaluating conditions as early as possible helps to exclude rows that will not be part of the result set.

In the case of queries with ORDER BY or GROUP BY, the optimizer might be able to avoid sorting by changing the access method of the first table to a scan of an index that provides correct ordering.

The optimizer will look to use the cheapest access method in every case, not just for ORDER BY or GROUP BY clauses. MySQL might also choose to “push” conditions to indexes, allowing the conditions to be evaluated without reading rows.

Note: See the slides titled “EXPLAIN type” in this lesson for a full list of the access methods.

Index Condition Pushdown Optimization

- Delegates conditions that can be evaluated on indexed columns to the storage engine.
- Enables MySQL to evaluate conditions without having to access data rows, to reduce disk I/O and CPU usage.
- Allows both conditions on a single index and conditions on earlier tables in a join to be “pushed down.”

	Normal Execution	Index Condition Pushdown
Storage Engine	<ol style="list-style-type: none">1. Reads the index entry2. Fetches and returns the row	<ol style="list-style-type: none">1. Reads the index and evaluates the pushed condition2. If the condition is met, fetches and returns the row
Server	<ol style="list-style-type: none">3. Evaluates the condition to elect whether to keep or discard the row	<ol style="list-style-type: none">3. Retrieves only rows that satisfy the index condition



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

With index condition pushdown (ICP), conditions are evaluated by the storage engine instead of by the server. MySQL can push conditions to an index if a condition compares a column that is in the index to a constant value or a column from a table earlier in the join sequence.

In a typical query execution, for each index entry that is read, the corresponding row is looked up in the table and returned to the server. The server must evaluate the conditions and decide whether to keep or discard the row.

With ICP, all conditions that depend on columns that are present in the index are evaluated in the index instead. So, for every row that does not qualify, MySQL does not have to read that row from the table, which often involves reading a page from disk. For rows that do qualify, there is no overhead because the conditions that were pushed to the index are not re-evaluated in the server.

Quiz



Which of the following is not a cost-based optimization?

- a. Subquery strategy
- b. Equality propagation
- c. Join order
- d. Access method

ORACLE

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

Answer: b

Topics

- How the optimizer works
- Understanding the query plan
- Optimizing queries



ORACLE®

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

Understanding the Query Plan

- Use the EXPLAIN command to investigate the final query plan.
- Prefix your SELECT, INSERT, UPDATE, or DELETE SQL statement with the EXPLAIN keyword
 - Example:

```
EXPLAIN SELECT Name, Continent FROM country WHERE Code = 'CAN';
```
- Analyze the output of EXPLAIN to understand how MySQL plans to execute your particular SQL statement.
 - EXPLAIN does not return data rows.



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

Example: EXPLAIN Output

The EXPLAIN command produces a number of columns that provide information from the optimizer about how MySQL will execute the statement.

```
mysql> EXPLAIN SELECT Name, Continent FROM country
-> WHERE Code LIKE 'C%'\G
***** 1. row *****
    id: 1
  select_type: SIMPLE
        table: country
    partitions: NULL
         type: range
possible_keys: PRIMARY
        key: PRIMARY
      key_len: 3
         ref: NULL
        rows: 20
     filtered: 100.00
       Extra: Using where
1 row in set, 1 warning (#.## sec)
```

Extended information available by executing SHOW WARNINGS.



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

The EXPLAIN statement for a SELECT query generates a warning that contains extra, “extended” information that is not part of the EXPLAIN output but which can be viewed by executing SHOW WARNINGS. For example, the EXPLAIN statement in the slide produces the following information:

```
mysql> SHOW WARNINGS\G
***** 1. row *****
Level: Note
Code: 1003
Message: /* select#1 */ select `world`.`country`.`Name` AS
`Name`, `world`.`country`.`Continent` AS `Continent` from `world`.`country`
where (`world`.`country`.`Code` like 'C%')
1 row in set (#.## sec)
```

The Message field in the SHOW WARNINGS output displays how the optimizer qualifies table and column names in the SELECT statement, what the SELECT looks like after the application of rewriting and optimization rules, and possibly other notes about the optimization process. This can include special markers such as <materialize> or <primary_index_lookup> which are not valid SQL and therefore you should not attempt to execute the rewritten query. This information replicates that available in the output of EXPLAIN EXTENDED, which is deprecated.

The following slides consider each of the columns in the EXPLAIN output in detail.

EXPLAIN Output

- `id`: A simple identifier for the `SELECT` statement
- `select_type`: Describes the type of `SELECT` being performed
- `table`: Displays the names of tables to be used in the access strategy
- `partitions`: The partitions from which records would be matched by the query
 - `NULL` for nonpartitioned tables
- `type`: Describes the access method the optimizer uses
- `possible_keys`: Provides the available indexes that MySQL was able to choose from when evaluating the access strategy
 - Contains `NULL` if there are no keys available



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

`EXPLAIN` returns one row of information for each table used in the `SELECT` statement. It lists the tables in the output in the order that MySQL reads them while processing the statement.

EXPLAIN Output

- **key:** Displays the actual key chosen to perform the data access
 - `NULL` is the output if there are no keys available.
- **key_len:** Provides the length (in bytes) of the chosen key.
 - The length is `NULL` if the key column is `NULL`.
- **ref:** Displays the columns in the key that are used to access data in the table, or
 - `const`, if comparing to a constant value
 - `func`, if comparing to a function expression
- **rows:** Displays the number of rows that MySQL expects to find.
- **filtered:** The percentage of rows filtered by the table condition.
- **Extra:** Extra information about the access method



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

- **ref value:** Produces a single constant value if the join has been optimized away.
 - The `ref` column contains `const` if the comparison is against a constant instead of a column value, for example, `CountryCode='IRL'`.
 - The `ref` column contains `func` if the comparison is against the result of a function expression. To determine which function is involved, issue `EXPLAIN EXTENDED` followed by `SHOW WARNINGS`.
- **rows value:** An estimate of the number of rows based on InnoDB statistics and the access method.

EXPLAIN Output: select_type Column

- SIMPLE: A single SELECT (no UNION or subquery)
- PRIMARY: The outermost SELECT
- UNION: The SELECT is the second or later statement of a UNION
- DEPENDENT UNION: The second or later SELECT in a UNION is dependent on an outer query
- UNION RESULT: The SELECT is the result of a UNION
- SUBQUERY: The first SELECT statement in a subquery
- DEPENDENT SUBQUERY: The subquery contains a reference to a table in the outer query
 - Also known as a *correlated subquery*
- DERIVED: The SELECT is part of a subquery within a FROM clause.



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

Other possible values in the select_type column include:

- MATERIALIZED: A materialized subquery
- UNCACHEABLE SUBQUERY: A subquery for which the result cannot be cached and must be re-evaluated for each row of the outer query
- UNCACHEABLE UNION: The second or later select in a UNION that belongs to an uncacheable subquery (UNCACHEABLE SUBQUERY)

EXPLAIN Output: type Column

The access method (from most efficient to least efficient):

- system: The SELECT statement is requesting data from an in-memory table that contains a single row.
- const: Contains at most one matching row, which is read at the start of the query
- eq_ref: Contains a single row that matches rows returned from previous tables
- ref: Contains one or more rows that match rows returned from previous tables
- fulltext: The join is performed by using a FULLTEXT index.
- ref_or_null: Similar to ref except MySQL performs an extra search for rows that contain NULL values. This optimization is often used to resolve subqueries.



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

- The const access strategy is used when you compare all parts of a PRIMARY KEY or UNIQUE index to constant values. Because there is only one row, values from the column in this row are regarded as constants by the rest of the optimization process. This access method is very fast because tables that are accessed by const are read only once.
- The eq_ref access strategy type is used only if both of the following conditions are met:
 - **Keys used:** All parts of a key are used by the join in the query.
 - **Unique key(s) are present:** The key is a PRIMARY KEY or UNIQUE NOT NULL.
- The ref access strategy is used when either of the following occurs:
 - **Leftmost part of join is used:** The join condition uses only the left-most part of a multicolumn key.
 - **Non-unique and non-null key:** The key used is not unique but does not contain any null values.

EXPLAIN Output: type Column

- `index_merge`: Multiple executions of `ref`, `ref_or_null`, or `range` accesses are used to retrieve key values matching various `WHERE` conditions.
- `unique_subquery`: The `WHERE` condition contains an `IN` clause that contains a subquery.
- `index_subquery`: Identical to `unique_subquery` except the values returned from the subquery are not unique
- `range`: Contains a `WHERE` clause that uses range operators
- `index`: Performs a sequential scan on all the key entries of an index
- `ALL`: Performs a sequential scan of all rows in the table



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

- `index_merge`: Used when results are combined from two indexes to form a single data set
- `unique_subquery`: Values returned from the subquery are unique.
- `range`: Operators include `>`, `>=`, `=`, `<>`, `<=`, `<`, `<=`, `IN`, `BETWEEN`, and `IS NULL`. Also includes `LIKE` comparisons if the argument to `LIKE` is a constant string that does not start with a wildcard character.
- `index`: Used when both of the following conditions are met:
 - No `WHERE` clause is specified, or the table used contains no index that would speed up data retrieval
 - All columns in the `SELECT` statement for this table are available in the index (that is, the index is a covering index)
- `ALL`: Used when no index is ordered by the columns that are referred to by any `WHERE` or `ON` condition. This is typically a very ineffective access method.

Note: For more information about MySQL access methods, see the slide titled “Access Method” in this lesson.

EXPLAIN Output: key Column and Index Hints

The key column in the EXPLAIN output displays the key MySQL uses to retrieve data. You can affect the optimizer's choice of key by using *index hints* in your SQL statements:

- `USE INDEX (key_list)`: Instructs MySQL to use only one of the possible indexes from `key_list` to find rows in the table.
- `IGNORE INDEX (key_list)`: Instructs MySQL not to use a particular index from `key_list`.
- `FORCE INDEX (key_list)`: Acts like `USE INDEX (key_list)` but with the addition that a table scan is assumed to be *very expensive* and is used only if there is no way to use one of the named indexes to find rows in the table.



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

The optimizer uses many factors to choose the best index to use for any query, and you can normally rely on the optimizer's decision.

If the optimizer occasionally chooses an incorrect index and you want to ensure that it uses a specific index for that query, use an index hint to specify that index. For example, if the selectivity on a particular index is generally low, the optimizer might not use that index. However, if the selectivity is high for the value or range that you use in a specific query, and you know that the optimizer might choose a different index in some circumstances, you can add the `USE INDEX` or `FORCE INDEX` syntax on that query to ensure the optimizer uses it.

Avoid using index hints unless you are certain that your index choice is better than the choice of the optimizer under all circumstances in which your statement runs. Before benchmarking, ensure that the optimizer has accurate and up-to-date index statistics by executing the `ANALYZE TABLE` statement, and ensure that the statistics are regularly updated either by periodically executing `ANALYZE TABLE`, or by configuring the InnoDB statistics persistence options appropriately.

EXPLAIN Output: Extra Column

Contains additional information about the query execution plan. For example:

- Using `where`: A `WHERE` clause restricts which rows to match against the next table or send to the client.
- Using `filesort`: The sort buffer is being used to perform a quicksort operation on the output rows.
 - There is no way to tell from `EXPLAIN` if the output fits in memory or if it requires a file-merge sort, but this information is available in the Performance Schema.
- Using `temporary`: A temporary table is being used to store the result set at some stage of the execution. Often seen when using `GROUP BY` and `ORDER BY` clauses.
- Using `index`: All operations on this table reference are completed in the index (covering index.) No rows need to be read.
- Using `index condition`: The optimizer uses the Index Condition Pushdown optimization.



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

EXPLAIN Output: Extra column

- Using index for group-by: An index is available to retrieve all columns of a GROUP BY or DISTINCT query and no row reads are required.
- Distinct: Searching stops after the first matching row is found.
- Not exists: A LEFT JOIN optimization is used.
- Range checked for each record (index map: N) : There is no suitable index but range access can improve performance.
- Using join buffer (*algorithm*) : Tables from earlier joins are read in portions into the join buffer by using either the Blocked Nested Loop or Batched Key Access algorithms, and the join with the current table is performed by using the buffered rows.
- Using MRR: Tables are read by using the Multi-Range Read optimization strategy. This reduces the number of random disk accesses for range scans by first scanning only the index and collecting the keys for the relevant rows.



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

Structured EXPLAIN

- EXPLAIN FORMAT=JSON outputs optimizer information in JavaScript Object Notation (JSON) format.
- Includes all the information from classic EXPLAIN in machine-readable format
 - Useful for tooling
 - Some output column names are different, but similar. For example `type` is `access_type` in JSON output.
- Provides extra information that is not part of the standard EXPLAIN output, including:
 - Attached conditions and explicit casts
 - Which key parts from a composite index are used (`used_key_parts` field)
 - Pushed index condition information (the `attached_condition` within an `index_condition` node)
 - Ordering and grouping (`ordering_operation` and `grouping_operation` nodes)
 - Subquery attachment points (`select_list_subqueries` nodes)
 - Detailed optimizer cost information



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

Example: Standard EXPLAIN

```
mysql> EXPLAIN SELECT * FROM country
-> WHERE Name = 'Russian Federation'\G
***** 1. row *****
    id: 1
  select_type: SIMPLE
        table: country
    partitions: NULL
        type: ALL
possible_keys: NULL
         key: NULL
      key_len: NULL
        ref: NULL
     rows: 239
filtered: 10.00
  Extra: Using where
1 row in set, 1 warning (#.## sec)
```



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

In the example in the slide, the standard EXPLAIN output for the query shows that 239 rows are examined (`rows` column), but only 10% of those are used to create the result (`filtered` column.). The standard EXPLAIN output does not tell you why this is the case.

Example: EXPLAIN FORMAT=JSON

```
mysql> EXPLAIN FORMAT=JSON SELECT * FROM country WHERE Name = 'Russian Federation'\G
***** 1. row ****
EXPLAIN: {
  "query_block": {
    "select_id": 1,
    "cost_info": {
      "query_cost": "53.80"
    },
    "table": {
      "table_name": "country",
      "access_type": "ALL",
      "rows_examined_per_scan": 239,
      "rows_produced_per_join": 23,
      "filtered": "10.00",
      "cost_info": {
        "read_cost": "49.02",
        "eval_cost": "4.78",
        "prefix_cost": "53.80",
        "data_read_per_join": "6K"
      },
      "used_columns": [
        ...
      ],
      "attached_condition": "(`world`.`country`.`Name` = 'Russian Federation')"
    ...
  }
}
```

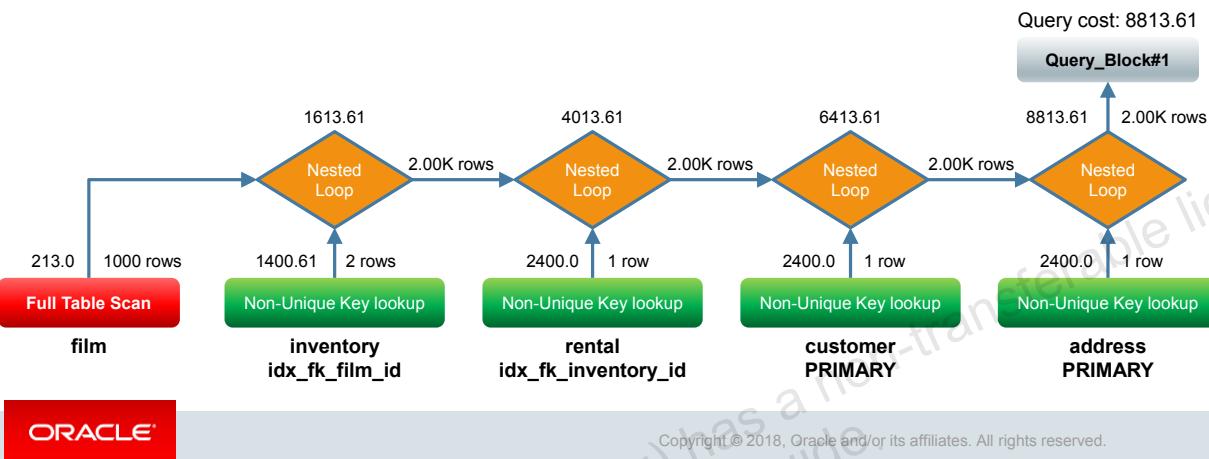


Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The JSON output shows how the `filtered` value is generated (`rows_produced_per_join` as a percentage of `rows_examined_per_scan`). Note also the detailed cost information provided by the optimizer (`cost_info` field).

Visual EXPLAIN in MySQL Workbench

```
SELECT CONCAT(customer.last_name, ' ', customer.first_name) AS customer, address.phone, film.title
FROM rental
INNER JOIN customer ON rental.customer_id = customer.customer_id
INNER JOIN address ON customer.address_id = address.address_id
INNER JOIN inventory ON rental.inventory_id = inventory.inventory_id
INNER JOIN film ON inventory.film_id = film.film_id
WHERE rental.return_date IS NULL
AND rental.rental_date + INTERVAL film.rental_duration DAY < CURRENT_DATE()
LIMIT 5;
```



MySQL Workbench uses JSON formatted EXPLAIN to provide a graphical view of the MySQL optimizer's query execution plan.

Quiz



Which field in the EXPLAIN output describes the method used by MySQL to access the data in the table or index?

- a. keys
- b. possible_keys
- c. select_type
- d. type

ORACLE

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

Answer: d

Quiz



Which of the following EXPLAIN type values is considered the most efficient access method?

- a. ALL
- b. const
- c. index_merge
- d. system

ORACLE

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

Answer: d

Quiz



Which EXPLAIN select_type value is also known as a correlated subquery?

- a. DEPENDENT UNION
- b. DEPENDENT SUBQUERY
- c. DERIVED
- d. SUBQUERY

ORACLE

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

Answer: b

Quiz



Which of the following statements forces MySQL to use the join order defined in the statement instead of that determined by the optimizer?

- a. FORCE_INDEX
- b. FORCE_JOIN
- c. STRAIGHT_JOIN
- d. USE_INDEX

ORACLE

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

Answer: c

Quiz



Which EXPLAIN Extra value indicates that a LEFT JOIN optimization has taken place?

- a. Distinct
- b. Not exists
- c. Using index
- d. Using_sort_union

ORACLE

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

Answer: b

Optimizer Trace

The optimizer trace utility provides extra information about the optimizer's decision-making process. To use the optimizer trace:

1. Enable tracing.
 - Execute `SET optimizer_trace="enabled=ON";`
2. Execute a query.
3. View the trace by querying the `INFORMATION_SCHEMA.OPTIMIZER_TRACE` table.
 - Output is in JSON format and includes:
 - The optimizer's analysis of the query
 - The execution plans the optimizer considered, including costs
 - The table join orders the optimizer evaluated
4. Disable tracing.
 - Execute `SET optimizer_trace="enabled=OFF";`



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

The output of the `EXPLAIN` statement shows an estimated query execution plan that the optimizer chooses for a specific query. If you want more detail about how the optimizer came to that decision, use optimizer tracing. The optimizer trace includes not only information about the actual query execution plan that it used to execute the query, but also all those plans that the optimizer considered and discarded. For example, if you want to understand why MySQL is using a full table scan when you expect it to use an index, the optimizer trace provides you with the information you need.

Topics

- How the optimizer works
- Understanding the query plan
- Optimizing queries



ORACLE®

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

Improving Query Performance

- Reduce the amount of data the query returns.
 - Minimize the number of columns in the `SELECT` statement.
 - Use a `WHERE` clause to filter irrelevant results.
 - Use a `LIMIT` clause to minimize the size of the resultset.
- Reduce the number of rows the server must filter.
 - Index relevant columns for fast lookup and ensure that the optimizer uses the indexes.
 - Delegate matching to the storage engine where possible (index condition pushdown).
 - Create indexes that can hold all required data for small, frequent queries (covering indexes).
- Rewrite the query.
- Create summary tables.



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

These strategies are covered in the following slides.

Data Retrieval: Best Practices

- Do not retrieve more rows than your application needs.
 - Limiting rows only in your application does not terminate the query after retrieving the number of rows required.
 - Using a `LIMIT` clause in your SQL statement terminates the query.
- Do not retrieve unnecessary columns.
 - Be careful with `SELECT *`, especially in joined tables.
 - The following statement returns all columns from all joined tables:

```
SELECT * FROM t1 INNER JOIN t2 USING(aID)
INNER JOIN t3 USING (bID);
```
 - Be specific about the tables you want to retrieve all columns from:

```
SELECT t1.*, t2.field, t3.field FROM t1
INNER JOIN t2 USING (aID)
INNER JOIN t3 USING (bID);
```



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

MySQL does not supply results on demand, like some other DBMS. Developers unfamiliar with MySQL assume that when they issue a query to return a specified number of rows (for example, to allow paging of query results) and then close the result set, it terminates the query. This is not the case. MySQL always generates the complete result set. A better approach is to use a `LIMIT` clause in the query, which saves network traffic because it only sends the specified number of rows to the client. However, `LIMIT` does not always reduce the number of rows that the server reads, so it does not always speed up the query performance at the server.

Take care when using `SELECT *` to return all the column data. This often includes columns that are not relevant, especially when joining tables, and adds unnecessary overhead. Using `SELECT *` prevents the optimizer from using any covering index that might cover a query that requests fewer columns. `SELECT *` returns the columns with the structure (and in the order) that exists in the table when you execute the statement. If you write `SELECT *` in your code, and the database structure changes, then your application might not receive the data in the same way that it expects. That is, the column definitions might be different and the columns themselves might be in the wrong order.

Filtering Data with WHERE

MySQL applies the WHERE clause in three different ways:

Method	Where filtering occurs	Reads rows?	EXPLAIN output (extra column)
Generic	Server	Yes	Using where
Covering index	Server	No	Using index
Index condition pushdown	Storage engine	No	Using index condition



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

Generic

Retrieve rows from the table, then filter unmatched rows. This happens at the server layer and requires the server to read rows from the table before it can filter them.

Covering Index

A covering index contains all the fields that the query needs to evaluate and return. Unmatched rows are filtered after retrieving each result from the index. This happens at the server layer, but it does not require reading rows from the table.

Index Condition Pushdown

Applies the conditions to the index lookup operation to eliminate unmatched rows. This happens at the storage engine layer. Index condition pushdown requires that all the fields required for condition evaluation are present in the index.

Indexing for Query Performance

The most important factor in query performance is the correct use of indexes. A good indexing strategy enables MySQL to:

- Avoid full table scans:
 - In a single table query, a full scan requires reading every row.
 - In a multi-table query, a full scan requires reading every single row from at least one table, and potentially all tables.
- Quickly:
 - Locate the first relevant result.
 - Exclude irrelevant results.
 - Find `MIN()` or `MAX()` values without examining every row.
- Sort and group data efficiently.
- Avoid row lookups entirely if covering indexes are available.



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

Indexing is the most important strategy for optimizing query performance. Sorting data rows in the table itself only provides one “view” into the table. Indexes allow multiple perspectives. The MySQL optimizer uses indexes to quickly match rows and filter out irrelevant results without having to do a full table scan.

Note: This course covers indexing strategies in the lesson titled “Optimizing Your Schema.”

Query Using WHERE Clause

```
mysql> EXPLAIN SELECT * FROM city
      -> WHERE Name='London' AND DISTRICT='Ontario'\G
***** 1. row *****
    id: 1
  select_type: SIMPLE
        table: city
     partitions: NULL
       type: ALL
possible_keys: NULL
         key: NULL
      key_len: NULL
        ref: NULL
     rows: 4188
  filtered: 1.00
   Extra: Using where
1 row in set, 1 warning (#.## sec)

mysql> SELECT COUNT(*) FROM city;
+-----+
| COUNT(*) |
+-----+
| 4079 |
+-----+
1 row in set (#.## sec)
```



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

Note that the number of rows estimated by the optimizer for the full table scan that the query requires is not equal to the actual number of rows in the `city` table.

InnoDB maintains only an approximate number of rows. This prevents MySQL from having to constantly access the table to determine the actual number of rows, which would affect performance.

Query Using Covering Index

```
mysql> ALTER TABLE City ADD INDEX `Name_Idx` (Name);
Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> EXPLAIN SELECT Name, ID FROM City
      -> WHERE Name LIKE 'Lon%'\G
***** 1. row *****
    id: 1
  select_type: SIMPLE
        table: City
       type: range
possible_keys: Name_Idx
      key: Name_Idx
  key_len: 35
      ref: NULL
     rows: 10
    Extra: Using where; Using index
1 row in set (0.00 sec)
```



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

This query uses a covering index, because all the column values it needs to match rows and return results are present in the index and MySQL does not have to read the `city` table to get this information.

The `Name` field is present in the index `Name_Idx`, and the `ID` field that is required in the resultset is the primary key. Every InnoDB secondary index implicitly contains the primary key (or, if there is no primary key, the internal unique row identifier.)

Query Using Index Condition Pushdown

```
mysql> EXPLAIN SELECT Name, District FROM City
-> WHERE Name LIKE 'Lon%\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: City
        type: range
possible_keys: Name_Idx
          key: Name_Idx
     key_len: 35
       ref: NULL
      rows: 10
    Extra: Using index condition
1 row in set (0.00 sec)
```



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

In this query, filtering can be performed by the storage engine, because the index contains all the information it requires. However, the server still has to read the rows, because it needs the District column in the results, which is not present in the index.

Costs of Indexing

There are costs involved in creating and maintaining indexes. Do not create unnecessary indexes.

- Data modifications are slower in tables that have many indexes.
 - Affected indexes must be updated to match the modified row.
 - This significantly impacts write-intensive workloads.
- Indexes consume disk space.
 - InnoDB stores data and index values together in the tablespace.
 - Be aware of filesystem limits.



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

Although indexes can greatly improve the performance of queries, they have a negative impact on data modification statements because they must be updated when the rows they refer to are changed. Do not create indexes that your queries do not use, and audit your indexes when you add, modify, or retire queries. Because each secondary index refers back to the primary key index that contains the table data, do not remove the PRIMARY index even if you think it is no longer used.

A useful way of finding redundant indexes is to query the `sys.schema_unused_indexes` view.

Indexing: Best Practices

- Consider indexes on columns that participate in
 - The WHERE clause
 - ORDER BY or GROUP BY clauses
 - Table join conditions
- Index columns with high cardinality (“uniqueness” of data).
 - For example, a gender column has low cardinality. A CustomerID column is likely to have high cardinality.
- Use the smallest appropriate data type for primary keys.
- Consider using prefix indexes on character string columns.
 - Quicker to compare and enable faster lookups
 - Require less disk I/O
- Do not create indexes that the optimizer will not use.



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

Just because a column appears in the query output, do not assume it needs an index.

The data types that you choose for a table’s primary key columns are an important consideration for efficiency and performance. The primary key is the ordering key for the PRIMARY index, which is the clustered index on which the table data is ordered. The primary key also uniquely identifies each row, so it is used in each secondary index to connect each index row back to the data row to which it refers. This means that the primary key column values exist not only in each row in the PRIMARY index, but also in each row in each secondary index, even if the index does not explicitly contain the primary key columns.

In some cases, you might create an index on a column and the optimizer might not use that index even if you refer to the column in the WHERE clause of a query. This is the case when the selectivity of an index is low, or when the statistics that the optimizer gathers do not accurately reflect the distribution of values in the column. For example, an index on a gender column has low selectivity because there are very few possible values for that column. Similarly, an index on an order_status column might have low cardinality (statuses might include opened, picking, packing, shipping, shipped, closed), and the majority of orders might be in the “closed” status. In such cases, the optimizer probably determines that a full table scan has a lower cost than using the low selectivity index (bearing in mind that, for each secondary index lookup, it must also perform a clustered index lookup) and it does not use the index.

Note: Every InnoDB table has a special index called the *clustered index* where the data for the rows is stored. Typically, the clustered index is synonymous with the primary key.

Leftmost Prefixes

- A composite index consists of two or more columns.
- MySQL can use any leftmost set of columns to match rows.
 - You do not need to provide separate indexes on those leftmost columns.
- Example: A `Staff` table, with a composite index on `department`, `team`, and `role` columns
 - Index rows are in department/team/role order.
 - Use the same index to search for the following column combinations:
 - `department, team, role`
 - `department, team`
 - `department`



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

One way to avoid creating unnecessary indexes is to understand how MySQL uses leftmost prefixes and not create a separate index when a leftmost prefix index already exists.

Similarly, you might find that a table that contains a few well-thought-out composite indexes might be more efficient than the same table with more single-column indexes, because a composite index might prove useful for several query types, even if it does not cover all of them.

Note: A leftmost prefix is not the same as a prefix index. The prefix index is described in the lesson titled “Optimizing Your Schema.”

Improving the Performance of SELECT Statements

If your SELECT statements are consistently slow, consider the following strategies:

- Avoid full table scans wherever possible.
- Consider adding indexes on:
 - Columns in the WHERE clause
 - Columns that participate in table joins
- Isolate and investigate individual parts of the query.
 - For example, does an expression evaluate more frequently than required?
- Use ANALYZE TABLE periodically to keep the optimizer statistics up to date.
- Learn storage engine-specific optimizations.
 - For example, InnoDB read-only transactions



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

To improve the performance of a slow SELECT query, the first thing to consider is indexing on the tables referenced by the query. In some cases you can use existing indexes to improve the query's performance, for example by rewriting the query or by splitting the query into two queries. Sometimes you might need to add new indexes.

Add indexes on columns used in the WHERE clause to improve the performance of condition evaluation, row filtering, and result retrieval. Avoid wasted disk space by using a small set of indexes that work with many related queries used by your application. Determine that the index is being used correctly by the optimizer, by issuing EXPLAIN on the statement. If the optimizer is not using the correct index, consider using optimizer hints.

Develop the habit of checking your SELECT statements to see how the optimizer rewrites them and use the same techniques when formulating your own queries.

Optimizing Table Joins

- Add indexes on columns that are contained in `ON` or `USING` clauses.
- Use `STRAIGHT_JOIN` to force the join order if the optimizer regularly chooses a join order that you know to be suboptimal.
- Use columns from a single table in `ORDER BY` and `GROUP BY` operations.
 - Enables the optimizer to use indexes on those columns
- Do not automatically assume that a subquery will perform better if you rewrite it as a join.



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

The optimizer is excellent at determining the join order, so you should only very rarely have to force it. If you use index hints, regularly check that the query plan you force is optimal: you might find that the data's distribution or usage patterns change such that the optimizer identifies a different plan as optimal, and if you upgrade to a newer version of MySQL, the optimizer might have improvements that remove any benefit from existing optimizer hints.

In early versions of MySQL, the performance of subqueries could often be improved by rewriting them as joins. Significant improvement to the optimizer means that this is usually not required in later versions, and some subqueries perform faster than joins.

Optimizing Sorting Operations

The following MySQL server status variables tell you how effective a statement with an ORDER BY or GROUP BY clause is:

- Sort_scan: The number of sorts that were performed by scanning the table.
- Sort_rows: The number of sorted rows.
- Sort_range: The number of sorts that were performed by using ranges.
- Sort_merge_passes: The number of merge passes that the sort algorithm has had to perform.
 - A merge pass requires a disk-based sort and is ineffective.
 - Consider increasing the size of the sort buffer if you see many Sort_merge_passes.



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

Sort_scan

- Helps you to identify sorting operations that do not use indexes.
- These queries are the most inefficient, as they scan entire tables to sort their records in the required order.

Sort_rows

- Helps you to identify queries that involve complex sorting operations.
- If this value is high compared to Questions, optimize your ORDER BY and GROUP BY queries to read the required records in sorted order.

Sort_range

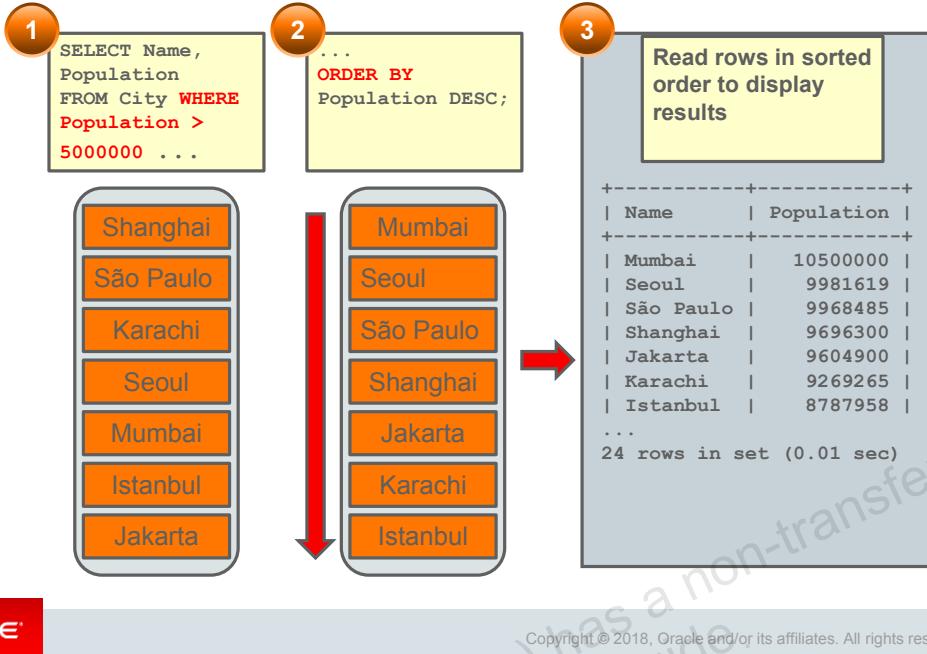
- Sort_range queries are more efficient than Sort_scan queries, but not as efficient as queries that sort based entirely on a key.
- If this value is high compared to Questions, investigate these queries to establish if they can use a key-based sort.

Sort_merge_passes

- When a query exceeds the available memory assigned in this buffer, it uses a disk-based sort (disk seek). The optimizer estimates how much space is required, but it can allocate up to the limit specified.
- If you see many Sort_merge_passes in the output of SHOW GLOBAL STATUS, consider increasing the sort_buffer_size to speed up ORDER BY or GROUP BY operations that cannot be improved by optimizing the queries.
- **Note:** This course covers the sort buffer in the lesson titled “General Statement Tuning.”

Order of Sort Operations

+



ORACLE

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

To display the names and populations of cities in the `City` table in descending order of population, the `SELECT` operation in step 1 must be followed by a sort operation in step 2.

Following the sort operation, a `Sort_scan` and `Sort_range` is incurred in step three.

- If step one results in `Select_scan`, step three requires a `Sort_scan`.
- If step one results in a `Select_range`, then step three requires a `Sort_range`.
- Both `Sort_scan` and `Sort_range` queries retrieve only the necessary rows in sorted order. Therefore there is no difference in the sort performance with `Sort_scan` and `Sort_range`.

Note: If the key the Optimizer uses for the sort operation is in the same order as the required results, the Optimizer bypasses step 2. However, do not assume that the Optimizer will use a particular key.

Optimizing Bulk DML Statements

- INSERT
 - Use multiple row `INSERT` statements instead of individual row `INSERTS`.
 - Use `LOAD DATA INFILE` to bulk load tables.
- UPDATE
 - Use `SELECT` statement best practices.
 - `UPDATE` is like `SELECT`, with the extra write overhead.
 - Execute multiple row `UPDATE` statements instead of individual row `UPDATES`.
- DELETE
 - `TRUNCATE TABLE tablename` is faster than `DELETE FROM tablename`.



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

The slide shows optimal methods for modifying large amounts of data. Bear in mind that any time you modify large amounts of data, the server must perform a lot of additional processing to manage that operation. This processing includes transactional overhead, logging (redo, undo, binary), buffer pool flushing, and the disk I/O required to support the operation. If you must perform large data modifications at one time, consider performing them in smaller batches.

`TRUNCATE TABLE` performs limited logging because it does not perform operations on rows. Rather, it recreates the table and all of the table's settings, including its `AUTO_INCREMENT` counter.

Creating Summary Tables

- Consider creating summary tables if your queries are very complex and do not perform well. For example, if your queries:
 - Need to read rows from many tables
 - Rely on complex expressions
 - Have complex sorting/ordering requirements
- Querying the summary table is faster.
 - Smaller number of rows
- The disadvantages of summary tables are that they:
 - Represent historical, rather than current, data
 - Need to be updated regularly
 - Manually, or via a cron job
 - Increase the amount of data that needs to be stored



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

The most significant disadvantage of summary tables is that they need to be updated regularly. It is important to consider how you will achieve this if you decide to use summary tables.

Quiz



Which of the following are not good strategies for using indexes?

- a. Index the columns that participate in the join condition.
- b. Index the columns that are in a function expression.
- c. Index all columns that are in the ORDER BY clause in a single table query.
- d. Index all columns that are in the SELECT column list but not in the WHERE clause.

ORACLE

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

Answer: b, d

Summary



In this lesson, you should have learned how to:

- Explain the role and operation of the MySQL query optimizer
- Determine the execution plan for a query by using the EXPLAIN statement
- List good strategies for improving query execution times
- Choose appropriate indexes to maximize query performance
- Rewrite suboptimal queries to take full advantage of indexes
- Use MySQL GUI tools to investigate optimizer behavior



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

Practices

- 10-1: Understanding the Query Execution Plan with the EXPLAIN Statement
- 10-2: Improving the Performance of a Query
- 10-3: Tracing the Optimizer



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

11

Optimizing Locking Operations



ORACLE®

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

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

Topics

- How MySQL locking works
- InnoDB lock types
- Metadata locks
- Viewing lock information with SQL statements
- Investigating locks by querying system databases



ORACLE®

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

Objectives



After completing this lesson, you should be able to:

- Describe how and why MySQL locks database objects
- List the different lock types InnoDB uses
- View basic locking information in the process list and `SHOW ENGINE INNODB STATUS`
- Retrieve detailed locking information by querying the Performance Schema

ORACLE®

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

Topics

- How MySQL locking works
 - InnoDB lock types
 - Metadata locks
 - Viewing lock information with SQL statements
 - Investigating locks by querying system databases



ORACLE®

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

Locks in MySQL

MySQL locks resources in the following ways:

- Server-level data locks
 - Table locks
 - Metadata locks
- InnoDB storage engine data locks
 - Locks rows by default
 - Can lock entire table if transaction accesses many rows
- Mutexes
 - Lower-level locks that apply to internal resources rather than to data



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

MySQL locks tables, rows, and internal resources to ensure consistency across all of the processes and threads that use those resources. This is what allows multiple connections and internal threads to appear to access the same data at the same time.

Some locks are held at the server level, and some at the storage engine level. In general, the server locks at the table level and the storage engine locks at a lower level. InnoDB locks rows by default, although it can also escalate row locks to table locks if a transaction must lock many rows. MySQL also supports user-level locks, such as those created with the `GET_LOCK()` function. User-level locks cause an application to wait until the lock is granted, but do not themselves lock any data.

Mutexes (short for mutual exclusions) are internal locks used for synchronizing low-level code operations. They ensure that only one thread can access each resource at a time. Resources that are protected by mutexes include log files and `AUTO_INCREMENT` counters. You can identify mutex contentions by querying for `synch` instruments in the Performance Schema `events_waits%` tables.

Implicit Locks

Some data operations lock data implicitly for the duration of a transaction or of a single statement.

- Transactional locks:
 - Statements that you execute after START TRANSACTION
 - Statements that you execute when the AUTOCOMMIT variable is off
- Single statements that block other statements without explicit locks:
 - ALTER TABLE ... ALGORITHM=COPY statements
 - Queries that you execute while the SERIALIZABLE isolation level is selected
 - Data modification operations
 - Locks are more noticeable on high-traffic systems.



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

Statements might block other statements even when your transactions are short or when you are using single-statement transactions, for example when AUTOCOMMIT is enabled.

If an UPDATE statement on a particular row takes 2 milliseconds to execute, then no other statement can modify that row within that period. On high-traffic systems that receive more than 500 updates per second, this means that statements are likely to be blocked.

If you select the SERIALIZABLE isolation level, each InnoDB transaction acquires a read lock for all rows read by any SELECT statement in that transaction, as if you used the LOCK IN SHARE MODE query option. These locks conflict with write locks acquired by UPDATE and DELETE statements, and can result in delays on high-traffic systems.

The READ-COMMITTED and REPEATABLE-READ isolation levels use multiversion concurrency control to avoid many such locks.

Note: This lesson covers multiversion concurrency control in the slide titled “Avoiding Locks with Multiversion Concurrency Control.”

Explicit Locks

You can lock data explicitly by using the following commands:

- Server locks:
 - LOCK TABLES
 - Acquires table locks for the current session.
 - FLUSH TABLES WITH READ LOCK
 - Locks all tables for all databases with a global read lock.
 - Release table locks with UNLOCK TABLES.
- InnoDB locks:
 - SELECT ... FOR UPDATE
 - Locks rows and corresponding index entries, as if this was an UPDATE.
 - Prevents other transactions from writing rows
 - Also prevents reads in certain transaction isolation levels.
 - SELECT ... LOCK IN SHARE MODE
 - Other sessions can read the rows, but cannot modify them until your transaction commits.



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

The InnoDB locks shown in the slide lock rows for the duration of the transaction. If the AUTOCOMMIT variable is enabled and you have not previously executed START TRANSACTION to begin an explicit transaction, each statement is automatically committed as a single transaction. In such cases the FOR UPDATE and LOCK IN SHARE MODE query modifiers do not lock rows beyond the duration of the SELECT statement.

Avoiding Locks with InnoDB Multiversion Concurrency Control

- Creates snapshots of the data at the beginning of a data modification statement.
- Enables consistent reads without locking rows.
- Prevents queries from seeing uncommitted changes.
- Works with the following transaction isolation levels:
 - READ-COMMITTED:
 - The snapshot contains all committed data at the time the statement begins.
 - Each new query within a transaction generates a new snapshot containing all committed data at that time.
 - REPEATABLE-READ:
 - The snapshot contains all committed data at the time the statement begins.
 - The transaction maintains the same snapshot for all queries in the transaction.
- Is stored in the tablespace in a structure called a *rollback segment*.



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

InnoDB multiversion concurrency control (MVCC) enables multiple transactions to access the same table data when modifications do not conflict. This means that if you run a batch `INSERT` operation and it takes a long time, a `SELECT` operation issued while the `INSERT` is still running operates on the version of the data before the `INSERT` started. Effectively, MySQL creates a snapshot at the beginning of the operation, which enables a consistent read of the data as it existed at that point. Snapshot-based consistent reads are enabled only when the transaction isolation level is `READ-COMMITTED` or `REPEATABLE-READ`.

If you insert and delete rows in small batches at about the same rate in the table, the purge thread can start to lag behind and the table can grow bigger and bigger because of all the “dead” rows, making everything disk-bound and very slow. In such a case, throttle new row operations, and allocate more resources to the purge thread by tuning the `innodb_max_purge_lag` system variable.

Topics

- How MySQL locking works
- InnoDB lock types
- Metadata locks
- Viewing lock information with SQL statements
- Investigating locks by querying system databases



ORACLE®

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

InnoDB Lock Modes

- Shared (*S*):
 - Permits the transaction to read a row
 - Allows other transactions to acquire shared (or intention shared) locks
- Exclusive (*X*):
 - Permits the transaction to write a row
 - Does not allow any other transaction to access the row
- Intention (*IS*, *IX*):
 - Permits the transaction to acquire a shared or exclusive lock at a later point
 - Must be acquired before the transaction acquires an *S* or *X* lock
 - Allows other transactions to acquire *IS* or *IX* locks
 - *IS* also allows transactions to acquire *S* locks.



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

Transactions acquire shared and exclusive locks on individual rows, and acquire intention locks on entire tables. Intention locks indicate that the transaction intends to acquire a row lock within the table at some point later in the transaction.

Examples:

- `SELECT ... FROM tbl1 LOCK IN SHARED MODE` acquires an *IS* lock on the `tbl1` table. At some later point, the transaction might make some decision based on a value in that table, and while it holds the *IS* lock, no other transaction can acquire an exclusive lock on any row in that table.
- `SELECT ... FROM tbl2 FOR UPDATE` acquires an *IX* lock on the `tbl2` table. At some later point the transaction might modify some row in the table, at which time it acquires an *X* lock on that row. While it holds the *IX* lock, no other transaction can acquire an *S* or *X* lock on any row within the table.

Lock Type Compatibility Matrix

	X	IX	S	IS
X				
IX				
S				
IS				

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

The table in the slide shows if a transaction can acquire a lock when another transaction holds a lock.

For example, if transaction *A* holds an exclusive lock on a row, and transaction *B* requests a shared lock on that row, those locks are in conflict and transaction *B* becomes blocked waiting for that lock.

On the other hand, if transaction *C* holds a shared lock on a row and transaction *D* requests a shared lock on the same row, those locks are compatible and transaction *D* acquires the lock.

Topics

- How MySQL locking works
- InnoDB lock types
- **Metadata locks**
- Viewing lock information with SQL statements
- Investigating locks by querying system databases



ORACLE®

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

Metadata Locks

- Manage concurrent access to database objects:
 - Tables
 - Schemas
 - Stored programs
 - Stored procedures, functions, triggers, and scheduled events
 - Tablespaces
- Prevent one session from performing a DDL operation on a database object in use by another session
 - Ensures data consistency
- Are acquired at the beginning of a transaction and released when the transaction completes



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

Metadata Lock Example

Session 1 acquires a metadata lock on table t1

```
mysql> START TRANSACTION;  
mysql> SELECT * FROM t1;
```

Session 2 cannot complete until the transaction in session 1 commits or is rolled back

```
mysql> ALTER TABLE t1  
-> DROP NAME;
```

The metadata lock is visible in the output of sys.processlist (or SHOW PROCESSLIST)

```
mysql> SELECT * FROM sys.processlist;  
...  
***** 2. row *****  
thd_id: 31  
conn_id: 6  
user: root@localhost  
db: locktest  
command: Query  
state: Waiting for table metadata lock  
time: 7  
current_statement: alter table t1 drop Name  
...
```

ORACLE

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

The example in the slide shows two sessions. The first session explicitly starts a transaction and queries table t1. The second session attempts to modify the structure of table t1 but cannot because the first session has acquired a metadata lock on the table. The second session waits until the transaction the first session completes or is rolled back. The presence of the metadata lock is visible in the state column in the output of sys.processlist or SHOW PROCESSLIST.

Topics

- How MySQL locking works
- InnoDB lock types
- Metadata locks
- Viewing lock information with SQL statements
- Investigating locks by querying system databases



ORACLE®

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

Identifying Locks in the Process List

- Locks are recorded in the `state` column in the output of `sys.processlist` or `SHOW PROCESSLIST`:
 - The table locked in `READ` mode:
State: Waiting for table level lock
 - The table locked in `WRITE` mode:
State: Waiting for table metadata lock
 - An InnoDB row (or table) lock:
State: updating OR
State: Searching for rows for update
- The process list does not display internal InnoDB information about locks.



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

The process list does not show details of transactional locks that originate in the storage engine. Such locks might include:

- Conflicting modifications (for example an `UPDATE` and a `DELETE` that affect the same row)
- Performing a `SELECT` with `LOCK IN SHARE MODE`
- Performing a `SELECT` with `FOR UPDATE`

In each case, you are likely to see the state `updating` in the process list if a statement is blocked by one of these conditions.

InnoDB cancels operations that are blocked for a number of seconds defined by the `innodb_lock_wait_timeout` setting. By default, this timeout is 50 seconds. If an operation times out, you receive the following error.

`ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction`

If you frequently receive this error, consider increasing the value of the `innodb_lock_wait_timeout` setting, but bear in mind that such timeouts might be a symptom of another problem, and that if you increase the value of that setting, you are enabling longer transactions that might cause even longer application waits.

Example: Table Lock Information in SHOW PROCESSLIST

```
mysql> SHOW PROCESSLIST\G
***** 1. row *****
Id: 14
User: root
Host: localhost
db: world
Command: Query
Time: 15
State: Waiting for table level lock
Info: INSERT INTO City VALUES (5000, 'Galway', 'IRL', 'Connaught', '80000')
...
```



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

After you execute a statement such as `LOCK TABLES City READ`, your connection holds an `IS` lock on the table so that all statements that attempt to modify that table are blocked. You can use `SHOW PROCESSLIST` to see currently blocked statements. The value in the `State` column in the example output shown in the slide tells you that another process has locked the table in `READ mode`. If a process locks the table in `WRITE mode`—acquiring an `IX` lock—you see the state `Waiting for table metadata lock`. When you modify the table structure with an `ALTER TABLE ... ALGORITHM = COPY` operation, it also holds a metadata lock on that table for the duration of the operation.

Displaying InnoDB Lock Information with SHOW ENGINE INNODB STATUS

```
-----
TRANSACTIONS
-----
...
LIST OF TRANSACTIONS FOR EACH SESSION:
...
---TRANSACTION 549222, ACTIVE 51 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 360, 1 row lock(s)
MySQL thread id 3, OS thread handle 0x7f6807ec8700, query id 61 ...
UPDATE table SET column = value WHERE ...
----- TRX HAS BEEN WAITING 51 SEC FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 58 page no 6 n bits 568 index `PRIMARY` of
table `database`.`table` trx id 549222 lock mode X waiting ...

```



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

In the example in the slide, an UPDATE statement is blocked by some other process that holds a lock on the affected rows. You can see the statement, its transaction ID, and the thread ID of its connection in the list of transactions, as well as the lock mode (in this case `lock_mode X`) for which the transaction is waiting. The transactions list contains other transactions and their statements and connections, so you can identify which transaction is causing the lock.

Although `SHOW ENGINE INNODB STATUS` is useful for displaying currently blocked statements, it is limited in its use for linking the blocked statement to the statement that is holding the lock. If you enable the InnoDB Lock Monitor, you see all row locks on all indexes for each transaction, so that you can cross-reference blocked transactions with the transactions that hold the blocking locks. On busy systems, there might be many thousands of such locks.

To enable the InnoDB Lock Monitor, set the following system variables:

```
SET GLOBAL innodb_status_output=ON;
SET GLOBAL innodb_status_output_locks=ON;
```

Topics

- How MySQL locking works
- InnoDB lock types
- Metadata locks
- Viewing lock information with SQL statements
- Investigating locks by querying system databases



ORACLE®

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

Viewing Metadata Lock Information in the Performance Schema

The `performance_schema.metadata_locks` table:

- Enables you to determine:
 - Which locks a session is waiting for
 - Which sessions currently hold those locks
- Requires you to enable `wait/lock/metadata/sql/mdl` instrumentation
- Includes the following columns:
 - `LOCK_TYPE`: The lock type from the metadata lock subsystem
 - `LOCK_DURATION`:
 - `STATEMENT` or `TRANSACTION`: The scope of the lock, at the end of which it is released
 - `EXPLICIT`: The lock must be explicitly released
 - `OWNER_THREAD_ID/OWNER_EVENT_ID`: The thread or event requesting the lock
 - `LOCK_STATUS`: The status of the lock



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

Interpreting metadata_locks.LOCK_STATUS

- GRANTED: A session requests a metadata lock and obtains it immediately.
- PENDING: A session is waiting for a metadata lock.
 - The status is updated to GRANTED when the session obtains the metadata lock
 - If the lock request times out, the status changes to TIMEOUT
 - If the lock request is canceled by the deadlock detector, the status changes to VICTIM
- When any granted or pending lock request is killed, LOCK_STATUS displays KILLED
- When a metadata lock is released, the corresponding row is deleted from the metadata_locks table.



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

Other possible values of LOCK_STATUS are PRE_ACQUIRE_NOTIFY and POST_RELEASE_NOTIFY. These are brief and signify that the metadata locking subsystem is notifying interested storage engines while entering lock acquisition or leaving lock release operations.

Example: Querying the metadata_locks Table

```
mysql> SELECT processlist_id, object_type, lock_status
-> FROM metadata_locks JOIN threads on (owner_thread_id = thread_id)
-> WHERE object_schema='employees' AND object_name='titles'\G
***** 1. row *****
processlist_id: 4
object_type: TABLE
lock_type: EXCLUSIVE
lock_status: PENDING
***** 2. row *****
processlist_id: 5
object_type: TABLE
lock_type: SHARED_READ
lock_status: GRANTED
```

This thread is waiting for an exclusive lock on the employee.titles table.

This thread has been granted a shared (read) lock on the employee.titles table.



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

Viewing Table Lock Information in the Performance Schema

The `performance_schema.table_handles` table

- Stores information about
 - Which tables the server currently has open
 - Which tables are currently locked and how
 - Which sessions are locking the tables
- Includes the following columns
 - `OWNER_THREAD_ID/OWNER_EVENT_ID`: The thread or event that owns the table handle
 - `INTERNAL_LOCK`: The table lock at SQL level:
 - READ, READ WITH SHARED LOCKS, READ HIGH PRIORITY, READ NO INSERT, WRITE ALLOW WRITE, WRITE CONCURRENT INSERT, WRITE LOW PRIORITY, or WRITE.
 - `EXTERNAL_LOCK`: The table lock at the storage engine level
 - READ EXTERNAL or WRITE EXTERNAL.



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

Example: Querying the `table_handles` Table

```
mysql> SELECT * FROM table_handles WHERE object_name = 't1'\G
***** 1. row *****

    OBJECT_TYPE: TABLE
    OBJECT_SCHEMA: locktest
    OBJECT_NAME: t1
    OBJECT_INSTANCE_BEGIN: 1041121211561212
    OWNER_THREAD_ID: 23
    OWNER_EVENT_ID: 4379
    INTERNAL_LOCK: READ
    EXTERNAL_LOCK: READ EXTERNAL

The SQL lock type
The storage engine lock type
The thread that acquired the handle
The event the caused the table handle to be opened
```



Viewing Table Lock Information in sys.innodb_lock_waits

```
mysql> SELECT * FROM
-> sys.innodb_lock_waits\G
***** 1. row *****
    waiting_query: insert into departments
    values ('d010', 'Distribution')
    waiting_lock_id: 9801258:55:4:3
    waiting_lock_mode: X,GAP
    blocking_trx_id: 9801257
    blocking_pid: 8
    blocking_query: NULL
    blocking_lock_id: 9801257:55:4:3
    blocking_lock_mode: X
    blocking_trx_started: 2017-10-13 14:34:03
    blocking_trx_age: 00:01:28
    blocking_trx_rows_locked: 19
    blocking_trx_rows_modified: 0
    sql_kill_blocking_query: KILL QUERY 8
    sql_kill_blocking_connection: KILL 8
1 row in set, 3 warnings (#.## sec)
```



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

The `sys.innodb_lock_waits` view provides information about the waiting query as well as the query that blocks it. You can use the `blocking_pid` column value to query the `threads` table in the Performance Schema to retrieve the thread ID for the blocking thread and then query the `events_statements_*` or `events_transactions_*` tables to determine which operation has caused the block.

Note: The statement shown in the slide produces warnings. This is because `innodb_lock_waits` depends upon tables in the Information Schema that are deprecated and due to be removed in a future version of MySQL. Oracle recommends continuing to use `innodb_lock_waits` to investigate table locks because the view will be updated in future releases to retrieve the same information from the Performance Schema.

Quiz



Which of the following provides the easiest way to find out which statement holds a lock on a table that is blocking another query?

- a. Querying the INFORMATION_SCHEMA.INNODB_TRX table
- b. Querying the performance_schema.table_handles table
- c. SHOW PROCESSLIST
- d. SHOW ENGINE INNODB STATUS

ORACLE

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

Answer: b

The table_handles table in the Performance Schema enables you to look up the thread that holds the lock in the ps.threads table. The information is also currently available in INFORMATION_SCHEMA, but the tables that store that information are deprecated and will be removed in a later version.

Summary



In this lesson, you should have learned how to:

- Describe how and why MySQL locks database objects
- List the different lock types InnoDB uses
- View basic locking information in the process list and `SHOW ENGINE INNODB STATUS`
- Retrieve detailed locking information by querying the Performance Schema

ORACLE®

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

Practices

- 11-1: Troubleshooting Blocked Queries
- 11-2: Investigating Metadata Locks



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

12

Tuning Replication



MySQL™

ORACLE®

Topics

- Replication overview
- Understanding replication lag
- Diagnosing replication lag
- Resolving replication lag



ORACLE®

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

Objectives



After completing this lesson, you should be able to:

- List possible causes of replication lag
- State the difference between I/O thread lag and SQL thread lag
- Identify replication lag using binary log file names and positions
- Identify replication lag in GTID-based replication
- Resolve replication lag

ORACLE®

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

Topics

- Replication overview
 - Understanding replication lag
 - Diagnosing replication lag
 - Resolving replication lag

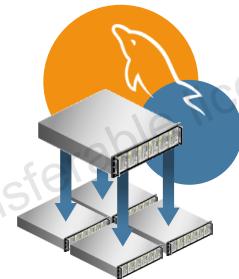


ORACLE®

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

MySQL Replication

- A MySQL feature that allows servers to copy changes between instances.
- Typically involves one *master* and one or more *slaves*
 - All writes occur on the master and are recorded in the binary log.
 - All reads occur on the slaves.
 - The slaves keep current by requesting the binary log from the master and applying its contents locally.
 - Each slave keeps track of how much of the log it has processed and, in the event of a network outage, resumes processing automatically when connectivity is restored.
- Supports other replication topologies, including:
 - MySQL Group Replication: Multiple masters arranged in a group.
 - Circular replication: Each server acts as both slave and master.



ORACLE

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

Note

- **Binary log formats:** MySQL supports statement-based, row-based, and mixed format logging as described in the “Binary Log Formats” slide later in the lesson.
- **MySQL Group Replication:** The MySQL Group Replication feature is a multi-master “update anywhere” replication plugin for MySQL with built-in conflict detection and resolution, automatic distributed recovery, and group membership.

Replication Use Cases

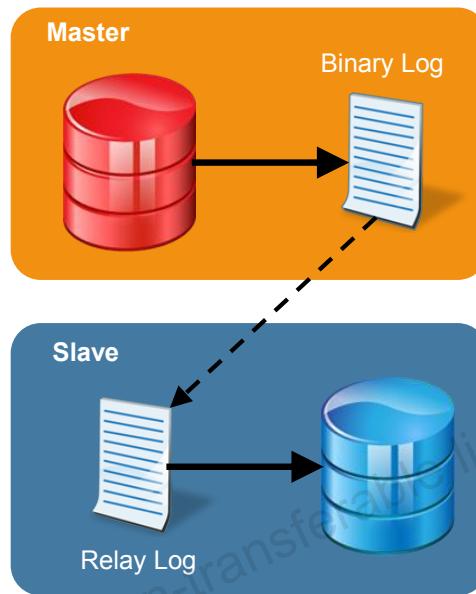
- Scale out solutions:
 - All writes and updates occur only on the master, for faster writes.
 - All reads occur on one of multiple slaves, for faster reads.
- Data security:
 - If a master crashes, you can promote a slave to be the master.
 - You can pause replication on any slave and execute backups without corrupting the corresponding master data.
- Reporting and analytics
 - Query-intensive reporting operations can take place on a dedicated slave without affecting overall application performance.
- Geographic distribution
 - Use replication to create a local copy of data for a remote site without permanent access to the master.



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

Replication Logs

- The master server
 - Executes statements that change data (writes)
 - Saves the changes in the **binary log**
- The slave server
 - Connects to the master to retrieve the binary log
 - Copies changes to its local **relay log**
 - Reads the relay log
 - Applies the changes



ORACLE®

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

Replication Log Events

The type of event that is written to the binary log depends on the setting of the `binlog_format` system variable:

- STATEMENT
 - The master writes SQL statements to the binary log
 - The slave executes the SQL to apply the changes
- ROW
 - The master writes the changed rows to the binary log
 - The slave copies the changes to the rows
- MIXED
 - Uses a mixture of STATEMENT and ROW-based logging
 - The storage engine decides which format to use for a given event.
 - Provides the best performance for most applications



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

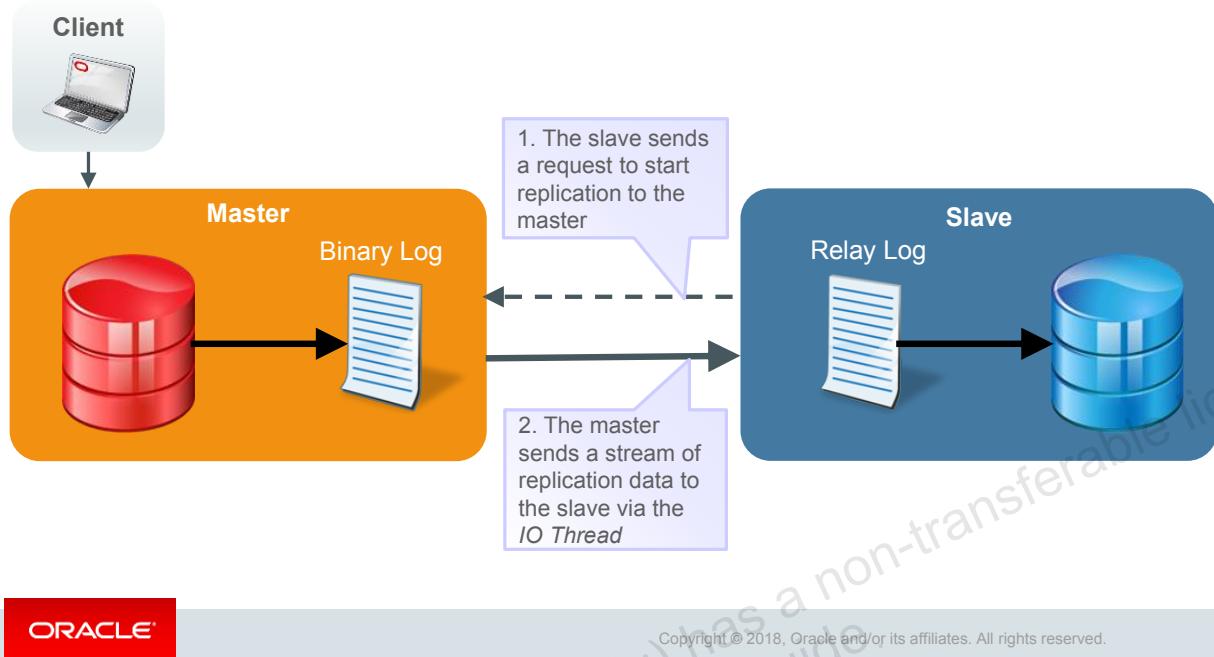
Role of the Master

- Accepts requests from clients to modify database objects
- Applies the changes locally and writes events to the binary log for slave to read later.
 - The *binlog dump thread* notifies slaves of changes and sends the contents of the binary log.
- Or, if *global transaction identifiers* (GTIDs) are enabled, the GTID of the last-executed transaction is stored in the `gtid_executed` system variable on each server and the slave sends its value to the master.
 - The master knows which transactions the slaves has executed.
 - The master sends only those transactions that the slave has not already executed.



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

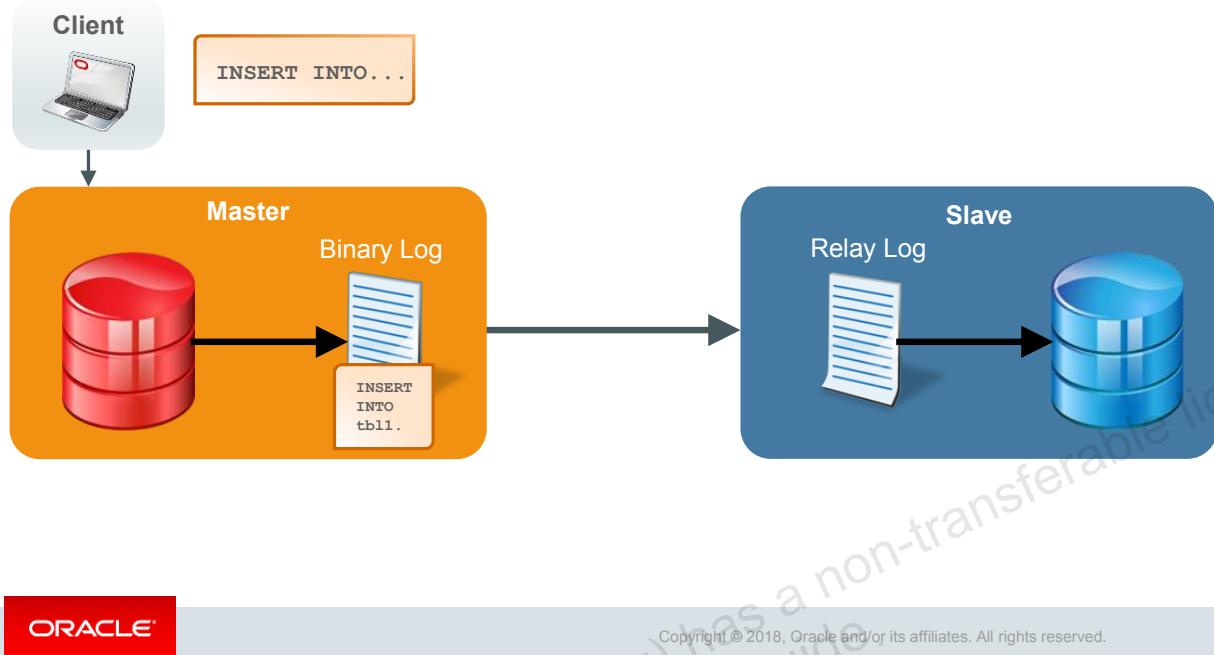
Role of the Slave



In the diagram in the slide, the slave sends a request to the master to start replication and the master responds by sending a stream of replication data to the slave via the IO Thread.

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

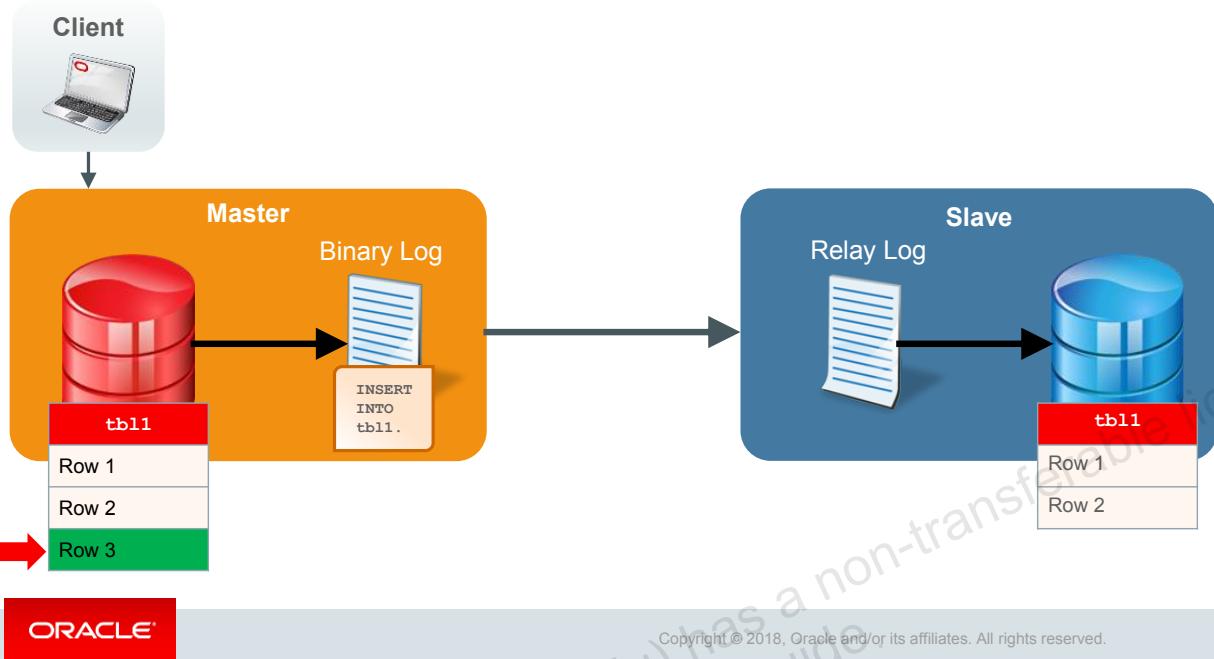
Role of the Slave



In the diagram in the slide, the client writes a new row to the database on the master, and the changes are recorded in the master's binary log.

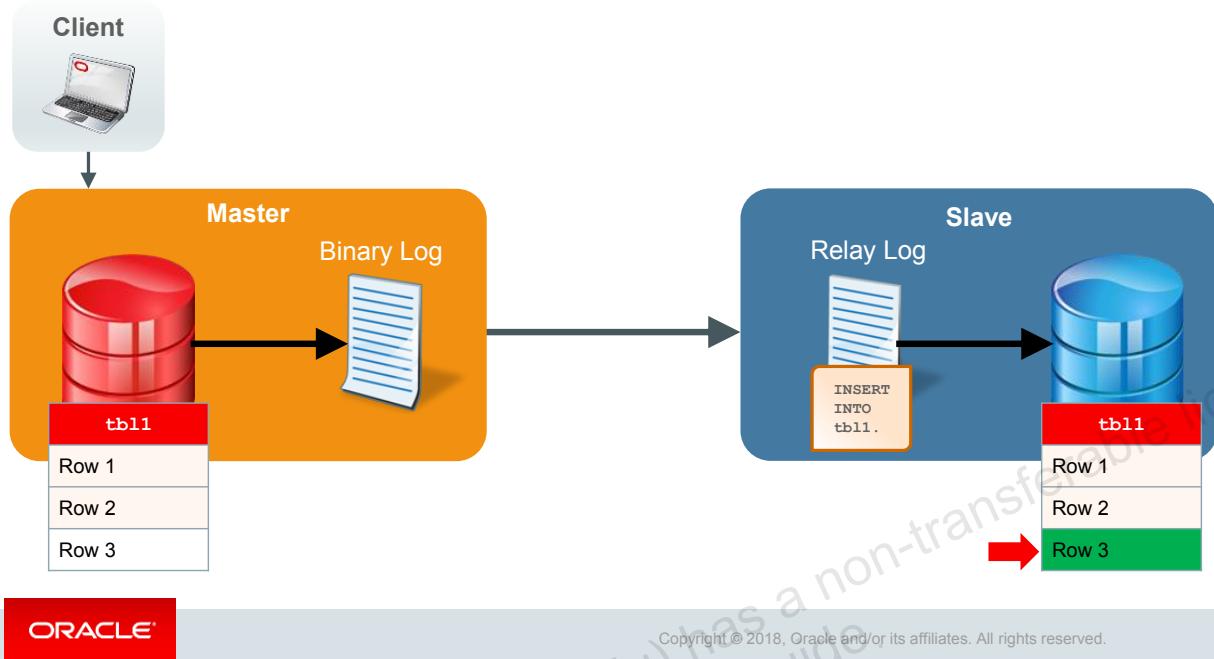
Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

Role of the Slave



In the diagram in the slide, the master's version of the database is now different from the slave version.

Role of the Slave



In the diagram in the slide, the slave applies the contents of the relay log to ensure that the slave version of the database is the same as the master.

Topics

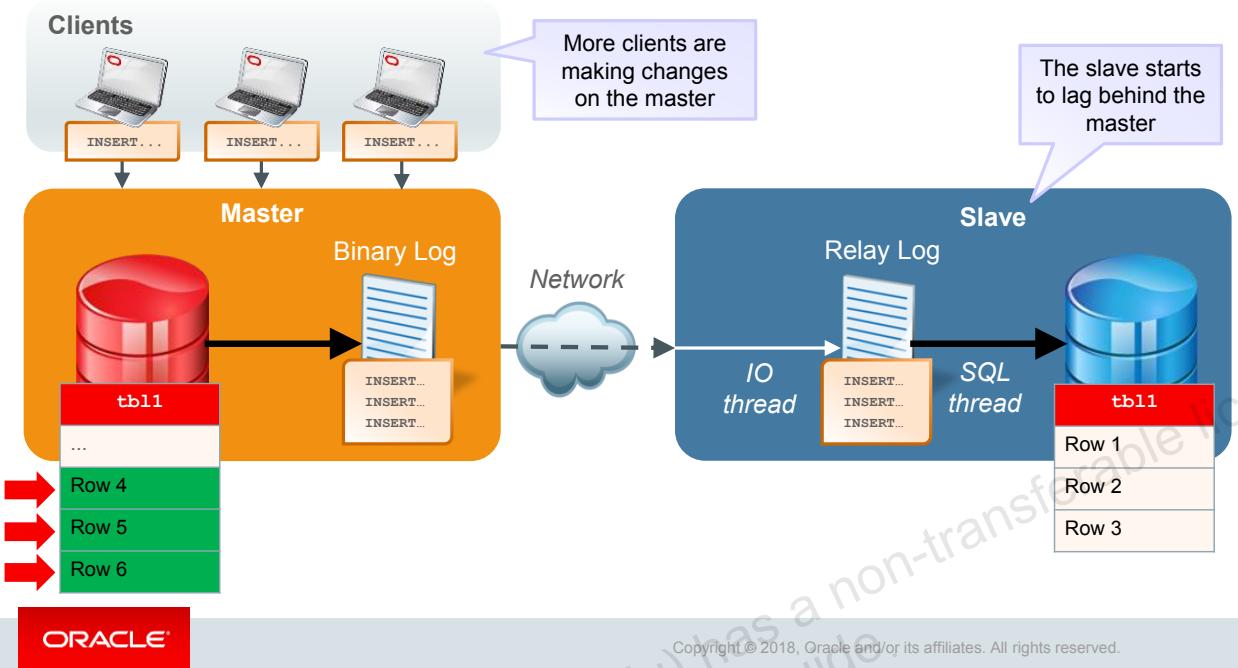
- Replication overview
- Understanding replication lag
- Diagnosing replication lag
- Resolving replication lag



ORACLE®

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

What Is Replication Lag?



In the diagram in the slide, more clients are connecting to the master and issuing data modification statements. The speed at which the slave can access those changes from the master's binary log and apply them starts to reduce. Unapplied changes build up in the slave's relay log and the slave's version of the database lags behind the master's.

Common Causes of Replication Lag

- *I/O thread lag* usually results from a slow network connection between a master and its slave(s)
 - Consider enabling `slave_compressed_protocol` to compress network traffic between the master and slave servers.
- *SQL thread lag* usually results from poorly-optimized queries:
 - Long-running transactions on the slave
 - Intensive I/O activity on the slave
 - No primary key on slave tables when using `ROW` or `MIXED` replication format



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

Quiz



Which of the following is the best definition of replication lag?

- a. A delay in the master being notified that a slave has applied a transaction
- b. The inability of a slave to clear the contents of its relay log
- c. The inability of a slave to connect to the master
- d. A delay in either the slave receiving or executing updates from the master, or both

ORACLE®

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

Answer: d

Topics

- Replication overview
- Understanding replication lag
- **Diagnosing replication lag**
- Resolving replication lag



ORACLE®

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

Diagnosing Replication Lag

The method you must use to diagnose replication lag depends on whether you are using standard binary logging or GTIDs:

- Standard binary logging: Compare the binary log file name and current position between the master and the slave(s).
- GTIDs: Compare the executed GTID set between the master and the slave(s).



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

Binary Log File Name and Position of the Master

Execute `SHOW MASTER STATUS` on the master to show the current log file name and position of the master.

- **File:** Current log file name
- **Position:** Current log file position

```
master> SHOW MASTER STATUS;
+-----+-----+-----+-----+
| File      | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+-----+-----+-----+-----+
| binlog.000002 | 121752008 |           |                 |                   |
+-----+-----+-----+-----+
```



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

Binary Log File Name and Position of the Slave

Execute `SHOW SLAVE STATUS` on the slave to show the current log file name and position of the slave.

- Status of the I/O thread:
`Master_Log_File` and
`Read_Master_Log_Pos`
- Status of the SQL thread:
`Relay_Master_Log_File` and `Exec_Master_Log_Pos`

```
slave> SHOW SLAVE STATUS\G
***** 1. row *****
Slave_IO_State: Waiting for master to send event
Master_Host: 127.0.0.1
Master_User: replication
Master_Port: 22808
Connect_Retry: 60
Master_Log_File: binlog.000002
Read_Master_Log_Pos: 121409852
Relay_Log_File: relaylog.000002
Relay_Log_Pos: 119819329
Relay_Master_Log_File: binlog.000002
Slave_IO_Running: Yes
Slave_SQL_Running: Yes
...
Exec_Master_Log_Pos: 120003004
Relay_Log_Space: 121226377
...
```

Status of the I/O thread

Status of the SQL thread



Comparing Binary Log File Name and Position

- Slave I/O thread behind replication master:

```
Bytes behind master = Position - Read_Master_Log_Pos  
= 121752008 - 121409852  
= 342156 bytes
```

- Slave SQL thread behind replication master

```
Bytes behind master = Position - Exec_Master_Log_Pos  
= 121752008 - 120003004  
= 1749004 bytes
```

- Slave SQL thread behind slave I/O thread

```
Bytes behind io      = Read_Master_Log_Pos - Exec_Master_Log_Pos  
= 121409852 - 120003004  
= 1406848 bytes
```



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

The sample outputs in the slide use the binary log file name and positions from the SHOW MASTER STATUS and SHOW SLAVE STATUS statements executed in the preceding slides to calculate the replication lag in terms of bytes.

While the above calculations are an accurate measure of the volume of binary log events that must be applied, it does not necessarily tell the whole picture of the lag in terms of time. In the above example, the I/O thread is only one transaction behind the replication master and the SQL thread is only one transaction behind the I/O thread. So while the SQL thread is more than 1.6MB behind the replication master, in reality the lag may be very short in terms of the time that it takes for updates applied on the replication master to be reflected on the slave.

If the binary log files between the master, the I/O thread, and the SQL thread are not the same, the size of the binary logs files must be taken into account. It is not possible from inside MySQL to determine the size of the binary logs, but provided all transactions are relatively small and no explicit flush of the binary logs have been made, the size can be estimated as the value of the max_binlog_size system variable (1 GB by default.)

Note: A transaction is never split across multiple binary log files, so a large transaction can cause a binary log file to be significantly larger than max_binlog_size.

Timing Replication Lag

- The Seconds_Behind_Master column in the output of SHOW SLAVE STATUS shows how far the replication slave is behind the master.
- It is not a reliable measure because it does not take into account:
 - The duration of long-running transactions
 - For example:
 - If a transaction takes one hour to execute on the master, Seconds_Behind_Master will be 3600 when the slave starts to execute the transaction.
 - If the same transaction takes another hour to execute on the slave, Seconds_Behind_Master will be 7200 by the time the SQL thread is ready to commit the transaction.
 - The SQL thread may then take only a second or two to catch up.
 - The amount of time the transaction spends in an idle state or performing temporary work on the master that is not required on the slave
 - In this instance, the transaction might take very little time to execute on the slave and Seconds_Behind_Master reverts to a small value almost immediately.



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

The value of Seconds_Behind_Master is found by calculating the difference between the current time on the slave and when the event started on the master. In the case where all transactions are fast, it is a reasonable measure of the lag. However, if there are long-running transactions, it might cause the lag to appear larger than it actually is. It is best to consider the Seconds_Behind_Master value over time.

Comparing GTID Sets

- Measure lag in GTID-based replication setups by monitoring the number of outstanding transactions
 - Use the `GTID_SUBTRACT()` function to calculate this value
- Consider the following metrics:
 - Executed GTIDs
 - Available on both master and slave
 - Shows which transactions have been executed on the instance
 - Retrieved GTIDs
 - Available only on the slave
 - Shows which transactions the I/O thread has received
 - Purged GTIDs
 - Available on both master and slave
 - Shows which transactions have been purged from the binary log
 - Only the transactions purged by the slave are relevant to detecting replication lag



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

Executed GTIDs

- From SHOW MASTER STATUS

```
+-----+-----+-----+-----+
| File | Position | Binlog_Do_DB | Binlog_Ignore_DB | Executed_Gtid_Set |
+-----+-----+-----+-----+
| binlog.000002 | 121752008 |           |                 | 96985d6f-2ebc-11e7-84df-08002715584a:1-134 |
+-----+-----+-----+-----+
```

- From SHOW SLAVE STATUS

```
...
Executed_Gtid_Set: 96985d6f-2ebc-11e7-84df-08002715584a:1-13
...
```

- From the global gtid_executed variable (for example, from the slave):

```
slave> SELECT @@global.gtid_executed;
+-----+
| @@global.gtid_executed          |
+-----+
| 96985d6f-2ebc-11e7-84df-08002715584a:1-132 |
+-----+
1 row in set (#.## sec)
```



Retrieved GTIDs

- From SHOW SLAVE STATUS

```
...  
Retrieved_Gtid_Set: 96985d6f-2ebc-11e7-84df-08002715584a:26-133  
...
```

- From the replication_connection_status table in the Performance Schema

```
slave> SELECT CHANNEL_NAME, RECEIVED_TRANSACTION_SET  
      -> FROM performance_schema.replication_connection_status;  
+-----+-----+  
| CHANNEL_NAME | RECEIVED_TRANSACTION_SET |  
+-----+-----+  
|           | 96985d6f-2ebc-11e7-84df-08002715584a:26-133 |  
+-----+-----+  
1 row in set (0.00 sec)
```



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

Purged GTIDs

From the global `gtid_purged` variable on the slave:

```
slave> SELECT @@global.gtid_purged;
+-----+
| @@global.gtid_purged           |
+-----+
| 96985d6f-2ebc-11e7-84df-08002715584a:1-25 |
+-----+
1 row in set (0.00 sec)
```



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

Example: I/O Thread Lag

```
slave> SELECT GTID_SUBTRACT(GTID_SUBTRACT('96985d6f-2ebc-11e7-84df-08002715584a:1-134', '96985d6f-2ebc-11e7-84df-08002715584a:26-133'), @@global.gtid_purged) AS MissingGTIDs;
+-----+
| MissingGTIDs |
+-----+
| 96985d6f-2ebc-11e7-84df-08002715584a:134 |
+-----+
1 row in set (#.## sec)
```



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

The `GTID_SUBTRACT()` function accepts two sets of global transaction IDs: *subset* and *set*. It returns only those GTIDs from *set* that are not in *subset*.

In the example in the slide, a single transaction (number 134 from the MySQL instance with `@@global.server_uuid= 96985d6f-2ebc-11e7-84df-08002715584a`) is outstanding. Two calls to `GTID_SUBTRACT()` are required as the previously purged GTIDs must also be subtracted from the GTIDs that have been executed on the replication master.

Example: SQL Thread Lag

```
slave> SELECT GTID_SUBTRACT(GTID_SUBTRACT('96985d6f-2ebc-11e7-84df-08002715584a:5-134', '96985d6f-2ebc-11e7-84df-08002715584a:26-132'), @@global.gtid_purged) AS MissingGTIDs;
+-----+
| MissingGTIDs |
+-----+
| 96985d6f-2ebc-11e7-84df-08002715584a:133-134 |
+-----+
1 row in set (#.## sec)
```



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

The example in the slide shows that two transactions are outstanding (numbers 133 to 134 from the MySQL instance with @@global.server_uuid = 96985d6f-2ebc-11e7-84df-08002715584a.)

Example: SQL Thread Lagging Behind I/O Thread

```
slave> SELECT GTID_SUBTRACT('96985d6f-2ebc-11e7-84df-08002715584a:5-133',
   '96985d6f-2ebc-11e7-84df-08002715584a:26-132') AS MissingGTIDs;
+-----+
| MissingGTIDs          |
+-----+
| 96985d6f-2ebc-11e7-84df-08002715584a:5-25:133 |
+-----+
1 row in set (#.## sec)
```

From SHOW SLAVE STATUS and/or executed_gtids

```
slave> SELECT GTID_SUBTRACT(RECEIVED_TRANSACTION_SET, @@global.gtid_executed) AS MissingGTIDs,
   -> IF(GTID_SUBTRACT(
   ->   RECEIVED_TRANSACTION_SET, @@global.gtid_executed) = '', 'YES', 'NO') AS CaughtUp
   -> FROM performance_schema.replication_connection_status;
+-----+-----+
| MissingGTIDs          | CaughtUp |
+-----+-----+
| 96985d6f-2ebc-11e7-84df-08002715584a:133 | NO      |
+-----+-----+
1 row in set (#.## sec)
```

From the Performance Schema replication_connection_status table

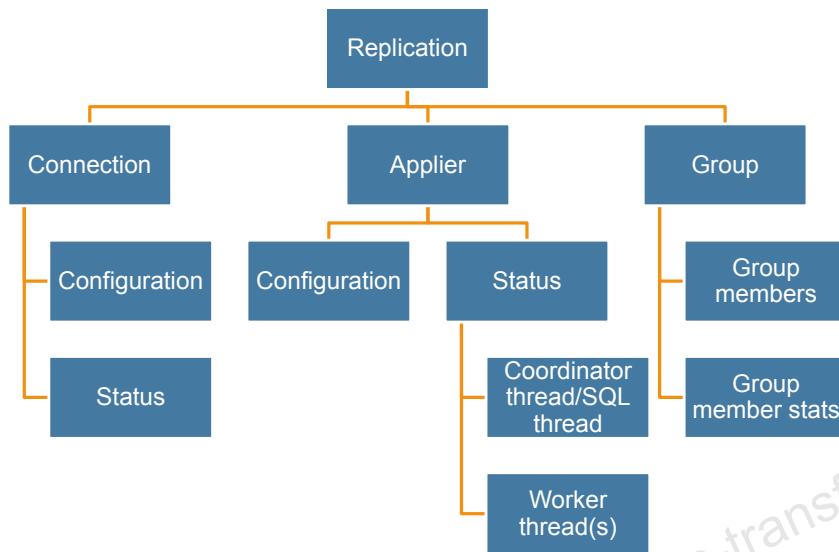


Copyright © 2018, Oracle and/or its affiliates. All rights reserved.

The first example in the slide requires only a single call to GTID_SUBTRACT() because both GTID sets originate from the same MySQL instance and therefore the purged GTIDs are irrelevant. It shows that a single transaction (number 133 from the MySQL instance with @@global.server_uuid = 96985d6f-2ebc-11e7-84df-08002715584a) has not yet been applied by the SQL thread.

The second example in the slide uses the Performance Schema replication_connection_status table to retrieve the same information, where the MissingGTIDs column value represents any transactions that have been received by the I/O thread but not yet executed by the SQL thread, and CaughtUp returns YES if the SQL thread has caught up with the I/O and NO otherwise.

Performance Schema Replication Table Hierarchy



ORACLE®

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

Performance Schema Replication Tables

- Tables that contain information about the connection of the slave server to the master:
 - replication_connection_configuration
 - replication_connection_status
- Tables containing general (not thread-specific) information about the SQL thread:
 - replication_applier_configuration
 - replication_applier_status
- Tables that contain thread-specific information in multithreaded slaves
 - replication_applier_status_by_coordinator
 - replication_applier_status_by_worker
- Tables that contain information about group members in MySQL Group Replication
 - replication_group_members
 - replication_group_member_stats



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

The Performance Schema replication tables contain information most relevant to GTID-based replication setups and not all the information from SHOW SLAVE STATUS – such as log file names and positions – is available. They are especially useful for programmatic access and preserving historical data.

Replication Support in MySQL Enterprise Monitor



ORACLE®

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

MySQL Enterprise Monitor provides a graphical display of replicated systems. It shows the status of any server participating in the replication, including maximum replication delay and the status of the slave I/O and SQL threads and enables you to drill down to retrieve more detailed information. It also provides support for managing MySQL Group Replication setups, as shown in the slide.

Quiz



Which of the following sources of information are not useful in a GTID-based replication setup? (Select all that apply.)

- a. The RECEIVED_TRANSACTION_SET column in the performance_schema.replication_connection_status table
- b. The value of Executed_Gtid_Set from SHOW MASTER STATUS
- c. The value of Read_Master_Log_Pos from SHOW SLAVE STATUS
- d. The innodb_flush_log_at_trx_commit option

ORACLE

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

Answer: c, d

Topics

- Replication overview
- Understanding replication lag
- Diagnosing replication lag
- Resolving replication lag



ORACLE®

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

Resolving I/O Thread Lag

- Common symptoms:
 - There is only one relay log file on the slave, but data is slow to replicate.
- Causes:
 - I/O thread lag usually involves a poor network connection between the master and slave(s).
- Consider:
 - Upgrading network hardware
 - Enabling `slave_compressed_protocol` to compress network traffic on the slave/master protocol
 - Both slave and master must support compression
 - Disabling `sync_relay_log`, `sync_relay_log_info`, and `sync_master_info`
 - When enabled, the slave frequently synchronizes data to disk
 - Should only be enabled if your disk has a battery-backed cache



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

Enabling `sync_relay_log`, `sync_relay_log_info`, and `sync_master_info` forces the slave to synchronize data to disk on every event. For example `sync_relay_log` ensures one write to the relay log per statement if `autocommit` is enabled, and one write per transaction otherwise. Enabling those options results in poor performance unless the disk has a battery-backed cache, in which case synchronization is very fast.

Resolving SQL Thread Lag

- Common symptoms:
 - There are multiple relay log files on the slave.
- Causes:
 - The SQL thread cannot keep up with the changes coming in via the I/O thread and there are several possible reasons for this.
- Consider:
 - Upgrading the slave hardware, if it is inferior to the master.
 - Enabling multithreading on the slave server.
 - Ensuring that the indexes on slave tables are identical to those on the master.
 - Auditing, and if necessary, rewriting poorly-performing queries.
 - Relaxing durability by setting `innodb_flush_log_at_trx_commit = 0`
 - If MySQL crashes, you could lose up to one second's worth of data on the slave
 - Disabling the query cache on the slave.



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

If you have a large query cache, it can cause transaction commit to take a long time, because every commit needs to scan the query cache to check for entries it needs to purge. For this, and many other reasons, the query cache is deprecated, and Oracle discourages its use.

Summary



In this lesson, you should have learned how to:

- List possible causes of replication lag
- State the difference between I/O thread lag and SQL thread lag
- Identify replication lag using binary log file names and positions
- Identify replication lag in GTID-based replication
- Resolve replication lag

ORACLE®

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

Practices

- 12-1: Diagnosing Replication Lag by Using Binary Log File Name and Position
- 12-2: Diagnosing Replication Lag by Using GTID Sets



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

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

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

Conclusion



MySQL™

ORACLE®

Course Goals

After completing this course, you should be able to:

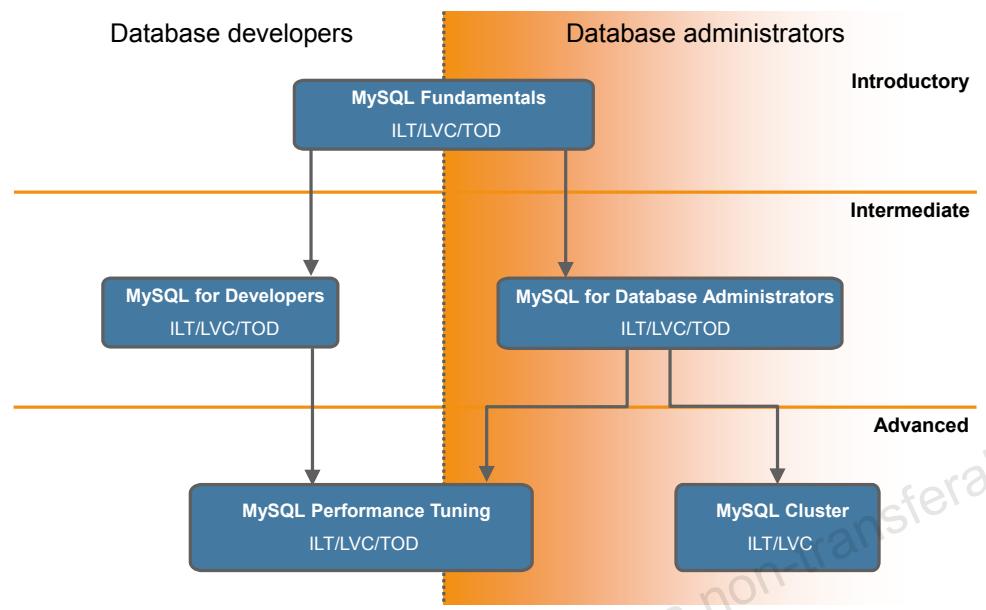
- Describe performance tuning concepts
- List factors that affect performance
- Use a range of performance tuning tools
- Configure and use the Performance Schema
- Tune the MySQL server instance
- Design a schema for optimal performance
- Understand how MySQL optimizes queries
- Identify and fix slow queries
- Diagnose and resolve common performance issues
- Identify and fix replication lag



ORACLE®

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

Oracle University: MySQL Training



ORACLE

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

Training Formats

- **Instructor-Led Training (ILT):** Delivered in a classroom with instructor and students present at the same location and time
- **Live Virtual Class (LVC):** Delivered by using video and audio through a web-based delivery system (WebEx) in which geographically distributed instructor and students participate, interact, and collaborate in a virtual class environment
- **Training On Demand (TOD):** On-demand training that takes traditional classroom training (complete with all classroom content including lectures, white boarding, and practice videos) and makes it available in a video-based, online format so that you can start your customized training at your convenience

For full details about MySQL training options, go to <http://education.oracle.com/mysql>.

MySQL Websites

- <http://www.mysql.com> includes:
 - Product information
 - Services (Training, Certification, Consulting, and Support)
 - White papers, webinars, and other resources
 - MySQL Enterprise Edition downloads (trial versions)
- <http://dev.mysql.com> includes:
 - Developer Zone (forums, articles, Planet MySQL, and more)
 - Documentation
 - Downloads
- <https://github.com/mysql>
 - Source code for MySQL Server and other MySQL products



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

Download MySQL Community Edition general availability (GA) and development releases from the <http://dev.mysql.com> website. This site also hosts the online documentation, which is an extremely valuable resource for both beginners and expert users of MySQL, and includes several example databases. Download trial versions of MySQL Enterprise Edition software, view detailed product information, and find out more about Oracle MySQL services at <http://www.mysql.com>.

Your Evaluation

- Courses are continually updated, and so your feedback is invaluable.
- Thank you for taking the time to give your opinions.



ORACLE®

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

Thank You

- Congratulations on completing this course!
- Your attendance and participation are appreciated.
- For training and contact information, see the Oracle University website at <http://www.oracle.com/education>.



ORACLE®

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

Q&A Session

- Questions and answers
- Questions after class
 - Get answers from the online reference manual at <http://dev.mysql.com/doc/mysql/en/faqs.html>.
- Example databases
 - Download the world, employee and other sample databases from <http://dev.mysql.com/doc/index-other.html>.



ORACLE®

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

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

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